available at www.sciencedirect.com
ScienceDirect
journal homepage: www.elsevier.com/locate/cose
Computers & Security

Reverse OAuth: A solution to achieve delegated authorizations in single sign-on e-learning systems

Jorge Fontenla González[*], *Manuel Caeiro Rodríguez*, *Martín Llamas Nistal*, *Luis Anido Rifón*

Escuela Técnica Superior de Ingenieros de Telecomunicación, University of Vigo, Campus Universitario, 36310 Vigo, Spain

ARTICLE INFO

Article history:
Received 2 April 2009
Received in revised form
8 June 2009
Accepted 8 June 2009

Keywords:
Access granting
Delegated authorization
E-learning
Password stress
Reverse OAuth
Service provider
Single sign-on
Web service

ABSTRACT

Current scientific and technological progress has led to the proliferation of e-learning systems known as *Learning Management Systems*. These systems consist of a central application for managing the sequencing of students' tasks, and also on several other educational applications that allow its users (teachers and learners) to communicate, carry out experiments, etc. However, despite the widespread use of these systems they show a usability problem when both kinds of applications require spare authentication processes. Indeed, users have to introduce several kinds of credentials, preventing them from focusing their efforts on their studies and increasing the so-called "password stress". Several initiatives such as OAuth or Delegation Permits have dealt with the problem of delegated authorizations, but their requirements are different from those that arise from an e-learning environment. In this paper we introduce Reverse OAuth – a protocol to enable the granting of authorizations to access protected resources in educational environments.

1. Introduction

Given the rapid scientific and technological progress, organizations such as universities and enterprises have envisaged the need to provide continuous formation to their students and employees (namely, life-long learning). This fact, together with the widespread use of broadband Internet connections, has led many institutions to grant access to educational resources through e-learning systems known as LMSs (*Learning Management Systems*). These systems deal with the administration, provision and control of educational resources and functionalities. Therefore, LMSs have achieved a breakthrough in education, as they allow covering educational needs avoiding spatial and temporal barriers.

Current LMSs can be considered as complex Web applications. Some of the best-known examples are Moodle (Web site of the Moodle project, 2009), Blackboard (Web site of the Blackboard project, 2009), LRN (Web site of the dotLRN project, 2009) and Sakai (Web site of the Sakai project, 2009). These systems typically provide a centralized environment to supply data (pdf documents, multimedia files, etc.) along with applications or tools to manipulate them. Nonetheless, the growing complexity of LMSs is leading to a design approximation in which the tools are split from the LMS itself (Fontenla et al., 2008; IMS Tools Interoperability Specification, 2009; Vogten et al., 2006). Tools become standalone Web applications which are *not part of* but are *used* by the LMSs (IMS Tools Interoperability Specification, 2009).

[*] *Corresponding author*. Fax: +34 986 814 073.
E-mail addresses: jfontenla@det.uvigo.es (J.F. González), manuel.caeiro@det.uvigo.es (M.C. Rodríguez), martin.llamas@det.uvigo.es (M.L. Nistal), luis.anido@det.uvigo.es (L.A. Rifón).

0167-4048/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2009.06.002

However, this approach also has some difficulties. Some of them inherently arise from the invocation of remote services: discovery of remote interfaces, transmission of real-time data, privacy and integrity of the messages, etc. Nevertheless these problems, although critical, are to a great extent solved by existing technologies.

Other kinds of difficulties are related to the usability of these systems and, due to the innovative approach of splitting an LMS and its tools, they have not been tackled yet. It would be desirable if the students did not have to authenticate themselves after the tools, provided that they have already been authenticated at the LMS. This can be seen as the counterpart of what happens in ‘‘old-fashioned’’ LMSs containing tools as well as the LMS itself: a student has to authenticate only once at the beginning of the session, but after that he can use the tools freely without having to authenticate again. This authentication principle in which a user can access many systems with a single authentication instance is frequently known as *single sign-on* (Introduction to single sign-on, 2009).

The study of a way to accomplish single sign-on within the LMS and the different Web tools is the main purpose of this article. We take as a starting point four single sign-on technologies, analyze their suitability to our e-learning environment and present our own proposal that solves the problems identified.

