SUNSHINE

Annex to D4.6 – Remote System Management service #2

Security Layer

WP 4 – Integration of SUNSHINE pilot smart urban services

Task 4.9 – Integration of new smart services with existing service infrastructures

Task 4.2 – Meter data management service

Task 4.3 – Remote system management service

Revision: v.1.0

Authors:
Stefano Pezzi (SGIS)
Luca Giovannini (SGIS)

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

Dissemination level	PU (public)
Contributor(s)	Stefano Pezzi (SGIS)
Reviewer(s)	Federico Prandi (GRAPHITECH)
Editor(s)	Raffaele De Amicis (GRAPHITECH)
Partner in charge(s)	SGIS
Due date	31/03/2015
Submission Date	31/03/2015

REVISION HISTORY AND STATEMENT OF ORIGINALITY

Revision	Date	Author	Organisation	Description
0.1	26/02/2015	Stefano Pezzi	SGIS	Document creation
1.0	20/03/2015	Stefano Pezzi	SGIS	Document finalization

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.
Moreover, this deliverable reflects only the author's views. The European Community is not liable for any use that might be made of the information contained herein.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

Table of content

Table of content 3
Acronyms 3
1 Introduction 5
2 Authentication 6
3 Authorization 6
3.1 RBAC 6
3.2 ABAC 8
3.2.1 XACML 10
4 Architecture 13
4.1 Services categories 14
4.1.1 SOAP API 14
4.1.2 POX API 14
4.1.3 RPC URI-tunneling 15
4.1.4 RESTful 15
4.2 Type of Sunshine services 15
4.3 XACML model for Sunshine's services 16
4.3.1 Subject 16
4.3.2 Resource 17
4.3.3 Action 17
4.4 Implementation 19
4.4.1 Reverse proxy and firewall 20
4.4.2 The service proxy 20
4.4.3 XACML repository and engine 22
4.4.4 Mutual Authentication 22
4.5 Software components 23
4.5.1 OpenDJ LDAP 23
4.5.2 WSO2 Identity Server 23
4.5.3 WSO2 Enterprise Service Bus 24

Acronyms

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

ABAC	Attribute Based Access Control
CSW	Catalogue Service for the Web
ESB	Enterprise Service Bus
HATEOAS	Hypermedia As The engine Of Application State
HR	Human Resource
IBAC	Identity Based Access Control
LDAP	Lightweight Directory Access Protocol
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
POX	Plain Old XML
RBAC	Role Based Access Control
RPC	Remote Procedure Call
REST	REpresentational State Transfer
SSL	Secure Socket Layer
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SoD	Separation of Duties
SOS	Sensor Observation Service
SPS	Sensor Planning Service
URI	Uniform Resource Identifier
WFS	Web Feature Service
WMS	Web Map Service
WSDL	Web Services Description Language
XACML	eXtensible Access Control Manipulation Language

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

1 Introduction

Security is a regularly desired characteristic of the interaction that happens between an information system and a user. Talking about security, there are three concepts that must be well understood:

- Identification
- Authentication
- Authorization

Identification deals with stating to the system who you are by supplying your identity data, which is a user name or a code. It's like saying your name at the telephone when you call someone.
Authentication is instead the way to prove that you are who you are asserting you are. The most common way is to provide a password together with your username, that is something that is shared between you and the system and no one else should know. Another way to authenticate yourself is not by telling it to the system, but by presenting to the system something that you *own*: a smart card, an ID card or an RSA token, for instance. Finally, you could present to the system something that you *are* rather then you have: i.e. your fingerprint or other bio-based features like the retina scan.
Authorization is what you can do with the system once it knows who you are and trusts your identity.
The paradigm of identification-authentication-authorization applied to the IT is quite natural to understand because it is clearly borrowed from the every-day life, even if sometimes some of these steps are hidden or slightly altered: for example, when you receive a phone call from a friend, you authenticate him by his voice; but when you buy some cigarette, probably the shopkeeper doesn't ask you your ID card because he can recognize directly from your aspect some characteristics of your identity on which the tobacco vending authorization is based: an automatic vending machine would have requested your card, instead.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

2 Authentication

Authentication is the validation of someone's stated identity. To validate we need some sort of proof, something only the use know, have or are. This is sometimes called a factor and it can be a password, a pin code, a card (that you insert, swipe or present), a driver's license, your biometric profile (e.g. fingerprint) or even a combination of all three.

The proof is basically more or less a sort of "guarantee" for the system that you are in fact the person you state you are. The word guarantee is between quotes since many of the things that we consider as proof, like a password can sometimes easily be guessed, stolen or handed over. The quality of the proof depends on the inalienability, in this case meaning how easy it is for others to use (e.g. impossible to take away or give up).

3 Authorization

The most widespread authorization model in IT world is the RBAC, Role Based Access Control, but it is also a static and old-fashioned model that is not fitting anymore the growing needs of new software architectures based on services, cloud and new type of clients.

3.1 RBAC

Role-based access control is an approach used to restrict access to authenticated users based on their role rather than on their identity. It is the first attempt to overpass the access control lists, a rudimentary mechanism for authorization were each user was assigned to one or more lists with the enabled user for the different functionalities of a system (IBAC security model). *Roles* decoupled the concept of identity from that of authorization: a user was enabled to functionality because of his/her role and not because of being someone.

In RBAC user assignment and permission assignment can be done separately and independently, and this is very important because this separation reflects the reality of

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

organizations. More over RBAC implements the principle of *separation of duties*[1], with the possibility of enforcing rules that prevent the association of a user to roles that create conflict of interest[2].

