



**Maintaining a sustainable Future  
for IT in Higher Education**

Thursday 16th June 2011

Poster Session

## **OAuth2lib Based Groups Management Tool for Authorization and Services Aggregation**

### **1. INTRODUCTION**

Over recent years we have witnessed the emergence and consolidation of a new generation of technologies around the concept of *federated identity and access management* focused to solve the problems arising from identification and authorization processes. However, although diverse authentication schemes look now as fully consolidated, authorization remains the next major issue to be solved, mainly due to their greater complexity, because, while authentication is basically a matter of "all or nothing", authorization requires a suitable granularity for each application or service.

Solving this problem is particularly relevant in the current context in which institutions can provide and make use of services across the net, many of which still using their own authentication mechanisms and internal authorization schemes. It is well known that the *integration of identities* is one of the patterns detected in the growing trend towards personal learning environments (PLE), closely linked to the *connection hub* pattern (participation in different networks and different levels of commitment) [1]. It can be said that both patterns have a technological equivalent in the authentication and role-based authorization managements.

### **2. OTHER ALTERNATIVES IN GROUP MANAGEMENT**

Externalizing the authorization of a service involves the need to maintain a control system of roles and permissions for different users groups. Managing groups -or *virtual organizations*-, is currently a common issue and several developments already exist such as *Groupier*, *GMT* or *Sympa*, so why not use one of these?

As for *Grouper* [2], this is not a solution that could be called "light", precisely. Even with the interface developed by CRU/ESCO, it remains very unfriendly (though it has improved somewhat with the new interface proposed by SURFnet). Anyway, the permissions management concerns only the group itself and its documents, not applications or services, and it is complex to utilize by end users.

The *GMT* (Group Management Tool) from SWITCH [3] has two important limitations: only global administrators can create groups, and new roles can only be defined in the original configuration, and therefore by a system administrator, exclusively.

Finally, *Sympa* [4] is an excellent manager for mailing lists that can be exposed through an API, but obviously this does not make *Sympa* a groups manager. In fact, the "groups management" is handled by means of a DokuWiki plugin and it is reduced to assimilate "Wiki manager = owner of the list" and "wiki users = list subscribers" [5].

In short, these tools are useful but end users need a lighter group manager they can manage without being forced to become technology experts, able to accommodate users of different institutions who want to have an easy way to autonomously create their own groups -students, members of a project and so on-, assigning them roles and permissions as they like.



Figure 1. Groups manager tool

### 3. DESCRIPTION OF THE GROUPS MANAGER

The groups manager presented here (Figure 1) is an evolution of a previous tool [6] and offers several distinctive features. Among others:

- It is a lightweight application, with a modular and well tested architecture. It is not built on any complex and cumbersome framework, but it uses a class library instead, *Flourish* [7], simple but powerful to use, with features such as ORM, Active Record and ACL support, among others.
- Very easy to use by end users. All functions are organized into three sections: groups and services, users and invitations, roles and permissions.
- Groups can be turned on and off, edited to change the name, characteristics, startup and end dates, to assign different services, and organize them hierarchically, as subgroups of a higher level, parent group (Figure 2). The hierarchy can be rearranged at any time simply by changing the parent or moving the group to the root level, directly.
- Integrated mechanism of invitation by email, which are inserted either manually or from a file or reading them both at the same time. The invitations are always for one of the established roles for the group, whether those are included by default or the owner of the group defined.

These features are designed to make end users able to autonomously manage their groups, regardless of the institution to which they belong, with the only requirement to be grouped together in a federation, such as the SIR [8] of RedIRIS, the Spanish NREN.