This paper is structured as follows. In Section 2 we describe a typical e-learning scenario where a new single sign-on technology is required, and list the main requirements that must be satisfied. Section 3 depicts an overview of OAuth, Delegation Permits, Shibboleth and OpenSSO, and analyzes them against the requirements of Section 2. In Section 4 we come up with a proposal of solution which we called Reverse OAuth, that solves the lacks identified in Section 3, and in Section 5 we describe its implementation process. We end up the article with Section 6, where we extract some conclusions.

2. Problem description

In the previous section we mentioned some general-purpose authorization technologies. Nevertheless, in order to analyze their suitability for our purposes we have to formalize the problem we are dealing with. This section offers a clear picture of what we want. Section 2.1 goes into some detail on the architecture we based our work on, while Section 2.2 provides several use cases concerning the use of delegated authorizations that could take place over this architecture. From these use cases, Section 2.3 extracts the main requirements of a solution.

2.1. Architecture description

Fig. 1 depicts the architecture under study. We can see three entities – the LMS, a Tool and a user:

- On the one hand, the LMS provides the core functionality of the educational system (e.g. authentication modules, databases to store personal data of the students, the logic to manage sequencing of tasks).

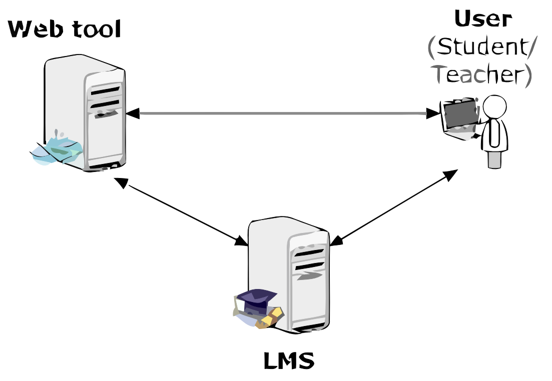


Fig. 1 – Architecture description.

- On the other hand, the Web tool is a standalone application providing a specific functionality that help users to carry out their tasks. Examples of Web tools can be a forum, an assessment application, a hydrodynamics simulator, etc.
- Finally the user accesses the LMS to carry out his/her tasks with the aid of the Web tool.

This decoupling of LMS, users and Web tools allows LMSs to extend their functionalities in an easy way, as the only requirement to add new features to the LMS is establishing an Internet connection with the appropriate Web tool. Thus an LMS can use many Web tools, but also a Web tool can be used by many LMSs. This solution implies that the development of both LMSs and Web tools can follow separate paths.

To that end, the developers of the LMS have to adopt certain specifications to support the interaction with the Web tool. The developers of the Web tool must also embrace these interaction specifications in order to allow communication with the LMS. This mutual interaction is represented in Fig. 1 with an arrow that links the LMS with the tool. Users take advantage of this mutual interaction to carry out the tasks indicated by the LMS by using the Web tool, which is represented in Fig. 1 by the arrows coming from the user. Notice that there may be more that one Web tool available, and that the user accesses the one that fits better with his/her needs.

The Web tool may (and should) have implemented some access control in order to prevent unauthorized users to access it. This brings up the possibility that only the students from a university are allowed to use the tool, but not those from another university. In this document we consider two ways to perform this access control:

1. Each user of the LMS has a working account at the tool.
2. The LMS has a working account at the tool, and its users are granted access as invited users. This mechanism is known as *delegated authorization*.

The first solution is neither scalable nor practical, as users have to remember many passwords (one passwords to access the LMS, plus one password per tool). Moreover, they are asked to authenticate themselves every time they want to access a tool. The second solution solves the drawbacks of the first one, as it is based on a delegated authentication mechanism. Hence, we have chosen it to base our single sign-on solution.

2.2. Use cases

In order to clarify the expected functionality of the architecture of Section 2.1 we introduce some use cases. All of them are based on a user that wants to carry out the activities of a ''Hydrodynamics'' subject of the University A. The theoretical material and the exercises required for the subject are available at an LMS hosted and managed by the University A.

