Grant Agreement no. 215056

Title:	***WP5 PICOS PHASE 2 Platform Description document***
Editor:	*Jean-Philippe Caradec (HPF)*
Reviewers:	*Christian Kahl, Johann Wolfgang Goethe-Universität, Germany (GUF)*
	Marek Kumpošt, Masaryk University, Czech Republic (BRNO)
Identifier:	*D.5.2.b*
Type:	*Deliverable*
Version:	*1.2*
Date:	*November 8th, 2010*
Status:	*Final*
Class:	*Public*

Summary

This document presents a high-level description of the PICOS platform developments for Phase 2.

It describes the required features for the 2nd version of the platform and their implementation to support the anglers Community

It describes the new platform feature set and their implementation to support the Gamers Community.

A chapter articulates the new design with the PICOS phase 1architecture and describes the new platform components. Examples of component Call flows are provided for better clarity.

At last the document provides high level information on the development process, the non-regression test suite, the platform software package, its dependencies on 3rd party open source software, its installation and configuration.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

Members of the PICOS consortium:

Johann Wolfgang Goethe-Universität (Coordinator)	Germany
Hewlett-Packard Laboratories Bristol	United Kingdom
Hewlett-Packard Centre de Competence France	France
Universidad de Málaga	Spain
Center for Usability Research & Engineering	Austria
Katholieke Universiteit Leuven	Belgium
IT-Objects GmbH.	Germany
Atos Origin	Spain
Deutsche Telekom AG	Germany
Leibniz Institute of Marine Sciences	Germany
Masaryk University	Czech Republic

The PICOS Deliverable Series

○ D2.1 Taxonomy	July 2008
○ D2.2 Categorisation of Communities	July 2008
○ D2.3 Contextual Framework	November 2008
○ D2.4 Requirements	November 2008
○ D3.1.1 Trust and Privacy Assurance for the Platform Design	April 2009
○ D3.2.1 Trust and Privacy Assurance of the Platform Prototype	November 2009
○ D3.3.1 Trust and Privacy Assurance of the Community Prototype	January 2010
○ D3.4.1 A summary of PICOS WP3 sub-phase 3.1 deliverables	September 2010
○ D4.1 Platform Architecture and Design v1	March 2009
○ D4.2 Platform Architecture and Design v2	September 2010
○ D5.1 Platform description document v1	October 2009
○ D5.2a Platform Prototype 2	May 2010

The PICOS project receives research funding from the Community's Seventh Framework Programme.

- D6.1 Community Application Prototype 1 December 2009
- D6.2a First Community Application Prototype v2 April 2010
- D6.2b Second Community Application Prototype v2 October 2010
- D7.1 a Trial Design Document December 2009
- D7.2a First Community Prototype: Lab and Field Test Report February 2010
- D7.1b Trial design document 2 October 2010
- D8.1 Legal, economic and technical evaluation of the first platform and community prototype April 2010
- D9.1 Web Presence February 2008
- D9.2.1 Exploitation Planning April 2009
- D9.3.1 Dissemination Planning April 2009
- D9.2.2 Exploitation Plan 2 March 2010
- D9.3.2 Dissemination Report 2 March 2010

The PICOS documents are all available from the project website located at http://PICOS-project.eu.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

The PICOS Deliverable Series

Vision and Objectives of PICOS

With the emergence of services for professional and private on-line collaboration via the Internet, many European citizens spend work and leisure time in on-line communities. Users consciously eave private information; they may also leave personalized traces they are unaware of. The objective of the project is to advance the state of the art in technologies that provide privacy-enhanced identity and trust management features within complex community-supporting services that are built on Next Generation Networks and delivered by multiple communication service providers. The approach taken by the project is to research, develop, build trial and evaluate an open, privacy-respecting, trust-enabling identity management platform that supports the provision of community services by mobile communication service providers.

The following PICOS materials are available from the project website http://www.PICOS-project.eu.

PICOS documentation

Slide presentations, press releases, and further public documents that outline the project objectives, approach, and expected results;

PICOS global work plan providing an excerpt of the contract with the European Commission.

PICOS results

PICOS Foundation for the technical work in PICOS is built by the categorization of communities, a common taxonomy, requirements, and a contextual framework for the PICOS platform research and development;

PICOS Platform Architecture and Design provides the basis of the PICOS identity management platform;

PICOS Platform Prototype demonstrates the provision of state-of-the-art privacy and trust technology to leisure and business community applications;

Community Application Prototype is built and used to validate the concepts of the platform architecture and design and their acceptability by covering scenarios of private and professional communities;

PICOS Trials validate the acceptability of the PICOS concepts and approach chosen from the end-user point of view;

PICOS Evaluations assess the prototypes from a technical, legal and social-economic perspective and result in conclusions and policy recommendations;

The PICOS project receives research funding from the Community's Seventh Framework Programme.

PICOS-related scientific publications produced within the scope of the project.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

Charter

Objectives of WP5

The objectives of this WP are to provide prototype implementations of the PICOS identity management platform which will be used within the community applications prototypes. These platform prototypes will address the requirements of all PICOS platform stake-holders, including community developers (responsible for defining different classes of communities), application developers (responsible for defining new community applications), community members, PICOS platform administrators and network operators. As such, the scope of these prototype implementations is expected to include the core functionality of the PICOS platform and APIs for developers of different classes of communities and for developers of community-based services, interfaces with network-based services (e.g., location servers, presence servers, messaging servers, etc), and administrative interfaces for platform management, for management of the communities that are hosted by the PICOS platform and for members of those communities. The output of this work will be working prototypes of the PICOS environment, as well as technical documentation of the different platform interfaces.

Description of work

This WP has taken as input the architecture and design document (D4.1) from WP4. From this definition, the scope of the prototype implementation has been defined. Additional requirements derived from the nature of the target user trials and target communities have also been taken into account when defining the overall scope of the prototype.
The prototype implementation has been composed of 6 distinct areas:

- Core PICOS platform (functional modules for identity management, community management, privacy and trust management)
- Community APIs for community designers and developers of community applications
- Interfaces to network-based services and resources (e.g. location servers, presence servers, messaging services, communication services)
- Interfaces to business support systems
- Administrative interface for PICOS platform management
- Portal interface for community administrators and community members

The output from this work has been a working prototype of major aspects of the PICOS platform system, as well as technical documentation of the programmatic interfaces.
To demonstrate its community agnosticism, two community applications have been developed on top of the WP5 platform. As a result, the two applications (mostly the mobile client application developed by the WP6 team) work in front of the same WP5 phase 2 platform.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

Task 5.1 Platform prototype 1 for Anglers V1 community

This task has taken as input the architecture and design document from task 4.1 and has delivered the first working prototype of the PICOS platform. The output of this task has been used by WP6 to construct community application prototypes and by WP7 to validate PICOS concepts and their acceptability from a user experience viewpoint. With these goals, the primary focus of this first prototype was on core functions of the PICOS platform, including the tools and interfaces that are used to create and manage communities (including the privacy and trust aspects) and to access community services.

Task 5.2 PICOS Platform Phase 2

It has been decided at the beginning of the project to exemplify the PICOS value proposition by supporting two different Communities. The major set of requirements for the phase 2 platform came from the support of Gamers Community. These requirements have been translated into generic and community agnostic new platform features.
The specification and design description of the platform phase 2 is the aim of the current document.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

Foreword

Deliverable of the WP5.2b various tasks is a collective work by the WP5 platform prototype development team, whose members are listed below.

We are very grateful to the members of PICOS who prepared earlier deliverables, which were key foundation work for the platform prototype development.

Many thanks to the PICOS WP5 team.

The Platform prototype development team

ATOS, GUF, HPF, HPL, IFM-Geomar, ITO, DT and UMA

PICOS D5.2b reviewers

Christian Kahl, Johann Wolfgang Goethe-Universität, Germany (GUF)

Marek Kumpošt, Masaryk University, Czech Republic (BRNO)

The PICOS project receives research funding from the Community's Seventh Framework Programme.

Table of Contents

The PICOS project receives research funding from the Community's Seventh Framework Programme.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

Table of Figures

Table of Tables

List of Acronyms

Abbr	*Abbreviation*
AES	*Advanced Encryption Standard*
API	*Application Programming Interface*
CA	*Certification Authority*
CS	*Client Server*
CSCF	*Call Session Control Function*
DRM	*Digital Rights Management*
DSA	*Digital Signature Algorithm*
Dx.y	*[PICOS] Deliverable: Work Package x, Deliverable y*
EPAL	*Enterprise Privacy Authorisation Language*
FTMGS	*Fair Traceable Multi-Group Signature*
GPS	*Global Positioning System*
GSM	*Global System for Mobile communications (originally Groupe Spécial Mobile)*
HTTP	*Hypertext Transfer Protocol*
ICT	*Information and Communication Technology*
ID	*Identity (also Identifier)*
IdM	*Identity Management (also Identity Manager)*
IM	*Instant Messaging*
IPSec	*Internet Protocol Securlty*
ISO	*International Organization for Standardization*
JSR	*Java Specification Request*
MAC	*Media Access Control*
OSI	*Open Systems Interconnection*
P2P	*Peer-to-peer*
P3P	*Privacy for Platform Preferences*
PA	*Privacy Advisor*

The PICOS project receives research funding from the Community's Seventh Framework Programme.

P-CSCF	*Proxy - Call Session Control Function*
pdf	*[Trademark] Portable Document Format*
PF	*PICOS Feature*
PICOS	*Privacy and Identity for Community Services*
PP	*PICOS Principle*
proxyWs	*Proxy Web Service part of the RPC gateway*
PUA	*Presence User Agent*
PUC	*PICOS Use Case*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
RSA	*Rivest, Shamir and Adleman*
S/MIME	*Secure / Multipurpose Internet Mail Extensions*
SDK	*Software Development Kit*
S-HTTP	*Secure Hypertext Transfer Protocol*
SIP	*Session Initiation Protocol*
SLA	*Service Level Agreement*
SSO	*Single Sign-On*
TLS	*Transport Layer Securely*
TOR	*The Onion Router*
TSA	*Time Stand Authority*
TTP	*Trusted Third Party*
URL	*Uniform Resource Identifier*
Wi-Fi	*[Trademark] Wireless local area network*
WP	*Work Package*
ZKP	*Zero Knowledge Proof*

The PICOS project receives research funding from the Community's Seventh Framework Programme.

1. Platform Phase 2 Project

1.1. Introduction

The PICOS Phase 2 has been defined as two separate and parallel development threads:

- The first thread has been defined to improve the usability of the Anglers application V1 and prepare the Anglers community trial in May 2010 with a new Anglers V2 application version.
- The second thread has been conducted to fulfil the Gamers Community requirements and prepare the Gamers trial in October 2010. The second thread has also taken into account the major feedbacks of the Anglers trial to globally improve the PICOS community services

1.2. Anglers V2 community support

In order to prepare the anglers trial in May, propositions have been made to improve the complete application (client + platform) mainly in terms of usability.
Most of the work has been performed on the client side to optimize the usability of the client features.

On the platform side, some development has been executed on:

- Support of special characters in PICOS application for language such as German or French (special characters not defined in ASCII encoding schemas).
- Support of multiple image types for content upload and thumbnail generation.
- Various fixes for the existing phase 1 functionality.

Due to the limited time for preparing the anglers V2 version (used for the trial), some of the identified enhancements, that were requesting too much time for implementation, have been integrated into the Gamers release. Since these enhancements were generic and not particularly related to the anglers community support, they could benefit to the Gamers application as well and be user tested during teh Gamers field trial.

1.3. Gamers community support

Supporting a new community to demonstrate the PICOS value proposition in various contexts has been part of the major PICOS objectives. The phase 2 PICOS project has developed an appropriate answer to the Gamers community using traditional project management for building:

- A list of gamers community requirements.
- A prioritization of the requirements.
- A team proposal to address these requirements.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

- A specification of the proposed solutions.
- A design, implementation and integration phase using a phase approach.

The rest of the document is describing the Gamers platform release.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

2. Evolutions of the Platform features for Gamers Community support

2.1. Major evolutions

The PICOS phase 1 Platform has been designed to be community agnostic. The phase 2 version keeps that design criteria considering that all the additional features can be used by any community and are not specific to (but important for) the Gamers community.

A lot of new capabilities are focusing on extending the use of policies to handle finer control on any object of the community and its community members.
The PICOS Policy management is based on a central Policy Engine and on components that request authorization for executing actions on Objects and enforce policy Engine response.

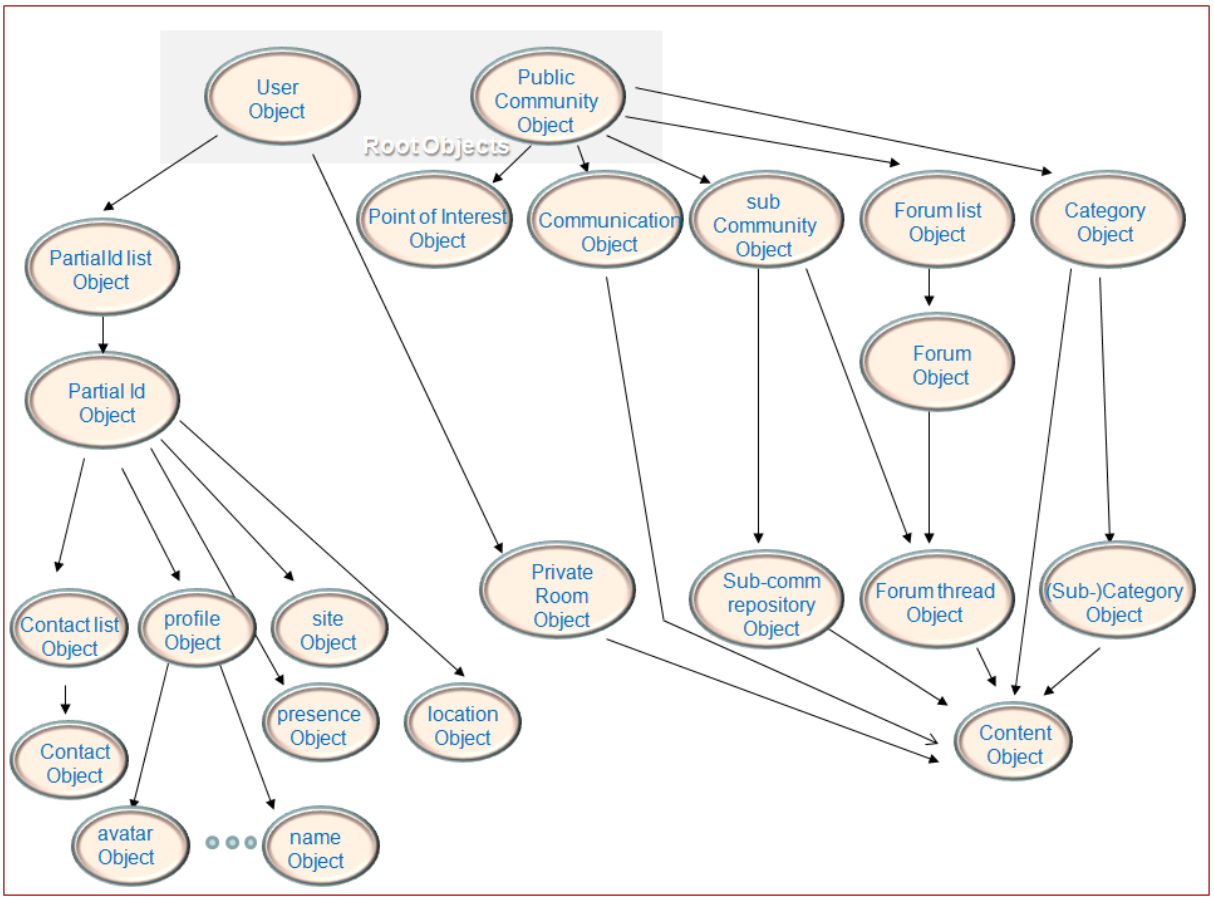
The Phase 1 Policy engine is re-used and enhanced to address new use cases that were developed for the Gamers community support.

The next 2 sections recap the fundamentals of the PICOS platform add their extensions for the Gamers Community support.

2.1.1. The PICOS Object model

The PICOS platform is defined as a set of Objects with attributes and children Objects. There are two root Objects which are the User Object and the public-community Object.

The PICOS project receives research funding from the Community's Seventh Framework Programme.



The user Object is the root element for any attribute that is user related.
The public-community Object is the root element for any Object that describes the overall community besides the member of the community.
Conventions have been defined to describe an Object by providing the complete "hierarchical path up to the object itself. The path describes a context to make the Object unique. Some objects are unique and are defined by a name type other can be instantiated and are referenced by a name type and an instance id.

Examples of PICOS Object descriptions

PICOS Object	Object path description
Particular Forum of the community	public-community.forum(id)
Particular content of a specific category	public-community.category(id1).content(id2)

Avatar attribute of a particular identity of a specific user	User(id3).partialId-List().partialId(id4).user-profile().avatar
Particular thread of the sub-community forum	public-community.sub-community(id5).forum().thread(id6)

2.1.2. The PICOS Policy Model

The policy engine is in charge of storing rules attached to various Objects or attribute of Objects as well as evaluate user actions based on the set of rules. It is typically a generic rule engine that embeds intelligence to evaluate rules and deliver a response (status) on the required action.

Each component is responsible for asking the policy manager to evaluate action on resources like user attributes (privacy) as well as community resources for user privilege. It is the caller component responsibility to enforce the response sent back by the policy manager.

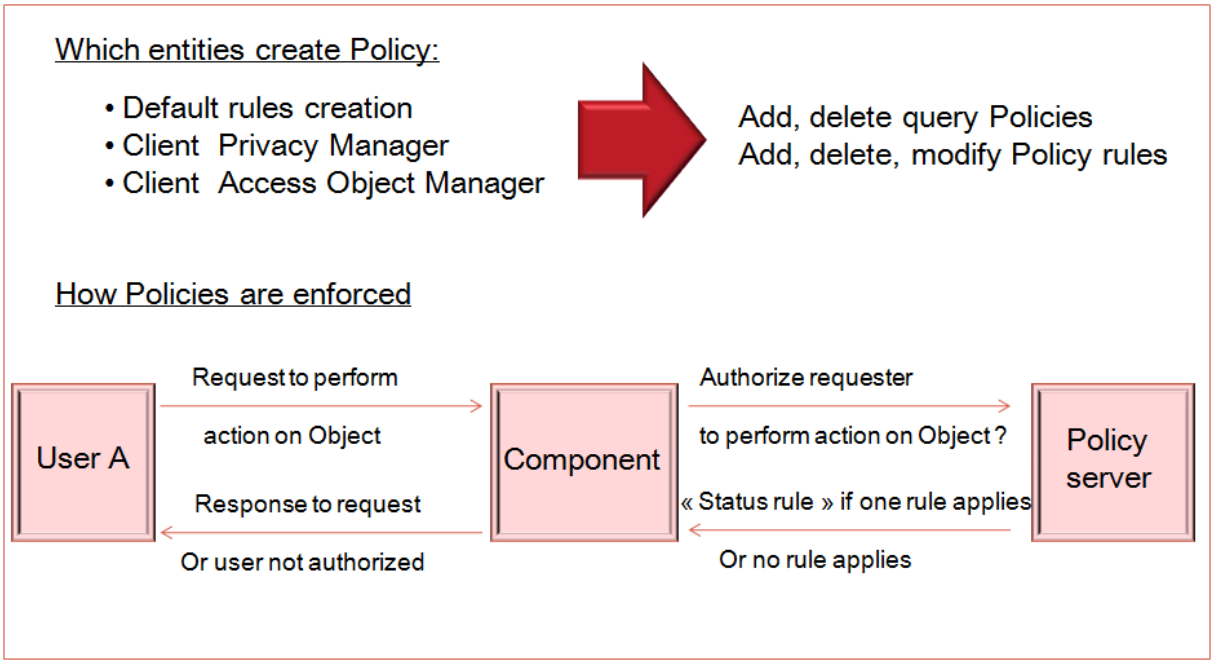


Figure 1: General mechanism for policy management and enforcement.

Managing user attribute privacy

User privacy management is the ability to control access to the User or PartialId Object attributes such as presence, user profile or particular attributes of the user profile**,** contact list as well as possibly content generated by the user.

The user privacy management is based on a set of policies that are either provisioned during the community creation (known as default policies), during the user registration, during the partial identity creation or when the End user decided to customize their own rules.

The platform is in charge of storing these policy rules attached to a specific attribute of the user and enforcing them when a User decides to perform actions on a particular resource.

Managing user privileges

Each member of the public community has specific privileges to access the different attributes of the public community objects. The privileges are context dependent and are mostly managed via the extended notion of identity role in these contexts.

The current contexts where roles apply as attributes are:

- public community (admin, member)
- sub-community (admin, member)
- forum (moderators contributor)
- Content (Owner)
- Point Of Interest (Owner)
- communication (creator, participant)

The PICOS platform is provisioned with default policy rules that describe which role can do which action in for which resource. The platform offers the ability to have specific privileges for a defined role.

The current default policies restrict access of user attributes to the attribute owner. These rules can be superseded by customized rules to selectively start opening the access to personal attributes.

User privilege will then be evaluated when an action on a particular resource is requested. There is no global authorization entity in the PICOS platform so platform components that are responsible for the function to be performed must enforce the policy Engine response.

Policy Model

	Major definitions
PICOS Resources	Any Object or attribute of an object managed by PICOS.

Policies	Set of rules attaches to a PICOS Resource.
Rule	A rule is defined as a set of conditions to be validated for a set of actions to perform on the resource. (example " Userxxxx is not allow to read " then we associate that rule to a resource like the presence attribute of a User (yyy) → then we have a policy : Userxxx is not allowed to read the presence attribute of the user (yyyy).
Rule conditions	Multiple independent conditions can be defined in a rule. As a condition is evaluated, its result must be true or false. All conditions of a rule must provide a result to true to validate the whole rule. The supported conditions are: identity of the requester(s) (who can perform the action) reputation of the requester date validity site

Table 1: Major Policy terms and definitions

Identity conditions

The table below describes all the possible identities that can be specified in an identity condition.

Possible Identity conditions	Description
No identity conditions	All members of the community
Id[]	One of multiple ids which can be either rootIds or partialIds or a mix
subCommunityId	All Members of a particular sub-community
subCommunityId→ role	A specific role of a particular sub-community
publicCommunity→ role	A specific role of the public community
Forum→ role	A specific role of the forum

Communication→ role	A specific role during a communication with a group

Table 2: Possible identity role in community Groups and Objects

A single identity condition can contains multiple types of user identity (groups, list of users, role) . When evaluated, the requester identity must comply with at least one type of user identity..("OR" condition). If these different types of user identities are set in separate conditions, the operand "AND" between conditions applies.

Reputation conditions

The possible reputation conditions that are supported:

- Requester reputation greater than a value.
- Requester reputation less than a value.
- Requester reputation inside a range.

Validity conditions

The possible validity conditions that are supported by the platform when setting or evaluating a policy are:

- Current date greater than a date.
- Current date less than a date.
- Current date inside a range.
- Current date inside a range that is recurrent. The frequency can be defined.

The date format is the one defined in xsd:dateTime also compliant with ISO 8601 (http://www.w3schools.com/Schema/schema_dtypes_date.asp).

An example of the dateTime structure: `2002-05-30T09:30:10`

The frequency is defined using the type xsd:duration (same link).

An example of the duration structure: `P5Y2M10DT15H.`

The example above indicates a period of five years, two months, 10 days, and 15 hours.

Site conditions

A site condition applies when the owner of the defined resource is located at a particular site. The site definitions are private to an identity of a User.

Rule action

- The action name is defined as a string, any "verb" that characterises the required action is stored
- The action result is defined as an enumeration type composed of:
  - allow: the action operated by the "who" on the resource can be done.
  - disallow: the action operated by the "who" on the resource cannot be done.
  - ask once: please ask the end user for permission.
  - ask always: please always ask the owner of the object before executing the action.

The rule action has evolved to add extra parameters that characterize the response to the request.

Example: the rule is: allow access to location attribute of an identity but with a weak precision. The "weak precision" is defined as an extra parameter that characterize the precision of the location information that should be returned when requested.

The definition of action parameter is defined in a generic way at the policy engine level and its usage depends on the resource (Object/attribute), the rule is attached to.

Evaluating a request (authorization)

When a authorization request is received (with the following parameters: requester, resource, action)), the policy engine starts by checking rules attached to the last level of the resource description. If multiple rules are defined, the engine takes the latest defined. If no rule applies, it checks the level above and so one up to the root Object.

EX: User A wants to read the avatar attribute of User B.

Hierachical Checked resources	Description
User(B).partialId-List().partialId(id4).user-profile().avatar	Check if User A can read the avatar of the identity id4
User(B).partialId-List().partialId(id4).user-profile()	Check if User A can read the whole user B profile of the identity id4
User(B).partialId-List().partialId(id4)	Check if User A can read all the user B attributes of the identity id4
User(B).partialId-List().partialId	check if User A can read the whole user

	B attributes of all identities

Table 3: Example of hierarchical policy evaluation

The hierarchical approach allows the definition of generic rules that can apply to multiple children attributes.

Note that multiple rules can be associated to the same resource. The last defined resource is always the one that is evaluated first and supersede any rule defined "before" or at higher level of the resource description

2.2. Extending the platform

A lot of additional capabilities have been implemented in the platform in order to offer a finer control on who, when and how user and community objects are created, accessed, modified or deleted.

2.2.1. Ability to share Contact Lists

The contact-List is an attribute of identities of the community member and, as such, rules can be associated to it.

By default the Contact-List attribute is private, meaning that only the owner of the Contact-List can access the attribute.
A community member has now the ability to retrieve the contact-List of one of his contact and optionally store the received contact as his own contacts. This is achieved by defining a rule that is associated to the Contact-List attribute of a particular identity (partialId). The rule will allow a set of identities of users to access the attribute or may require permission when the attribute is accessed (read).
Note that if a user has access to somebody's Contact-List, he can get it, extract some contact to make them become his own contacts but cannot share somebody's Contact-List as a whole with other members.

Example

I have a Contact-List that I want to share with some of my contacts. I create a rule that I associate to the Contact-List attribute of one of my identities. This rule contains an identity condition where I define the list of identities that is allowed to see my Contact-List attribute and actions/statuses to allow (status) read (action) of the attribute;
When the attribute is accessed (action=read) , the rule is evaluated and the identity condition is checked. In that case if the requester is part of the identity list, the condition is met and the rule can apply (action= read => status= allow). The requester can read my Contact-List.

2.2.2. Access restriction to Public-Community Objects

The PICOS object model has two root Objects: the User Object and the public-community Object.

Default policies are defined for User Objects and public-community Objects. The Phase 1 platform has implemented the ability to define different policies per instances of the User Objects and attributes like a specific identity of a particular User. The Phase 2 offers the ability to define custom access rules for public-community Objects and attributes as well: I push a new content in a public content repository and I apply a rule to that content so that only my contact can see/read the content or to specify that the content is accessible during a period of time.

The policy engine evaluates rules whereas the servers, in charge of managing the request, enforce the policy engine decision. The policy engine enforcement has been added for the following objects:

Community Objects	Resource description
Public community repository	public-community. category(id)
Public community Content	public-community. category(id).content(id)
Public community Forum	public-community. forum(id).
Public community Forum thread	public-community. forum(id).forum-thread(id)
Public-community Point Of Interest	public-community. poi(id)
Sub-community	public-community.sub-community(id).
Sub-community Forum	public-community.sub-community(id).forum().
Sub-community thread	public-community.sub-community(id).forum().thread(id)
Sub-community content	public-community.repository(id).content(id)

Table 4: New community Objects that support instantiation for Policy definition

Note that any function that list a set of objects (getCategoryContentList, getForumList, ...) takes into account the possible "per object " access rules to only send back the list of Objects that the requester is allowed to see.