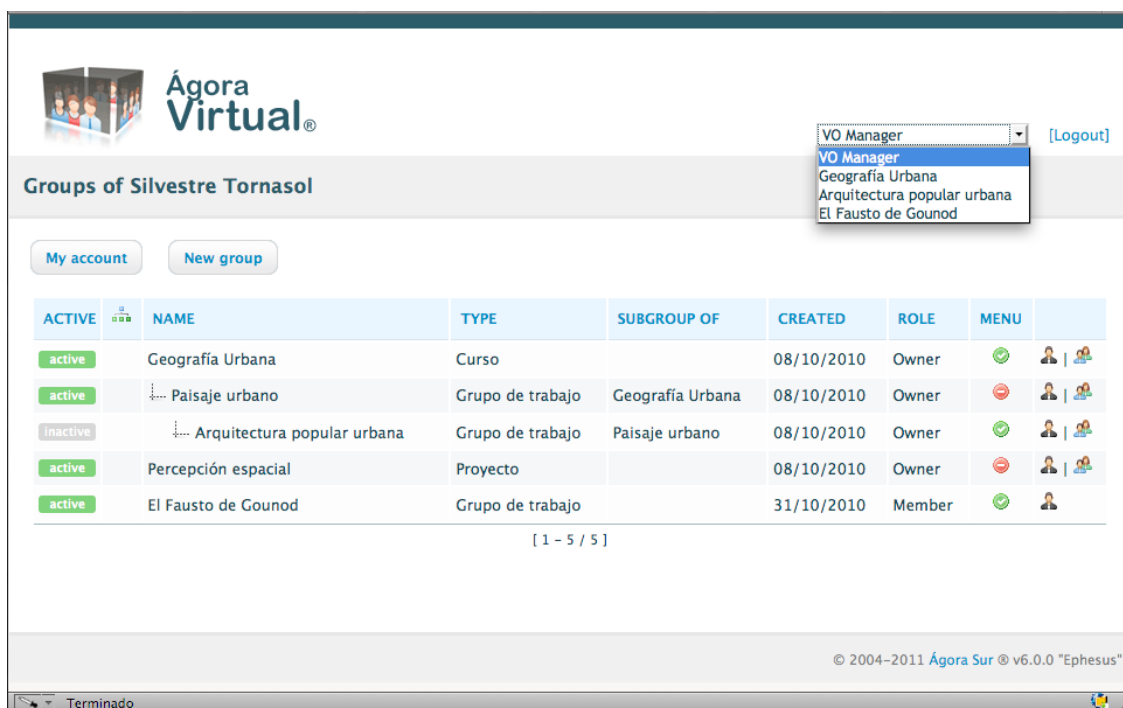


Figure 2. User's groups

## 4. ROLES AND PERMISSIONS

To manage the authorization a role-based ACL (access control list) mechanism has been chosen. Most of the existing ACL implementations (phpGACL, Spiff Guard, CakePHP ...) are global so their tables keeping the permissions must also be aware of the kind of object they are referring to. In our case, however, the ACL does not need to be global because the group defines from the start which objects the user has access. Therefore, the ACL only has to control what actions are allowed within the group.

The group owner can easily manage the group's roles, edit existing or add new ones, activating a given role for the functions it deems appropriate, for example "edit wiki", "evaluate heading" depending on the services the group has assigned (Figure 3).