In order to help the users with the exercises, two Web tools are available. The first one finds the speed of a fluid by solving the Navier–Stokes equations under some boundary conditions, while the second one estimates its friction coefficient according to conditions such as temperature, pressure, etc. These tools are hosted and managed by the University B.

At a certain point of the lesson, the user must answer an online questionnaire. This functionality is provided by a Web assessment tool allocated at University C. The user can use any of the simulators of University B together with the assessment tool in order to arrive at the solutions. The time limit of the questionnaire is 1 h.

The use cases based on this scenario will help to clarify the concepts involved in the architecture and emphasize its potential advantages, and will also allow us to establish a formal list of requirements (see Section 2.3).

Use case 1. When the user wants to attempt the questionnaire, the LMS provides him/her with hyperlinks to the assessment tool and to the two simulators. From the moment he/she clicks on the link to the questionnaire, he/she has 1 h until the time to answer the questions is over. During that period he/she can use any or both fluid simulators, according to his/her needs. When the hour has passed by, he/she is denied to continue answering the exam, and his/her qualification is displayed.

Use case 2. When the user accesses the activity involving the online questionnaire, the LMS provides him/her with hyperlinks to the three tools (the assessment tool and the two simulators). When clicked, the LMS establishes a negotiation with the tool in question in order to grant the user free and transparent access to the tool for 1 h.

While the user completes the questionnaire the LMS can receive feedback from all the tools, concerning what the user is doing. For example, it can receive information regarding how many times the user has changed the answer in a specific question. All this information is collected by the LMS, which can be used to modify the authorization (e.g. if some question has been answered correctly, the LMS can grant him/her access to an optional part of the exam), or simply to produce statistics.

When the hour has passed by, the qualification is transferred from the tool to the LMS, where it is stored in a database.

Use case 3. Just before the user tries to access a tool (either the assessment tool or a simulator) the tool receives an incoming connection by the LMS, meant to establish the length of the usage by the user and the files he/she is allowed to access. This negotiation process does not involve sensitive data concerning the user, just access parameters.

Once these parameters have been negotiated the user gains access to the tool. While he/she completes the questionnaire, the tool reports events to the LMS concerning his/her operation. When the user clicks on the ''Log-out'' button of the assessment tool, or when the hour has passed by, the assessment tool notifies one final event to the LMS involving the final qualification of the user.

2.3. Enumeration of requirements

After the description of the architecture under study and the use cases we are now ready to list more formally the main requirements of a solution. A subset of the following requirements have been identified as very urgent by some e-learning commissions (Klobucar, 2008).

Requirement 1: Interoperability. The LMS must be able to interoperate with a Web tool even if they are in different network domains. This is necessary because in general the Web tool is not managed by the same entity of the LMS, and hence it is located at a different network domain. This requirement is implicit in the three use cases above.

Requirement 2: Access Transparency. It should be possible for the users of the LMS to access a tool without being prompted to authenticate after it, given that previously they had already authenticated themselves after the LMS. The LMS should grant delegated authorizations to its users to use the tool. Access transparency is part of Use case 2.

Requirement 3: Privacy. The tool cannot have access to sensitive data regarding the user hosted at the LMS, but only to those supplied by the LMS itself. In Use case 3 of Section 2.2, Universities B and C do not have access to sensitive information regarding the student of University A (e.g. name, email address, telephone number).

Requirement 4: Choosability. A user of the LMS should be able to access whenever he wants, except errors in the communications network, to the tool he prefers among those available and offered by the LMS. The election is hence constrained by his/her needs, personal preferences, limitations, etc. This requirement is enclosed in Use case 1.

Requirement 5: Granularity. A user should be able to access particular resources (e.g. concrete pieces of data, concrete functionalities) at the tool with different levels of permissions (e.g. read only, read/write, execution only). This ''granularity'' is necessary if we want to ensure that the data of one user cannot be accessed by another user without permission. In Use case 1, for example, a student should only have access to his/her own instance of the questionnaire, and not to anyone else's.

Requirement 6: Simplicity. As far as possible, the solution must be simple and scalable. Furthermore, such solution should involve no more actors than the LMS, the tool and the user, which would make it independent from the correct working of external entities.