It is used by the majority of enterprises with more than 500 users. As said before, with the coming of SOA architecture, RBAC has shown the limitations of this schema, in particular:

- RBAC does not know the concept of **context**. If someone has a role, then he will get the permissions associated with that role. There is no restriction in place based on the concept of context. A context could be the channel used (internet, LAN), time of day, legal entity, the user agent (browser, mobile phone) etc. An account using a wireless home PC might not deserve the same permissions as the same account in an office using a centrally managed workplace. But such a concept does not exist in RBAC terms (unless you define lots of extra roles: developers working at home, developers working in the office...but what if developer works usually at the company's premises and seldom from home?).
- A big issue (but that is not specifically related to SOA) is that there is no possibility for a **fine user profiling**. An example is authorization based on the skills of an employee instead of the single fact that the user is connected to a role. RBAC doesn't know the difference between junior or senior employees within the same role, unless you define separate roles. And creating almost duplicated roles is not the right way to implement RBAC.
- In RBAC to define authorizations you still have to manage every user account and bind these accounts to roles. Unknown accounts can only be linked to a default role, like *guest* or *customer*. But users coming in through the internet channel are not only guests or customers, but they can also be employees, partners, external service providers, you name it. There may be plenty accounts you don't want to manage, because they don't belong to your security domain. Unless you manage different portals for different roles and have some form of identity management within each portal environment, RBAC is of no use. Federation could be a solution, but that only limits interaction possibilities to trusted partners you federate with. Besides, most

[1] Separation or segregation of duties (SoD) is a general security concept (that can be applied to Information Technology as well) that states the need of having more than one person to accomplish a single task in order to prevent possible abuses, errors and fraudulent behaviors.

[2] A user, for instance, can't be associated to a "purchasing" role and to an "approving" role at the same time and RBAC helps in defining roles that are mutually exclusive.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

federation solutions use shadow accounts within the security domains, thereby creating a new identity management problem because of all duplications.

3.2 ABAC

ABAC, Attributes Based Access Control, is a "next generation" authorization model that provides a dynamic, context-aware and fine-grained solution. Today's systems are complex and various and roles are no longer sufficient for dealing with such a complexity unless producing a plethora of roles that makes the system impossible to be managed. Whilst RBAC is based on a single-dimensional[3] characterization of the authorized entities into roles, ABAC introduces a multidimensional categorization where dimensions can be added with no limits.

Attributes are characteristics of the three main entities involved in the security mechanism:

- subject attributes
- resource attributes
- action attributes

A fourth group of attributes comes from the "environment" in which the access request was done.

The **subject** is the user requesting the access to an information asset. Among the attributes that describe a subject are the user name or id (this brings back IBAC model to ABAC) and role of the user (and this brings back RBAC model too). These attributes usually are stored in the LDAP directory or in HR management system.

The **resource** is the information asset on which the user wants to do something. The type of resource can be very different depending on which information system we are talking about.

A sensor can be a resource in the case of Sunshine's platform and an example of a resource's attribute can be the pilot that owns the sensor. If the subject's attributes can be quite standard and furthermore they are retrieved from standard containers, resources' attributes strongly depend on the application context.

The **action** is what the user wants to perform. Common actions' attributes are the read/write type of action because this characteristic is shared by almost every information system, but in Sunshine's context we can make a more detailed modelization, in particular for the OGC services that are characterized by well-known operations.

[3] Actually RBAC has evolved introducing few more dimensions.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

The **environment** is the context in which an access request is made and common attributes are the time in which the request happens or the location of the requestor in that moment, the type of device that is being used, the level of protection of the channel or protocol used, etc.
These are the input parameters used by the ABAC model, but the proper authorization is expressed by means of a **rule** or, better, by a **policy** that is a set of rules. That's why this model sometimes is called ruled-based or policy-based access control.
The rules are stored in a repository and are processed by a rule engine that takes as input the so-called request context.

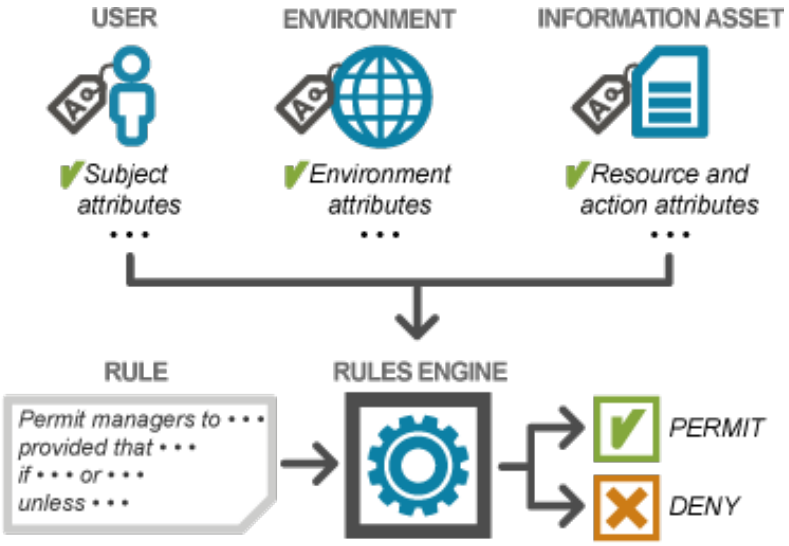

Figure 1 - Basic concepts of ABAC[4]

Most of the advantages of ABAC reside among the disadvantages of RBAC, but not only and here we have listed some of them even from the theoretical point of view:

- **fine grained authorization**
- **minimum information usage**; authorization decision can be made only on strictly necessary attributes, avoiding the unveiling of other details that potentially could be used for incorrect actions; the needed attributes can be calculated ad hoc (i.e. to apply an authorization rule based on the age of the user, it could be enough to know if the user is over or under 18 y, and not to know exactly his/her birthday);

[4] Source: www.axiomatics.com

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

3.2.1 XACML

XACML is an OASIS' standard that was first ratified in 2003 (v.1.0) and last reviewed in January 2013 (v.3.0). It specifies an XML coding for implementing an ABAC authorization model. This means that XACML responds to a long list of requirements where the most important are shown below:

1. provide a method to encode rules and policies and their combinations;
2. provide a method to encode a decision request/response.
3. provide a data flow model for the authorization process that represents its reference architecture.