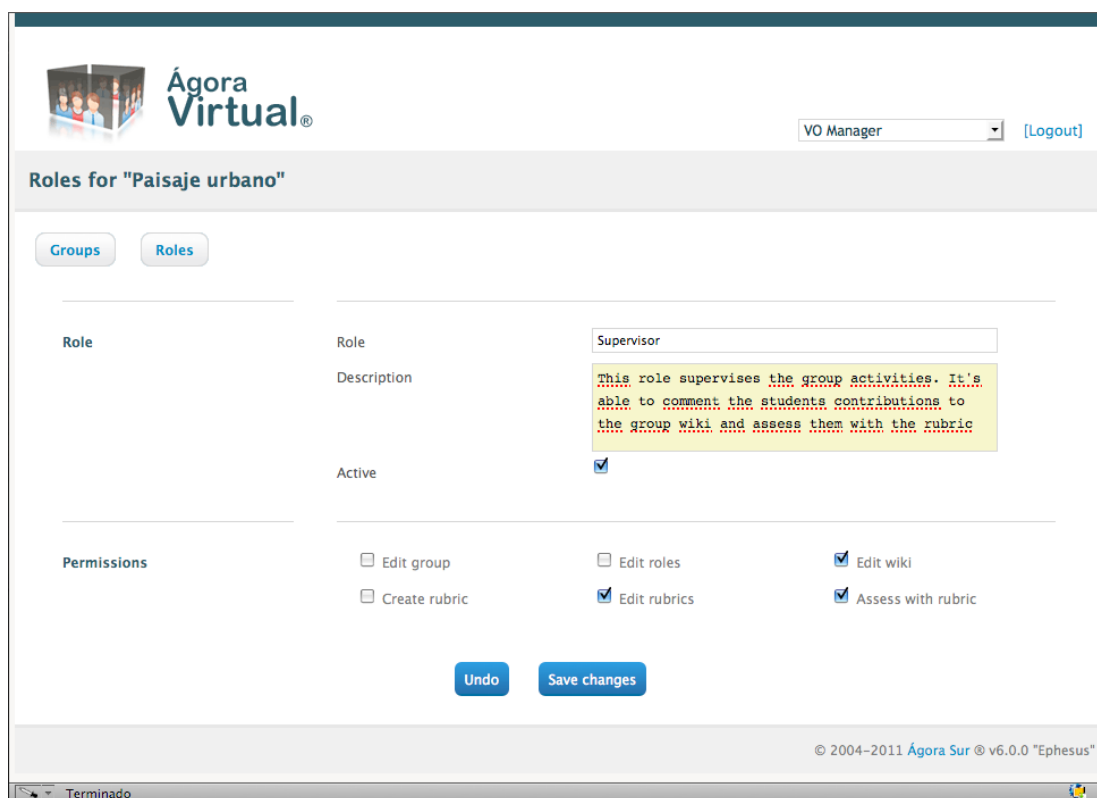


Figure 3. Role and permissions editing

Naturally, the problem is how to make the application aware of what a given role means and, above all, what the group owner wants that role means in that group. Therefore, we need to see how and where to link each role to the functions of the application. Of course, it could be done in the application itself, but then the application should be aware in advance of all the possible roles, and as a result the group owner would not be free to specify new roles to his liking. It is easier -and more logical to manage by an end user- that the group manager knows the functions available in the applications and not the other way around. This means that when a new service is added to the environment, the system administrator should let the group manager know the appropriate permissions for the new service (for example, for a wiki they could be "read", "edit" and "administer"), so they are available for the groups owners use them in their roles as they like.

## 5. AN ARCHITECTURE FOR SERVICES AGGREGATION

As an example of use is proposed a service architecture in which users of several institutions are able to get shared access to a formative assessment tool or *Rubric*.

The *Rubric* (Figure 4) consists of an array, of one or more dimensions, for skills and knowledge assessment, and provides a framework for teachers and students have a clear idea about the criteria to be applied in the evaluation [9]. Quite the opposite of other simple HTML editors available on the network, the tool used permits to assign criteria and indicators giving them specific weights, among other features.

Rubric					
<a href="#">Back</a>					
Collaborative work - Haddock ,					
Criteria	Indicators				Value
<i>Contribution to the group</i>					
Searching information	Doesn't search information	Collects little information or it is not relevant	Knows to look for information suitable for the most part	Is able to collect a lot of information relevant to the topic	7.5/10
	2.5	5	7.5 ✓	10	
Sharing information	Does not share any information	Shares little or irrelevant information	Shares enough relevant information	He shares much and relevant information	7.5/15
	3.75	7.5 ✓	11.25	15	
<i>Responsibility</i>					
Participating in group discussions	Not involved in the group debates	Participates little or irrelevant things	Makes important contributions	Is proficient, making many contributions and all relevant	6.67/10
	0	3.33	6.67 ✓	10	
Distribution of tasks	Expects others to do the job	Others must remind him his part of the work	It is not usually necessary to remind him his task	Always does his part of the job	15/15
	3.75	7.5	11.25	15 ✓	
<i>Rating the point of view of others</i>					
Cooperation	Frequently faces or altercates his peers	Sometimes discuss minor issues	Is not polemic within the group	Never polemic, solves all conflicts positively	5/10
	2.5	5 ✓	7.5	10	
Decision Making	Expects things to be solved alone	Identifies himself only with his friends	Usually considers all points of view	Always helps the team to make the right decision	11.25/15
	3.75	7.5	11.25 ✓	15	
<b>Total</b>					<b>52.92/75</b>


© 2004-2011 Ágora Sur © v4.1.1 "Priene" -  RSS

Figure 4. Rubric tool

The rubric activity is naturally organized according to groups: for example, the teacher can evaluate a competence in a group of students, but students could score themselves in a peer evaluation as well. Obviously, the service could manage the groups internally, but outsourcing the groups management they may also be used by other applications or services without having to recreate each of them.