Requirement 7: Dynamic Reconfiguration. It should be possible for an LMS to modify the characteristics of an ongoing authorization to access a tool. In Use case 2, the LMS may need to modify the authorization during runtime if the user must be granted access to an optional part of the questionnaire.

Requirement 8: Expiry. The LMS must be able to grant authorizations for a finite period of time. In case that this validity time is not enough it should be possible to extend it. Our example suggests the need of a time limit for authorizations, as the session with the assessment tool must terminate after 1 h.

In a general case, the validity time of an authorization should not always be set to a predefined period. Instead, it should be determined depending on the resources to be accessed, the purpose of the authorization, its beneficiary, etc.

Requirement 9: Awareness. The LMS should be able to track the activities of each user at a tool. Indeed, as pointed in the Use case 2, the LMS can grant access to additional resources depending on the information it receives from the tools.

Requirement 10: Pseudonimity. Provided that a tool should not have access to the name of the user by Requirement 3 (Privacy), it should provide some mechanism to set identifiers to its users in order to differentiate their activities. This is necessary, for example, if the assessment tool of our example wants to send reports to the LMS concerning the activities of a concrete user.

Requirement 11: Confidentiality. Sensitive data sent between the user, the LMS and the tool must be kept confidential against eavesdropping attacks. For example, the information with which the LMS grants access to the assessment tool in Use case 2 must be kept confidential, as an attacker could intercept and reuse it to get unlimited access to the tool.

Requirement 12: Integrity. It must be possible to detect illicit modification on the messages sent between the user, the LMS and the tool. It is important, for example, that the request for a 1 h access to the assessment tool cannot be illegally modified into a request for 2 h access.

Requirement 13: Authenticity. The delegated authorization mechanism must detect whether the user, the LMS or the tool have been impersonated. When an important learning activity takes place, such as the questionnaire of our example, it is important that both the LMS, the student and the tool are sure that none of them has been impersonated.

Requirement 14: Single-use Authorizations. The user cannot reuse any expired authorizations previously granted by the LMS to access the protected resource again. The tool must be able to detect expired authorizations and consequently deny the access to the resource. This is obvious in our example, as a student should not be allowed to redo the exam by reusing the authorization he/she was previously granted.

3. Related works

From the increase of popularity of the Web 2.0 more and more Internet sites can interoperate to achieve a brand-new functionality. An example of this is NetVibes (NetVibes Web site, 2008), that allows collecting information from other sites where the user has an account (e.g. Gmail, Yahoo! Mail) to display it together. Nevertheless, these sites show an important privacy problem. NetVibes, for example, requires from the user the password of his e-mail account. From that moment, the trust the user has in that NetVibes does not make illicit use of the password (e.g. read private mails, send mails impersonating the user, change the password) plays a key role.

The case of NetVibes is quite frequent. In view of this scenario, some single sign-on solutions have flourished. Single sign-on (Introduction to single sign-on, 2009) is a way to perform access control that allows a user to access several software systems authenticating just once. Single sign-on is not a specific technology or protocol, but a methodology. It does not involve concrete protocols or technologies, which allows using some of the already existing ones.

Among the benefits of single sign-on we must emphasize the following two:

1. Reduction of the effort needed to remember multiple passwords (i.e. *password stress*). As a consequence, the few that have to be remembered can be more secure.
2. Reduction of the time needed to re-enter credentials for the same user.

In our context the users are students and teachers, and the systems requiring authentication are the LMS and the Web tools. Single sign-on is specially interesting in this scenario, as users have already authenticated after the LMS and, *from their point of view*, additional authentications after each tool should not be necessary.

Among the different ways to achieve single sign-on we have selected four that are quite relevant for our purposes. These four techniques are OAuth (OAuth specification, 2009), Delegation Permits (Hasan et al., 2008), Shibboleth (Shibboleth Web site, 2009) and OpenSSO (OpenSSO Web site, 2009). All of them have been proposed to grant delegated authorizations in single sign-on scenarios involving Web applications.