A high level sketch of the XACML ecosystem is drawn in Figure 2 which shows on the right side the role of the main components[5] that appears as boxes on the left one. The user icons represent the entity asking for the access to the resource (the document icon) and the administrator of the authorization policies.

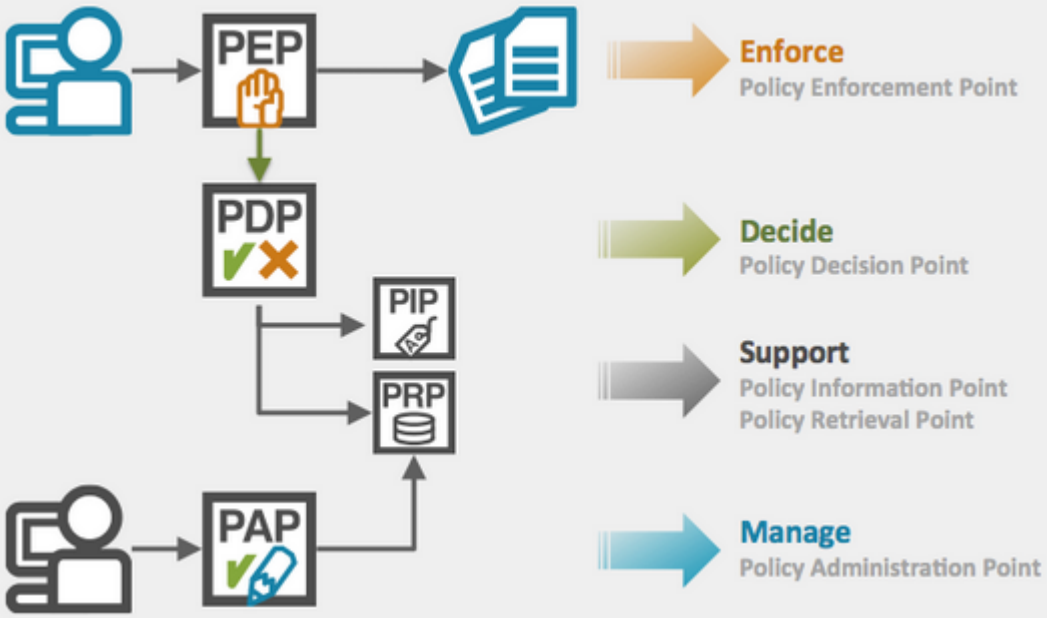

Figure 2 - The XACML ecosystem[6]

The full data flow diagram is shown in Figure 3. To understand it, it's worth introducing some XACML jargon[7]:

[5] The PRP was mentioned in the first versions of the standards and even though in the latest is not present anymore, merged into the PDP.

[6] Source: www.axiomatics.com

[7] XACML terms are not always the same used in "ISO/IEC 10181-3 – Open System Interconnection- Security framework for open systems: Access control framework – Part 3", but they have a clear correspondence.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

PAP (Policy Administration Point): The system component that creates a policy or a policy set. Usually, it is implemented with a repository and a GUI that offers functionalities to create a policy in an assisted way, test it, insert it into the repository, activate or deactivate it and generally manage a store that can become very dense of rules.

PDP (Policy Decision Point) The system component that evaluates applicable policy and renders an authorization decision.

PEP (Policy Enforcement Point) The system component that performs access control, by making decision requests and enforcing authorization decisions.

PIP (Policy Information Point) It is a component that is able to look for any attribute the context handler needs to let the PDP evaluate the policies. Therefore it is a general source of attributes and can be implemented by standard or custom the software. An example of PIP can be even a LDAP where is possible to retrieve all the attributes related to the subject.

resource The device or data element for which the access is required.

policy A set of rules that define a single access-control strategy for a resource.

context handler The component that produce a XACML encoded access request starting from the native request (decision request). It handles also the response of the PDP eventually transforming it into the canonical XACML response format from a native one. It interacts with PIP to retrieve the attributes needed by the PDP.

obligation It is not the main answer of the PDP about the access request, but a directive to the PEP that must be processed before or after the access being approved. An example can be the sending of a notification to the resource's owner that is being accessed (in case of the **allow** option) or the logging of the unauthorized request (when the evaluation of the request brings to a **deny**).

attribute A generic characteristic of the resource or the subject or the action or, finally of the environment.

The Security Framework defines the basic concepts of access control and the abstract specification of an open security mechanism.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