The architecture consists of three blocks: identity providers within the federation, the group management service (GS) and the service(s) provided, the *Rubric* tool in this case (Figure 5).

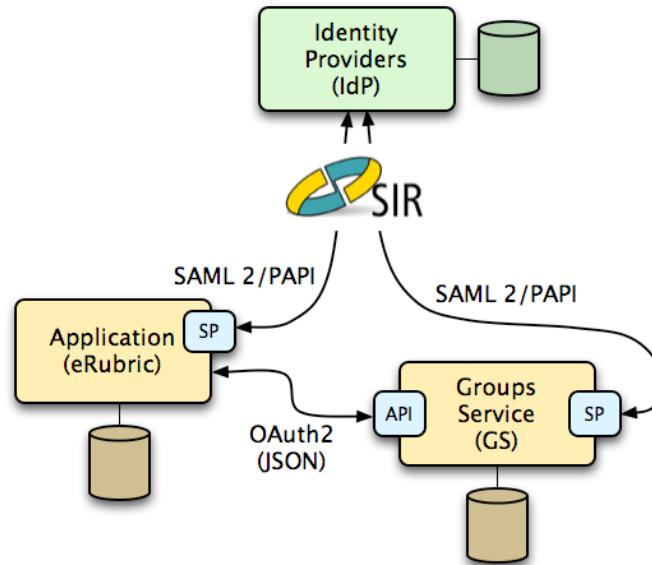


Figure 5. Services aggregation architecture

In this architecture, a possible sequence of authorization could be (Figure 6):

1. The user enters the Rubric service and is authenticated by an Identity Provider that issues a PAPI or SAML 2 assertion.
2. The Rubric requests to the Group Manager the list of groups the user belongs to.
3. The rubrics of these groups are shown to the user.
4. The user attempts some action on a rubric: view, edit, evaluate...
5. The Group Manager is asked for the user permissions within the group owning the requested rubric (which will depend on their role in that group).
- 6-7. The user is allowed to execute the action (i.e. to assess a student) according to the received permissions from the Group Manager.

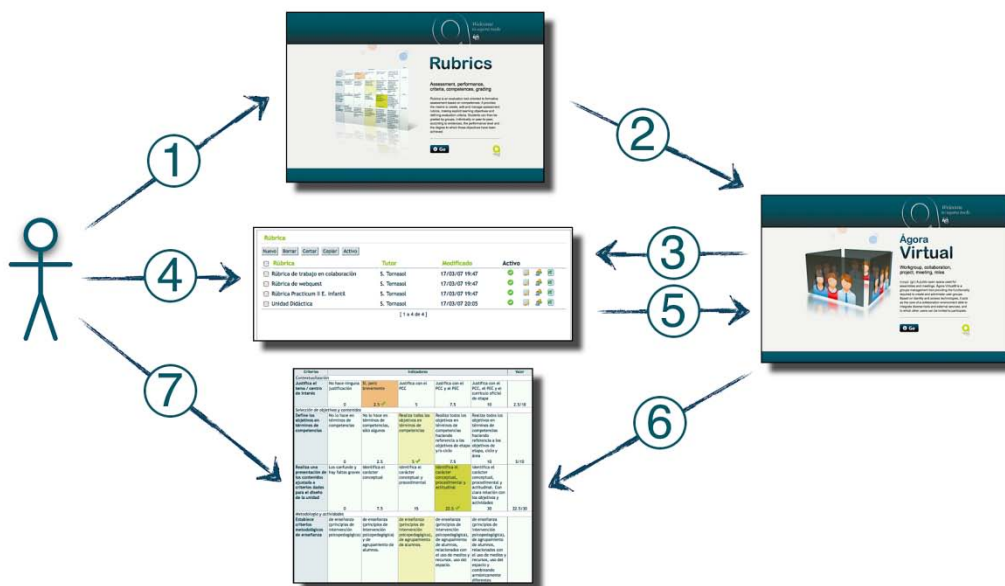


Figure 6. Sequence of authorization

## 6. THE OAUTH2LIB LIBRARY

Therefore, to be useful as authorization source, the group management service must have some mean to provide applications with relevant information about groups, members, their roles and permissions using a protocol able to ensure security and integrity. For this purpose our setup uses the authentication and authorization protocol named OAuth [10].