Emerging technologies such as OpenID (OpenID Web site, 2009) try to give solutions to the problem of delegated (or federated) *authentication*. These technologies are based on delegating the authentication of users of a service provider to an external third-party identity provider. Once the authentication process is complete the service provider has full control over what users are allowed to do. In other words, the access policy at the service provider cannot be altered by external providers. On the contrary, when dealing with delegated *authorizations* the access policy can depend on external service providers with which a trust relationship exists. These external providers can "invite" their own users to access the service provider according to their needs, even if they do not have a working account. *Delegated authentication* technologies fall out of the scope of this article. Our problem is about *delegated authorizations*.

In the following we give a brief introduction to OAuth, Delegation Permits, Shibboleth and OpenSSO.

3.1. *OAuth*

The OAuth initiative (OAuth specification, 2009) has arisen as a method to achieve delegated authorizations. The version 1.0 of the specification has been published in October, 2007. Ever since then, the interest for OAuth has increased to the extent that many companies have adopted or are considering to adopt it (OAuth first summit, 2009) (e.g. Google, Yahoo!, MySpace), and a strong developing community has been built around it due to its openness (OAuth developing community, 2009).

Using OAuth a user can grant to a Web application tickets to access protected resources hosted in another site, without having to trust any set of credentials. These protected resources may be data (e.g. pictures, documents), actions (e.g. create a new thread in a forum, send an email) and, in general, any URL with access restrictions.

Plenty of information about OAuth can be found in its official Web site (OAuth Web site, 2008), and therefore here we will just give a brief summary. The operation of OAuth, depicted in Fig. 2, works on an architecture made up by three main actors:

- ***Service Provider***: Web application that allows the access to protected resources via OAuth.
- ***User***: person with an account at the Service Provider.
- ***Consumer***: application that uses OAuth to access the Service Provider on behalf of the User.

A typical example of the use of OAuth, provided by the specification itself (OAuth specification, 2009), takes place when a person (User) tells a printer Web site (Consumer) to print a photo stored on another site (Service Provider). Firstly, the user signs into the printer web site and places an order for prints. Secondly, the printer Web site asks the name of the site where her photos are stored. Thirdly, the printer Web site redirects the user to the photo site, where he/she signs into his/her account and is asked if he/she really wants to share the photos with the printer. If he/she agrees, he/she is sent back to the printer site which can now access the photos. At no point the user shares his/her username and password with the printer site.

The internal operation of OAuth is based on the use of single-use tickets, or ''tokens'' in OAuth's terminology. For these tokens to be valid for accessing the resource, they must be formerly authorized by the User. OAuth differentiates two kinds of tokens. Access Tokens allow their owner to get the protected resource. Request Tokens, on the other hand, allow their owner to get an Access Token.

One peculiarity of OAuth is its ''all or nothing approach''. OAuth can be used to grant full access to all the resources hosted at the Service Provider. The latter can restrict the access priviledges of the Consumer by self-initiative (e.g. allowing read-only access to some resources), but it is not part of OAuth itself.

3.2. *Delegation Permits*

One kind of Web application that has been proliferating during the last few years are mashups (Yan, 2007). Mashups allow to collect up data from different data sources, and combining them in completely new ways. The problem arises when data require access control. For example, we could think of a mashup that accesses the user's personal photos stored at a photo hosting site, and places them in the maps provided by Google Maps. In this case some access control is needed at the photo hosting site, as the mashup should not access user's data without his/her permission.

Delegation Permits (Hasan et al., 2008) deals with the problem of granting authorizations to mashups in order to access protected resources. Their authors define it as ''*a scalable, stateless delegated authorization protocol*''. However, despite the fact that Delegation Permits was originally thought to grant authorizations to mashups, its operation can be generalized to other kinds of systems. The most remarkable differences between Delegation Permits and OAuth involve the actors and messages that take part in the protocol. Basically, Delegation Permits works over the architecture summarized in Fig. 3, where five main entities are identified:

- *Mashup*: application that wants to access the Back-end Service.