Example:

I create a Point Of Interest (POI) that I want to open to some of my contacts. I create a rule that I associate to the POI attributes. This rule contains an identity condition where I define the list of identities that is allowed to see the POI attribute and actions/statuses to allow (status) read (action) of the attribute. I associate the rule to that specific POI (the resource (the POI) description contains the id of the POI (public-community.poi(id)). The rule is only valid for that POI.
When the POI attribute is accessed (action=read), the rule is evaluated and the identity condition is checked. In that case if the requester is part of the identity list, the condition is met and the rule can apply (action= read => status= allow). The requester can read the created POI.

2.2.3. Enhance policies with new date condition

When a rule is created, an optional condition related to Date can be inserted. This is mainly used to prepare categories / content/forum creation and make the object available starting from a particular date or during a limited amount of time. A single rule can then contain condition on identify (who is involved in this rule) and date condition (when this rule applies)

As defined in the previous chapter, the date condition supports a frequency parameter (every week-end), a single date parameter and/or a period of time parameter (inside or outside the period of time).

For a single rule, multiple date conditions can be defined implying that, at least, one condition must match.

Example:

I create a category to store content that will be used in a Game. I want to allow access to the category only at a specific date and during 3 days. I will then create multiple rules that will be associated to newly created Category:

- Rule 0: Default rule is: *member of the public community can read the category*
- Rule1: Insert a rule to supersede the previous one: *member of the public community cannot read the category*
- Rule 2: Then add the rule with date condition: *member of the public community can read the category if the current date is included in the defined period*

If a user lists the available categories, there will be a read access to the "newly created" category and rule 2 will be evaluated. If the date is included in the defined period the rule 2 is valid and then the created category is displayed in the list. If not, the rule 2 date condition is not valid so rule 2 cannot be used. Rule 1 is then evaluated and access is not allowed so that the category is not listed in the available categories

2.2.4. Enhance policies with new site condition

After defining the "who" and "when" conditions, a site condition ("where") can also be defined. The site condition matches if the owner of the object is located at a specific location. The site is privately defined by the owner of the resource and referenced in the rule definition:

Example:

As described in "Figure 2: Use case for site condition in Policy creation", I want to allow others to see my location attributes only when I am at "home"

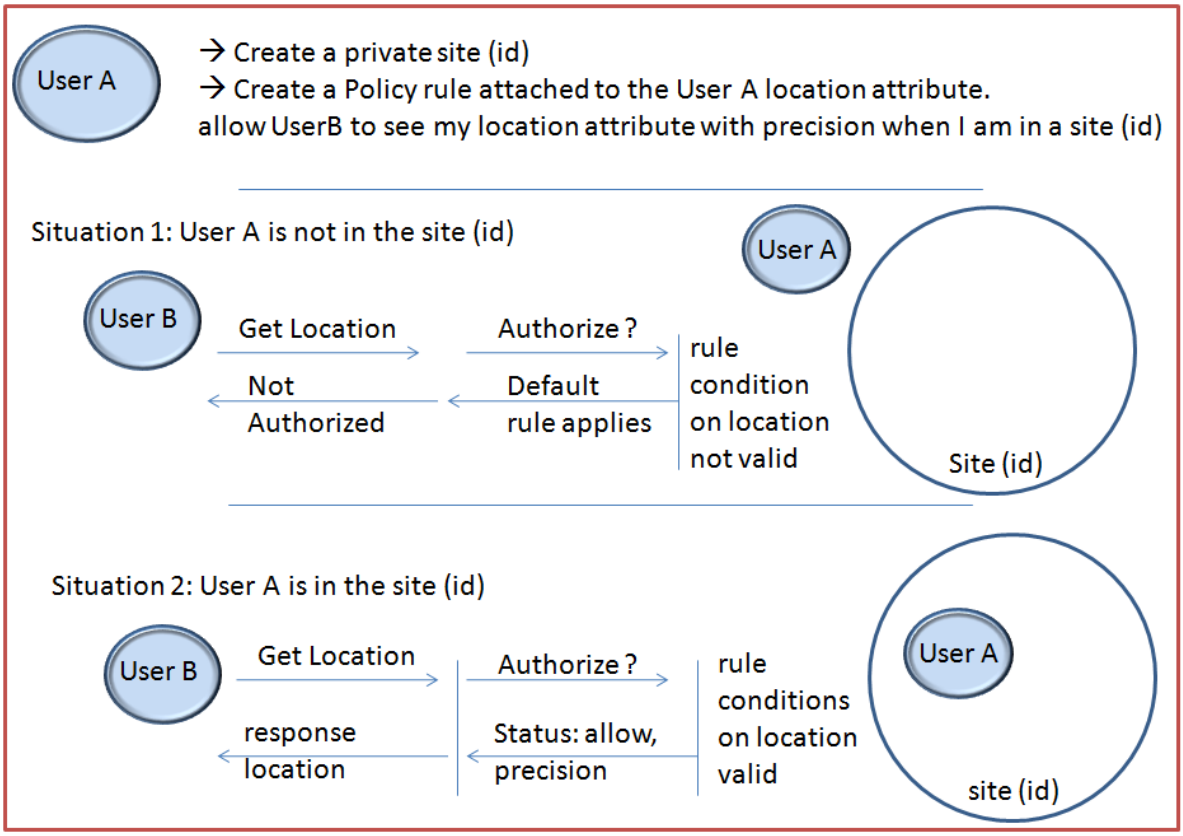


Figure 2: Use case for site condition in Policy creation

I define the "home" site (GPS location for the center of the site and a size) and I create a rule by inserting the site reference(id) as the site condition.
If User B wants to access User A location, the policy engine check rules attached to User B location attributes, detects that there is a site condition defined with the site "home" and evaluate if User A is in the site. If yes (situation 2), the condition is valid and if all the other conditions (here identity= User B) of the rules are valid then the rule applies.
For the Gamers community, the capability is used to automatically switch the blurring parameter on and off depending on when the user is located.
Multiple sites can be defined in a single condition implying that if the owner is in one of the defined sites, the condition is valid.

2.2.1. Support authorization request for any Object

A rule definition can include the following status:
"before performing this action, you need to get approval from the resource owner".

This is configured by defining a status: askOnce or askAlways associated to an action

In phase 1, the authorization request was handled (notification sent to the client and update of the rule) by the components and was only supported for the presence and location attribute of an identity.

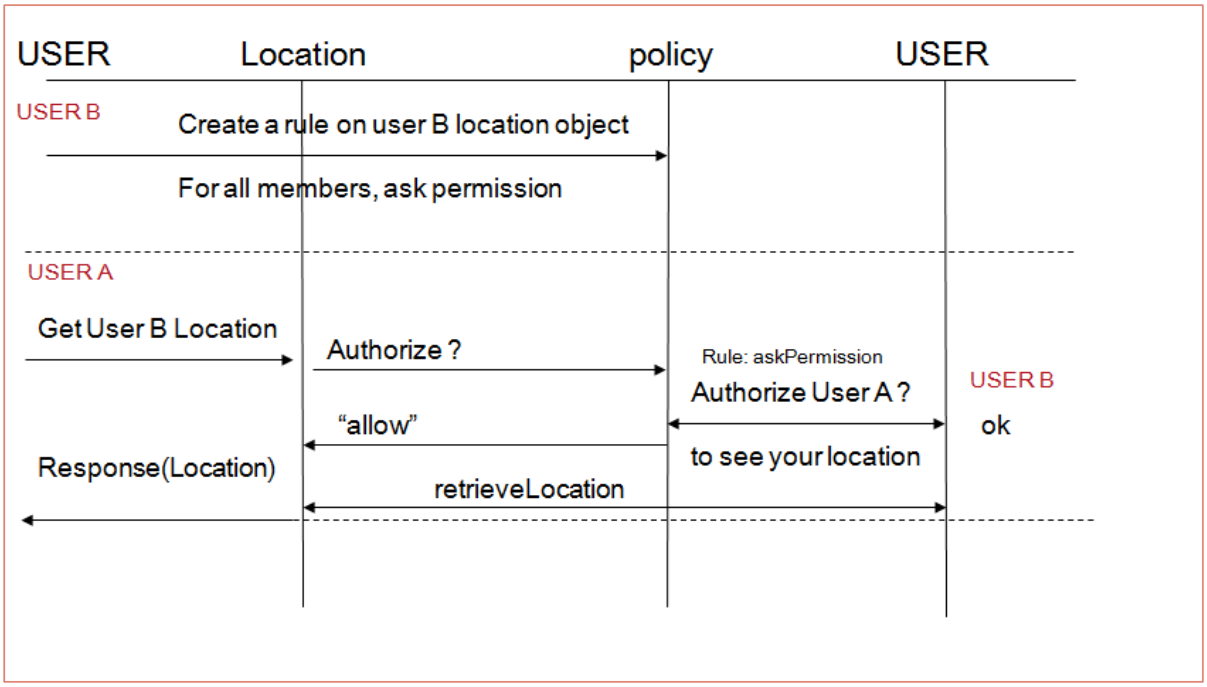


Figure 3/ the authorization request is generically handled by the Policy server

The phase 2 has generalised the support of authorization request for any Object and is used for location, presence, contact List or profile attributes of a user object

2.2.1. Enhance status of policy rules.

A rule is defined as a set of conditions and a list of couples (action, status). The rule can be formalised as follow.

If (all condition are true):

- Then for action 1 return status1
- Then for action 2 return status2
- ...
- Then for action x return status

Typically status values can be : allow, disallow, askOnce, askAlways.

The phase 2 introduces parameters attached to the status. they characterize how the “action” should be handled.

The new feature is used in phase 2 to specify the precision of the location information when a user allows others to access his/her location information.

Example: UserB allow user A to see my location at home with good precision

Resource: location attribute of User B
Condition
- Identity : user A
- Site : at home

Couple (action status):
- Action = read, status=allow,
  - status parameter name = precision.
  - Status parameter value = good.

The status parameter has been defined with a generic model as an array of couple of (name, value) in order to be used in many situations.

2.3. Content and Forum thread Access History

User can contribute to the community by pushing public content to public-community or sub-community repositories or pushing new post in forum threads.

The platform offers now the ability to keep a history of who has accessed the resource and when. This capability is available on a per content (provide history of accesses to a particular content) basis or on a per thread basis (provide history of accesses to a particular thread).

The interface is as followed:
- getContentAccessHistory (category, content, requester)
- PcgetThreadAccessHistory (forum, thread, requester)
- ScGetContentAccessHistory(sub-community, content, requester)
- ScGetContributionAccessHistory(sub-community, thread, requester)

2.4. Add Point of Interest (POI)

The platform allows the definition of Points of Interest (POIs). A point of Interest is basically a location and an associated POI type. The POIs can be “internet coffee” , “hotspots”, shop, restaurant, or museum.

POIS are defined using GPS coordinates (using decimal degree format)

These Points of interest can then be listed (per type) and shown on a map (client feature). A facility is offered to locate the nearest POIs around the requester.

POIs are public objects and are directly attached to public-community Object (the POI is an attribute of the public-community object). However, the owner (the creator) of the POI can define access rule by using the defined resource: public-community().poi() or public-community().poi(id) where the id is the id of the POI. All the facilities of the PICOS rules can be used to define policies for POIs (identity condition, date condition, reputation condition...).
A private POI is a POI where a policy has been attached that limit the access to the creator.

Once POIs are defined, user can recommend them to (a list of) community members. Community members have the ability to disable these recommendation notifications if they don’t want to be disturbed with.

The interface proposes the following capabilities

- AddPOI
- deletePOI
- updatePOI
- getPOIAttributes
- getPOIList
- recommendPOI
- set/getReceiveRecommendationStatus
- searchNearbyPOI
- notifRecommendedPOI (notification sent by the platform to recommend the POI)

2.5. Advertising of Commercial Points of Interest

POIs can be defined with an associated advertising profile. These POIS are then considered as commercial POIs (e.g. a shop). If there is no associated advertising profile to a POI, it is not considered as a commercial POI and cannot be advertised using the two scenarios that are described below.

.
The advertising profile contains profile information about the target user “segment” for that commercial POI.

The advertising profile is composed of:

- The target gender (optional) .
- The age interval (optional).
- A set of keywords and associated commercial announcement (mandatory).

- An area around the location of the POI in which, advertising is valid. (mandatory).

.

The advertising profile is used to deliver advertising that matches the member profiles. If no target gender is defined, then the gender of the receiver can be male or female, if no target age interval is defined, then the age(age interval) of the receiver is not checked.

Two advertising scenarios can be demonstrated:

- Scenario 1: The most valuable advertising is pushed to the user depending where he is.
- Scenario 2: When a user reads contribution of a forum, extra advertising contribution that match contribution, POI keywords and user interest can be inserted in the list.

2.5.1. Scenario 1: Advertising based on location

If the user has enabled the location sensor, the client application periodically sends some location updates, providing new GPS coordinate data.
When the platform receives a location update related to a particular user, it checks what are the commercial POIs around this user and try to match the POI advertising profile with the user profile for the following profile fields:

- User profile age versus advertising profile age interval (User profile age must be inside the advertising profile age interval
- User profile Age interval versus advertising profile age interval (User age interval must be inside the advertising age interval.
- User gender versus advertising profile gender.
- The POI must have one keyword that is part of the User profile hobby list
- The POI coordinates and the size parameter must form a circle that “contains” the user location coordinates

If multiple POIs match the above criteria, the platform selects the one that has the highest number of matches between the list of POI keywords and the list of hobbies. The second criterion is the proximity of the POI for the user.

By default the user does not receive advertising notifications. He must enable its reception in the client application settings.

The POI advertising is enabled when a user enters an area centered on the POI. Each time, a notification is sent to an identity, the platform remembers it to avoid pushing multiple times the same notification. If the user leaves the area and re-enter it again, he might receive the notification related to that specific site again.

Once the POI is selected, a notification is sent to the user related to the commercial POI.
Note that only the best POI is selected and pushed but as one POI advertising notification cannot be sent multiple times unless the user exits and re-enters the advertising zone, advertising for other POI might be sent as well later on, when location is updated. Strategies on pushing advertising might be refined to be well accepted by the end user.

2.5.2. Scenario 2: Advertising in forums

When a user read a forum thread, contributions (or post) are displayed. Each contribution contains text. For each contribution, the platform detects any matching between the contribution text words, the list of keywords of defined commercial POIs and the list of hobbies of the thread reader. Any non null matching will lead to the insertion of a virtual advertising contribution that provides information on the advertised POI. This insertion is valid for a particular reader and is not stored in the platform.

2.6. Notification of new content

The platform offers a generic subscribe/notify service. The subscription related to a particular Object called resource and an action on this object (EX: I subscribe to receive an alert when community members read a content that I own: the resource is the specific content and the action is read.

Subscription and un-subscription is managed by the End User via the client application.
The platform sends alerts for specific sets of couple (resource, action) which are:

Objects	Action	Alert	Attributes
Public repository	create	Alert when a new repository is created	Category attributes
Specific public repository	Read , write	Alert when a specific repository is read or modified	Category attributes
Content in repository	create	Alert when a new content is added to a repository	Category attributes
Specific content in repository	Read, delete	Alert when a specific content is read or deleted	Content attributes
Public forum	create	Alert when a new forum is created	Forum attributes
Specific public forum	Read, delete	Alert when a specific forum is rated or deleted	Forum attributes
Public forum thread	create	Alert when a new thread is created in a particular forum	Forum attributes
Specific public forum thread	Read, delete	Alert when a specific forum thread is read or deleted	Forum-thread attributes
Public forum contribution	create	Alert when a new contribution is added to a specific thread	Forum-thread attributes
Specific Public forum contribution	Read, delete	Alert when a specific contribution is read or deleted	Contribution attributes
Content in sub-community	create	Alert when a new content is created in a specific sub-community	Content attributes

The PICOS project receives research funding from the Community's Seventh Framework Programme.

Sub-community thread	create	Alert when a new thread is created in a sub-community	Thread attributes
Sub-community contribution	create	Alert when a new contribution is created in a specific thread	Contribution attributes

Table 5: Community Object and actions on which Alert are sent if a user subscribes to them

The resource description model for subscription re-uses the same description used for policy resources

When the action on the resource occurs, the platform sends a notification (alert) to the client application with the resource description, the action and attributes related to the object (as define in the above table)

The platform offers the following interface:

- Subscribe (resource, action, subscriber)
- unSubscribe (resource, action, subscriber)
- getSubscriptionList (subscriber)
- notifyAccessElement (resource, action)
- NewContent notification (sent to the client)

Note that the service could also be used by the client for objects not handled by the platform but described using the same convention than any platform object. In this case, the platform would just push notifications to those (identities) who have subscribed to the proper couple (resource, action)

2.7. Add shared alarms

The Phase 2 platform supports the ability to create Alarms (date, text, list of destination). When the alarm timer pops, the text is sent as a notification to the list of defined destinations. By default the alarm creator is part of the destination.
The platform offers the following interface:

- createAlarm
- deleteAlarm
- updateAlarm
- Notification for alarm.

2.8. Meet with nearby Users

The platform offers the ability to search for community users around a provided location. It can be the requester location or any location like a POI location. The provided location must be defined with GPS coordinates. The result contains information on users who are close to the location. Note that

only users who have allowed the requester to see their location attributes will be part of the list. If the policy specifies to ask for permission, the user is not selected because it is not viable to ask for permission to all members of the community around the user.
For the search, only the last locations of the users are taken into account and there is no access to the phone to real-time retrieve the location.

The new interface is as followed:

- SearchNearbyUsers (location, size of the location, max number of response)

2.9. Enhanced presence with planned statuses

The members of the community can define enriched status information attached to period of time that will be displayed like in an agenda. The status will be composed of a text string and an interval during which the status applies. The status can optionally be recurrent.

Statuses are attributes of a user/partialId and as such are defined as a policy resource:

- User(id).partialId-list().partialId(id).status(id)

Default policy for the resource is that only the owner of the resource can read / write / delete the resource.

Statuses can be shared with other community users by setting policies attached to the status resource using possible conditions on identity, validity or location.

Note that the access to statuses is defined by the policies associated to the presence attribute as the status is perceived as an extension of the presence information

Note that the presence information is composed of an enumeration of a status (on-line, off-line, discreet) and a presence note. The presence note is automatically updated with the enriched status test when it applies.

2.10. Enhanced chat capabilities

The platform now supports the ability to share content during a chat by either selecting one image and transfers it to the list of participants or by taking a picture and push it over the chat channel.

The second enhancement relates to the ability to look for previous archived chat and display exchanged messages and content and check participant list of these past chats.

The interface provides the ability to/

- Get a list of chats, the user was part of as chairman or as participant.

The PICOS project receives research funding from the Community's Seventh Framework Programme.

- Get the list of messages/content for the closed chat
- Get the list of participants of the closed chat.

Note that an identity will only see chat, he is allowed to see. Default rules allow participant or chairman of a chat to access a chat. These rules can be changed on the platform side to change access rights.

2.11. Support offline notifications

The platform offers a notification mechanism to inform users of new events. However, this mechanism requires having the user logged in to the community to be able to receive the notifications.

The platform now offers a way to get some of the notifications that arrived when the user was off-line. The notifications are stored and can be retrieved at login in and processed by the client application. This new mechanism applies to a sub-set of the notifications:

- Authorization Request for all resources (presence, location, contact list (requires a user answer))
- New content availability
- Sub-community invitation (requires a user answer)
- Sub-community change admin (requires a user answer)

The mechanism does not apply to all notifications because the information of some notifications is outdated / superseded by the information retrieved during the login of the user (presence or location updates, message alerts) or because some notification are valid at a specific time and do not make sense afterwards (like chat invitation)

Note that during the login process, the client application is informed whether or not there are pending notifications. The client retrieves the pending notifications and acts upon.

Example of usage:

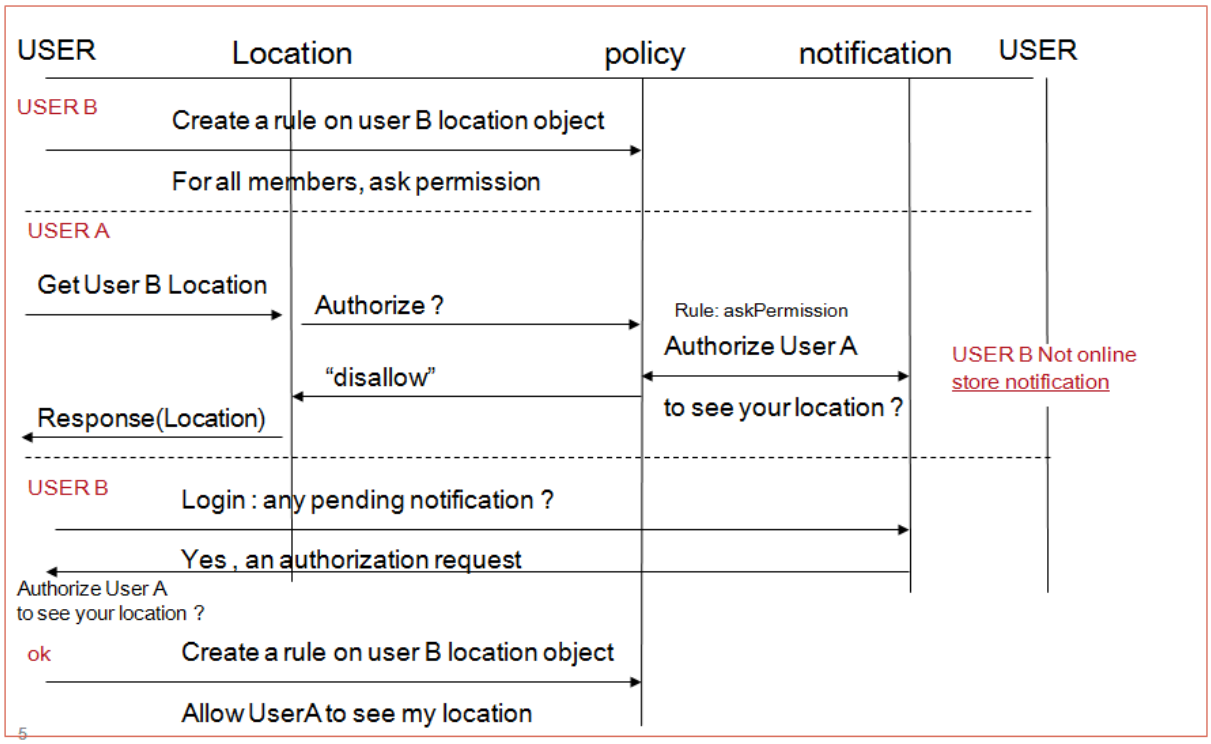


Figure 4 Usage of off-line notification process

User A adds a contact (User B) . Once it is done, the client requests presence and location information. For presence and location the default is that an authorization request is sent to User B. However if User B is off-line, he cannot answer the authorization request. The immediate User A request is denied until User B can answer.
In this case, the authorization request is kept by the platform and when User B logs in, he will get the request and can respond. The rule is updated and when User A requests for presence or location, the server will deliver '(or not depending on User B response) the presence / location information.

2.12. Shared Desk

A Shared Desk is a content repository that is created by a member of the community. The creator can then decide to invite other members to access his shared desk. The creator of the shared Desk has full capabilities to manage the list of invitees whereas invitees have the capability to read the content but cannot contribute to the shared Desk.
The shared desk interface is equivalent to the handling of private sub-community. The shared desk is then perceived as a new type of sub-community.

3. New Platform Design for Gamers Community support

3.1. Architecture

The PICOS platform Phase 1 architecture has been kept while introducing new or enhanced components. The client (mobile) / server (platform) model remains based on a duplex RPC model rather than defining complex protocols. The value is that the client application calls methods as if the server was a "local" library and the server calls mobile methods as if the mobile was a server lib removing the need to define complex protocols.

Two new components have been added: the site component and the subscription component

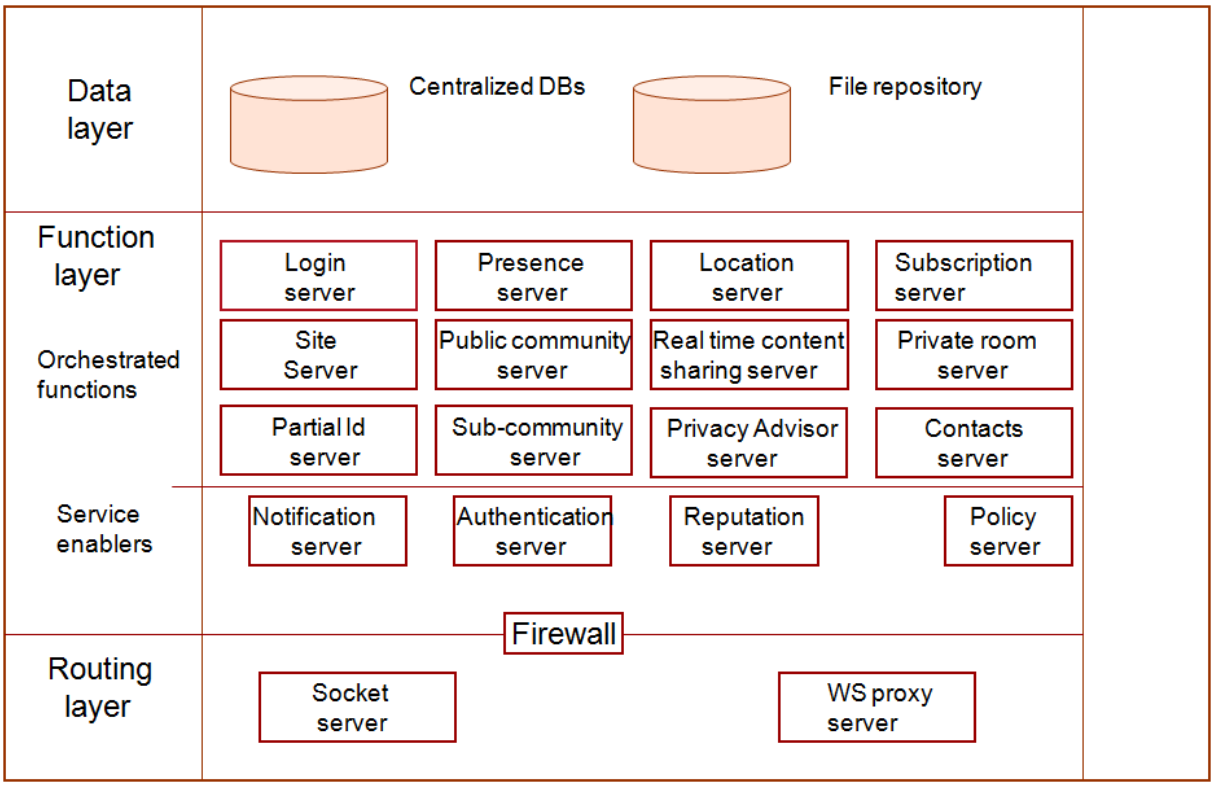


Figure 5: Architecture functional blocks

Components are designed to be independent from each others, so they can be distributed on multiple servers, and can easily customizable in terms of inter component communications and component storage.

3.1.1. Mobile access to the WP5 platform

The WP5 PICOS platform implements a set of components for community management with enforced capabilities around identity, policies, privacy and trust. These components are called by the WP6 client application to implement the PICOS use cases.

The WP5 PICOS platform interface is defined as a web service interface. However to ease the client development, a client RPC library is provided in the handset J2ME environment to access the WP5 PICOS platform using a Remote Procedure Call (RPC) rather than managing protocols.