3.2.1.1 Data flow model

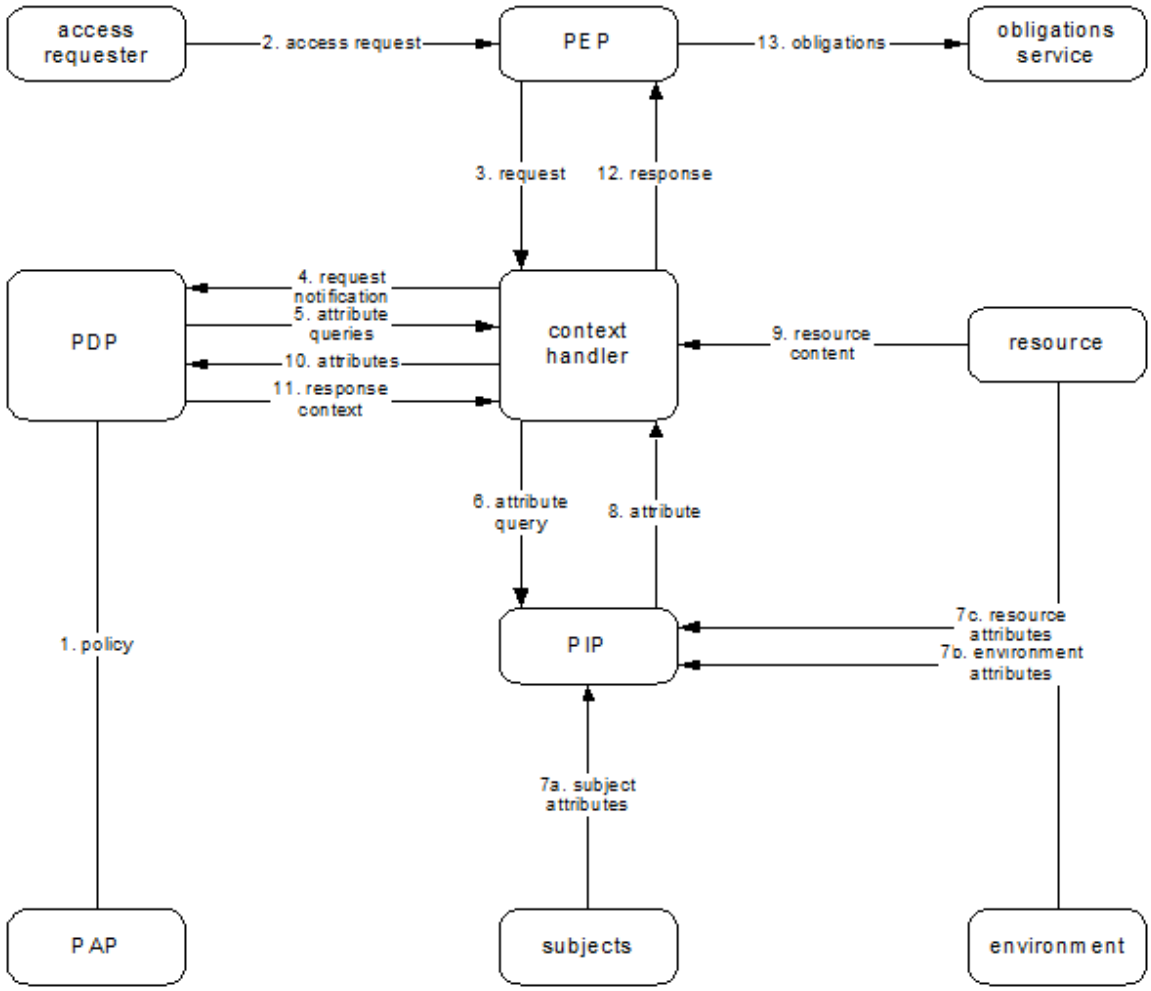

Figure 3 - XACML data flow[8]

The model operates by the following steps.

1. ***PAPs*** write ***policies*** and ***policy sets*** and make them available to the ***PDP***. These ***policies*** or ***policy sets*** represent the complete ***policy*** for a specified ***target***.
2. The ***access*** requester sends a request for ***access*** to the ***PEP***.
3. The ***PEP*** sends the request for ***access*** to the ***context handler*** in its native request format, optionally including ***attributes*** of the ***subjects***, ***resource***, ***action***, ***environment*** and other categories.
4. The ***context handler*** constructs an XACML request ***context***, optionally adds attributes, and sends it to the ***PDP***.

[8] source: http://docs.oasis-open.org/

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

5. The ***PDP*** requests any additional ***subject***, ***resource***, ***action***, ***environment*** and other categories (not shown) ***attributes*** from the ***context handler***.
6. The ***context handler*** requests the ***attributes*** from a ***PIP***.
7. The ***PIP*** obtains the requested ***attributes***.
8. The ***PIP*** returns the requested ***attributes*** to the ***context handler***.
9. Optionally, the ***context handler*** includes the ***resource*** in the ***context***.
10. The ***context handler*** sends the requested ***attributes*** and (optionally) the ***resource*** to the ***PDP***. The ***PDP*** evaluates the ***policy***.
11. The ***PDP*** returns the response ***context*** (including the ***authorization decision***) to the ***context handler***.
12. The ***context handler*** translates the response ***context*** to the native response format of the ***PEP***. The ***context handler*** returns the response to the ***PEP***.
13. The ***PEP*** fulfils the ***obligations***.
14. (Not shown) If ***access*** is permitted, then the ***PEP*** permits ***access*** to the ***resource***; otherwise, it denies ***access***.

This is the complete data flow with all the optional variants.

3.2.1.2 Sunshine's flow model

In Sunshine context some aspects of the general flow are not implemented because in this context not necessary. Some of these peculiarities are listed below:

1. PIP is not implemented as a real separated component and all the attributes are retrieved by the context handler or by some custom classes that the context handler can execute.
2. Context handler and PDP actually overlap.
3. There's no necessity of using obligations, so the PDP response will be always a simple allow or deny.

In the next chapter the Sunshine architecture will be described in details.

4 Architecture

The Sunshine's platform architecture is service oriented and, moreover, we have different kind of services that are exposed by the system.
Services are used by the Sunshine's client application, but are available for others applications too.
In this context services must be secured and an authentication service must be made available to client applications. The authentication service must also be equipped with auxiliary services in order to let the users manage their own accounts (change password, change user's details, manage password expiration).