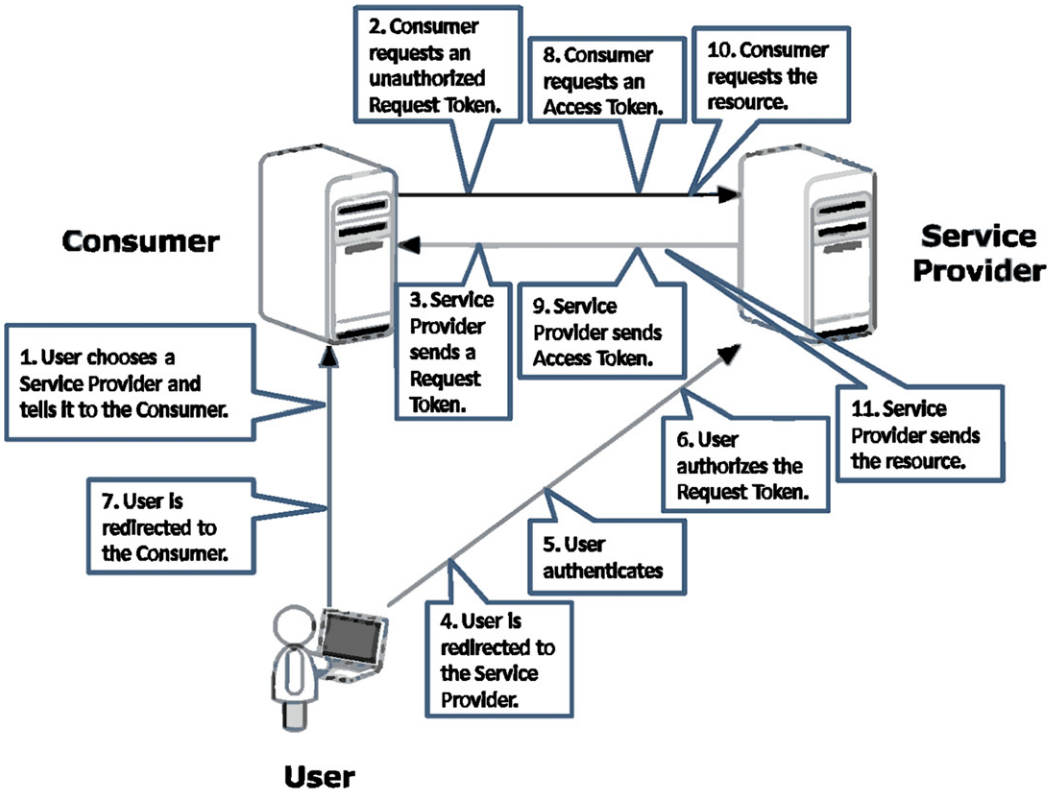

Consumer
Service Provider
User
1. User chooses a Service Provider and tells it to the Consumer.
2. Consumer requests an unauthorized Request Token.
3. Service Provider sends a Request Token.
4. User is redirected to the Service Provider.
5. User authenticates
6. User authorizes the Request Token.
7. User is redirected to the Consumer.
8. Consumer requests an Access Token.
9. Service Provider sends Access Token.
10. Consumer requests the resource.
11. Service Provider sends the resource.

Fig. 2 – Message exchange in OAuth.