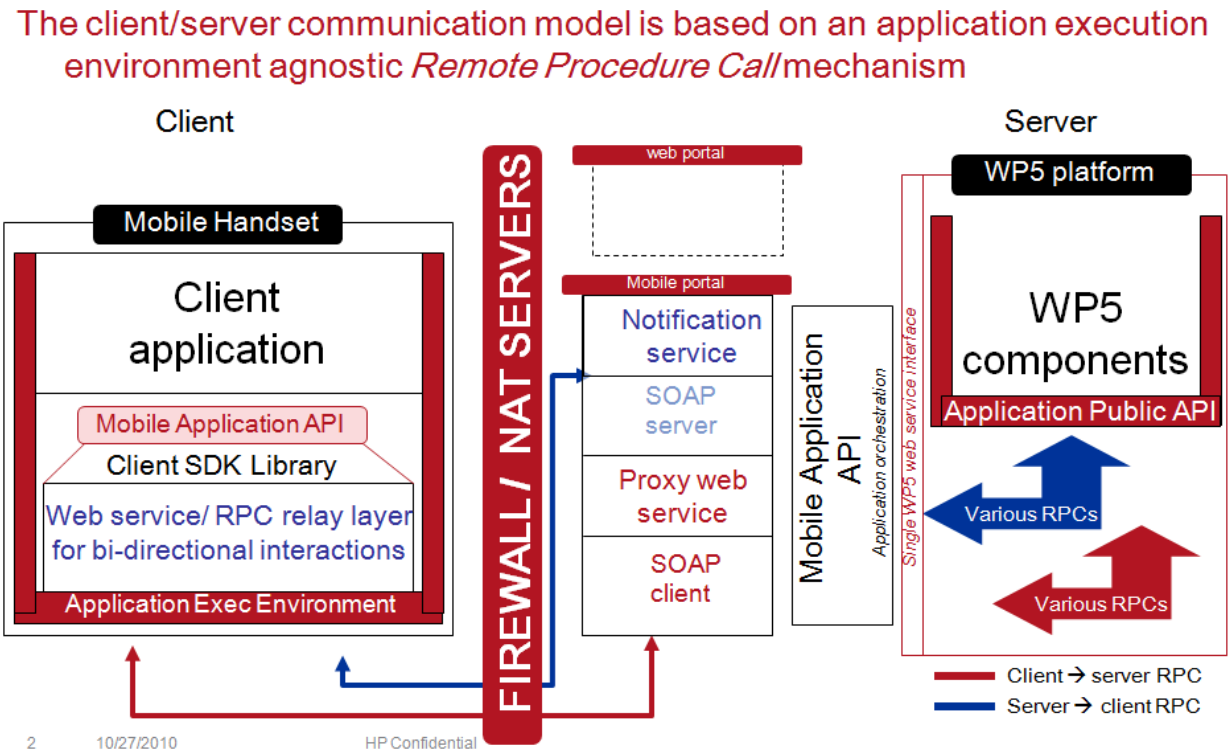


Figure 6: Mobile Client / server overall model

The WP5 PICOS platform embeds a RPC gateway that acts as a front end signalling relay for all the mobile interactions towards the WP5 services. The RPC gateway is a HP asset brought by the HP France team. This gateway also manages the access control to the platform by enforcing the authentication of the requester and only relaying the requests when login credentials have been

validated (except for register and login methods). All exchanges between the client application and the wp5 platform (up to the proxy) are carried over a secure channel using https.

3.1.2. ASP Deployment architecture

The PICOS platform is connected to internet and therefore must be protected against attacks. The RPC gateway server is the front-end access of the platform for client RPC request and is set to be installed in the DMZ zone of the service provider network. The rule is that there is no internet traffic that can directly reach the PICOS components integrated in the operator private LAN and protected by the service provider firewall. This firewall filter accesses from the mentioned RPC gateway.

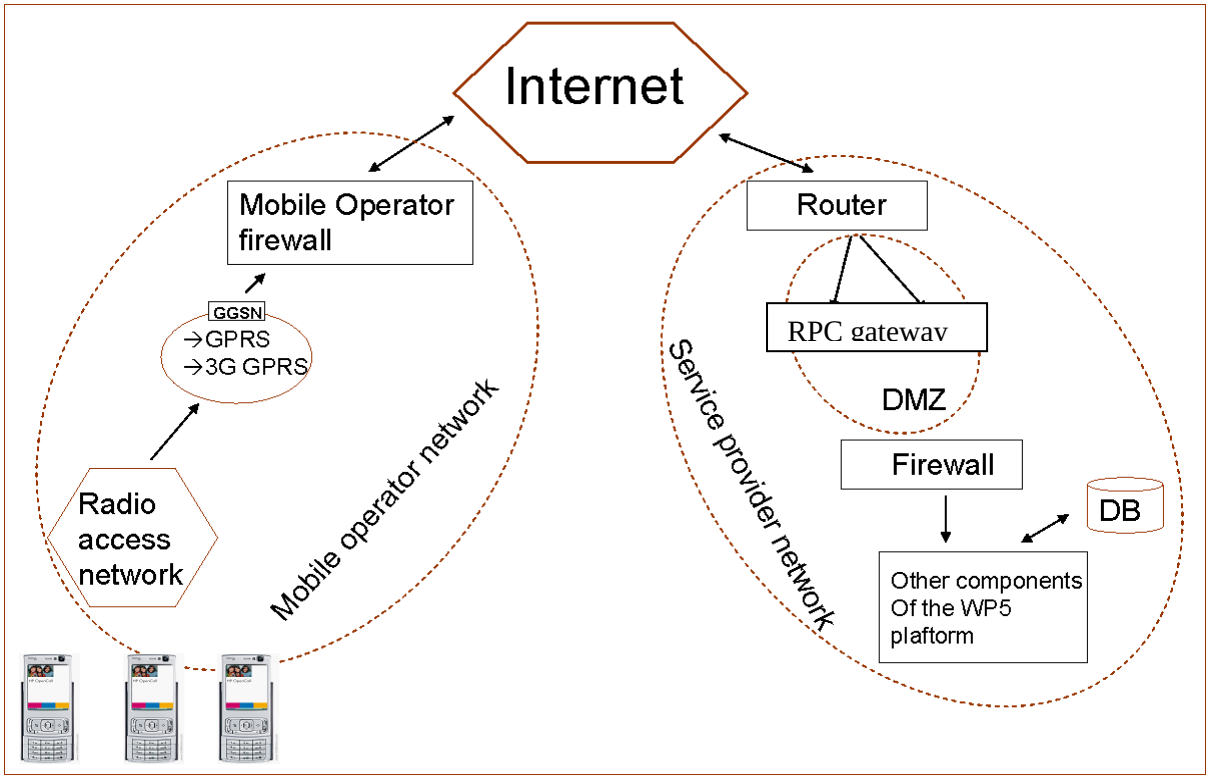


Figure 7: deploying the PICOS platform and the PICOS application

3.1. Inter-component communication

The PICOS Phase 2 project has kept a flexible WP5 platform inter-component communication platform based on a RPC access model

The platform defines for each component the location and the access method. By changing the topology configuration file, the system can be re-shaped to run components in different systems enabling different distribution strategies.
As a single shared method is used to communicate between components, various RPC model have been implemented to see the impact on performance. Each component has been slightly adapted to support the 3 RPC models, considering that a few lines per component is impacted by the RPC models.

Note that a configuration file exists that defines which RPC model is used to access which component. Easy customization is possible.

3.1.1. The web service interface

Each component acts as a web service server. The requests are decoded by PHP soap libraries and methods are called by the PHP soap server. Soap requests are sent over http and the http interface is completely hidden by the PHP library.

```
ini_set("soap.wsdl_cache_enabled", "0");

$server = new SoapServer ("presence.wsdl");

// User web service
$server->addFunction(SOAP_FUNCTIONS_ALL);
$server->handle();
```

The component interface is defined in a WSDL document (that may also reference xsd files for Object type definition) that formalizes the interface i.e. its methods and the method parameters as well as the URL of the web service server.

Additional information are inserted in the SOAP HEADER like the “client version” parameter

3.1.2. The RPC based on PHP serialize format

PHP offers a library to serialize any data structure and translate it into a string. This facility is used to convey a RPC request composed of the method name and of the method parameter data structure into an http post.
Each component is then launched when the request is received, decode the couple (method, parameters) and call the method.

```
if ($_SERVER['HTTP_USER_AGENT'] == "PHP-rpc" ) {
   // must be a web service request
   $rpc       = unserialize($HTTP_RAW_POST_DATA);
   $paramArray = $rpc->paramArray;
   $methodName = $rpc->methodName;
   $clientVersion= $rpc->clientVersion;
   $response= call_user_func ($methodName, $paramArray);  //  call the requested method
```

```
    ob_start();
    echo "&result=".serialize($response);
    $oblen = ob_get_length();
    header('Content-Length: ' . $oblen);

}
```

The heart of the request processing is independent from the access method
Note that this RPC model can be used in a distributed architecture

This RPC has been tested as the one that offers the best performances among the tested RPCs

3.1.1. The RPC based on PHP script launching

Rather than using http protocol to convey the RPC, the component is launched like any PHP script (“PHP component.PHP argument1) with the RPC as its parameter (argument1)
PHP offers a library to serialize any data structure and translate it into a string. This facility is used to encode the RPC request, composed of the method name and the method parameter data structure. Each component is then launched, decode the couple (method, parameters) and call the method. Note that the argument is urlencoded to encode any space character

```
   $rpc        = unserialize(urldecode($argv[1]));
    $paramArray = $rpc->paramArray;
    $methodName = $rpc->methodName;
    $response= call_user_func ($methodName, $paramArray);
    echo "&result=".serialize($response);
```

The heart of the request processing is independent from the access method and then the called function is the same than the one called when other RPC methods are used.
.
The RPC response is returned as component output and its Object structure is PHP serialized.

3.2. Hardware configuration and performance

The objective of the PICOS platform is to demonstrate the value proposition developed in the PICOS project and demonstrate it in a trial with no more than 100 users.
It has been decided to provide a system that offers the optimum flexibility in terms of distribution. The selected language is offering a fast coding approach to validate the PICOS concepts. These two choices do not optimize the overall performance of the system but system response time remain acceptable when compared to the networking response time over mobile accesses and to some possible optimization to reduce the number of request/replies exchanged between the client and the server.

Some performance data points (per method response time) are provided for information and have been measured on the following system:

- 4 CPUs Xeon (5160) with 4 MB L2 cache running at 3 GHz and 8GB of RAM
- 64 bit Linux redHat el 5.4.

3.3. Component design

3.3.1. PHP language

PHP is a very popular and very rich scripting language with a large community of developers. Its main advantages are:

- Fast coding for prototypes
- Large community of developers (forum, code examples, open source)
- Simple and dynamic integration of web service interfaces.
- Capabilities not supported by other languages
- Tight integration with web servers for web page handling or http interface handling

Its drawbacks:

- Not a formal language for Object definition (errors may be found during run time whereas other languages find the errors during compilation)
- Do not support threads (workaround exists)
- Do not support timers (workaround exists)
- Not as efficient as other language like Java.

3.3.2. Component type

All components are written using PHP language. They support one of the two possible component architectures:

1) The font-end / back-end architecture

The components of that type are split into two separate elements: a front-end and a back-end.
The front-end part implements the component interface (RPC) and interacts with the backend to have the request being processed.
The back-end part is acting as a socket server and receive request to be processed. They are started and must remain up and running all the time. The back-end listens to a TCP port to receive RPC requests

Advantages:

- Fastest response time as

- the backend PHP scripts are loaded once and not during request processing
- All the data structures are in memory and ready to be used.

2) The simple script.

A model "a la web" where the URL of the http "post" references the PHP script which is launched by the apache module. Any context data related to the request must be loaded prior any processing. The script exits when the response is sent.

Advantages:

- Simplest architecture.
- Each request is processed as a separate thread. As the PHP language does not support threading, it is the only way to support component re-entrance.
- No need to have the component started, it is automatically started when a request is received by the apache module

3.3.3. Component interfaces.

A WSDL document has been defined on a per component level to describe the interface using the WSDL description language and xsd types.

Note that the client application sees a single virtual WSDL file (PICOS.wsdl) but in reality, the platform interface is defined by 16 WSDL document and 13 type definition documents shared by multiple interface description documents

Some of the methods are not accessible outside the platform. The proxy component filters which methods can be accessed by the client.

3.3.4. Proxy component

Component summary	
Main functions	• Translate RPC request into web service requests, does platform access control • Rout web service request to the proper component. • Does access control of all request
Component type	Script launched when a request is received
Storage	None
Object/Attributes of objects the component is responsible for	N/A
Interface	HTTP Serialize RPC,
Phase 2 modifications	• Support of alternate RPCs • Centralized traces

	• Change the RPC structure for better performance • Remove URL encoding of body requests • Support of utf-8 characters • Use central PICOS library

The PICOS platform embeds a proxy Web Service (proxyWs) server that receives all the Remote Procedure Calls requests from the mobile client application. This proxy is in charge of translating RPC requests into web service requests, controlling and enforcing the identity of the requester and routing the request to the proper component.

To achieve this, the RPC API offers a way to pass the rootId and the authorization token that is analyzed by the proxy. Each web service request to the PICOS must contain at the root level a parameter called "requester". The requester parameter can contain either a rootId or a partialId. It is the proxy responsibility to make sure that the requester identity is in line with the RPC rootId parameter otherwise the request is rejected.

The WS proxy formats and transmits the request to the proper PICOS functional component.

The proxy is also in charge of making the WP5 PICOS platform appear as a single huge web service server by hiding the description of component interfaces behind a virtual WSDL interface.

The proxy can also redirect a client request that needs to be routed .to an external web service server. Note that the proxy is web service agnostic. It converts client requests into various RPC access models depending on the accessed component

Receiving mobile client application RPC request

The client application communicates with the platform via the proxy web service server. Rather than defining a protocol interface, all the accesses to the PICOS platform are done through a client RPC library which offers a single entry point to send RPC requests.

Sending server RPC request to the mobile client application

The PICOS platform offers a RPC mechanism to call methods that are developed inside the mobile client application. The mobile client application embeds a client RPC library that supports this dual RPC model. This client RPC lib connects the notification manager of the PICOS platform and waits for RPC calls to be received. A RPC call is defined as a method name and a more or less complex parameter structure. The data structure is serialized and passed over the connection handled by the client RPC library.

A response can be built on the client application that also includes a more or less complex data structure. This data structure must be compatible with mapping rules between the execution environment data object and the complex types defined in the WSDL interface document of the notification manager.

Example of call flows

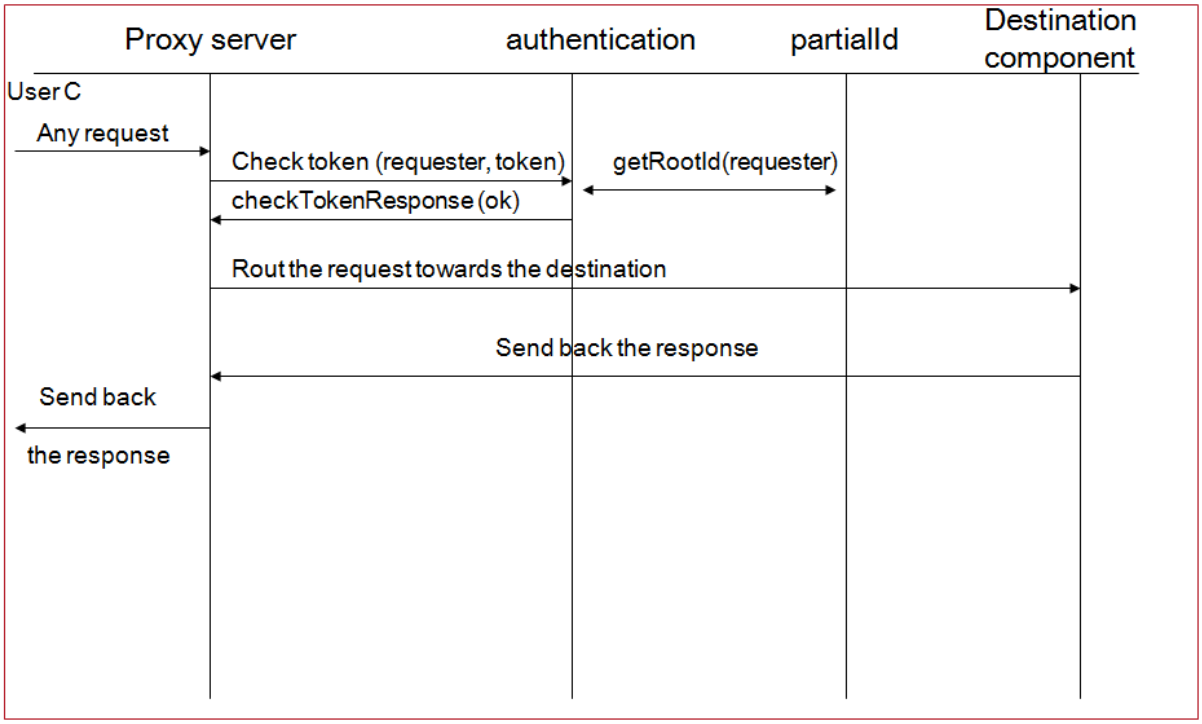


Figure 8: call flow involving the proxy server used for authentication and access control.

3.3.5. Registration component

Component summary	
Main functions	Manage the registration of the new user as a member of the public community
Component type	Script launched when a request is received
Storage	None
Object/Attributes of objects the component is responsible for	N/A
Interface	Web service (registration.wsdl), HTTP Serialize RPC, Script RPC
Phase 2 modifications	• Support of alternate RPCs • Centralized traces • Trap of unrecoverable errors
Performance (response time)	• Register 250 ms • unregister: 100 ms

A User can become a member of the public community after registration. The registration is a self user provisioning procedure (as opposed to manual operator provisioning) that leads to immediate possible login and access to the PICOS platform capabilities once it is complete. During the registration, the application can define a full or a partial definition of the End User profile as well as its privacy rules.

At least a login name/password and a pseudo (part of the user-profile) must be defined. The platform checks that the login name is unique among the member of the public community. It also checks that the pseudo is unique among those used for rootId profile as well as partialId profiles.

A new user context is created in each component that deals with user attributes. A set of policy rules (based on default policies) is attached to the attributes of that particular user (identified by the rootId). These rules can then be customized by the User using the client application.

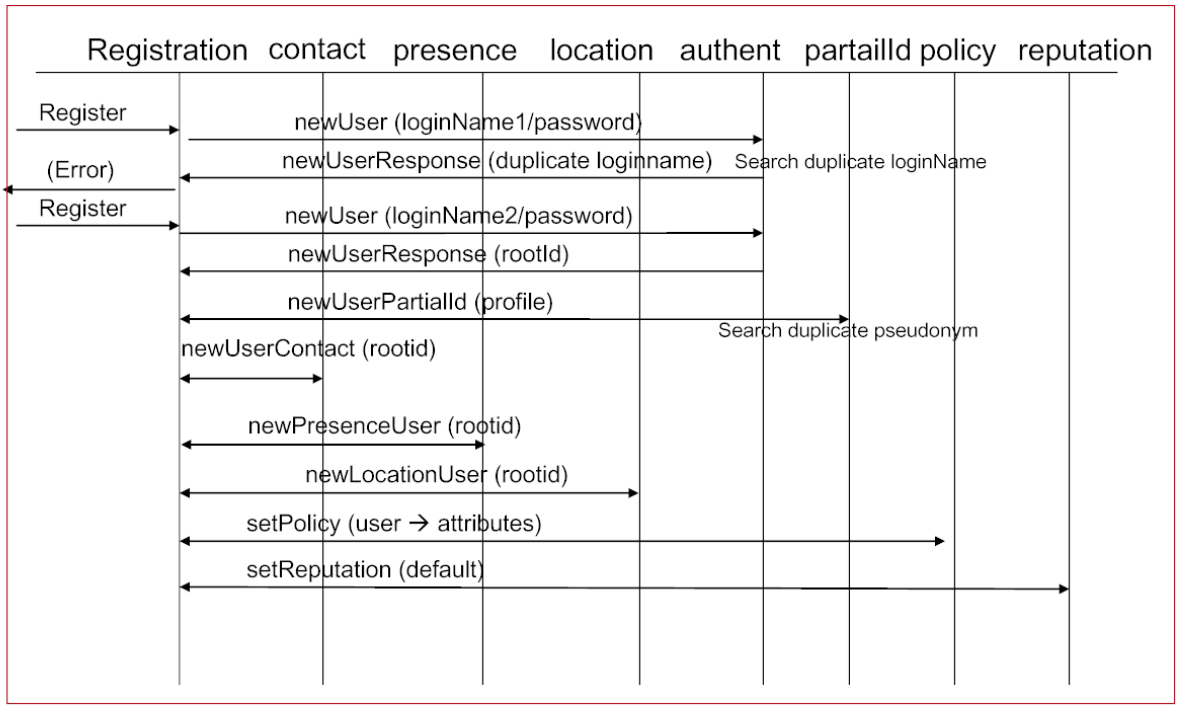
The registration component implements a transaction for registration and a roll back in case of failure in one of the component

The platform also provides a procedure to un-register which deletes most of the user context.(user-profile, identities, private room) Note that user contributions to Forums are not deleted and by default content published in sub-communities and in the public repositories are not deleted as well.

When a user un-registers, all partial Identities associated to that User are deleted. Contact lists or sub-community member lists that contain one of the partialId identities are updated and their new definitions are immediately available. However no notification is sent to the client application to update real time the client application screens.

No additional step is necessary to provision the End User in the platform.

Example of call flows



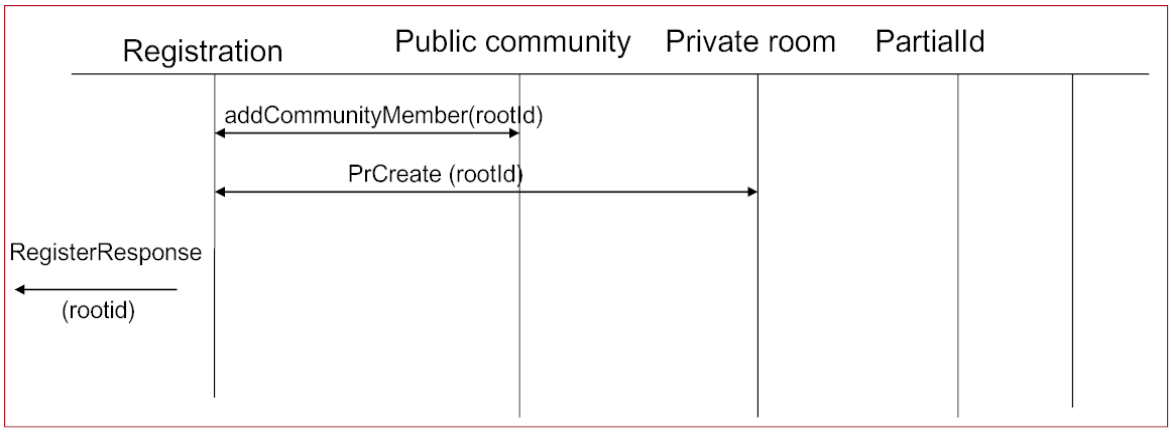


Figure 9: Intra-platform register call flow

3.3.6. Login component

Component summary	
Main functions	Manage login and logout of a user
Component type	Script launched when a request is received

Storage	None
Object/Attributes of objects the component is responsible for	N/A
Interface	Web service (login.wsdl), HTTP Serialize RPC, Script RPC
Phase 2 modifications	• Support of alternate RPCs • Centralized traces • Trap of unrecoverable errors
Performance (response time)	• Login : 200 ms • Logout: 5ms

This component manages the logon / logout of a user to the public community facilities and the provisioning of platform access control. The login interacts with the authentication component to validate the login and generate credentials.

Any login (as well as any client/server exchange) via the mobile client application is using a secure channel (https/sslv3).

The login interacts with the partialId, contact and public community components to deliver back all the necessary information in a single request

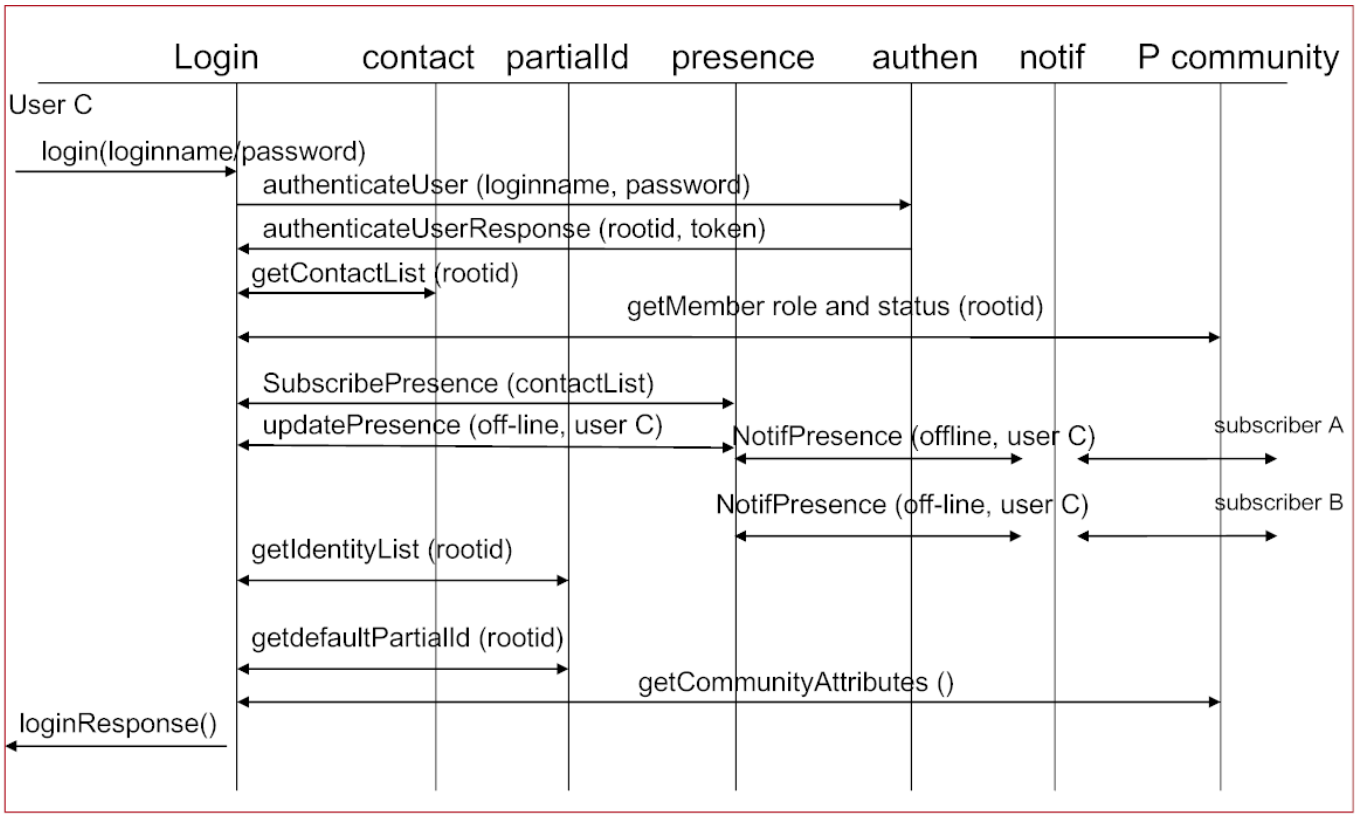


Figure 10: Login intra-platform call flow

Logout

The following diagram represents the interactions between the login component and the other WP5 components upon the receipt of the login request.