OAuth (Open Authorization) is an open standard to include security in authentication of web and desktop applications. Broadly speaking, it consists of a tool that allows a client application to act on behalf of a particular user to obtain resources that are owned by the same and are stored in a particular resource server.

The basics of the OAuth architecture are:

- *Client Application*: An application that requests and uses the user information for a particular purpose. In this case the client application is the *Rubric* service.
- *Authorization Server*: The server that collects information from the assertion of the client application and the resource to which access is desired and returns a code of access to resources to be used in the resource server.
- *Resource Server*: A server that provides access to resources with a particular access code. In the use case of the *Rubric* service both servers are located in the Groups Manager server.

Version 2 of OAuth is a completely new protocol and is not backwards compatible with previous versions. Among other changes, there are new methods for obtaining an access token. In the *Rubric* case, the required profile is defined as “SAML 2.0 Bearer Assertion Grant Type Profile for OAuth 2.0” [11] in the current draft 13, an evolution of the previous “Autonomous Client Profile” of draft 11. This flow does not require user action because the client presents an assertion -such as a SAML assertion- to the authorization server in exchange for an access token.

Therefore, this profile is well suited to this use case because, instead of using the user directly in all access requests, such approval is delegated into user attributes that are contained in an assertion. For this reason, in addition to factors discussed above, this profile provides the figure of the Identity Provider, which will be responsible for generating the assertions of users based on their attributes. In our case, this function is performed by an Identity Service as the SIR of RedIRIS.

The steps performed to provide access to some restricted resources through OAuth2 are the following (Figure 7):

**1. The client obtains a proper assertion**

The client application -the Rubric service in this case-, communicates with an identity provider, the SIR, so that the user logs on this provider and this will generate an attributes assertion with a duration determined by the configured lifetime.

**2. Getting the Access Token**

The client application -the Rubric, again- sends the assertion obtained in the previous step along with the *scope* or objective it wants to access (some entry to the Group Manager API) and its credentials as a client application. These credentials must have been obtained by the application before starting the authorization flow, through its registration and configuration in the authorization server. All these data make the authorization server able to ensure that: a) the requesting client service is already registered and is not a fraudulent application that could misuse of data, b) it is authorized to access the specified scope; and c) the user, represented by the assertion, has enough privileges to access the resource.

If all of the security conditions are satisfied, the authorization server returns an access token, which will allow access to the specified scope in the resource server through the specific lifetime, specified in advance at the server configuration.

**3. Obtaining resources**

The client application sends the access token to the resource server and if it is valid and has not expired, the server will respond with the action requested by the client application, in this case a JSON string with the required information.