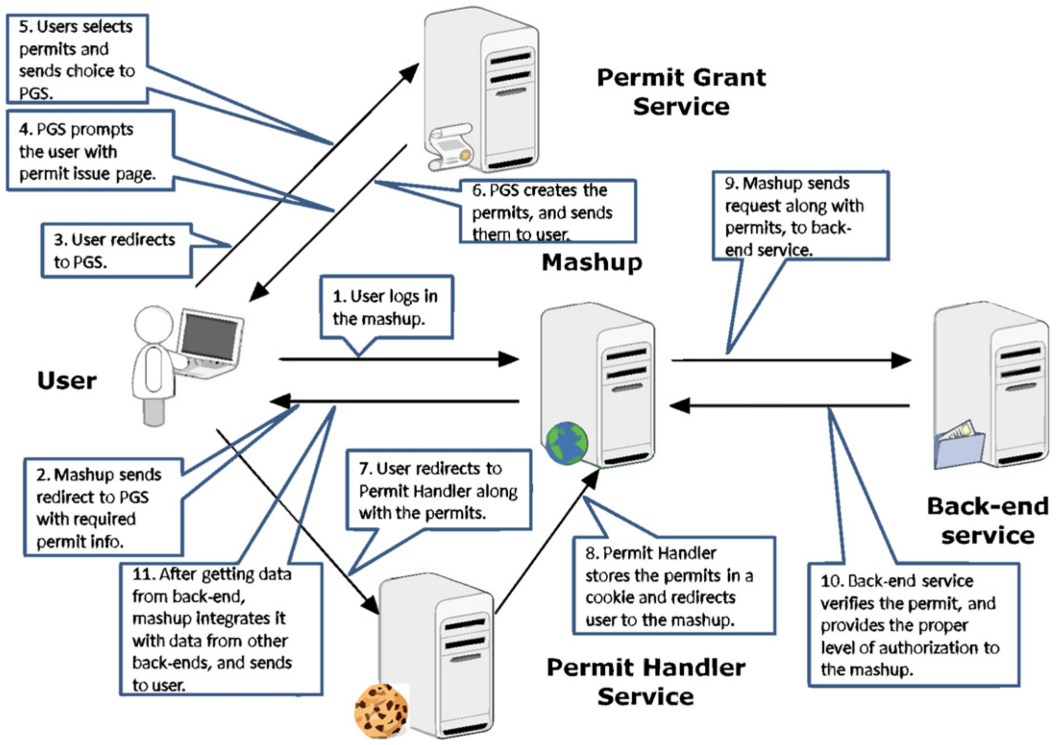

5. Users selects permits and sends choice to PGS.
4. PGS prompts the user with permit issue page.
3. User redirects to PGS.
Permit Grant Service
6. PGS creates the permits, and sends them to user.
9. Mashup sends request along with permits, to back-end service.
Mashup
1. User logs in the mashup.
User
Back-end service
2. Mashup sends redirect to PGS with required permit info.
7. User redirects to Permit Handler along with the permits.
8. Permit Handler stores the permits in a cookie and redirects user to the mashup.
10. Back-end service verifies the permit, and provides the proper level of authorization to the mashup.
11. After getting data from back-end, mashup integrates it with data from other back-ends, and sends to user.
Permit Handler Service

Fig. 3 – Message exchange in Delegation Permits (figure adapted from Hasan et al. (2008)).

- ***User***: user of both the Mashup and the Back-end Service.
- ***Back-end Service***: application that hosts the data the Mashup wants to access.
- ***Permit Grant Service***: application that deals with the expedition of authorizations to access the Back-end Service.
- ***Permit Handler Service***: application running at the Mashup, responsible for managing the authorizations once they are received from the Permit Granting Service.

The main architectural difference with OAuth is that in Delegation Permits five entities are involved instead of only three. The entities of OAuth have a rough equivalent in Delegation Permits (User–End User, Consumer–Mashup, Service Provider–Back-end Service), but there are two more actors: the Permit Handler Service, and the Permit Grant Service. While the Permit Handler Service only provides a further optimization of the performance of the protocol by the use of cookies (and hence is an expendable entity), the Permit Grant Service supplies the core functionality of the protocol. It provides a centralized environment where the End-User can grant or deny the authorization to the Mashup. However, the Permit Grant Service is a single point of failure and potential bottleneck, which may not scale well.

3.3. *Shibboleth*

The Shibboleth project (Shibboleth Web site, 2009) is the main effort carried out by the Internet2 consortium (Internet2 home page, 2009) to bring a middleware addressing issues in authentication and authorization, in order to make secure inter-institutional services possible and practical. The word *shibboleth* itself has historical association with linguistic passwords that identify people as members of a group. Ever since the release of the first version in 2000 more than 60 organizations use Shibboleth, including Napster and Elsevier's ScienceDirect (Shibboleth enabled applications and services, 2009).

The primary function of the Shibboleth system is to support transparent access to the resources of multiple sites among which there is a trust relationship, which is known as a *federation*. Shibboleth pays special attention to enabling independent organizations to federate in order to extend their capabilities. The working of Shibboleth is based on SAML 2.0 (SAML Web site, 2009), an XML-based standard for exchanging security information between partner domains. The information is encoded as attributes, pairs key-value containing information about the User or its role at his/her home institution (e.g. ''branch–telematics'', ''profession–teacher''). SAML has been developed by the OASIS standards working group (OASIS Web site, 2009).