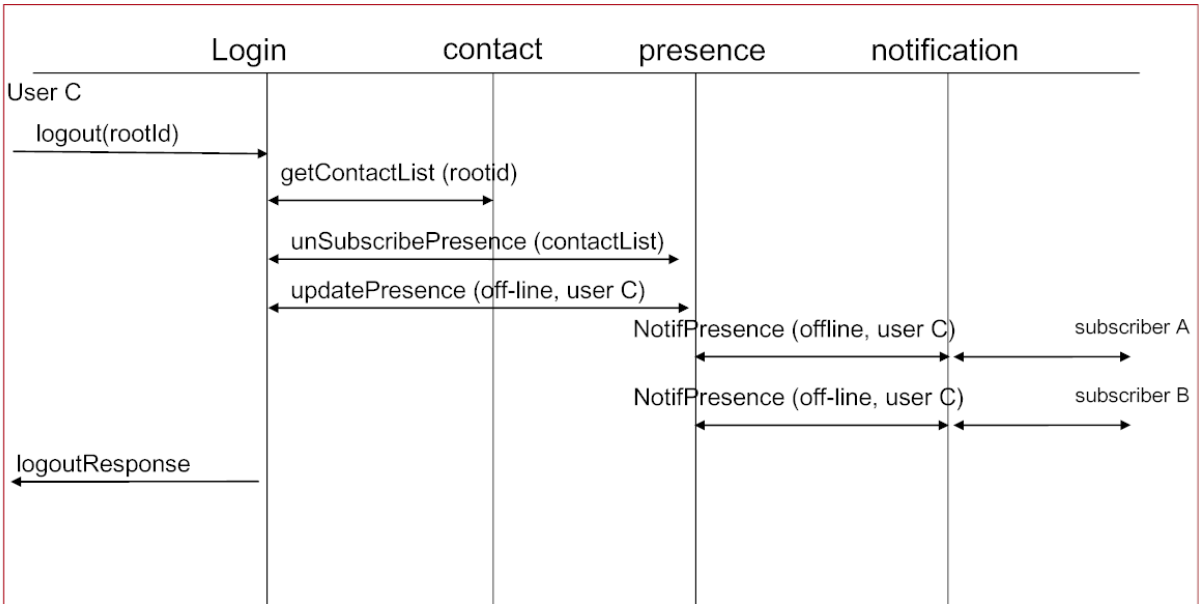


Figure 11 Logout intra-platform call flow

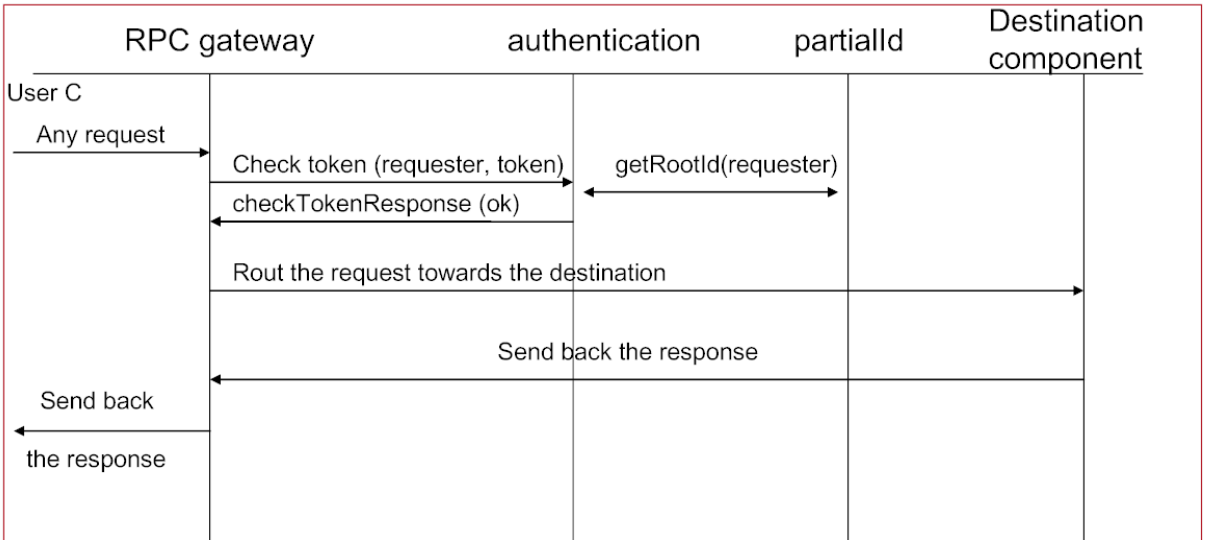


Figure 12 access control done by the RPC gateway on almost any request

3.3.7. Authentication component

Component summary	
Main functions	Authenticate the user, manage the session token
Component type	Script launched when a request is received
Storage	Use a specific configurable file repository to store user related authentication information
Object/Attributes of objects the component is responsible for	User credentials
Interface	Web service (authentication.wsdl), HTTP Serialize RPC, Script RPC
Phase 2 modifications	• Support of alternate RPCs • Centralized traces • Trap of unrecoverable errors • Configurable repository
Performance (response time)	• checkToken (authenticate) 4 ms

The authentication of the End User is achieved by the verification of the pair username/password that is transmitted over a secure access at login. A token is returned in the login response that must be provided for each sub-sequent request. The secure channel can be dropped and re-established upon activity, this without impact on the login token. The token remains valid until the End user logs out or if a valid login procedure is restarted.

There is a single authentication method supported by the platform (user/password over secure channel)

The platform has a limited notion of login session beyond the validity of the token and the presence status (associated to the rootId) that is automatically updated to “online” status with “User has logged in” as note after login and to “off-line” status and “User has logged off”.

The authentication model is kept simple because the client/server exchange for password validation is performed on top of an encrypted https channel.

Example of call flows

Please refer to the proxy call flows to see how the authentication server is involved for checking token or teh login call flows for allocating token or registration call flow for creating a new login.

3.3.8. PartialId Component

Component summary	
Main functions	Manage identities of the use, the associated profile static information as well as the user alarms
Component type	Script launched when a request is received

Storage	Use a specific configurable file repository to store user related identities and profile information
Object/Attributes of objects the component is responsible for	• PartialId Object • PartialId profile • User profile • Alarm Objects
Interface	• Web service (partialId.wsdl), • HTTP Serialize RPC, • Script RPC
Phase 2 modifications	• Support the new status feature 'add /remove status' • Move authorization request processing to teh policy server • Centralized repository for user data storage • Centralized traces • Trap unrecoverable errors
Performance (response time)	• createPartialId : 70 ms • getPseudo : 3 ms • getRootId 2 ms • getProfile: 8 ms • updateProfile: 6 ms

Users register to the public community and create then a primary identity. In order to keep anonymity, End users can create additional identities called partial Identities The platform is defined in such a way that end user can select either the primary identity or added partial Identities to use the platform services. Note that the client application may restrict the use of the primary identity but it is an application choice.

User and partialIds are different objects with partialId Object owning only a sub-set of the User attributes. PartialId automatically inherit from user Object for attributes that are generic to the user (location, some user profile attributes…).

However, attributes like location that are not redefined at the partial Id level can still be accessible using a partialId.

A user profile is associated to an identity and contain sensitive static user information thus excluding privacy rules, reputation, presence or location. The primary identity profile contains the full definition of the user information. The partial Id profile can only redefine a sub-set of the profile attributes. As an example, the gender of the user is defined at the primary identity profile level and all partial Identities attached to that user will see the same gender without being able to modify it on a per partialId profile level.

The primary identity is created at registration time whereas partialId can be created at any time after login. Creating a partial identity also creates a specific context for the user-profile, the presence, the privacy rules and the reputation.

These identities are used to reference the user in any operation in the public community. Identities are externally known through the notion of pseudonyms which is the only mandatory field of the profile to complete. From a platform standpoint, they can be changed. These identities are internally referenced

using either the rootId (for primary identity) or a partialId which are not supposed to be revealed to the End User. RootId and partial Id are then internal IDs which have a specific syntax to preserve uniqueness.

The partialId component has implemented the ability for users to create alarms (or post it with date) that can be shared with his contacts. The alarm is created with possibly a list of destinations. When the alarm pops, a notification is sent via the notification server to each destination.
The partialId component is launched when a new request is received and exits when the response is transmitted.

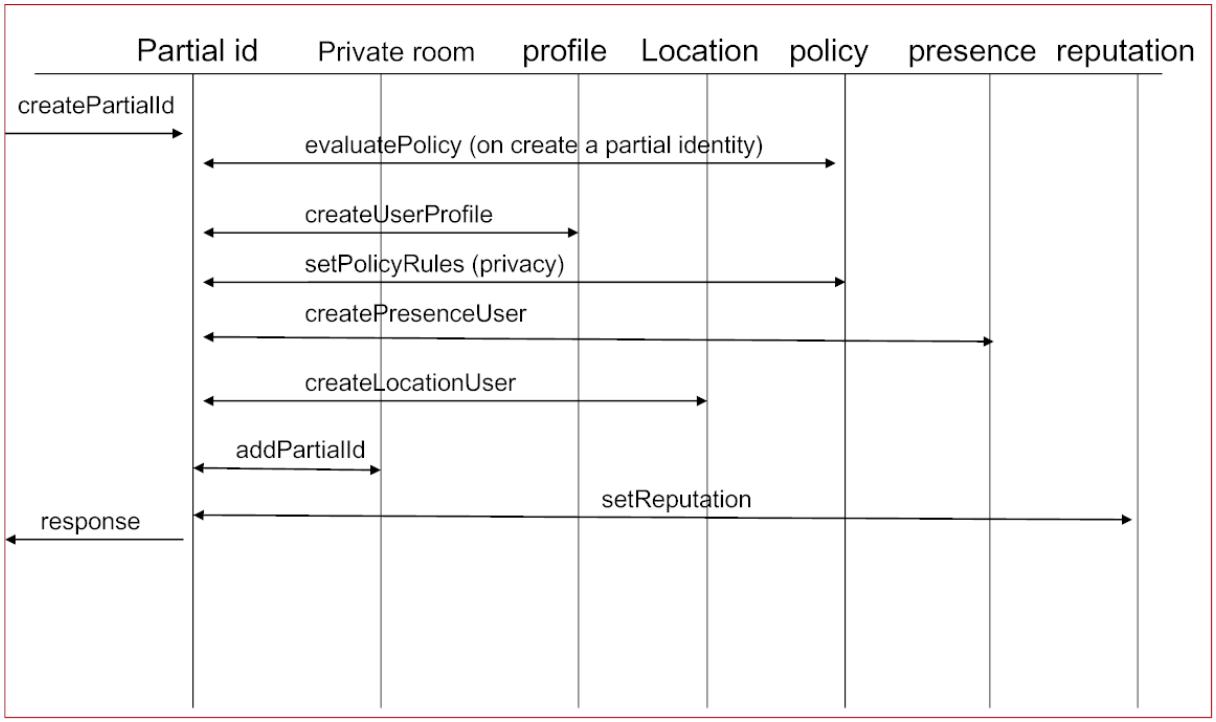


Figure 13: create a partial identity call flow

getIdentityProfile

For partialId, combine the rootId profile (attributes that are generic to the user) and the partialId profile. (Attributes that are specific to a partial identity) and check if there is a global policy for the profile or a per attribute policy.

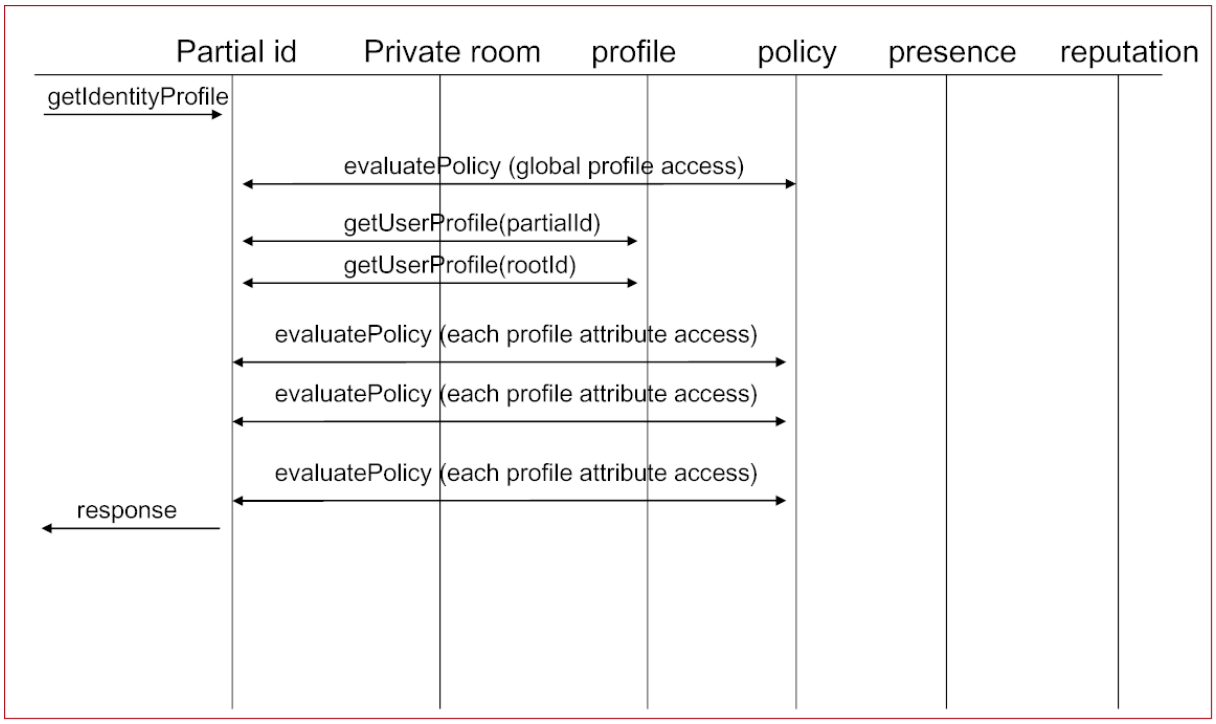


Figure 14: get Identity Profile call flows

The PICOS project receives research funding from the Community's Seventh Framework Programme.

deletePartialId

Remove an identity of a user: remove identity context in most of the platform compoenent

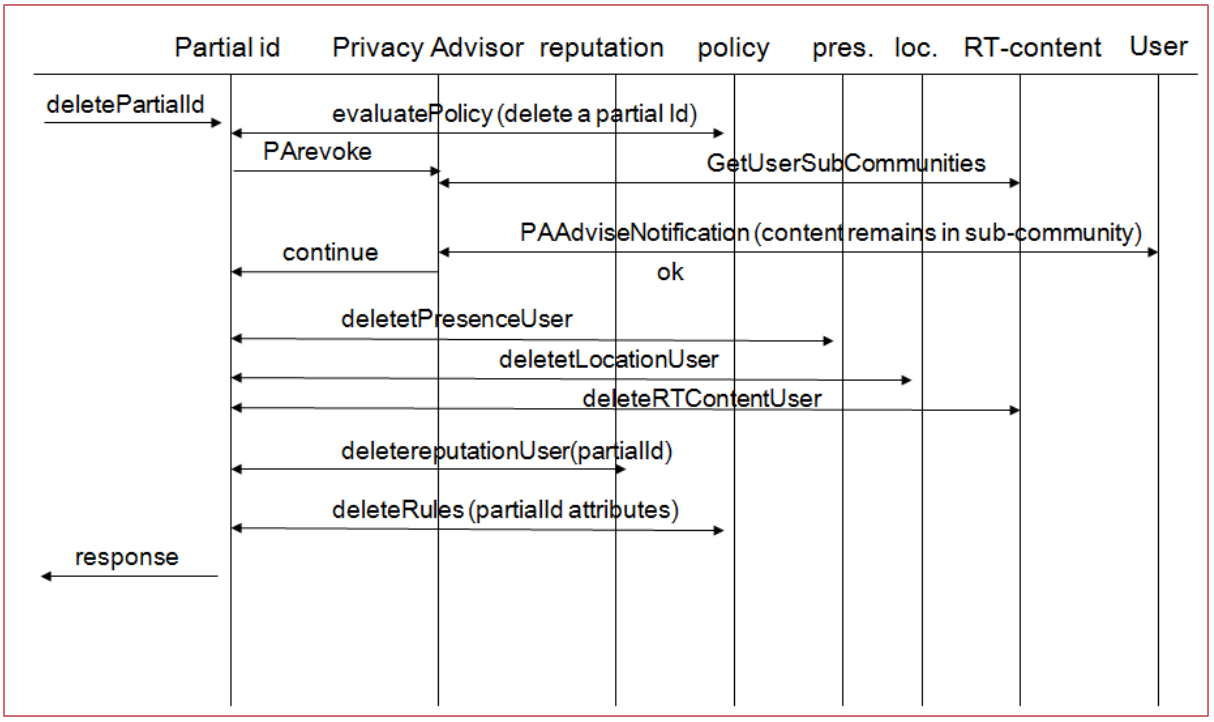


Figure 15: delete a partial Id call flow

getidentityList

Get the list of identities attached to the partialId. The response includes the profile for each identity

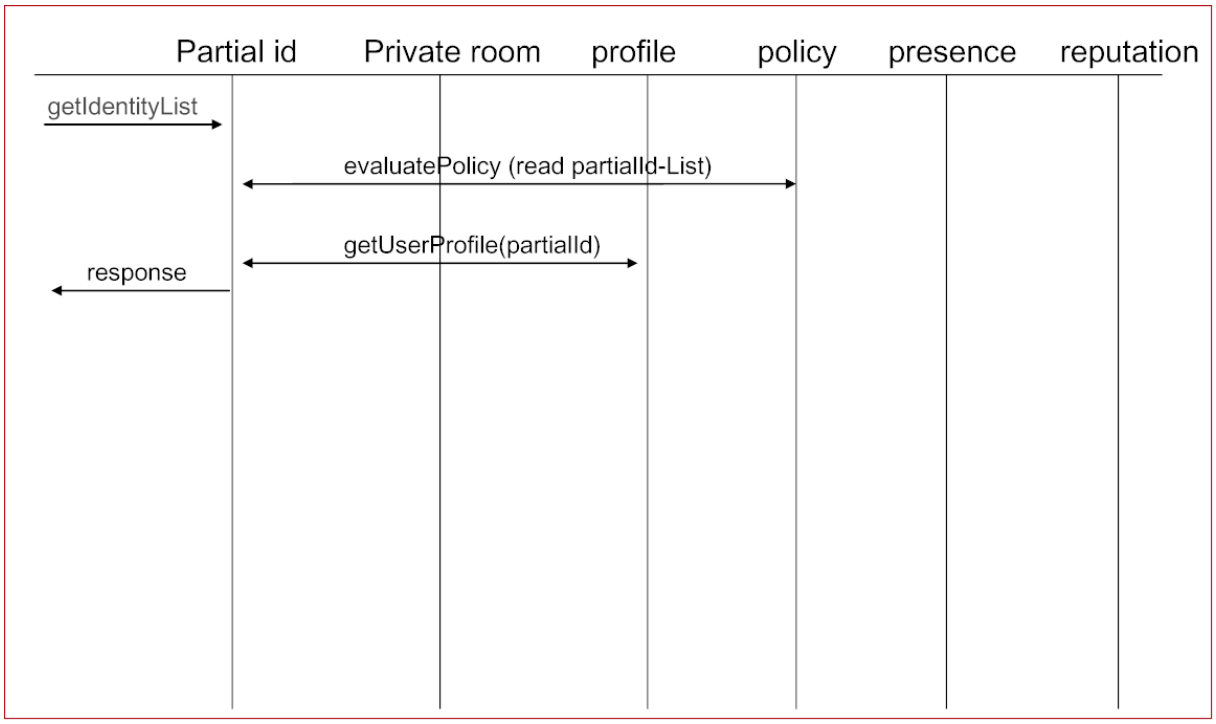


Figure 16: getIdentityList call flow

getRootIdWithPartialId

Retrieve the rootId (user) attached to the provided partialId (identity)

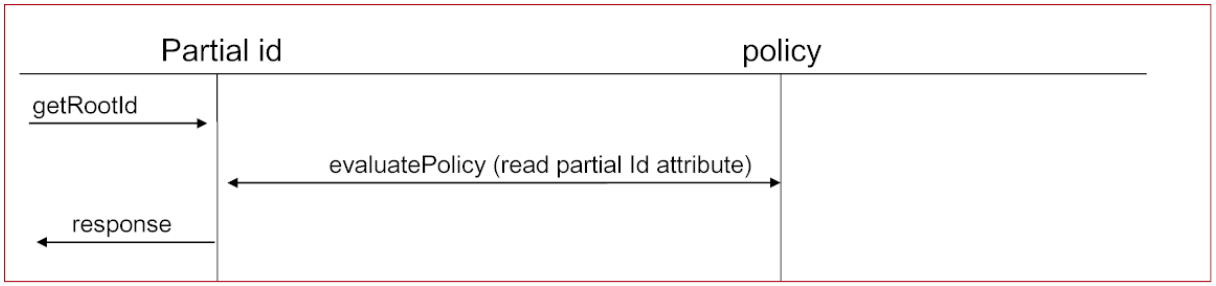


Figure 17 getRootIdWithPartialId call flow

The PICOS project receives research funding from the Community's Seventh Framework Programme.

3.3.9. Location

Component summary	
Main functions	Manage the location attribute of a User or a partial identity
Component type	Script launched when a request is received
Storage	Use a specific configurable file repository to store user related location as well as POIs information
Object/Attributes of objects the component is responsible for	• Location attribute of the User Object • Location attribute of the partialId Object • Points of Interest attributes of the public-community Object
Interface	• Web service (location.wsdl), • HTTP Serialize RPC, • Script RPC
Phase 2 modifications	• Support the management of Points of Interest and advertising profile • Support the searchNearByUsers • Move authorization request processing to the policy server • Centralized repository for user data storage • Trap unrecoverable errors • Centralized traces
performance	• subscribeLocation 17 ms • updateLocation (with notification) 40 ms • getLocation (depends on client response time) • addPOI: 9 ms • getPOIList: 10 ms

Location attribute is attached to the primary identity (rootId). A user A has the ability to update his own location and see user B location information if user B privacy rule allow this access. A user A has the ability to subscribe to user B location so that it will receive location notification when the user B location is updated.

Privacy rules allow the definition of rules that may request user B authorization when a user requests location or subscribe to location information. These default rules are provisioning by the application as global community policies.

Location can be queried using a partial id. So if user A partialId X subscribes to user B partialId W location, only when the user B location is modified, the user A will receive a notification with partialId X and partialId W as context.

When the location is updated, the location of all the identities is updated.

Privacy rules may restrict location information access to a list of users or a sub-community or a role in a community context. The client application is controlling these “per instance” policy rules using the policy interface to provision these rules.

Longitude, latitude must be expressed using decimal degrees. Decimal Degrees is displayed as the degrees in normal value, with both minutes and seconds in decimal format, as a degree value.

The definition of the location includes a precision field that is set by the client application. The location server doesn’t do any processing on this except that it passes that parameter as part of the location data. Precision can be set on a per requester level. It is generically attached to the location and it is up to the client application to update that parameter when necessary to preserve privacy.

The location information of a User is stored in a file repository. How any query of location for a user B will generate a RPC call to the client application to retrieve the last location GPS coordinates. The last known location is provided back when the user B is not online

Example of location call Flows

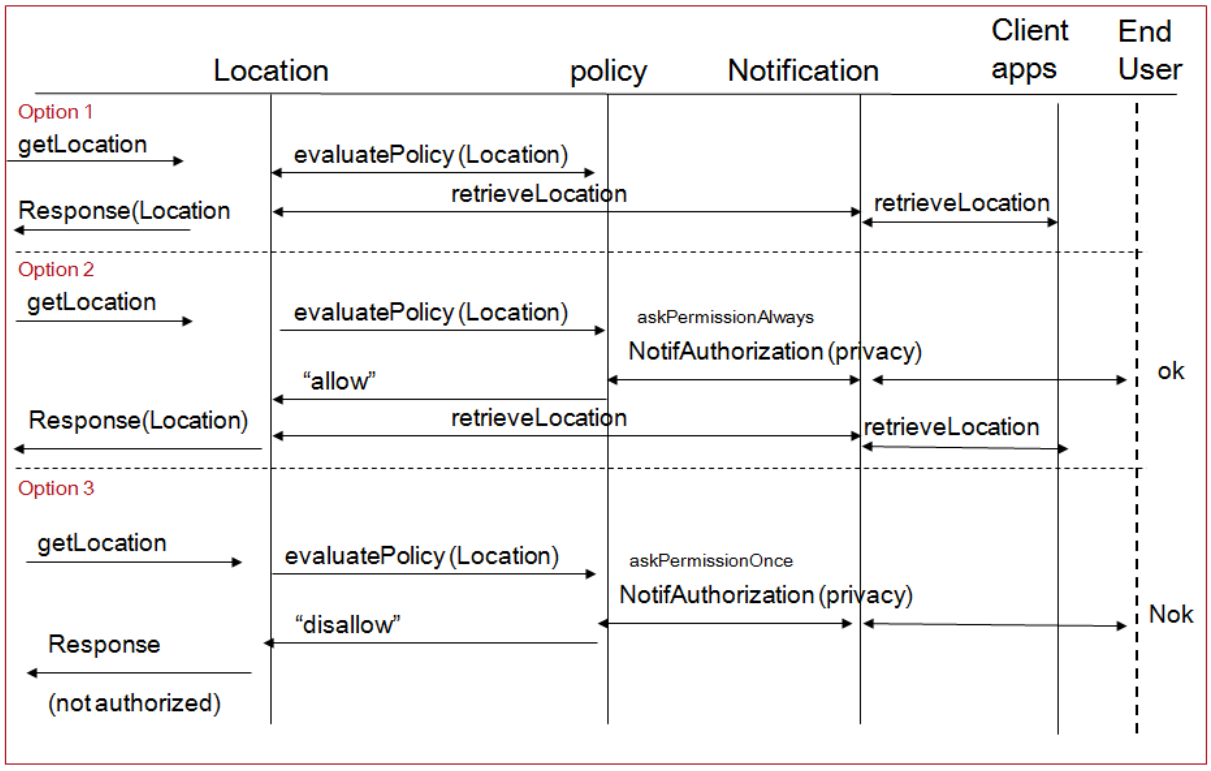


Figure 18: getLocation call flow

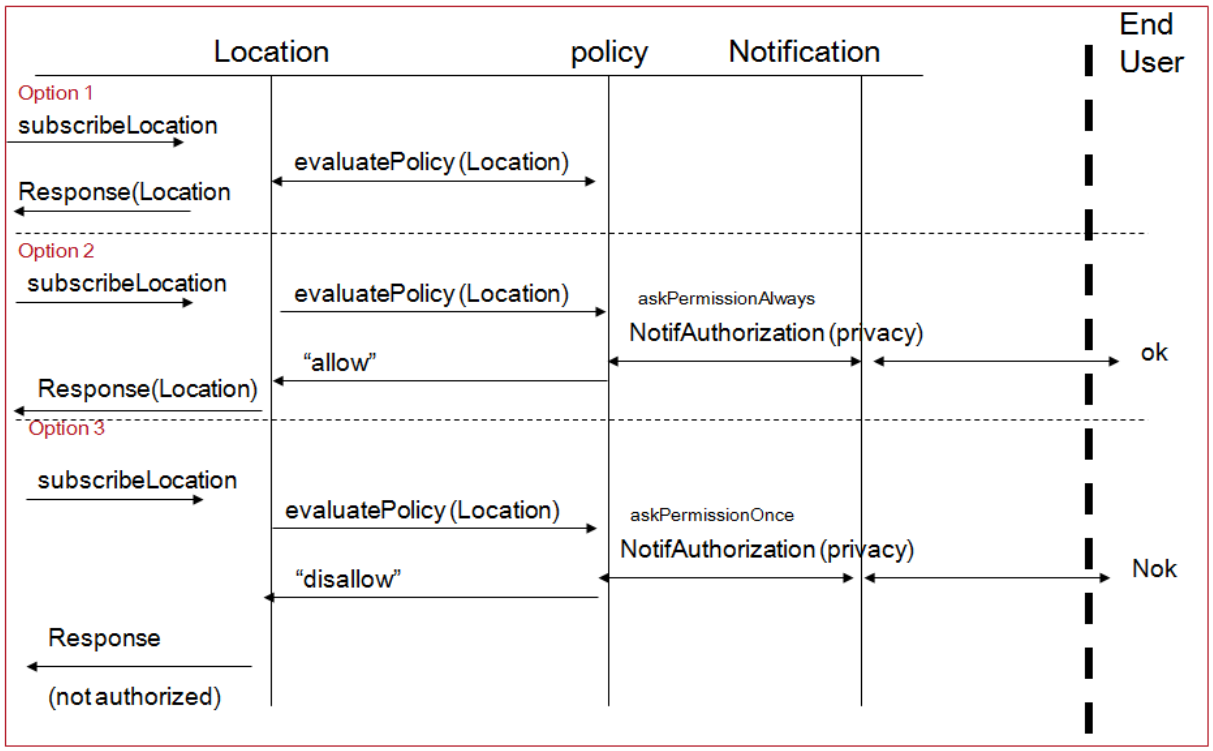


Figure 19 subscribeLocation call flow

3.3.10. Presence

Component summary	
Main functions	Manage the presence and statuses of a User or a partial identity
Component type	Script launched when a request is received
Storage	Use a specific configurable file repository to store user related presence and status information
Object/Attributes of objects the component is responsible for	• Presence attribute of the User Object • Presence attribute of the partialId Object • Status attribute of the User/partialId
Interface	• Web service (presence.wsdl), • HTTP Serialize RPC, • Script RPC
Phase 2 modifications	• Centralized traces • Support the new status feature 'add /remove status' • Move authorization request processing to teh policy server • Centralized repository for user data storage • Trap unrecoverable errors
performance	• subscribePresence 27 ms • UpdatePresence 40 ms

	• getPresence 140 ms • setStatus: 9 ms • getStatusList: 10 ms

Users register to the public community and are known under the primary identity but can also create multiple partial identities attached to the same user.