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

4.1 Services categories

HTTP-based APIs are growing more and more, and Sunshine's platform is a good example of their intensive utilization. By now, the basis of comparison for the various type of APIs are RESTful services, but a quite subjective definition of the typology of the services makes easily the use of terms like "almost RESTful", "quite RESTful" and so on. Therefore we wish to introduce a clear classification in order to avoid miscomprehensions. This classification takes the RESTful features as parameters of evaluation, that is:

1. Clear identification of the resources. Instead of talking to a single endpoint, clients make request to the targeted resources.
2. Management of the resources by means of HTTP verbs. HTTP offers several methods, some safe and some that can change the status of the targeted resource. RESTful services use these verbs in the most coherent manner, at least for the CRUD operations: i.e. GET for read operations, POST for insert ones, DELETE for deletions and PUT for modifications.
3. Self-descriptive messages (stateless and use of HTTP headers)
4. Hypermedia controls (HATEOAS): response messages contain links that tell the client what it can do with the resource in that particular status. This is a particular useful feature that decouples the client from the URI scheme used by the server: actions can be added or changed.

4.1.1 SOAP API

This kind of services is not RESTful at all. No resources are identified by URIs, but only the service endpoint has one. Operations are not mapped into HTTP verbs, but are contained inside the SOAP body. Messages are not understandable by relaying parties. The interface is described by a specific document WSDL and HTTP is used just as (one of the possible) transport.

4.1.2 POX API

These are services that exchange messages encoded in XML, but not wrapped inside a SOAP envelope or other messaging protocol like AtomPub.
Apparently more simpler then SOAP, actually transfers to the application the tasks carried out by the infrastructure components.
For the REST features, they have the same score of SOAP APIs.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

4.1.3 RPC URI-tunneling

These types of APIs expose resources, but the operations are tunnelled through action parameters inside the URIs: in particular write/delete operation can be tunnelled through safe HTTP operation like GET. Often this category is referred to as a "KVP encoded GET request".

4.1.4 RESTful

RESTful APIs should satisfy each level depicted in §4.1, but normally the HATEOAS level is the less implemented and its specification still have some gaps (no hints on the verbs that have to be used with suggested URIs, not clear URI qualification relationships …). So in this context we use the RESTful (Level2) notation to denote APIs that satisfy all RESTful constraints but the last.

4.2 Type of Sunshine services

All OGC APIs can be classified as RPC RI-tunneling APIs as they use GET and POST HTTP methods regardless the safeness of the operations requested (an SOS' GetCapabilities can be requested through a POST even if it is indeed a safe operation).
When using GET method, resources can be seen as part of the URIs even if when using POST they are completely hidden inside its XML payload and in such sense OGC services are more similar to POX APIs.
Moreover OWS services implementation often

	RESTful (Level2)	RESTful (Level 3)	SOAP	POX	RPC URI-tunneling
OGC WMS				X	X
OGC WFS			X	X	X
OGC SOS			X	X	X
OGC SPS			X	X	X
OGC CSW			X	X	X
grouping	X				
scheduling	X				
GB ThirdParty		X			

Table 1 - List of Sunshine's services

The importance of satisfy or not RESTful principles is related to the easiness of defining rules for the evaluating authorization and of developing software that compose the XACML context. In fact, when compliance to RESTful specifications is guaranteed, retrieving resources (explicitly part of the URI) and understanding if a request is safe or unsafe (simply looking at the HTTP verb) is very simple. Moreover the characteristic of the messages of being self-explaining and

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

understandable by intermediate parties (like secure layer components) allows a pluggable secure architecture.
Of course, seen other types of APIs can be secured, but the relaying parts must look inside messages and have the specific knowledge of the service to deduce the resources involved and the actions: without understanding what is doing a POST of RESTful API to a certain resource, I can classify this operation as unsafe and adequately protect it, while a GetObservation sent with a POST of the encoded XML request must be parsed to be identified like a safe operation.

4.3 XACML model for Sunshine's services

In order to apply the XACML authorization mechanism to the Sunshine's platform, we did an analysis of which attributes were needed to base the authorization decision on. In this analysis base on the use cases, the environment is not involved, while the other entities are (subject, resource, action).
We also focused on the opportunity of creating an abstraction-layer that insulates the policy-writer from the details of the application domain. To achieve this goal, that is also one of the requirements of the XACML specifications, in the case of OGC services, we have defined the resources and the actions (and the relative attributes) strictly following the OGC information model (ORM, OGC Reference Model).
For the RESTful services, for the actions we naturally borrowed the different HTTP verbs in which these services express themselves.

4.3.1 Subject

For the subject, besides the standard attributes that are used in the LDAP's **inetOrgPerson** Object Class (the most general and widespread object class that holds attributes about people) like:

- givenname
- sn
- mail
- uid
- user
- password[9]

we defined other properties by means of a subsidiary class with these attributes:

- Class: **project**, **external**
- Category: **reviewer**, **tech partner, pilot**
- Typology: **administrator, manager, reader**
- Pilot: **Trento, Rovereto, Zagreb**, …

[9] Password is a special kind of attribute that you can never get from a LDAP server.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

- Partner: **Sinergis, Graphitec**, **Hepesco**, …

4.3.2 Resource

Resources depends on the type of service: for OGC services we started from the information model that is at the fundament of the OGC specifications and for the RESTful services we had to consider the specific domain that is faced by the single service. As the concept of resource is tight with that of RESTful service, this is quite a trivial process to do.

4.3.2.1 OGC Services

For OGC services resources are shown in the following table:

Type of OGC service	correlated resource	workspace	codespace
WMS	Layer	(list of pilots' code)	n/a
WFS	FeatureType	(list of pilots' code)	n/a
SOS	Procedure	n/a	n/a
	FeatureOfInterest	n/a	(list of pilots' code)
	Offering	n/a	n/a
	Observation	n/a	(list of pilots' code)
SPS			

We also define an attribute called "workspace" that is not really defined by the ORM model (it's the qualifier of the full qualified name of the Layer or the FeatureType); this applies only to WMS and WFS services, while for SOS service there is a similar concept that is used only for Observation and for Feature of Interest. Note that the range of these two similar attributes is not properly the same, but they can be correlated to each other.