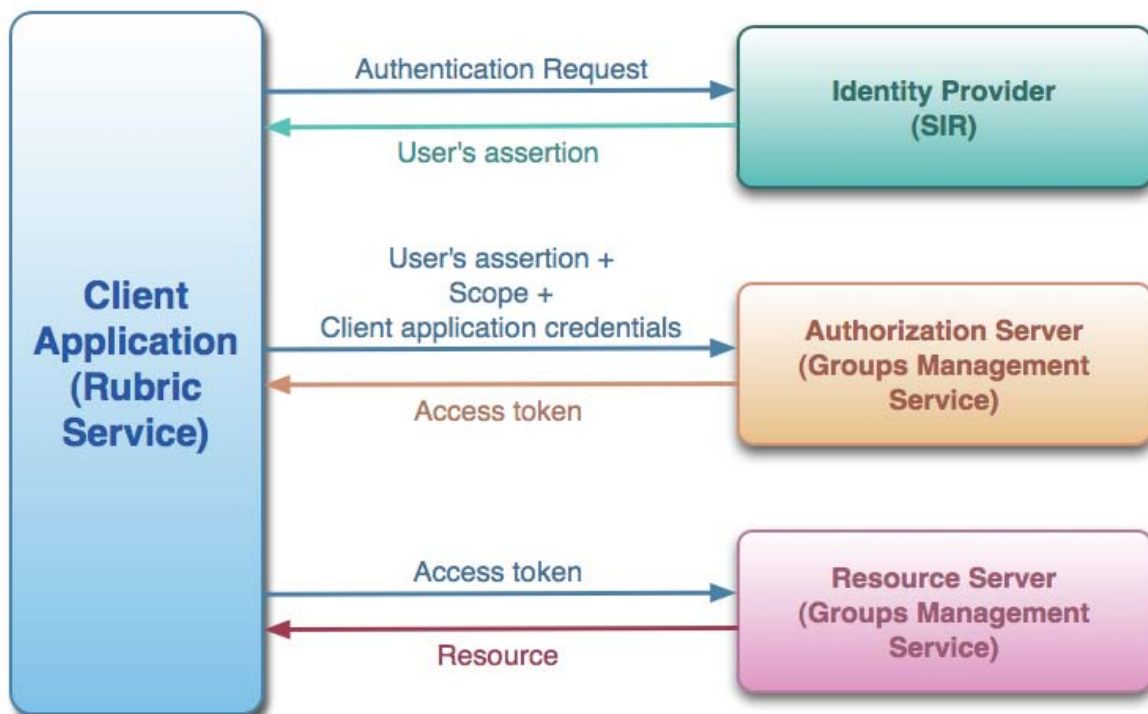


Figure 7. Using the OAuth2 assertion flow

The core component to implement this protocol in the Rubric service is *oauth2lib* [12], an open source library developed by RedIRIS that implements the assertion profile of OAuth protocol version 2. In addition, other safety and usability features have been added for a more secure and functional operation of the library:

- Integration with federated systems, as RedIRIS SIR, by means of SAML2 and PAPI assertions.
- Easy addition of functions to process different types of responses and integration of new resources through the modular organization of the architecture of the library.
- Use of XML for simple and personalized settings.
- Establishment of trust relationship between authorization server, client applications and resource servers, preventing other fraudulent servers or applications could be taken for valid ones.
- Registering of client applications and scopes in both the authorization and resource servers using XML files to ensure that access is allowed only to those having adequate privileges for doing so.
- Ability to define different access policies according to the assertions and the attributes received.

## 7. CONCLUSIONS

Identity technologies make possible a scenario consistent with the evolution of the technological and educational environment in which services can be offered to the community through a federation scheme, with the consequent saving of resources and a positive impact on good practices of collaboration between institutions. However, besides the already known authentication schemes, this model requires a service in which users can rely to manage their groups and a secure mean of transferring information between services. The architecture presented for the specific case of a Rubric service is an example of this model, in which the end user has full capability to create and manage their own groups. From the assertion issued by the identity provider, the RedIRIS OAuth2lib library makes the applications able to retrieve the required information about authorizations available to the user according to his role in the group, to adjust their response to these authorizations.

## 8. REFERENCES

- ❖ Wilson, Scott : Patterns of personal learning environments. (Educational Cybernetics: Journal Articles) University of Bolton Institutional Repository.  
[http://digitalcommons.bolton.ac.uk/cgi/viewcontent.cgi?article=1005&context=iec\\_journals\\_pr](http://digitalcommons.bolton.ac.uk/cgi/viewcontent.cgi?article=1005&context=iec_journals_pr)
- ❖ <http://www.internet2.edu/grouper/>
- ❖ <http://www.switch.ch/aai/support/tools/gmt.html>
- ❖ <http://www.sympa.org/contribs/sympaauth>
- ❖ In our view, it should be the other way around: instead of using the lists as a source of groups, we should make use of the ability of Sympa to use any groups as sources for their lists.
- ❖ Ágora Virtual: Una propuesta de entorno colaborativo y de enseñanza sobre interfaces OSID. Boletín de RedIRIS núm 76, abril 2006  
<http://www.rediris.es/difusion/publicaciones/boletin/76/enfoque1.pdf>
- ❖ <http://flourishlib.com/>
- ❖ <http://www.rediris.es/sir>
- ❖ Cebrián, M.; Accino. J.A.; Raposo, M.: Formative evaluation tools for the European Area of Higher Education: ePortfolio and eRubric. EUNIS Conference 2007. Grenoble (France).  
<http://www.eunis.org/events/congresses/eunis2007/CD/pdf/papers/p85.pdf>
- ❖ <http://oauth.net>
- ❖ SAML 2.0 Bearer Assertion Grant Type Profile for OAuth 2.0:  
<http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-03>
- ❖ <http://www.rediris.es/oauth2/>