Presence attribute can be attached to the primary identity or to a partial identity. A user A has the ability to update his own presence and see user B presence information if user B privacy rule for presence allows this access. A user A has the ability to subscribe to user B presence so that it will receive presence notification when the user B presence is updated.

Privacy rules allow the definition of rules that may request user B authorization when a user request presence or subscribe to presence. These default rules are provisioning by the application as global community policies. Then the client application can customize these rules on a per primary/partial identity level.

Privacy rules may restrict presence access to a list of users or a sub-community. The client application is using the policy interface to provision these rules.

Presence can be attached to a partial Id. So if user A partialId X subscribes to user B partialId W presence, only when the user B partialId presence is modified, the user A will receive a notification with user A partialId X as context.

Presence of the default identity is automatically updated at login (online) and logout (offline). Subscription to presence update of the contacts in the contact list is also automatically achieved, providing the primary identity as the user context.

Example of call flows

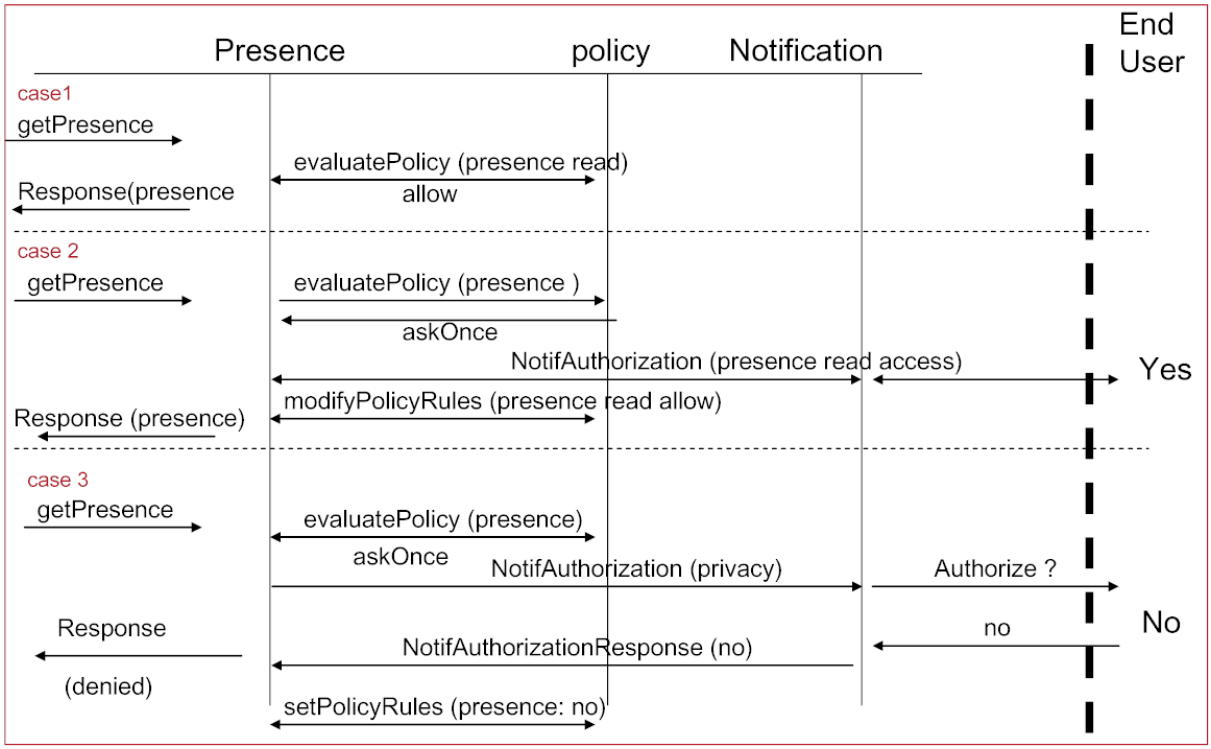


Figure 20. getPresence call flow

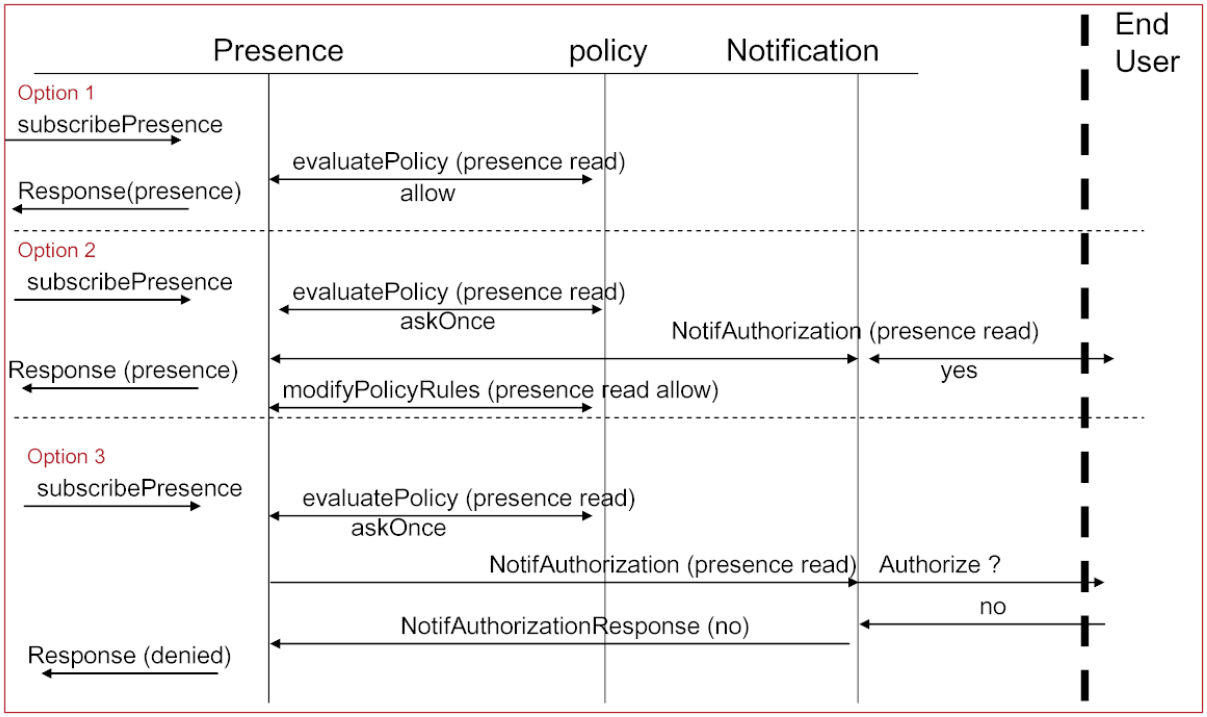


Figure 21: subscribe presence call flow

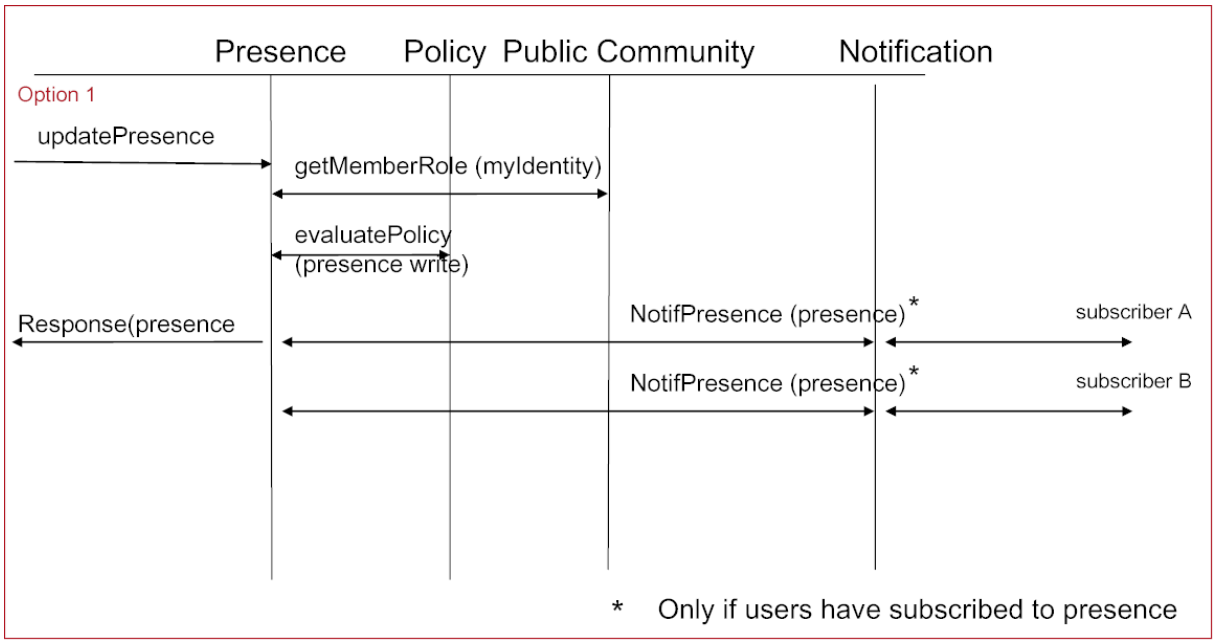


Figure 22 Update presence call flow

3.3.11. Notification / Socket server

Component summary	
Main functions	Allow platform notifications to be forwarded up to the handset via the Nat servers/firewalls
Component type	Frontend/backend architecture where the notification is the frontend and the socket server is the backend
Storage	Use a specific configurable file repository to store the "per user" pending notifications
Object/Attributes of objects the component is responsible for	None
Interface	Socket for pass through for the socket server and all types of RPCs for the notification component
Phase 2 modifications	• Support pending notifications when users are offline. • Define the 6 new notifications in the notification WSDL. •
performance	

The notification component is in charge of delivering notifications to the client application running on the handset.

It implements a front end web service server and uses a back-end socket server to interact with the handset.

The possible set of notifications has been defined in a WSDL file and the notification server acts as a web service server and relay the request up to the client.

The client RPC lib is in charge of receiving the notifications and delivering it to the client as a RPC request.

The notification component implements a generic RPC model. The delivery of the RPC request to the client application is generic. The RPC request can contains complex data structure as parameter of the notification request and the response from the client application can also contain a complex data structure that is delivered back to component requester.

Note that some notifications are informative. Others might request some End User response.

The socket server is part of the routing layer of the architecture. As such it is deployed in a DMZ zone and sends/receives direct internet traffic.

The socket server participates to the notification service. It receives connection request from client application (via the client SDK library) and allow RPC requests sent from WP5 components to be forwarded up to the client. The client response is sent back through teh same channel up to the component that sent the request.

Example of call flows

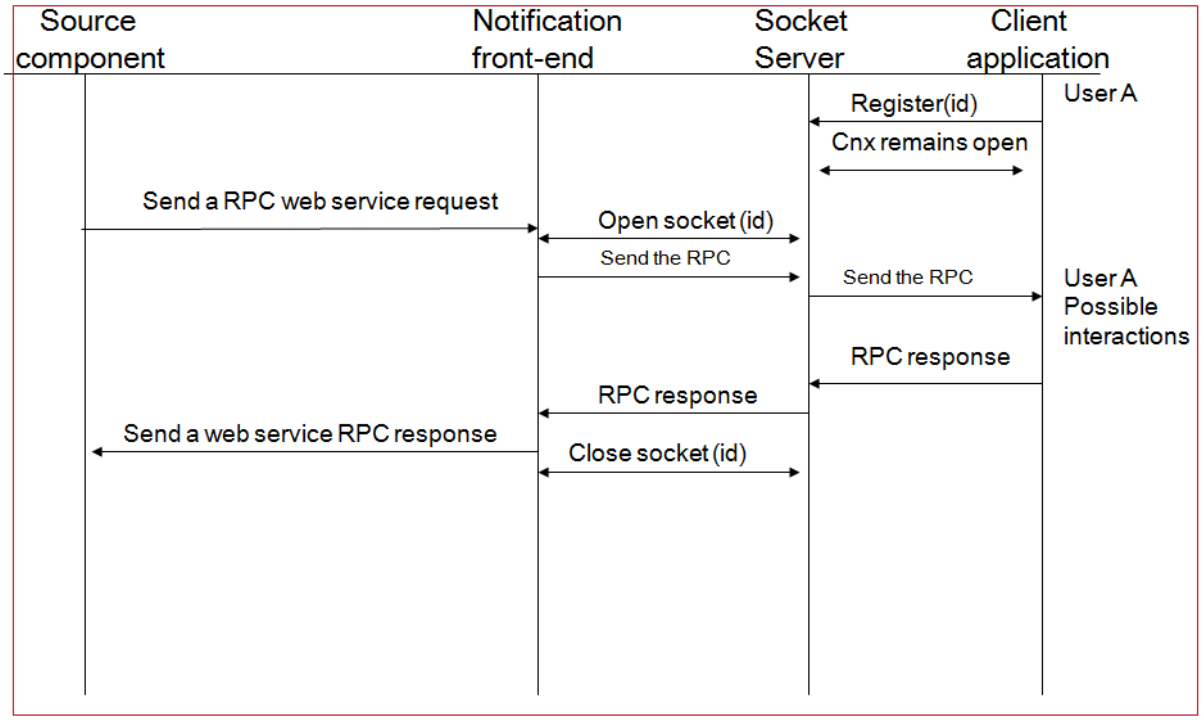


Figure 23: send a RPC request to the client.

3.3.12. Contact

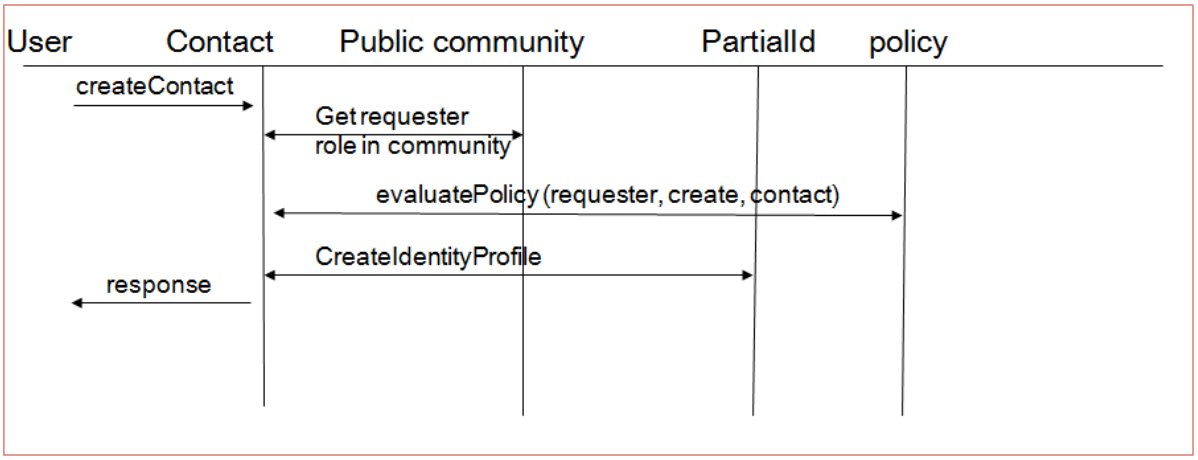
Component summary	
Main functions	Manage the policies attached to the PICOS resources
Component type	Script launched when a request is received
Storage	Use a specific configurable file repository to store contact information
Object/Attributes of objects the component is responsible for	• Contact-List Object • Contac Object
Interface	• Web service (presence.wsdl), • HTTP Serialize RPC, • Script RPC
Phase 2 modifications	• Support sharing of contact-List • Centralized traces • Centralized repository
performance	

The contact component is in charge of storing contacts in a contact list attached to either rootId or partialIds.

The component also stores contact profiles equivalent to user-profile which is filled by the owner of the contact.

Contact profiles are separated from the real profile of an identity of a user. It is a client application function to initiate the profile creation with public information of the member.

Example of Contact call flows:



3.3.13. Policy

Component summary	
Main functions	Manage the policies attached to the PICOS resources
Component type	Script launched when a request is received
Storage	Use a specific configurable file repository to store all policies
Object/Attributes of objects the component is responsible for	• Privacy rules of the User/partialId Object • Privilege rules of the User/partialId Object
Interface	• Web service (login.wsdl), • HTTP Serialize RPC, • Script RPC
Phase 2 modifications	• Generalize authorization request for the askOnce status • Add site conditions. • Change the storage of resources • Centralized repository for policy data storage • Trap unrecoverable errors • Centralized traces
performance	• evaluatePolicy 3 ms • setPolicy 40 ms • queryPolicy 140 ms • setStatus: 9 ms • getStatusList: 10 ms

The policy manager is in charge of storing and evaluating rules associated to resources

Policy information is stored in a multi-level table that uses the description level (array of couple (name, id)) as indexes in the policy table.
The model allows the creation of infinite number of levels in the table offering optimum flexibility.

If a resource is defined as an array ((name1, id1), (name2, id2), (name3), (name4, id4), any attached rule is stored in the table:

$resources['name1'] ['id1'] ['name2'] ['id2'] ['name3'] ['name4'] ['id4'] []

The value of the PHP language is that this table accessed is done by creating the above "string" and then uses a facility to execute the string. This provides a very efficient access to rule information and limited degradation when the data base of rules increases.

Identity Condition evaluation:

The Policy engine assumes that the user roles of the requester are provided by the calling component.

Reputation Condition evaluation:

If the Policy engine evaluates a rule with a reputation condition, it is in charge of querying the reputation component on the reputation of the requester

Site Condition evaluation

If the Policy engine evaluates a rule with a site condition, it is in charge of querying the site component to retrieve the site attributes and querying the location component to retrieve the location of the resource owner

Policy Storage

The Policy server stores root object and associated Objects/attributes into a single file separating contexts and optimizing the context loading when the policy engine is loaded. Each file stores a subset of the complete resource table i.e. all subsequent resources and rules attached to the table *$resources['name1'] ['id1'] ['name2'] ['id2']*.

The PICOS platform is provisioned with policy rules that describe which role can do which action for which resource. These default rules can be easily customized as they are gathered into a single PHP script that is launched during initial configuration time (See installation and configuration chapter page 105).

Example of call flows

Call flows executed by all components before performing a requested action.

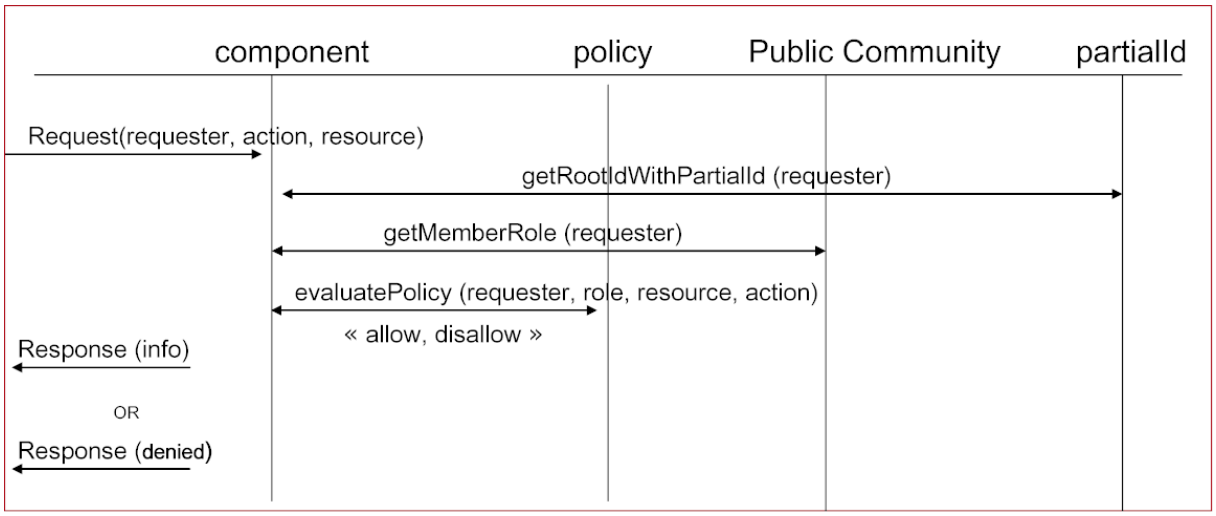


Figure 24: Policy enforcement involving all components and the policy server

3.3.14. Site

Component summary	
Main functions	Manage the site attribute of a user/identity
Component type	Script launched when a request is received
Storage	Use a specific configurable file repository to store user related site information.
Object/Attributes of objects the component is responsible for	• Site Object
Interface	• Web service (site.wsdl), • HTTP Serialize RPC, • Script RPC
Phase 2 modifications	• New component • Support site object management (create, delete, get list, get attributes) • Support centralized repository of user data • Support Centralized traces • Trap unrecoverable errors

The site component is in charge of managing creation and deletion of personal site. A site is defined as an area around a location.
Site are private Objects that cannot be shared. It role is to create rules with site conditions. Site could also be referenced when GPS doesn't work well (like inside the buildings)
The site component offers a classical interface for managing sites:

- Create Site

- Delete Site
- Get Site attributes
- Get Site List

3.3.15. Subscription

Component summary	
Main functions	Manage the subscription to action on community resource
Component type	Script launched when a request is received
Storage	Use a specific configurable file repository to store user related subscription information
Object/Attributes of objects the component is responsible for	• Relates to all community objects
Interface	• Web service (subscriptionWebService.wsdl), • HTTP Serialize RPC, • Script RPC
Phase 2 modifications	• Support object subscription management • Support centralized repository of user data • Support Centralized traces • Trap unrecoverable errors
performance	• Subscribe 4 ms • notifyAccessedElement (without notif): 4 ms • notifyAccessedElement (with one notif): 10 ms

The subscription component is designed to offer a generic subscription model for being alerted when an action has been performed on a community Object.

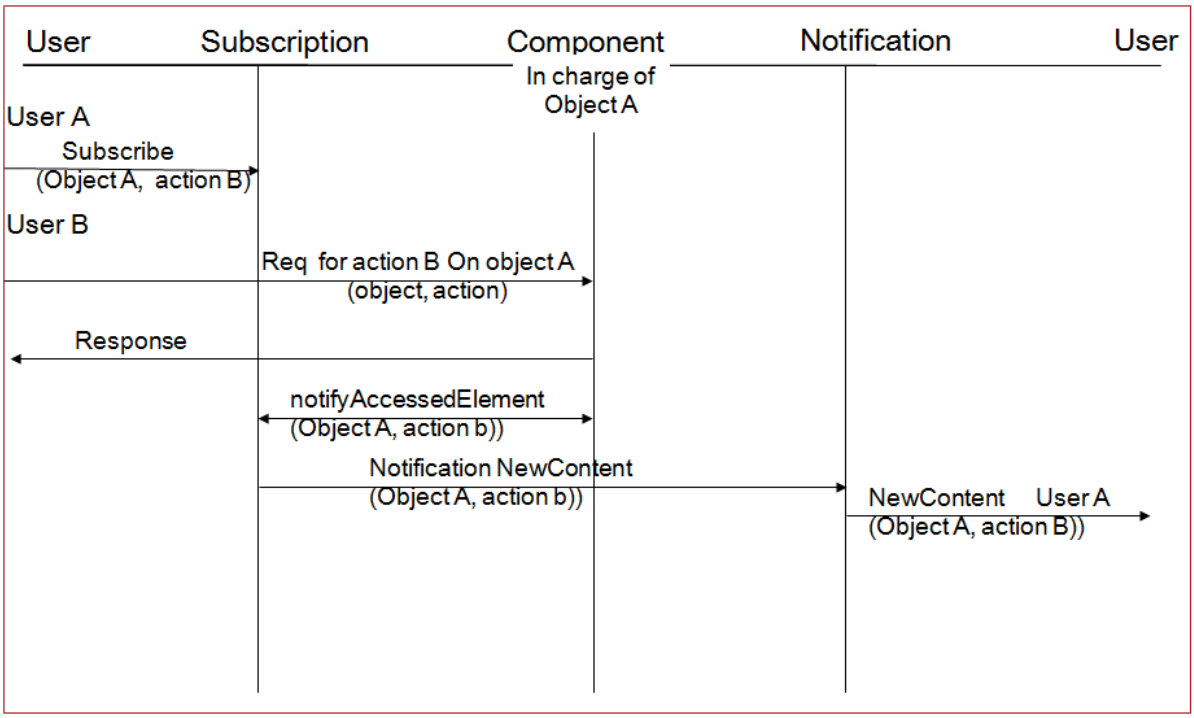


Figure 25: overall call flow for subscription service

The component is charge of the Object A receives a request to perform an action on this object (ex read the Forum Object). The component informs the subscription component of this action.

The subscription component checks if there are subscribers to this couple (object, action) and generate notification to all subscribers
The subscription component uses the notification service to deliver the alert up to the end user including some object attributes.

The subscription component doesn't checks whether or not the couple (resource , action) is valid.
The component re-uses the data structure implemented for policy where the same resource (Object) description is used.