4.3.2.2 RESTful Services

Type of RESTful service	related managed resource
grouping	Group
	Device
scheduling	

4.3.3 Action

Actions are kept as general as possible, in order to fulfil the goal of an abstraction layer: for RESTful services this is well accomplished because we use the generic HTTP verb, but for OGC

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

services we use the service's operations. However this can be considered sufficiently general as we are talking about standard services.

4.3.3.1 OGC Services

The actions defined for OGC services correspond to the operations offered by the services. The attributes related to all kind of OGC services are:

- service (WFS, CSW, WMS …)
- operation (GetCapabilities, GetMap …)
- version
- type (read, write)

Those that depend on the type of service are:

- SOS/GetCapabilities
    - section (ServiceIdentification, ServiceProvider, OperationsMetadata, Contents)

service	operation	version	type
WMS	GetCapabilities	1.1.0, 1.3.0	read
WMS	GetMap		read
WMS	GetFeatureInfo		read
WFS	GetCapabilities		read
WFS	GetFeatures		read
WFS	DescribeFeatureType		read
WFS	Transaction		write
CSW	GetRecordById	2.0.2	read
CSW	GetRecords	2.0.2	read
SOS	GetCapabilities	2.0	read
SOS	DescribeSensor	2.0	read
SOS	GetObservations	2.0	read
SOS	InsertObservation	2.0	write
SOS	GetResultTemplate	2.0	read
SOS	InsertResultTemplate	2.0	write
SPS			write

Table 2 - Attributes extracted for OGC services actions

4.3.3.2 RESTful Services

For RESTful services the actions overlaps completely with the HTTP verbs that the service may use, that is:

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

- GET
- POST
- PUT
- DELETE

We do not include those verbs like PATCH or HEAD that are not used by the large part of RESTful service and aren't used at all by the Sunshine's services.
For these services we do no advise the need of defining any attributes, but we think that the action itself is enough in order to define the access decision.

4.4 Implementation

To implement the security layer and in particular to put in place the XACML model, we did introduce in the system architecture three open source software components that play the following roles:

- user directory
- XACML repository and engine (PAP &PDP)
- service proxy (PEP)

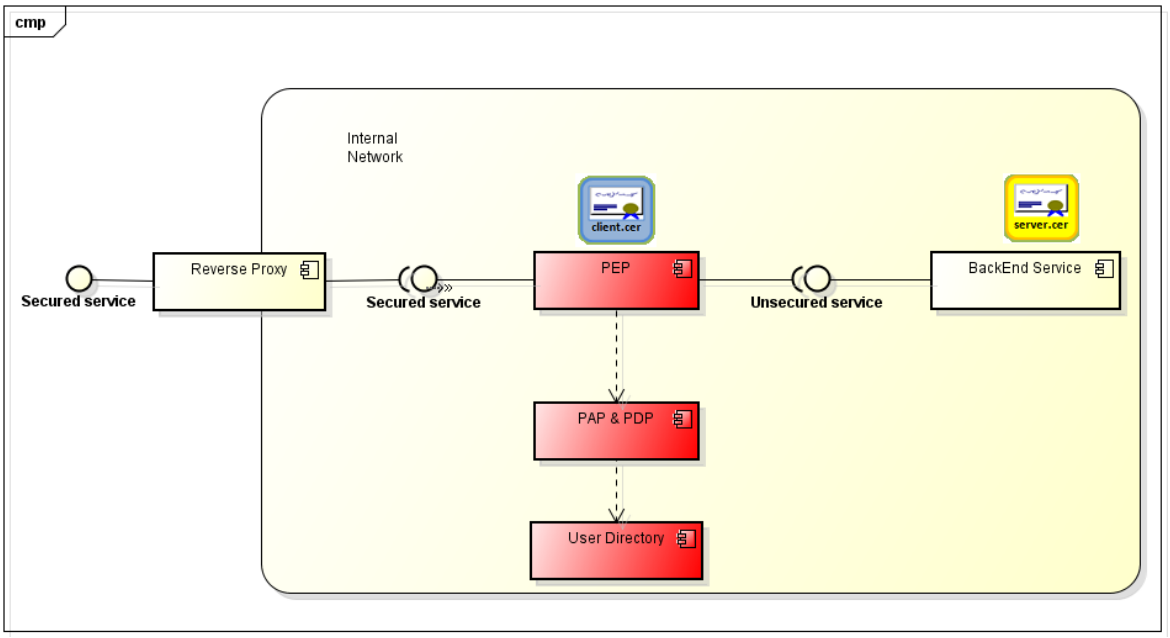

Figure 4 - High level security layer architecture

In the diagram of Figure 4 all service interfaces (secured or unsecured, inside or outside) are exposed by means of an SSL protocol and configured in such a way that is not possible to bypass the security policy. In other words, even from inside the LAN you can't call directly the back-end

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

services (that would be unsecured) because the service proxy server and the back-end service server are mutually authenticated.

4.4.1 Reverse proxy and firewall

The backend services are all installed in the private network behind a firewall and they are exposed on the internet via a reverse proxy. To be precise, the reverse proxy forwards the request not directly to the backend services, but to the ESB that acts like a further proxy and have the request make another hop before reaching the real service provider.
The reverse proxy is implemented with the specific configuration directive for the Apache web server that creates an additional security layer albeit rather weak.
The reverse proxy also delivers TLS encryption for secure communication, supporting HTTPS protocol for almost all services. However this feature is not exploited as a *single* point in which concentrate the TLS encryption for several hosts, because also many of the outbound request towards to backend services are encrypted, especially those forwarded to the ESB.