Subscription information is stored in a multi-level table that uses the description level as index in the table.
The model allows the creation of infinite number of level in the table offering optimum flexibility.

If a resource is defined as array ((name1, id1), (name2, id2), (name3), (name4, id4), any subscription is stored in :

$elements['name1'] ['id1'] ['name2'] ['id2'] ['name3'] ['name4'] ['id4'] []

The value of PHP is that this table accessed is done by creating the above "string" and then uses a facility to execute/evaluate the string as if it was a PHP statement. This provides a very efficient

access to subscribe information and limited degradation when the data base of subscribed objects increases as well as support an “infinite” depth of the table.

The subscription component is designed for sending notifications to End users via the client application but could be extended to offer a subscription service between the platform components

Actually, the public community (for forum Object, forum-thread, contribution, category, content Objects) and the sub-community (for sub-community content, thread Object) are generating the “notifyAccessed” request

3.3.16. Public community

Component summary	
Main functions	Manage public repositories and forum as well as public community members
Component type	Front end / back end architecture
Object/Attributes of objects the component is responsible for	• Public community Category Object • Public community Forum Object • Public community member Object
Interface	• Web service (login.wsdl), • HTTP Serialize RPC, • Script RPC
Phase 2 modifications	• Centralized traces • Support content publisher revocation • Support category creator revocation • Support Forum creator revocation • Enforce rules per instance of category, forum, forum-thread, contribution Objects • Support content access history • Support forum-thread access history • Support notification of new action on public community objects. •
performance	• CreateCategory 25 ms • CreateForum 30 ms • CreateForumThread 30 ms • Create Contribution (with 4KB content) 130 ms • addContentToCategory (with 30 KB content) 180 ms

Once the User is registered and has logged in, he can access the different functions of the public community. The public community offers two major services; one service for asynchronous communication between members of the public community i.e. Forums and one service to store and retrieve content to/from the public repository.

Accessing Forums

Members of the public community can create Forum of discussions or use existing forum which might have been provisioned before the start of the community.

Once the forum is created with at least one moderator, members of the public community can start creating forum threads by pushing an initial message. Content can be attached to forum thread contributions as well. Contribution can then be pushed, deleted and edited by the owner of the contribution.

User can start seeing the list of forum with title and description, select one forum and see the list of threads of discussion in the forum. They can enter a thread of discussion and decide to contribute to the on-going discussion. Each contribution has attributes including the publisher pseudo and its reputation/number of owner contributions as well as the rating/number of vote of the contribution by itself.
Each member of the public community can read a contribution and rate it whereas contribution owners can decide to edit or delete their contribution.

User can participate to Forum using one of their identities.

The moderator has the right to delete/ edit any contribution to update the text or remove attachment. The contribution is now locked and cannot be edited again. Moderators can also copy a thread from one forum to another one.

The client application can decide to allow association of privacy rules to each contribution that is published. Privacy rule can apply to the contribution by itself or to sensitive attributes of the contribution like the publisher information..

Policy enforcement has been modified to allow rule definition per public community Object instance (define a specific rule for a particular forum)

Accessing the public store

A public repository is made available to push content to a general store. Content might be subject to editing before being made available to the member of the public community.

Members of the public community can create and open categories where public content is stored or create new ones. Categories can be cascaded (category attached to a category). Publisher of the content have the access right to edit or delete content they have published.

Content is defined as a set of attributes (the meta data of the content) and the content data by itself. A Meta Content is defined as an array of content.

Meta Content can be stored into categories.

A category has attributes that can be modified. For a multimedia description of the category, a Meta Content can be attached to the category.

The client application can decide to allow association of privacy rules to each content they publish via the policy manager. Privacy rule can apply to the content by itself or to sensitive attributes of the content like the location information.

Policy enforcement has been modified to allow rule definition per public community Object instance (define a specific rule for a particular content)

Example of call flows

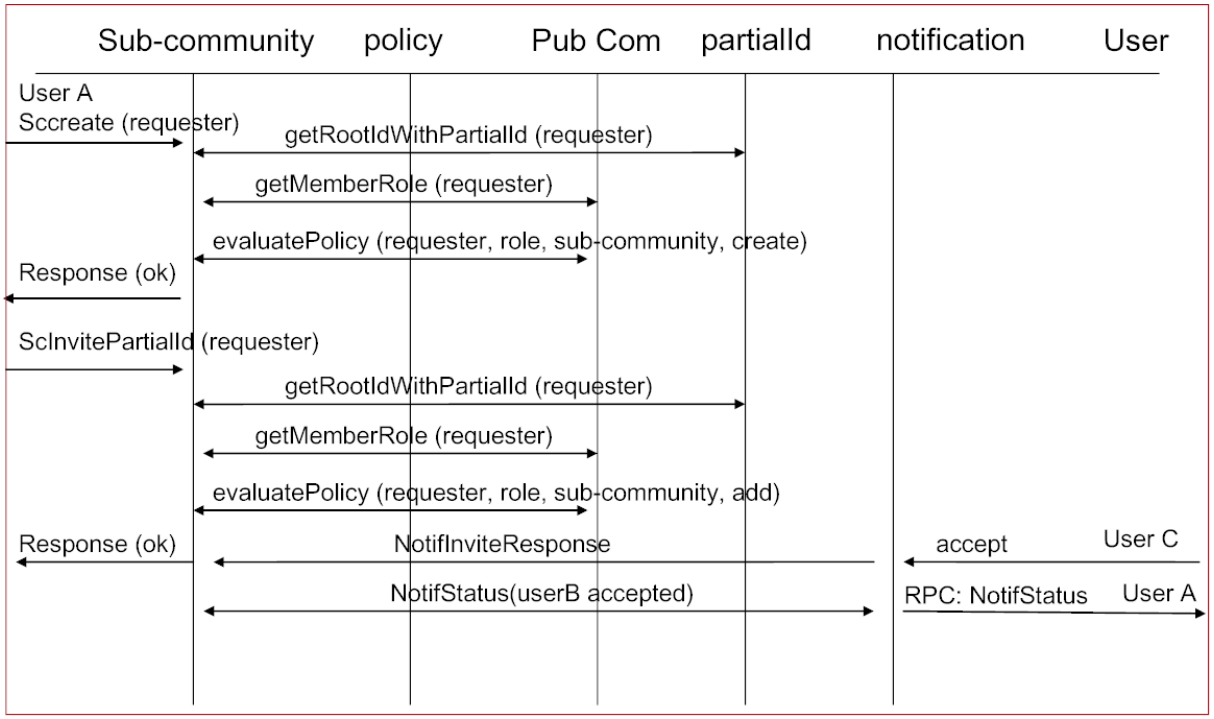


Figure 26: create sub-community call flow

3.3.17. Real time (RT) content sharing

Component summary	
Main functions	Manage the real time communications between users manage ana synchronous Message Inbox
Component type	Front end / back end architecture
Storage	Use a specific configurable file repository to store user related communication and user inbox information
Object/Attributes of objects the component is responsible for	• communication Object • User inbox Object
Interface	• Web service (login.wsdl), • HTTP Serialize RPC, • Script RPC

Phase 2 modifications	• Support access to archived chats. • Support content sharing over chat • Parallelisation of invitation and message transmission • Support centralized repository of user data • Support Centralized traces • Trap unrecoverable errors
performance	• Create communication 13 ms • sendMessage 80 ms (3 participants)

The real time content sharing component is in charge of setting up instant group chats or group content sharing communications.

The communication establishment is achieved via an invitation process of a set of members of the public community that may come from the contact list of the communication creator.

Users that accept to be part of the group chat accepts to dedicate their time for that chat. When a user accepts to be part of the communication, all participants receive a notification that he has accepted to be part of the communication.

Messages and content pushed by one member are immediately delivered to those who have accepted to be part of the communication.

The communication can then be closed by the creator of the communication. Participant can leave the communication at any time. The remaining participants are informed on any participant leave.

No content or messages are stored once the communication is closed.

To avoid serial process of multiple invitations that could lead to timeout, the Real Time content sharing component has implement a thread emulation process (thread are not supported by PHP) to support concurrent invitation processes (invitation notification waiting for user approval).

The same model has been put in place for the transmission of messages.

Example of call flows

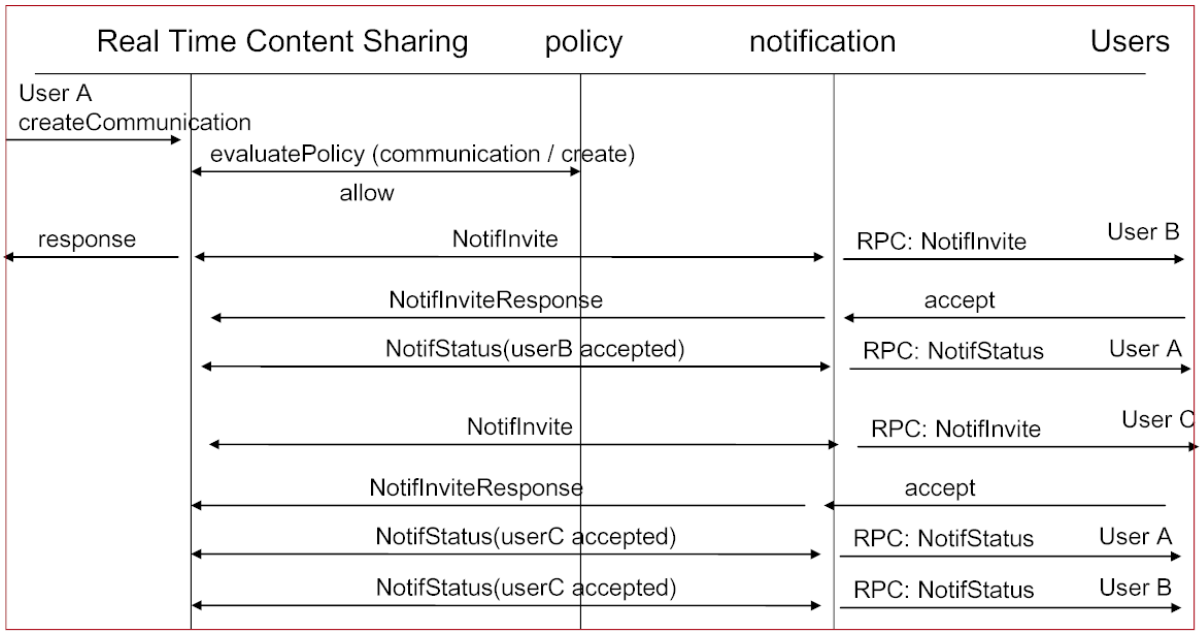


Figure 27; create a communication (chat) call flow

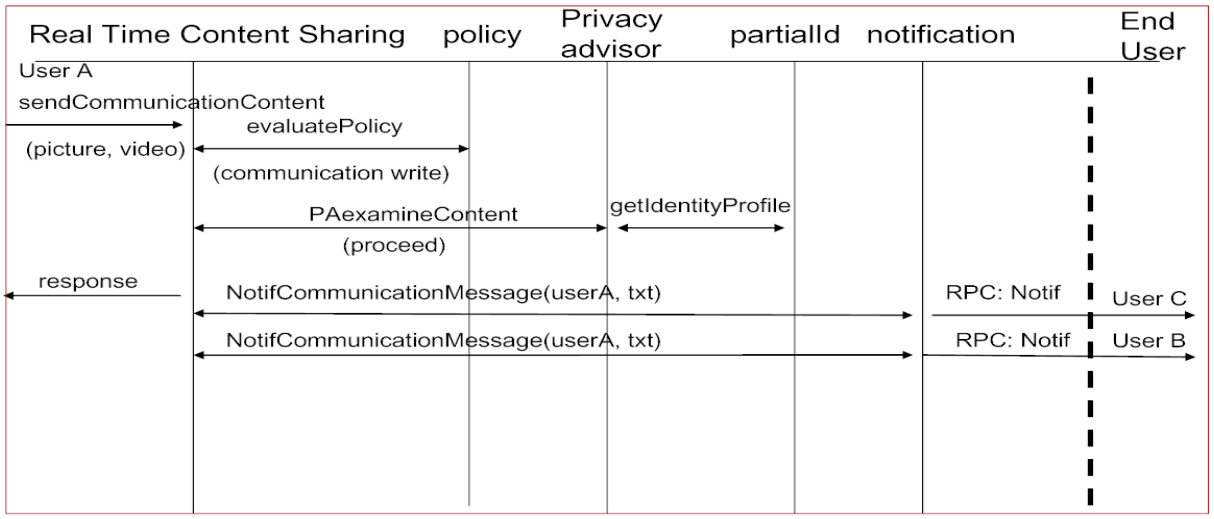


Figure 28: Send Communication Content call flow

3.3.18. Private room (My Files)

Component summary	
Main functions	Manage user's private room with files (and catch reports for anglers)

Component type	Front end / back end architecture
Object/Attributes of objects the component is responsible for	• Private Room Object • Private Room Content Object
Interface	• Management Web service (PrivateRoomWebService.wsdl), • Content WebService (PrContentWebService.wsdl) • XML-RPC
Phase 2 modifications	• Centralized traces • Centralized WebService Factory
performance	• No performance tests available

The private room component is responsible for managing the users' private room and the corresponding content objects.

Once a user registers to the community a single private room is created for this particular user.

Users can use the private room to store private data like files (or catch reports in the angler version) in a secure place which is not accessible for other users. The user can modify and prepare his content and then transfer it either to various sub-communities or to a category inside the public community.

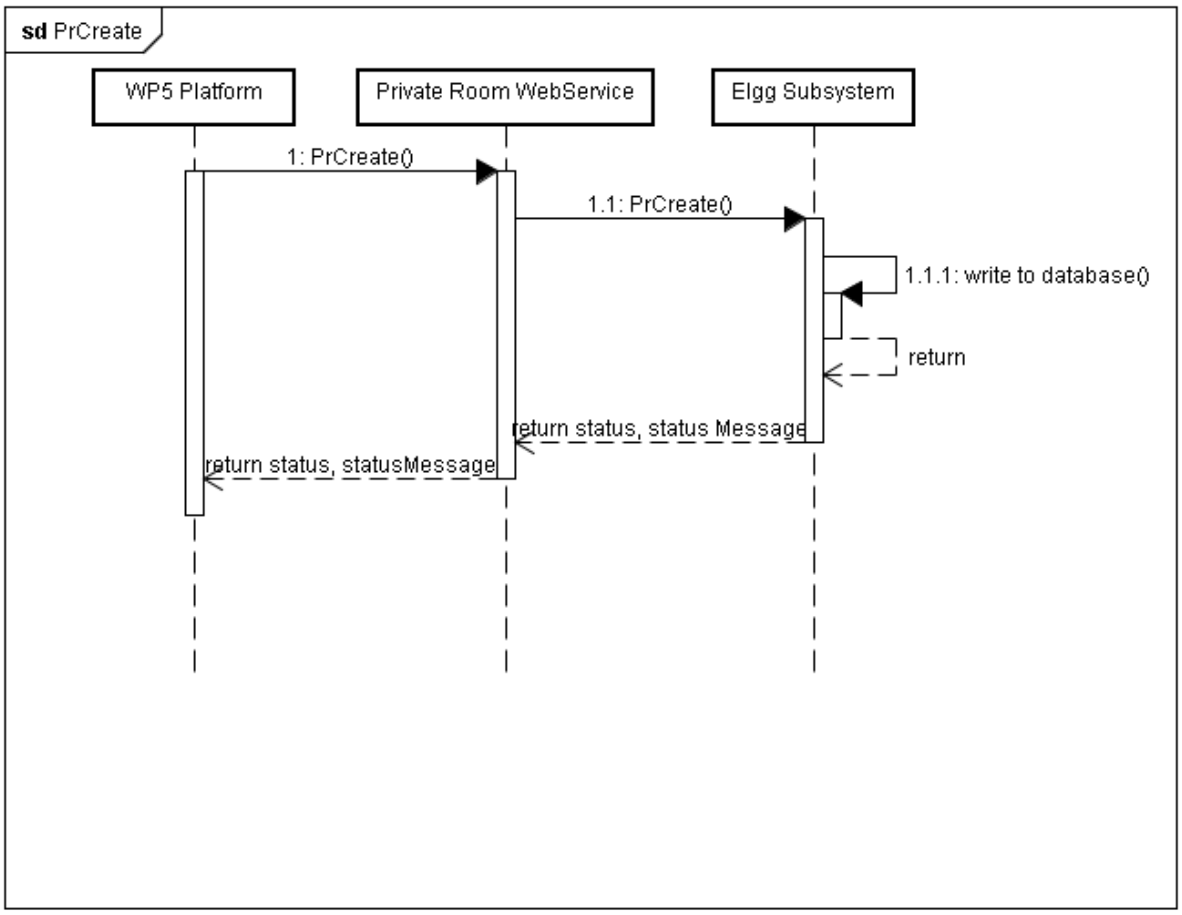
When the user unregisters the private room will be closed and content deleted

The private room component is a front end / back end component where the front end part implements the web service interface and the communication with the other PICOS components whereas the back-end part is dealing with the interface with the ELGG sub-system.

Publishing Content

The user of the private room can upload e.g. pictures to his private room. He can then e.g. edit the title and publish this picture to a sub-community or a category of the public community. During this process he has the possibility to add access restrictions either time based, user based or both. The file will then be only accessible during the given timeframe or by the specified users.

Example of Call Flows



3.3.19. Sub-community

Component summary	
Main functions	Manage the sub-communities with files, forum (and catch diaries for anglers)
Component type	Front end / back end architecture
Object/Attributes of objects the component is responsible for	• Sub-Community Object • Sub-Community Forum Thread Object • Sub-Community Repository Object • Sub-Community Content Object
Interface	• Management WebService (SubCommunityWebService.wsdl), • Content WebService (ScContentWebService.wsdl) • XML-RPC
Phase 2 modifications	• Support content publisher revocation

	• Support Forum-Thread creator revocation • Enforce rules per instance of forum-thread, contribution Objects • Support content access history • Support forum-thread access history • Support notification of new action on public community objects. • Shared Desk functionality • Centralized traces • Centralized WebService Factory
performance	• No performance tests available

The sub-community component architected as a front end / back end component using a XML-RPC communication between them.

It is responsible for managing the sub-communities and the corresponding content objects as well as the forum-thread object.

A user can freely create private or public sub-communities. Other users can freely join public sub-communities where as for membership in a private sub-community an invitation by the creator is needed.

Shared Desk

The shared desk is not implemented as an own component. Because the shared desk is a kind of special type of sub-community, this component is responsible for managing the shared desk as well.

A user can freely create a shared desk. Other users can only be invited to participate in this special kind of group.

In contrast to the normal sub-communities only the creator of the shared desk is able to publish content files. The other members can only read the content but cannot discuss it or post own content to the shared desk.

Accessing Forum Threads

Members of a sub-community can create a discussion and take part in a discussion (thread) which has been started by another member before.

A thread is created by giving a title and an initial message with an optional content attachment. The corresponding contributions can then be deleted and edited by the owner of the contribution.

Users start seeing the list of threads in the forum. They can enter a discussion and decide to contribute to the on-going discussion. Each contribution has attributes including the publisher pseudo and its reputation/number of owner contributions as well as the rating/number of vote of the contribution by itself. Each member of the sub-community can read a contribution and rate it as long as access is not restricted for him.

The sub-community creator has the right to delete/ edit any contribution to update the text or remove attachment.

Most of the default forum policy rules are defined using the policy manager and pre-provisioned before the start of the service.

For the gamers' platform policy enforcement has been added to threads so that the creator of a thread can restrict access either time based, user based or both.

Accessing the Repository

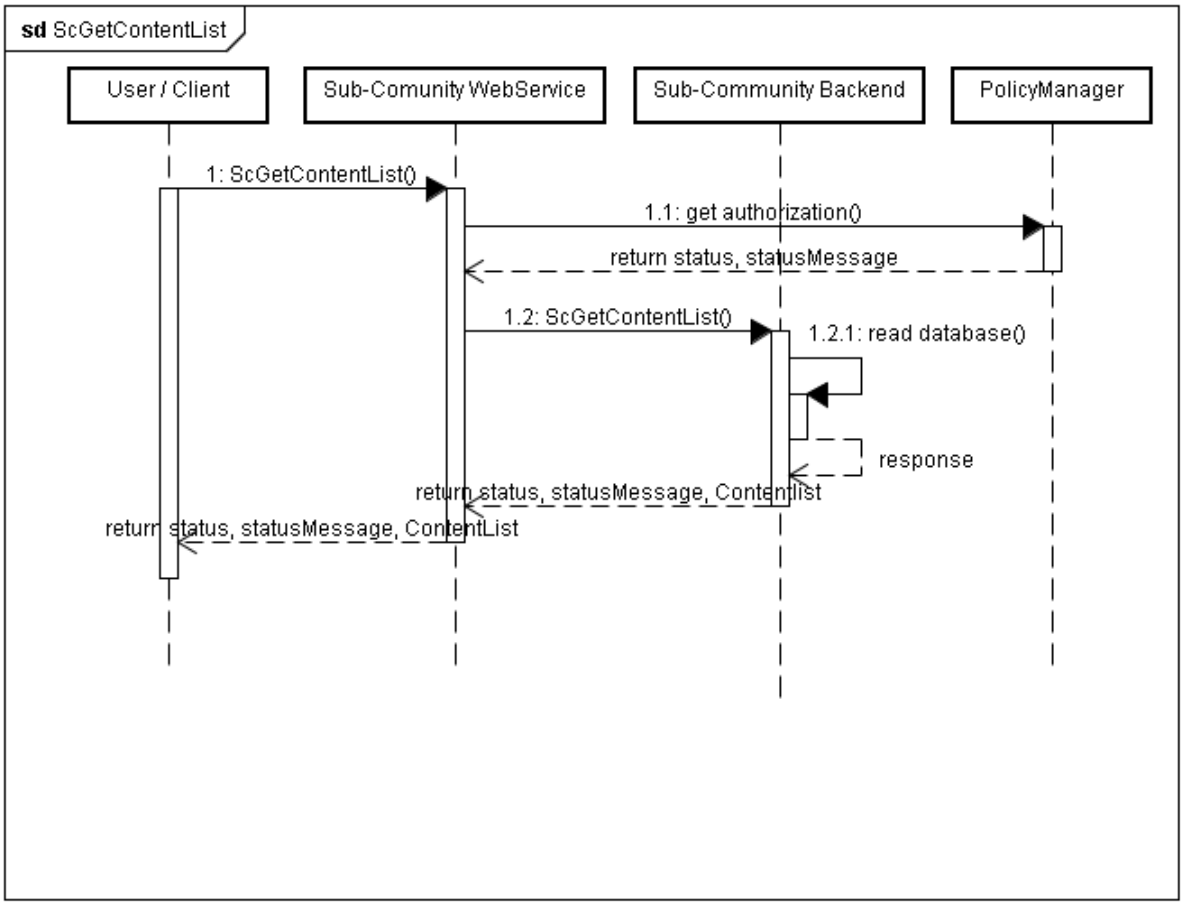
A content repository is made available to push content to a store inside the sub-community. Content might be edited inside the private room before being made available to the members of the sub-community.

Publisher of the content have the access right to edit or delete content they have published.

Content is defined as a set of attributes (the meta data of the content) and the content data by itself.

For the gamers' platform policy enforcement has been added to files inside the repository so that the publisher of a content item can restrict access either time based, user based or both.

Example of Call Flows



3.3.20. Logging server

Component summary	
Main functions	Log the component events , retrieve event using filtering criteria
Component type	Front end / back end architecture
Storage	Use a specific configurable file repository to store user related logging information
Object/Attributes of objects the component is responsible for	• none
Interface	• Web service (login.wsdl), • HTTP Serialize RPC, • Script RPC
Phase 2 modifications	• No major enhancement. • Support centralized repository of user data • Support Centralized traces • Trap unrecoverable errors
performance	• logEvent 3 ms

The logging server is used by the components to store information about platform events. The event details contain a clear text describing the event as well as a data structure that contains the characteristics of the event. Any logging requester can select one or multiple criteria in the data structure for event filtering.

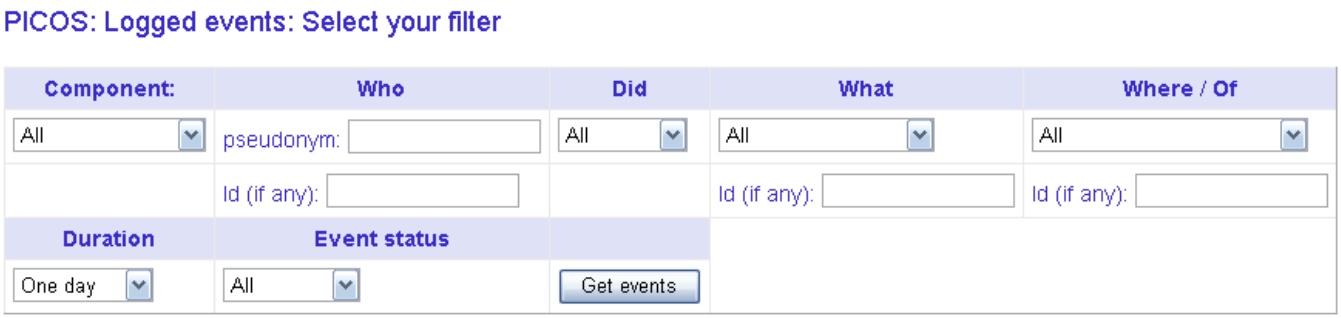


Figure 29: admin console screen for selecting a filter to get events

Filter criteria can be:

- Which component logged the event
- Who performed the action (name and id)
- Which action was performed
- What resource was involved (name and possibly an id

- Where the resource is located or who is the owner of the resource (name and possible an id)

The event logging component returns then all events that match all defined criteria.

Event are incrementally stored into a file (one file per day), making the event logging performing well. Decoding consume more CPU but might be done off-line.

The component offers the ability to retrieve events over multiple days

3.3.21. Centralized PICOS library

In phase 1, some methods were duplicated among the components, especially the encapsulation of the RPC access to component methods. A set of re-usable methods have been centralized in a "PICOS_util" directory and PHP scripts are included by components, thus reducing the PICSO foot print, code duplication and centralizing some functions like inter component communication for easy customization

3.3.1. Configuration scripts

The general configuration of the platform is achieved via a centralized script that includes all information related to :

- The host server for the platform and each separate component
- The URL access paths to support the RPC calls and web console
- The TCP socket ports for the frontend/backend components
- For each component, the configuration file contains:
  - The path to store the traces
  - The path to store the persistent data
  - The path to launch the component PHP scripts
  - The described methods of each component and its access control over the proxy.

Another script (createCommunity.PHP) is used to initially create the community by:

- Setting the default rules for each object
- Creating an admin user to login the admin console

3.3.2. Operation scripts

Operation can be achieved using the admin console or 3 basic scripts are provided to perform action like

- checkServer.PHP (check that all frontend/backend components that need to be started are up and running)

- stopServer.PHP stop all frontend/backend components
- restartServer.PHP: start all frontend/backend components

3.3.3. The PICOS platform admin console

In order to manage the platform, a web admin console is defined allowing the management of various administrative functions as well as operation functions.

Note that the purpose of the admin Console is not to offer a web access to community members

The admin console is accessible using the following URL:

> <root platform URL>/adminConsole/index.php
> In the installation described in the document the root platform URL is :
> https://hostname/webservices/PICOS/

Note: the root platform URL (can be changed during installation, thus changing the PICOS admin console access URL.

The admin console is a web application that is accessed with a browser and that uses the web service interface of the platform components to perform actions or retrieve data.

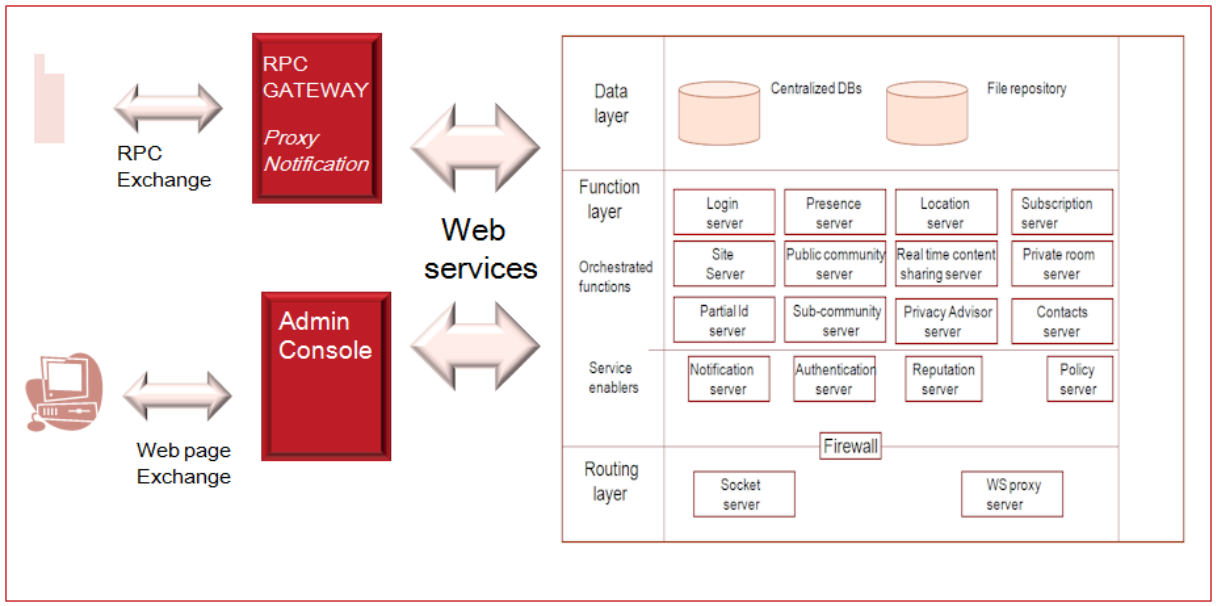


Figure 30: Comparing mobile access and admin console accesses to the platform

The PICOS project receives research funding from the Community's Seventh Framework Programme.

The access to the admin console requires a login with admin capabilities. Note that the admin console login is designed to work even if the complete platform is not started. The admin role allows the user to perform all functions and access any user context. SSL channels are used to secure the transport of sensate information over Internet.

During the initial installation and configuration, an admin login is automatically created (login: padmin, password: padmin). During the first login to the admin console, it is recommended to create a new admin user and delete the default one (or change his password)

The admin console offers a main navigation using a left menu

Figure 31: admin console web screen showing the public repositories (categories)

The admin console has been designed to administer the community, fill the repositories with valuable content or moderate the forum and cleaning up the discussion when necessary. It is also designed to operate the platform with restart platform function or statistics retrieval.

The admin console contains the following features (in bold new features added in phase 2)

features	Description
Public community attributes	Get, update the general announcement, the terms and conditions
Public community members	List of members Search member Revoke member or identity of a member Display member details Register new users / **update profile** **Create / update user identities and associated attributes** **List policies attached to user/identities and attributes**
Public community repository	Create / delete category Add / delete (meta) content in category List categories/sub-categories **Create and List policy rules attached to categories and content**
Public community forum	Create / delete forum Add / delete discussion in forum List forum /discussions **Create and List policy rules attached to forum and forum-thread**
Platform usage	Display logging Statistics Most active users
Platform operator	Check servers Restart servers
Point of interests	Create/delete/ update POIs Manage advertising profile. **Create and List policy rules attached to POIs**
Send/receive messages	Send message to any member of the community Get message from any member to manage complaints using the reply feature of the client

4. Implementation strategy

4.1. Definition

Brainstorming has been conducted between partners to refine the value proposition select which uses cases had to be implemented in the first prototype and select the necessary components accordingly.

An internal Gamers requirement document has been written.

4.2. Specification

Once the scope of the platform was started to be shaped, it was critical to define the platform functionality and its interface from an external standpoint so that the application team could start define the end User application.
A functional specification document (main feature and interface description) of 400 pages has been written and reviewed. All review comments have been gathered into a Microsoft excel form with a status and action per comment.

4.3. Design

The design phases consisted in defining the overall architecture and the internal API between the orchestrated layer and the service enabler components. Some of the component API methods are directly accessed by the client, other are reserved for WP5 platform internal usage. These methods cannot be called by the client. This method access control is done by the RPC gateway.

Call flows between components have been produced. Their goal was validate component needs against the proposed internal API and evaluate further integration tests to be performed during the integration phase as they were showing component dependencies.

The internal and external interfaces have been defined using the WDSL language. From a Client application standpoint, the platform was perceived as having a single WSDL document that was describing the platform interface.

4.4. Coding

The WP5 PICOS platform has been written using web technologies including web services and the PHP language. The PHP language is a high level scripting language that offers outstanding new flexibilities in the development of new applications. The web service support is well integrated and allows in a few lines of code to generate web service requests that is managed by the platform or by an external web resource. The PHP language is well recognized for rapid prototyping of complex applications. The PHP language is supported on various platforms including windows and Linux platforms.

The WP5 PICOS platform is based on Linux RedHat operating system and various open source software have been installed including apache, PHP, libpng, libjpeg and ELGG .
The RPC gateway, which is a HP asset, is also required to translate the client RPC calls into web services requests.

Each component owner was responsible for doing their component module testing.
The component testing has been achieved by creating a PHP script for each web service entry point of the component. Each PHP test script was offering a rudimental web interface to fill the parameters of the request and test various scenarios.
For components that were using other component services, emulation of these components were made available.
These test scripts were also made available to WP5 partners as well as WP6 so that they could create platform context (such as create new users) before the client application was able to perform it.

The delivery model for the complete platform feature set has been an incremental delivering of features over 10 releases for the phase 1 and 5 for the phase 2..

4.5. Integration

Due to the light weight used application execution environment, it was possible to provide early in the process a WP5 PICOS platform with all the components either with limited (emulated) capabilities or with the finalized component. Step after step, the emulated components have been replaced with final components.

Five platforms have been installed from the very beginning, each platform being accessible from outside and allocated to one partner so that risk and test impact was minimized. The five “platforms” are running on a single server.
Each partner could then develop their own component on their system and could do a remote integration of their component by accessing the platform and by reconfiguring the platform so that their component was used instead of the emulated one.
Each component owner was responsible for the integration with service enablers components (policy, profile, logging..) and HPF was responsible for validating the component in the platform using the test suite.

The private room and sub-community components have been integrated to the platform by running most of the platform components on HPF Grenoble site and by running the private room and sub-community components on servers in Germany and by re-configuring the platform to properly locate the remote components.

4.6. Test strategy

In order to validate the 200 web service requests defined in the WP5 PICOS platform interface, a test suite has been developed that covers around 230 usage scenarios with a 100 % tested method coverage. The branch coverage has not been evaluated.

The test suite offers a way to directly test the PICOS platform components (emulation of WP5 component interactions). It also offers an emulation of the client to query the platform services over the RPC gateway. The same tests can be run in these two environments.

In order to test sophisticated scenarios where the WP6 client application is supposed to receive notifications or request, the test suite embeds an emulation of the client application for the notification process. The test suite has then the ability to test various client application answers.

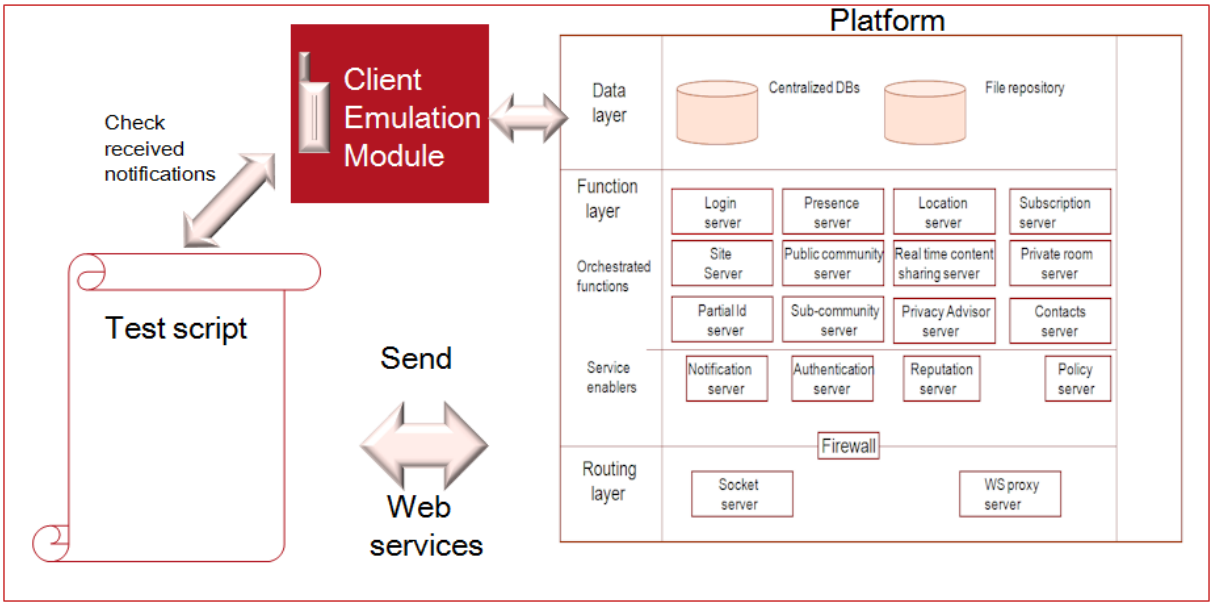


Figure 32: Complete chain emulation for non reg test suite.

The WP5 team uses the same tracking system than the WP6 team, easing the bug correction process.

See below the actual number of direct tests per component:

component	Number of test scripts	number of methods
registration	8	2
login	7	2
contact	6	7
location	26	18
partialId	20	18
policy	0	8
presence	17	10
Private room	10	10

Privacy Advisor	6	5
Public community	66	39
reputation	6	12
RT content sharing	6	19
Sub Community	37	27
Notifications	3	11
Site	9	5
Subscription	4	3
Authentication	0	6
Total	231 tests	202 methods

4.6.1. Registration

Test name	Purpose
PICOS-WP5-PROTO1-REGISTRATION-001	Basic Registration
PICOS-WP5-PROTO1-REGISTRATION-002	Registration with all parameters
PICOS-WP5-PROTO1-REGISTRATION-003	Registration with login already used
PICOS-WP5-PROTO1-REGISTRATION-004	Registration with pseudo already used
PICOS-WP5-PROTO1-REGISTRATION-005	Registration while server is down
PICOS-WP5-PROTO1-UNREGISTRATION-001	Un-registration
PICOS-WP5-PROTO1-UNREGISTRATION-002	Un-registration with a non registered rootID
PICOS-WP5-PROTO1-UNREGISTRATION-003	Un-registration while server down

4.6.2. Login

Test name	Purpose
PICOS-WP5-PROTO1-LOGIN-001	Login
PICOS-WP5-PROTO1-LOGIN-002	Login with wrong loginName
PICOS-WP5-PROTO1-LOGIN-003	Login with wrong password
PICOS-WP5-PROTO1-LOGIN-PARTIALID-001	Login, create partialId logout and

	login again
PICOS-WP5-PROTO1-LOGIN-PARTIALID-002	Login, create partialId, set default partialId logout and login again
PICOS-WP5-PROTO1-LOGOUT-001	Logout
PICOS-WP5-PROTO1-LOGOUT-002	Logout with wrong rootId

4.6.3. partialId

Test name	Purpose
PICOS-WP5-PROTO1-PARTIALID-CREATE-001	PartialId creation
PICOS-WP5-PROTO1-PARTIALID-DELETE-001	Delete partialId
PICOS-WP5-PROTO1-PARTIALID-GET-ROOT-001	Get rootId of partialId
PICOS-WP5-PROTO1-PARTIALID-GET-LIST-001	Retrieve list of identities
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-001	Get identity profile
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-002	get profile from another user, rule allow
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-003	get profile from another user, no rule to allow --> should see the displayname only
ICOS-WP5-PROTO1-PARTIALID-SEARCH-PSEUDO-001	create partialid and search pseudo
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-004	get profile from another user, rule for askOnce authorize ok
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-005	get profile from another user, update to add age attributes, rule to allow only the age get profile
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-006	use partialId, get profile from another user, update to add age attributes, rule to allow only the age get profile
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-007	use partialId, get profile from another user, update to add age attributes, rule to allow only the age get profile then modify the rules

PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-008	**Get profile from another user, update to add age attributes, rule to allow user-profile get profile**
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-009	**Get profile from another user, update to add age attributes, rule to allow user-profile to all members & get profile**
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-012	**update a root profile, get profile**
PICOS-WP5-PROTO1-PARTIALID-GET-PROFILE-013	**update a root profile, update a prtialId profile, get a partialId profile**
PICOS-WP5-PROTO1-PARTIALID-ALARM-001	**set one alarm**
PICOS-WP5-PROTO1-PARTIALID-ALARM-002	**set one alarm, delete alarm**
PICOS-WP5-PROTO1-PARTIALID-ALARM-003	**Set one alarm, edit alarm, delete alarm**
PICOS-WP5-PROTO1-PARTIALID-ALARM-004	**set three alarms, getAlarmList, delete alarm**

4.6.4. Presence

Test name	Purpose
PICOS-WP5-PROTO1-PRESENCE-GET-001	get presence for unauthorized user
PICOS-WP5-PROTO1-PRESENCE-GET-002	get presence for unauthorized user with rootId/partialId
PICOS-WP5-PROTO1-PRESENCE-GET-003	get presence with authorized and unauthorized users
PICOS-WP5-PROTO1-PRESENCE-SUB-002	subscribe, update and get notification
PICOS-WP5-PROTO1-PRESENCE-SUB-003	rule allow, subscribe, update presence
PICOS-WP5-PROTO1-PRESENCE-SUB-004	subscribe, update and get notification for authorization and presence
PICOS-WP5-PROTO1-PRESENCE-SUB-005:	subscribe, update and get notification for authorization and presence with partialId

PICOS-WP5-PROTO1-PRESENCE-SUB-006	subscribe and get notification for authorization , refuse
PICOS-WP5-PROTO1-PRESENCE-SUB-007	
PICOS-WP5-PROTO1-PRESENCE-UPD-001	update then get presence to check result
PICOS-WP5-PROTO1-PRESENCE-USB-001	subscribe, update and get notification then unsubscribe
PICOS-WP5-PROTO1-PRESENCE-STATUS-001	**create status**
PICOS-WP5-PROTO1-PRESENCE-STATUS-002	**create status, delete status**
PICOS-WP5-PROTO1-PRESENCE-STATUS-003	**create status, getStatusList, delete status**
PICOS-WP5-PROTO1-PRESENCE-STATUS-004	**create status, getPresence, delete status**
PICOS-WP5-PROTO1-PRESENCE-STATUS-005	**create status, getStatusList, update status, delete status**
PICOS-WP5-PROTO1-PRESENCE-STATUS-006	**create status, getStatusList from #identity, set rule, same getStatusList, delete status**

4.6.5. Location

Test name	Purpose
PICOS-WP5-PROTO1-LOCATION-GET-001	get location for unauthorized user
PICOS-WP5-PROTO1-LOCATION-GET-002	get location for unauthorized user with rootId/partialId
PICOS-WP5-PROTO1-LOCATION-GET-003	mix of authorized and unauthorized access to multiple location information
PICOS-WP5-PROTO1-LOCATION-GET-004	Mix of authorized and unauthorized access to multiple location information
PICOS-WP5-PROTO1-LOCATION-GET-005	add a rule with askOnce, get location, check rule modification
PICOS-WP5-PROTO1-LOCATION-SUB-003	subscribe, update and get notification
PICOS-WP5-PROTO1-LOCATION-SUB-003	subscribe, update and get

	notification
PICOS-WP5-PROTO1-LOCATION-SUB-005	subscribe, update and get notification for authorization and presence with partialId
PICOS-WP5-PROTO1-LOCATION-UPD-001	update and get location for data validation
PICOS-WP5-PROTO1-LOCATION-UPD-002	update and get location for the client
PICOS-WP5-PROTO1-LOCATION-USB-001	subscribe, update and get notification then unsubscribe
PICOS-WP5-PROTO1-LOCATION-POI-001	**Add POI**
PICOS-WP5-PROTO1-LOCATION-POI-002	**Add POI and delete POI**
PICOS-WP5-PROTO1-LOCATION-POI-003	**Add POI get list delete POI get list**
PICOS-WP5-PROTO1-LOCATION-POI-004	**Add POI get POIAttributes delete POI**
PICOS-WP5-PROTO1-LOCATION-POI-005	**Add POI get POIAttributes update POI getPOIAttributes delete POI**
PICOS-WP5-PROTO1-LOCATION-POI-006	**Add POI setRecommendation true , recommend POI resetRecommendation**
PICOS-WP5-PROTO1-LOCATION-POI-007	**Add POI updatelocation and send a searchNearbyPOIs**
PICOS-WP5-PROTO1-LOCATION-POI-008	**Add a commercial POI updatelocation**
PICOS-WP5-PROTO1-LOCATION-POI-009	**Add a commercial POI Enable advertising, updatelocation**
PICOS-WP5-PROTO1-LOCATION-POI-011	**Add a commercial POI with hobbies, Enable advertising, updatelocation**
PICOS-WP5-PROTO1-LOCATION-POI-012	**Add a commercial POI with hobbies, Enable advertising, updatelocation**
PICOS-WP5-PROTO1-LOCATION-SEARCH-001	**Search nearby Users one user**
PICOS-WP5-PROTO1-LOCATION-SEARCH-002	**Search nearby Users: one user in the area**
PICOS-WP5-PROTO1-LOCATION-SEARCH-003	**Search nearby Users: two users in the area**
PICOS-WP5-PROTO1-LOCATION-SEARCH-004	**Search nearby Users: center with address two users in the area**

4.6.6. privacy advisor

Test name	Purpose
PICOS-WP5-PROTO1-PA-EXAMINE-CONTENT-001	update the profile with a familyName and Add content to a category with this family name in the txt content and accept the PA advise
PICOS-WP5-PROTO1-PA-EXAMINE-CONTENT-002	update the profile witrh a familyName and Add content to a category with family name and refuse the advise
PICOS-WP5-PROTO1-PA-EXAMINE-CONTENT-003	update the profile with a familyName and Add content to a category with family name in description and refuse the advise
PICOS-WP5-PROTO1-PA-EXAMINE-CONTENT-002	update the profile with a familyName and Add content to a category with family name in title and refuse the PA advise
PICOS-WP5-PROTO1-PA-EXAMINE-CONTENT-005	open a chat, update the profile, send a message with personal data , accept the pa advise
PICOS-WP5-PROTO1-PA-EXAMINE-CONTENT-005	open a comm, update the profile, send a message with personal data , refuse the pa advise
PICOS-WP5-PROTO1-PA-REPUTATION-001	create sub Community and invite partialId, then lower the reputation

4.6.7. RT content sharing

Test name	Purpose
PICOS-WP5-PROTO1-RT-CONTENT-SHARING-001	Communication creation and closing
PICOS-WP5-PROTO1-RT-CONTENT-SHARING-002	Communication closing
PICOS-WP5-PROTO1-RT-CONTENT-SHARING-003	Open connection and add participant

PICOS-WP5-PROTO1-RT-CONTENT-SHARING-004	remove a participant
PICOS-WP5-PROTO1-RT-CONTENT-SHARING-MSG-001	open connection with invitees and send a message to invitees
PICOS-WP5-PROTO1-RT-CONTENT-SHARING-MSG-002	open connection with invitees and send a message to invitees, get message list

4.6.8. Private room

Test name	Purpose
PICOS-WP5-PROTO1-PRIVATEROOM-ADDCONTENT-001	Add a content to the private room
PICOS-WP5-PROTO1-PRIVATEROOM-ADDCONTENT-002	Add/delete a content to the private room and use GetContentList
PICOS-WP5-PROTO1-PRIVATEROOM-ADDCONTENT-003	Add content , create a sub community copy content from pr to sc
PICOS-WP5-PROTO1-PRIVATEROOM-CHANGECONTENT-001	Add/delete to the private room, get a content change and check content
PICOS-WP5-PROTO1-PRIVATEROOM-GETCONTENT-001	Add/delete and get a content to the private room and check content

4.6.9. Public community

Test name	Purpose
PICOS-WP5-PROTO1-PC-001	Set and get member status
PICOS-WP5-PROTO1-PC-002	Change community member role
PICOS-WP5-PROTO1-PC-003	Get Terms and Conditions
PICOS-WP5-PROTO1-PC-004	Create community
PICOS-WP5-PROTO1-PC-005	Get community attributes
PICOS-WP5-PROTO1-PC-006	Update community attributes (policy

	not updated
PICOS-WP5-PROTO1-PC-007	Update community attributes (policy text updated)
PICOS-WP5-PROTO1-PC-008	Add community member
PICOS-WP5-PROTO1-PC-009	Revoke community member
PICOS-WP5-PROTO1-PC-010	Get user role
PICOS-WP5-PROTO1-PC-011	Check if user is a community member
PICOS-WP5-PROTO1-PC-CATEGORY-001	Category creation
PICOS-WP5-PROTO1-PC-CATEGORY-002	Retrieve category list
PICOS-WP5-PROTO1-PC-CATEGORY-003	Delete category with wrong category ID
PICOS-WP5-PROTO1-PC-CATEGORY-004	Category creation with 2 meta contents
PICOS-WP5-PROTO1-PC-CATEGORY-005	Sub category creation
PICOS-WP5-PROTO1-PC-CATEGORY-006	Retrieve category list from a sub category
PICOS-WP5-PROTO1-PC-CATEGORY-007	Delete sub category
PICOS-WP5-PROTO1-PC-CATEGORY-008	Update category attributes
PICOS-WP5-PROTO1-PC-CATEGORY-009	Get category presentation
PICOS-WP5-PROTO1-PC-CATEGORY-010	create category, unregister, check “isFounderRevoked” bit
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-001	Add content to a category
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-002	Add content (previously added to the private room) to a category
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-003	Get list of contents in a category
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-004	Search content per publisher name
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-005	Search content per location
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-006	Delete content
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-007	Get content
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-008	Add content that is located in a sub-community
PICOS-WP5-PROTO1-PC-CATEGORY-	Get access history of a content

CONTENT-011	
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-012	add content, set rule, Get content, set rule Get content
PICOS-WP5-PROTO1-PC-CATEGORY-CONTENT-013	create category, unregister, check "isFounderRevoked" bit
PICOS-WP5-PROTO1-PC-FORUM-001	Forum creation
PICOS-WP5-PROTO1-PC-FORUM-002	Forum deletion
PICOS-WP5-PROTO1-PC-FORUM-003	Forum deletion with wrong forum ID
PICOS-WP5-PROTO1-PC-FORUM-004	Change forum attributes
PICOS-WP5-PROTO1-PC-FORUM-005	Change forum attributes with a wrong forum ID
PICOS-WP5-PROTO1-PC-FORUM-006	Retrieve forum list
PICOS-WP5-PROTO1-PC-FORUM-007	Change forum admin
PICOS-WP5-PROTO1-PC-FORUM-008	Change forum admin with wrong forum ID
PICOS-WP5-PROTO1-PC-FORUM-009	create forum, unregister, check isFounderRevoked bit
PICOS-WP5-PROTO1-PC-FORUM-THREAD-001	Forum thread creation
PICOS-WP5-PROTO1-PC-FORUM-THREAD-002	Forum thread creation with wrong forum ID
PICOS-WP5-PROTO1-PC-FORUM-THREAD-003	Move a forum thread to another forum
PICOS-WP5-PROTO1-PC-FORUM-THREAD-004	Move a forum thread to another forum with a wrong forum ID
PICOS-WP5-PROTO1-PC-FORUM-THREAD-005	Delete a forum thread
PICOS-WP5-PROTO1-PC-FORUM-THREAD-006	Delete a forum thread with a wrong forumThreadId
PICOS-WP5-PROTO1-PC-FORUM-THREAD-007	Retrieve list of forum thread in a forum
PICOS-WP5-PROTO1-PC-FORUM-THREAD-008	Retrieve list of forum thread in a forum with a wrong forum ID
PICOS-WP5-PROTO1-PC-FORUM-THREAD-009	**Forum thread creation addrule to created forum-thread**
PICOS-WP5-PROTO1-PC-FORUM-THREAD-010	**create forum, create forum thread, unregister, check isFounderRevoked bit**
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-001	Contribution to forum thread
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-002	Contribution to forum thread with wrong IDs

PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-003	Change contribution to forum thread
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-004	Change contribution to forum thread with wrong IDs
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-005	Delete contribution to forum thread
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-006	Delete contribution to forum thread with a wrong contribution ID
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-007	Get contribution list and attachments
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-008	Get contribution list with wrong forumThreadId
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-009	Get list of attachments with wrong contentContributionID
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-010	Search contributions by publisher name
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-011	Search contributions by keyword in txt
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-012	**Delete contribution to forum thread done not by the owner**
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-013	**Delete contribution to forum thread done by the owner not the moderator**
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-014	**Create forum, create forum thread, create contrib unregister, check isFounderRevoked bit**
PICOS-WP5-PROTO1-PC-FORUM-THREAD-CONTRIB-015	**Create forum, create forum thread, create contrib , get threadcontribution list get access history bit**

4.6.10. Reputation

Test name	Purpose
PICOS-WP5-PROTO1-REPUTATION-RATE-001	Rate a content with rootId as entity owner
PICOS-WP5-PROTO1-REPUTATION-RATE-002	Rate a content with partialId as entity owner

PICOS-WP5-PROTO1-REPUTATION-RATE-003	Rate a content with the owner partialId
PICOS-WP5-PROTO1-REPUTATION-multiplerateforsameuser-001	Rate multiple time a content with the same rootId as requester
PICOS-WP5-PROTO1-REPUTATION-multiplerateforsameuser-002	Rate a content twice with partialId as entity

4.6.11. Sub-Community

Test name	Purpose
PICOS-WP5-PROTO1-SUBCOMMUNITY-CR-001	create sub Community and invite partialId
PICOS-WP5-PROTO1-SUBCOMMUNITY-CR-002	create public sub Community and get public sub-comm list
PICOS-WP5-PROTO1-SUBCOMMUNITY-CR-003	**Create private sub Community and get public sub-comm**
PICOS-WP5-PROTO1-SUBCOMMUNITY-CR-004	**Create private sub Community unregister and get user list sub-comm**
PICOS-WP5-PROTO1-SUBCOMMUNITY-001	Change Sub Community Administrator
PICOS-WP5-PROTO1-SUBCOMMUNITY-002	Change Sub Community Administrator not with the owner and close not with the owner
PICOS-WP5-PROTO1-SUBCOMMUNITY-003	Change Sub Community Administrator not with the owner.
PICOS-WP5-PROTO1-SUBCOMMUNITY-CR-003	create private sub Community and get public sub-comm
PICOS-WP5-PROTO1-SUBCOMMUNITY-INV-001	create sub Community and invite partialId
PICOS-WP5-PROTO1-SUBCOMMUNITY-JOIN-001	create sub Community and join the community
PICOS-WP5-PROTO1-SUBCOMMUNITY-REM-001	create sub Community and invite partialId
PICOS-WP5-PROTO1-SUBCOMMUNITY-	Add content to Sub Community, then