4.4.2 The service proxy

In respect to the theoretical XACML data flow, in Sunshine's architecture the PEP's role is played by WSO2 Enterprise Service Bus.
As every Service Bus does, WSO2 ESB offers a great variety of features, but in this context we take advantage of few of them, in particular the **routing** and the **mediation** functionalities and the support to a generic authorization mechanism. Routing is the capacity of directing the incoming request messages throughout various units of processing that are hosted by the ESB and that can be assembled into sequences of elaborations and/or to an external end-point. Mediation is all kind of processing that happens to the message between the client that issued the request and the final back-end service provider.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

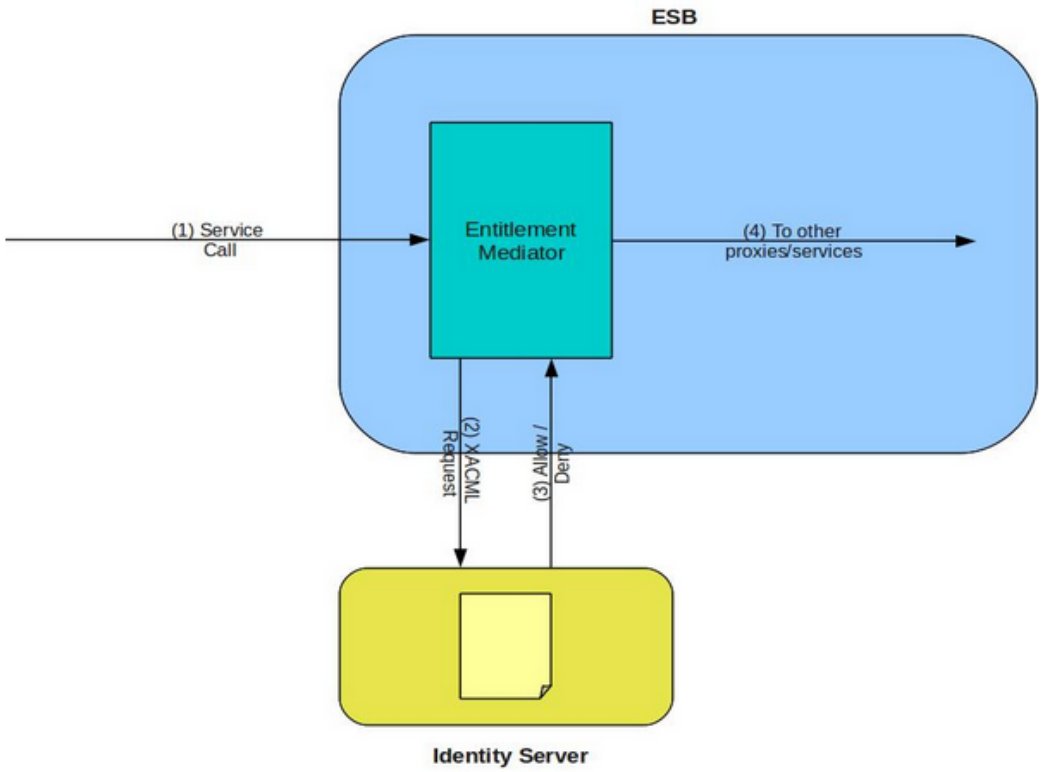

Figure 5 - Role of WSO2 ESB

Generally speaking, an ESB can act like a proxy for a back-end service and usually, before passing the incoming request to the final executor, some other steps can be applied. This is the mediation feature that in WSO2 jargon is implemented by defining a **mediation sequence** (or simply sequence), that is a list of **mediators** that are executed in order. A mediator is a unit of processing that acts on a message that has entered the service bus, doing whatever the developer wants. Various kinds of mediators are already available in WSO2 ESB and you have only to configure their specific parameters in order to use them. A special type of mediator is the **Entitlement Mediator** that intercepts requests and evaluates the actions performed by the user against an XACML policy. This mediator is able to produce an **XACML request context** to be sent to a PDP that will respond with a **permit/deny** answer and that the mediator will enforce in respect to the client access request.
Besides telling the Entitlement mediator which XACML policies evaluator is to be contacted, you must instruct by telling it:

- which sequence execute in the “permit” scenario;
- which sequence to execute in the “deny” scenario.

For the sake of completeness, the XACML authorization response is not exactly of the on/off type: in XACML there is also the possibility to receive an **obligation** or **advice** from the PDP that is an

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

additional action to be executed before or after an access is permitted. Consequently the mediator can be instructed with a sequence to be executed synchronously that elaborates the obligations and that returns a further consent/deny to be considered for the final access authorization.
The Entitlement Mediator is capable of extracting standard attributes from the incoming message to create a XACML request context, but since we need custom attribute we had to use an extension hook given by WSO2 ESB. In fact, it is possible write a custom java class that is used instead of the default implementation to extract the attributes to enable the policy evaluation.

4.4.3 XACML repository and engine

Another important component of the architecture is the PDP that receives the XACML context request from the Entitlement Mediator. Although WSO2 ESB support a standard protocol to communicate with a PDP (WS-XACML, Web Service Profile for XACML) it is not widely accepted and so, even though in theory we could choose another vendor component for playing the PDP role, our choice fell on WSO2 Identity Server (WSO2 IS). Doing so, we configured the communication to happen via a proprietary protocol (Thrift API).
WSO2 IS is a complete identity server that offers different kind of features and XACML support is just one of these. In the Sunshine's context we use it just playing this role, that is, as a PDP, but also as PAP. WSO2 IS offers a GUI for creating in an assisted way rules and policies and then storing them in a repository.