CONTENT-001	get it and check it is the same
PICOS-WP5-PROTO1-SUBCOMMUNITY-CONTENT-002	Add 2 contents to Sub Community, check list of contents, delete 1 content, check list
PICOS-WP5-PROTO1-SUBCOMMUNITY-CONTENT-003	Add content to Sub Community, then change its attributes
PICOS-WP5-PROTO1-SUBCOMMUNITY-CONTENT-004	Create forum thread in Sub Community and delete it not by the owner of the thread but by an admin guy
PICOS-WP5-PROTO1-SUBCOMMUNITY-CONTENT-005	Create forum thread in Sub Community , get list of threads and delete it
PICOS-WP5-PROTO1-SUBCOMMUNITY-CONTENT-006	**Copy content from a Sub Community to another**
PICOS-WP5-PROTO1-SUBCOMMUNITY-CONTENT-007	**Add 2 contents to Sub Community, check list of contents, delete 1 content, check list**
PICOS-WP5-PROTO1-SUBCOMMUNITY-CONTENT-008.PHP	**Add 2 contents to Sub Community, add a rule to the first content check list of contents,**
PICOS-WP5-PROTO1-SUBCOMMUNITY-CONTENT-009.PHP	**Add 2 contents to Sub Community, get content, get content access history**
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-001	Create forum thread in Sub Community
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-002	Create forum thread in Sub Community and delete it
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-003	Create forum thread in Sub Community and delete it not by the owner of the thread
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-004	Create forum thread in Sub Community and delete it not by the owner of the thread but by an admin guy
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-005	Create forum thread in Sub Community , get list of threads and delete it
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-006	Create forum thread in Sub Community , get list of threads unregister and check revoked bit
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-007	**Create forum thread in Sub Community , add a rule on forum thread, check enforcement**
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-CONTRIBUTION-001	**Create forum thread in Sub Community and contribute to the discussion**
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-	Create forum thread in Sub Community

THREAD-CONTRIBUTION-002	and contribute to the discussion with an invalid partialId
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-CONTRIBUTION-003	Create forum thread in Sub Community and contribute to the discussion with an invalid partialId
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-CONTRIBUTION-004	Create forum thread in Sub Community and contribute to the discussion then change the contribution with an invalid partialId
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-CONTRIBUTION-005	Create forum thread in Sub Community and contribute to the discussion gt the list of contribution
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-CONTRIBUTION-006	Create forum thread in Sub Community and contribute to the discussion get the list of contribution then delete
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-CONTRIBUTION-007	Create forum thread in Sub Community and contribute to the discussion , change the contribution and get the contribution list
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-CONTRIBUTION-008	Create forum thread in Sub Community and contribute to the discussion , change the contribution and get the contribution list at last get content
PICOS-WP5-PROTO1-SUBCOMMUNITY-FORUM-THREAD-CONTRIBUTION-009	**Create forum thread in Sub Community ,contribute to the discussion gt the list of contribution, get access history**

4.6.12. Contact

Test name	Purpose
PICOS-WP5-PROTO1-CONTACT-001	Contact creation
PICOS-WP5-PROTO1-CONTACT-002	Contact creation, contact not found
PICOS-WP5-PROTO1-CONTACT-003	Contact deletion
PICOS-WP5-PROTO1-CONTACT-004	Get contact list
PICOS-WP5-PROTO1-CONTACT-005	Contact creation twice
PICOS-WP5-PROTO1-CONTACT-006	Update Contact profile

4.6.13. Site

Test name	Purpose
PICOS-WP5-PROTO1-SITE-001	Add site
PICOS-WP5-PROTO1-SITE-002	Add site, delete site
PICOS-WP5-PROTO1-SITE-003	Add site, delete with invalid id and delete site
PICOS-WP5-PROTO1-SITE-004	Add site, get site, delete site
PICOS-WP5-PROTO1-SITE-005	Add site, get site, edit, get site, delete site
PICOS-WP5-PROTO1-SITE-006	Add site, get site, edit, get site, delete site
PICOS-WP5-PROTO1-SITE-007	Add site, set location , set rule with site, get location (within site) , delete site
PICOS-WP5-PROTO1-SITE-008	Add site, set location , set rule with site, get location (outside site) , delete site
PICOS-WP5-PROTO1-SITE-009	Add multiple sites, get sites delete site

4.6.14. Subscription

Test name	Purpose
PICOS-WP5-PROTO1-SUBSCRIPTION-001	subscribe to an element
PICOS-WP5-PROTO1-SUBSCRIPTION-002	subscribe to an element, get element list
PICOS-WP5-PROTO1-SUBSCRIPTION-003	3 subscribe to an element, get element list, unsubscribe, get subscribeList
PICOS-WP5-PROTO1-SUBSCRIPTION-004	subscribe to an element, send NotifyAccessElement

4.6.1. Notifications

Test name	Purpose
PICOS-WP5-PROTO1-NOTIFICATION-PENDING-001	subscribe to an alement Store notification, isPendingNotification
PICOS-WP5-PROTO1-NOTIFICATION-PENDING-002	Store notification, logout and login
PICOS-WP5-PROTO1-NOTIFICATION-PENDING-003	Store notification, logout and login and get pendingNotifications

5. Installation and configuration

In order to ease the dissemination of PICOS result, the Phase2 platform has been shaped to be easily installable and configurable in a LINUX RedHat el5 machine.

5.1. Hardware dependencies

There are no real dependencies on hardware, especially for demo or trial support. Supporting a production configuration will require some extra work to size the amount of memory, of LAN interfaces for high availability and disk arrays for DB and file repositories

5.2. Software dependencies

The PICOS installation assumes the following configuration

- Linux Redhat el 5.4 (apache included) and above
- PHP 5.2.10 and above
    - The default PHP that comes with the redHat el 5.4 should be upgraded to support this version
    - PHP modules like PHP socket, PHP soap, xlm-rpc and graphics libraries modules must be enabled.
- MySQL 5.1.37

PICOS is also using the ELGG open source. This software installation has been integrated into the PICOS installation.

Note that for any other Linux distribution or version of open source software, the following installation and operation guide might have to be modified. Such a study is out of scope for this document.

PICOS uses MySQL database for its prototype. Any commercial usage of the PICOS platform requires a license agreement with the providers of MySQL.

5.3. Installing and configuring the PICOS system

5.3.1. Linux login for installation and configuration

You need to install and configure the PICOS platform as root user .

5.3.2. Root name

A **root name** must be select that will be used for creating directories, database….. Multiple platforms can be installed on the same server, having different MySQL database. The root name used in the example is **"PICOS"**

5.3.3. Create the PICOS root directories

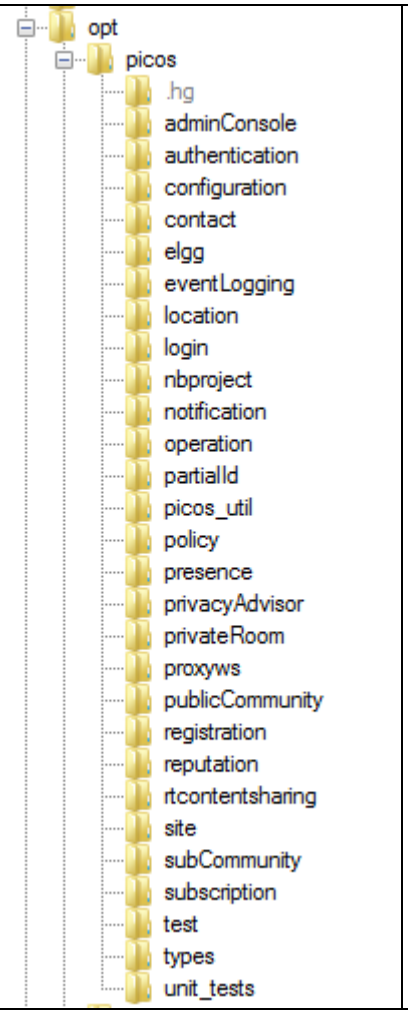
Note that the repository names used for installing and running PICOS can be changed. However changing the names requires some adaptations in the HTTPD configuration and in the PICOS_config.inc file. Unless there are good reasons to change , the names should be the ones that are proposed in this installation description.

Step 1. Log in as `padmin` user.

Step 2. Create a directory for PICOS installation. Use the root name. "/opt/"root name" i.e. /opt/PICOS

```
# mkdir /opt/PICOS
# chmod  775 /opt/PICOS
# mkdir /var/lib/PICOS
# chmod  775 /var/lib/PICOS
# chown apache:apache /var/lib/PICOS
# mkdir /var/log/PICOS
# chmod  775 /var/log/PICOS
# chown apache:apache /var/log/PICOS
```

5.3.4. Untar the PICOS file



Step 1. Copy the PICOS tar file in the /opt/PICOS directory.

Step 2. Untar the PICOS file in the /opt/PICOS directory

```
# cd /opt/PICOS
# gunzip PICOS_PLATFORM_V_34.tar.gz
# tar -xf PICOS_PLATFORM_V_34.tar
#
```

All the PICOS PHP scripts are now installed with the proper security setting

Table 6: picos directory creation after the untar.

5.3.5. Change the PHP.ini file

Step 1. Edit /etc/PHP.ini.

Step 2. Change the amount of Max Memory per PHP script.
.

```
memory_limit = 256M
```

Step 3. Change the Error logging.
.

```
error_reporting = E_ALL & ~E_NOTICE
```

5.3.6. Change the httpd configuration

Step 1. Create a PICOS.conf in /etc/httpd/conf.d with the following setting.
Configure the directories used. Change the root name if required

```
Alias /webservices/PICOS/repository/adminConsole/contents/
/var/lib/PICOS/adminConsole/contents/
Alias /webservices/PICOS/ /opt/PICOS/

<Directory /opt/PICOS/>
    Options Indexes FollowSymLinks
    DirectoryIndex index.php
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>

<Directory /var/lib/PICOS/adminConsole/contents/>
    Options Indexes FollowSymLinks
    DirectoryIndex index.php
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
```

item	Description
Alias /webservices/PICOS/repository/adminConsole/contents/ /var/lib/PICOS/adminConsole/contents/	The directory "*/var/lib/PICOS/adminConsole/contents/"* contains pages that are temporary created for the web console and must be accessed via http get. It is assumed here the selected directory for storing the data is "/var/lib/PICOS"

	It is also assumed that the "PICOS_url" is "http://localhost/webservices/PICOS"
Alias /webservices/PICOS/ /opt/PICOS/	An Alias so that PICOS PHP script can be referenced in url and launched by apache. It is assumed that the directory used to store all the PHP script is "/opt/PICOS".

5.3.7. Configure SQL tables in MySQL

The installation assumes that the MySQL has been installed on the same server and a root user has been created (root/"password") and will be used by ELGG to access the database

Step 1. Run the sql script to create tables. The string "password" must be replaced with the password of the root user. Select a database name. Use the root name

.

```
# cd /opt/PICOS/ELGG
# MySQL -u root –p"password"
MySQL> create database PICOS;
MySQL> use PICOS;
MySQL> source ELGG.sql
MySQL> source PICOS.sql
```

Step 2. Change some "datapathes" in the SQL database. The path must reference the directory that contains the ELGG software.
The example assumes that /opt/PICOS has been used to untar the PICOS software

Step 3. For the "dataroot" the example assumes that /var/lib/PICOS is used to store all PICOS data.

.

```
MySQL> update ELGG_datalists set value='/opt/PICOS/ELGG' where name
   = 'path';
MySQL> update ELGG_datalists set value='/var/lib/PICOS/ELGG/' where
   name = 'dataroot';
MySQL> exit
#
```

5.3.8. Configure ELGG

Step 1. Change the ELGG setting.PHP script. Update the user/password information for SQL access and change if necessary

the ELGG database name, here “PICOS”.

.

```
# cd /opt/PICOS/ELGG/engine
# vi settings.PHP


// Database username
$CONFIG->dbuser = 'root';
// Database password
$CONFIG->dbpass = 'rootpassword';
// Database name
$CONFIG->dbname = 'PICOS';



# cd ..
# PHP install.PHP
```

5.3.9. Configuring PICOS

Step 1. The configuration file uses the name "localhost" to reference the current server. Make sure that "localhost" is configured in /etc/hosts.

Step 2. The minimum is to update the external IP address of the server. A server may have an internal private address and an external public address. Please configure the external IP address with the real IP address

Step 3. Configure the root name in the $instance variable.

```
# cd /opt/PICOS/configuration/
# vi PICOS-config.inc
```

```
<?php
$PICOSdir = preg_replace('{/\w+$}','/',dirname(__FILE__)) ;
require_once('methodDefinition.inc');

$storeTraces=true;
$instance='PICOS';
$datapath='repository/' ;

$host_ip="localhost";
$external_ip="X.Y.Z.W";

//
// Socket configuration
//
define('CST_COMPONENT_SOCKET_PUBLICCOMMUNITY', '10001');
define('CST_COMPONENT_SOCKET_REPUTATION', '10002');
define('CST_COMPONENT_SOCKET_EVENTLOGGING', '10003');
define('CST_COMPONENT_SOCKET_SHARING', '10004');
define('CST_COMPONENT_SOCKET_NOTIFICATION', '10000');
```

Step 4. Some socket connections with TCP port are used internally. Default values are provided. If the default values are in conflicts with other applications on the server, they must be changed.

Step 5. The 'CST_COMPONENT_SOCKET_NOTIFICATION' constant defines the socket port that the mobile must open to receive the notifications

5.3.10. Restart the httpd server.

```
# /sbin/service httpd restart
```

5.3.11. Start the PICOS server.

Note that the PICOS server must be started to create the community. Make sure that each of the 5 server have an allocated Process ID

```
# cd /opt/PICOS/operation
# ./restartServer.sh
 Reputation Server (pid: 26883)...
 Public community Server (pid: 26880)...
 Real Time Content Sharing Server (pid: 26882)...
 Notification Server (pid: 26864)...
 Event Logging Server (pid: 26884)...
#
```

5.3.12. Create the community

The community must be initially created using a script:

- It sets the default rules for the community.
- It creates a PICOS user with admin capabilities that will be used to access the admin console.
- It configures the WSDL files paths with the proper root name and store all paths into a file /var/lib/PICOS/configuration/wsdls.ctx

```
# cd /opt/PICOS/configuration
# PHP createCommunity.PHP
setPolicies for user  attributes
setPolicies for partial Id  attributes done
setPolicies for public community forums
setPolicies for public community attributes
set default Policies for sub-community
setPolicies for communication
The community is now created with padmin as the initial administrator.
#
```

```
The community and its basic configuration is now created. All its parameters
(users, categories, forum, POIs...° can be modified using the admin Console.
```

5.3.1. Restarting the Servers using the admin Console.

The stop/restart server operations is performed using the admin Console. In order to allow the admin Console ("apache" user) to stop and start the PHP servers, the servers must be stopped using

command line scripts (as the server linux processes are not owned by “apache” user) and restarted via the admin console (operation achieved under the user “apache”.

Step 1. Stop the servers

```
# cd /opt/PICOS/operation/
# ./stopServer.sh
```

Step 2. Start a browser.and use the following url: http://hostname/webservices/PICOS/adminConsole/index.PHP

Step 3. Login the adminConsole using the default admin user (padmin/padmin)

Step 4. Select in the left menu “Operating the platform”. The name of the server are displayed with no associated PID.

Step 5. Click on the “restart the platform” button. The Notification server may take some time to start. If no PID appears, click on “Check the server components” menu item to refresh the screen

Figure 33: Admin console screen for operating the PICOS platform

5.3.1. Define community attributes

The PICOS project receives research funding from the Community’s Seventh Framework Programme.

Before login in using the mobile, the attributes (title, description, announcement and terms and conditions of the community must be defined. Please use the admin console

Manage the Public community → update Community attributes

5.3.2. Checking the PICOS platform

A test suite has been developed to validate the platform from a functional standpoint.

The test suite chains more than 200 tests.

Step 1. Start one test to make sure that the fundamentals of the platform are ok

```
# cd /opt/PICOS/test/registration
# PHP  PICOS-wp5-proto1-registration-001.PHP proxy
PICOS-WP5-PROTO1-REGISTRATION-001: Registration with mandatory
   parameters
PICOS-WP5-PROTO1-REGISTRATION-001: Test OK
#
```

Step 1. Run the complete non-regression test suite.

```
# cd /opt/PICOS/test/
#
# ./launch-tests-proxy.sh

PICOS-WP5-PROTO1-REGISTRATION-001: Registration with mandatory
   parameters
PICOS-WP5-PROTO1-REGISTRATION-001: Test OK
PICOS-WP5-PROTO1-REGISTRATION-003: Registration with login already
   used
PICOS-WP5-PROTO1-REGISTRATION-003: Test OK
PICOS-WP5-PROTO1-REGISTRATION-004: Registration with pseudo already
   used
PICOS-WP5-PROTO1-REGISTRATION-004: Test OK
PICOS-WP5-PROTO1-UNREGISTRATION-001: Un-registration
....
#
```

5.4. Operating the PICOS platform

The operation of the platform is performed via the admin console web access.

5.4.1. Start / stop frontend/backend server components

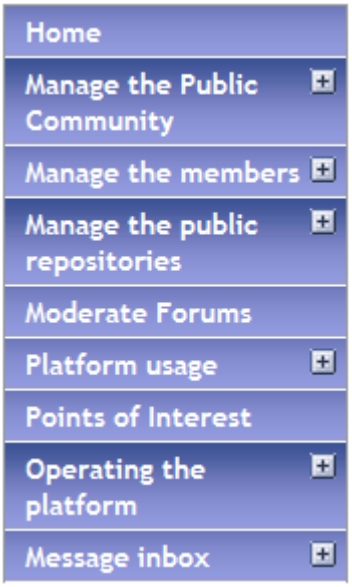
	The major operations that can be performed via the admin Console are: • Managing Community Objects (categories, forum, POIs) • Manage Users including revocation of identities or users. • Status and restart of the frontend/backend servers • Statistics of community usages • Moderate forums • Interact with users.

Table 7: Main Admin Console Menu for web navigation.

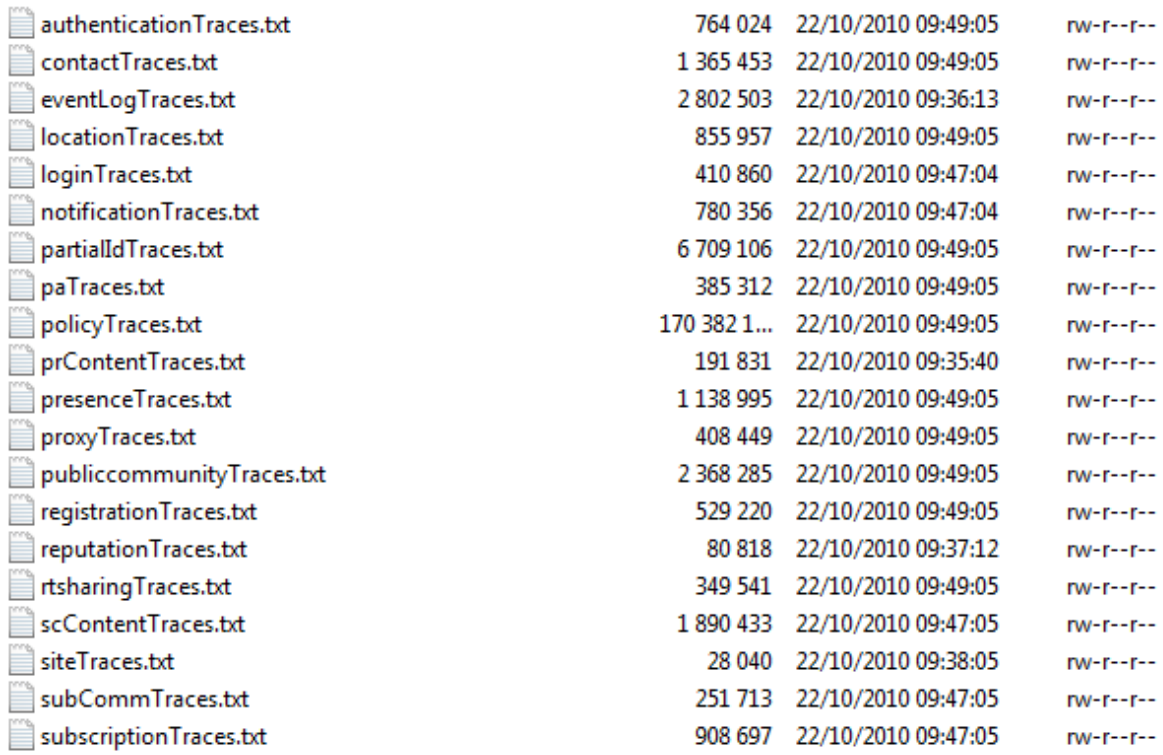
The frontend/backend components are started and remain active LINUX processes. Once started, there is an auto-restart mechanism to restart each component separately in case of component failure.

Note that components can also be started using command line with root user but then started processes are owned by the "root" user. Hence, the "apache" user (owner of console admin request) does not have the ability to stop processes started by root.

So "command line" start/stop of servers should not be used for final start/operation of the platform as it blocks the start/stop admin console capability.

5.4.2. Logging / tracing

All traces are stored in the "/var/log/PICOS/ directory (the path can changed in the PICOS-config.inc file)



authenticationTraces.txt	764 024	22/10/2010 09:49:05	rw-r--r--
contactTraces.txt	1 365 453	22/10/2010 09:49:05	rw-r--r--
eventLogTraces.txt	2 802 503	22/10/2010 09:36:13	rw-r--r--
locationTraces.txt	855 957	22/10/2010 09:49:05	rw-r--r--
loginTraces.txt	410 860	22/10/2010 09:47:04	rw-r--r--
notificationTraces.txt	780 356	22/10/2010 09:47:04	rw-r--r--
partialIdTraces.txt	6 709 106	22/10/2010 09:49:05	rw-r--r--
paTraces.txt	385 312	22/10/2010 09:49:05	rw-r--r--
policyTraces.txt	170 382 1...	22/10/2010 09:49:05	rw-r--r--
prContentTraces.txt	191 831	22/10/2010 09:35:40	rw-r--r--
presenceTraces.txt	1 138 995	22/10/2010 09:49:05	rw-r--r--
proxyTraces.txt	408 449	22/10/2010 09:49:05	rw-r--r--
publiccommunityTraces.txt	2 368 285	22/10/2010 09:49:05	rw-r--r--
registrationTraces.txt	529 220	22/10/2010 09:49:05	rw-r--r--
reputationTraces.txt	80 818	22/10/2010 09:37:12	rw-r--r--
rtsharingTraces.txt	349 541	22/10/2010 09:49:05	rw-r--r--
scContentTraces.txt	1 890 433	22/10/2010 09:47:05	rw-r--r--
siteTraces.txt	28 040	22/10/2010 09:38:05	rw-r--r--
subCommTraces.txt	251 713	22/10/2010 09:47:05	rw-r--r--
subscriptionTraces.txt	908 697	22/10/2010 09:47:05	rw-r--r--

Figure 34: Various trace files (one per component)

File Names can be modified in the PICOS-config.inc file (a single file can be used). The files contain a trace of each request with parameters. The tracing can be disabled by setting the variable `$storeTraces=false in the PICOS-config.inc file (/opt/PICOS/configuration/PICOS-config.inc)`

Note that any trace file can be deleted at any time without impacting the component that uses it for tracing.

5.4.1. Changing the Community default rules

The <root-name> /configuration/createCommunity.PHP is the script that populates the Policy Engine. It sends web service request (addPolicy) for each Object/resource of the community. Default Policies are created and attached to Objects not instances of Objects

Each Policy is independent from the others. Changing the default policies (including rule definition must be done carefully knowing that the last rule defined for a policy has the highest priority and will be used assuming that conditions are valid.

The community default rules can be changed once the community has been launched but these new policies will apply to newly created object and there is no mechanism to apply these rule on already created objects.

5.4.1. Distributing the components on multiple servers

This facility has been used for debugging purpose within the team and is unlikely to be used for demo purpose as it degrades the overall response time.

The inter component communication uses a common repository (file) to locate where the components are installed. This file is generated during the installation with default paths (all components are located on the same server). There is a web interface to configure the paths to enable such distribution. The provided interface has been used for integration of components that were running on different servers. In more secure environment, this web access should be secured.

To change the component distribution, please change the WSDL path of each component using the following URL:

http://hostname/webservices/PICOS/configuration/configurewsdl.php

You need to restart the PICOS platform once the new WSDL URLS have been configured.

5.5. Using PICOS platform in commercial deployments

The provided platform, is not intended to be used, as it is, for commercial deployments.

The platform is provided for demo and research activities purpose only.

Should this work be evolved to support a community in real deployment it should require licensing and usage agreements for 3rd party software like ELGG open source, or the RPC client lib / HP RPC gateway.
The operation model as well as the overall platform feature- set should be re-evaluated to support real deployment. Such study is out of the WP5 scope.

5.6. A common platform for the two Communities

Special effort has been done so that the WP5 PICOS phase 2 platform can support both Anglers V2 and Gamers client applications as all functional evolutions have been defined and implemented in order to be backward compatible.

6. Conclusion

The development of the phase 2 has demonstrated the community agnosticism of the platform re-using all previously defined capabilities to serve the Gamers community.
Efforts have also been focused on improving major problems found during the anglers trial (like the handling of offline notifications)

The generic and centralized PICOS policy implementation simplifies the deployment of privacy and access control improvements that are made available to all Object of the community.

The hierarchical description of all community and users Objects and attributes allows a hierarchical evaluation of Object rules but also of Parent policies when necessary.

The selection of an agile methodology as well as web technologies like web services and PHP language were key decisions to successfully deliver the platform on time within a short schedule and build this fairly complete mobile service. PHP uses dynamicity a lot like for web service decoding easing the integration of evolving interfaces; Some of the policy engine data structure management would have been very difficult to implement using more traditional languages. The richness of the PHP environment makes complex operation easy to implement without compromise on the capabilities and the control of system interfaces.
The result is a very rich platform which supports more than 200 methods that can be called by the client , by the admin Console or by any of the platform components

The test suite has been an incomparable tool for validation of iterative versions of the platform. Written in PHP, it was allowing the definition of complex scenarios involving numerous interactions with platform components, emulating client request/response as well as notification decoding and answers. This test suite has helped delivering the platform early in the process so that the application team could start its server integration. 200 Web based test scripts (one per method) were also made available to WP6 team so that they could emulate client behaviours not yet implemented and test a particular function. Final Platform evolution (improvements regarding installation and operability) have been done late in the process without any major software regression.

The global client server architecture has used innovative new models of communications that allow client application developers to focus on user experience rather than protocol implementation. On the server side, web service interfaces are mature and easy to use technologies to speed up prototyping of research concepts.
Mobile networks still have constraints that can degrade the user experience if strict rules are not followed in the development of the client server interface. It is crucial to know how the client structures its user interface to adapt the platform interface accordingly (one user action should lead to a maximum of one server interaction) and limit its service orchestration responsibility.
In the PICOS platform prototype, more than 200 methods were made available through a simple and duplex RPC model between the client and the server.

The PICOS platform has been the result of a dense phased project led by European ambitious team members with different cultures and backgrounds. By evidence, the PICOS collaborative work has

been very efficient in shaping its value proposition and in driving its implementation. The shared richness of such experience is the first and important step towards dissemination of the PICOS fundamentals. Each member of the team will become an “evangelist” of the PICOS result.

The PICOS project receives research funding from the Community’s Seventh Framework Programme.