4.4.4 Mutual Authentication

To complete the securization of the architecture, the potential gap of security that regards the communication between the ESB and the backend services, is covered with mutual authentication of the servers or better certificate-based mutual authentication or two-way authentication.
This technique allows each of the two servers involved to be sure of other's identity even if they resides in the same internal LAN. In this configuration the connection between ESB and back-end server only if the client (ESB) trusts the server's digital certificate and the server (the application server where the back-end services are deployed) trusts the client's certificate. The exchange of certificates is carried out by means of TLS and the certificates are located in their respective keystores.
In Figure 6 the dialogue that is established between the two parts is described: in our case the only difference is that no certificate authority is involved in the passages, but the certificates are self-signed ones and are checked against their trust stores.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

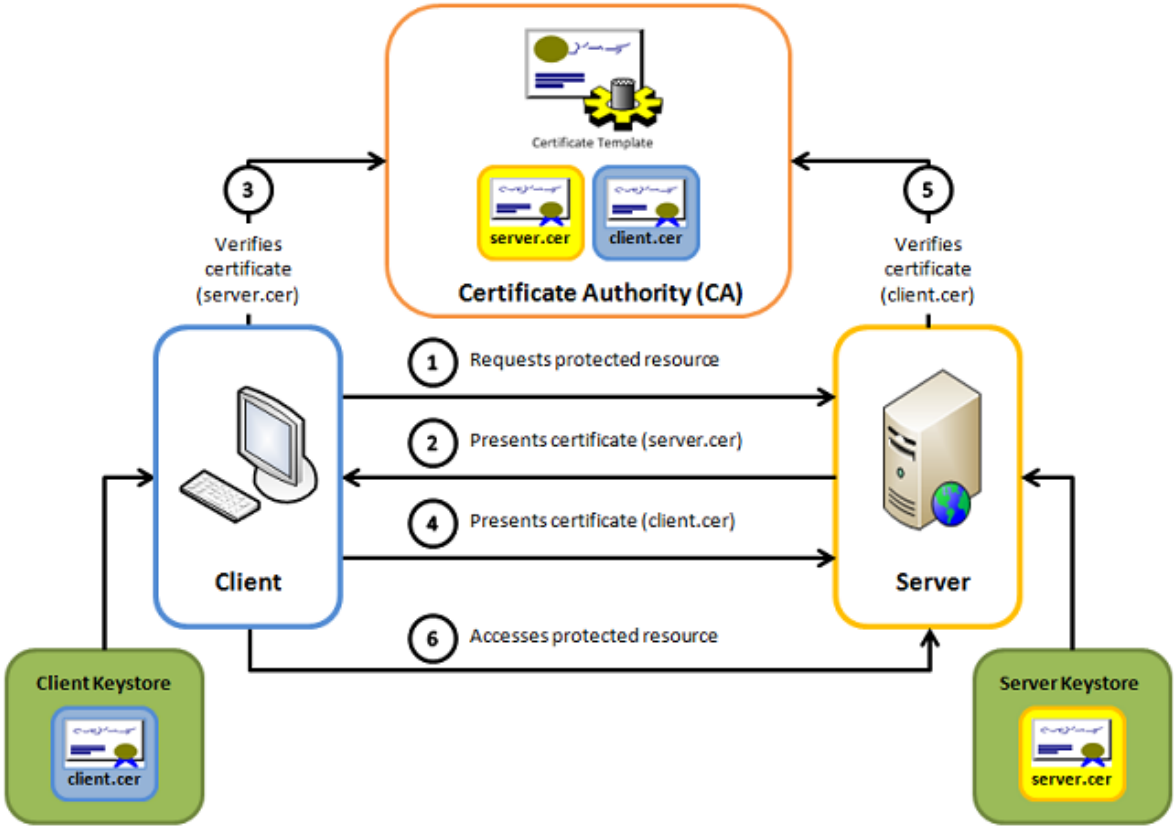

Figure 6 - Mutual TLS authentication

4.5 Software components

4.5.1 OpenDJ LDAP

OpenDJ is a fork of former project, OpenDS, and has similar roots as the Oracle Unified Directory, as it was inherited from Sun Microsystems.
OpenDJ is an LDAP directory server written completely in Java. All modules are 100% Java based and require at least Java 6. OpenJDK 8 compatible. This software is released under the CDDL license and it has good documentation and worldwide commercial support.

4.5.2 WSO2 Identity Server

WSO2 Identity Server provides sophisticated security and identity management of enterprise web applications, services, and APIs. Identity Server acts as an Enterprise Identity Bus able to manage identities over systems and networks, both locally and in the cloud.
WSO2 Server IS, has nine steps about its process flow:

1. The service provider sends an authentication request to the identity server.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).

2. The inbound authentication components forwards to message to the authentication framework after the processing it.
3. The authentication framework sends it to the local authenticator component and/or the federated authenticator component.
4. If federated authenticators are use the request is sent to the external applications.
5. The local authenticators and federated authenticators sent a response back to the authentication framework.
6. If there are any provisioning requests the response is sent to provisioning framework.
7. The user store is updated based on the provisioning request.
8. The response is sent back to the inbound authentication component.
9. And finally the response is sent back to the service provider.

Below there are some features which WSO2 IS provides: System and User Identity Management, User and Groups Provisioning, User and Groups Management, Entitlements Management and XACML 2.0/3.0 Support.

4.5.3 WSO2 Enterprise Service Bus

An enterprise service bus (ESB) is a software architecture construct that enables communication among various applications. The advantage is that instead of having to make each of your applications communicate directly with each other in all their various formats, each application simply communicates with the ESB, which handles transforming and routing the messages to their appropriate destinations. WSO2 ESB is 100% open source and is released under Apache Software License Version 2.0, one of the most business-friendly licenses available today. . Using WSO2 ESB you can perform filtering, transforming, and routing SOAP, binary, plain XML, and text messages that pass through systems by HTTP, HTTPS, mail, etc.

"This project is partially funded under the ICT Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation Framework Programme by the European Community" (http://ec.europa.eu/ict_psp).