Shibboleth works in an architecture made up by four actors (see Fig. 4):

- ***Identity Provider***: software running at the home institution of the User within the federation. The Identity Provider is responsible for authentication to access the protected resources at the different Service Providers across the federation.
- ***Service Provider***: institution of the federation that stores the protected resource the User wants access to. The Service Provider relies on the Identity Provider to verify the identity of the User.

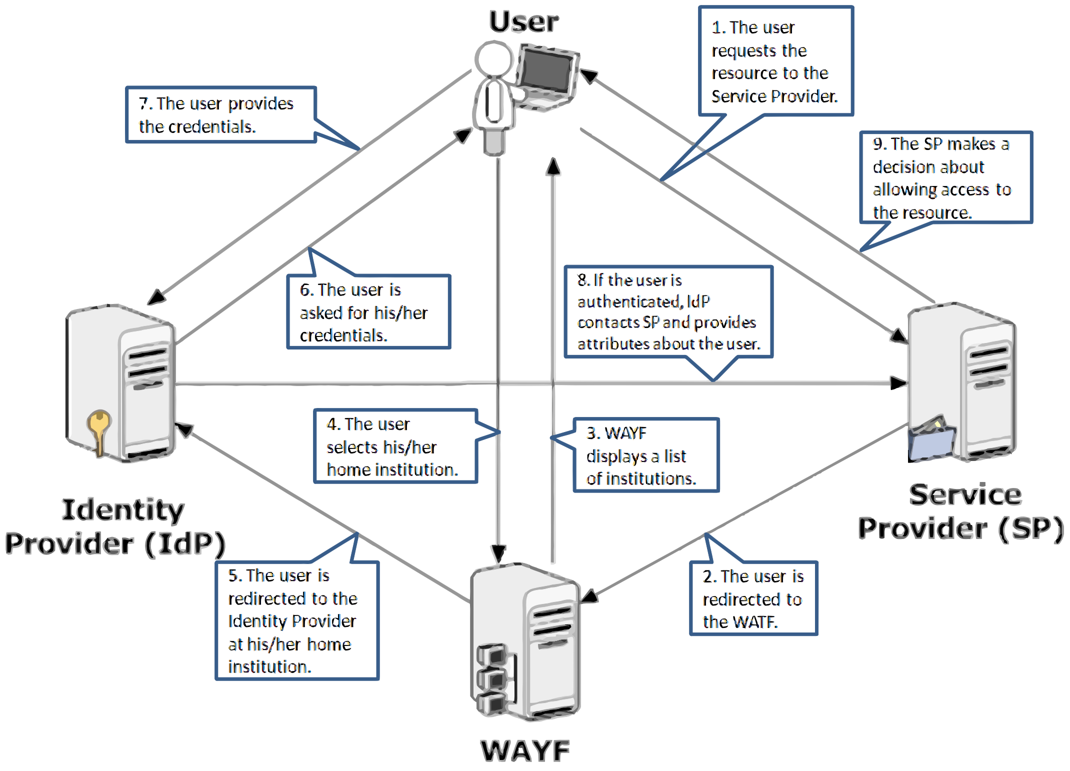

User
1. The user requests the resource to the Service Provider.
7. The user provides the credentials.
9. The SP makes a decision about allowing access to the resource.
6. The user is asked for his/her credentials.
8. If the user is authenticated, IdP contacts SP and provides attributes about the user.
4. The user selects his/her home institution.
3. WAYF displays a list of institutions.
Identity Provider (IdP)
Service Provider (SP)
5. The user is redirected to the Identity Provider at his/her home institution.
2. The user is redirected to the WATF.
WAYF

Fig. 4 – Message exchange in Shibboleth.

- *User*: person with a user account at the Identity Provider and that wants to access a protected resource at the Service Provider.
- *WAYF (Where Are You From)*: centralized service of the federation that allows the User to choose his/her home institution.

When the User wants to access the protected resource, he/she navigates to its URL using his/her Web browser. The Service Provider redirects the browser to the WAYF, where the User can select his/her home organization. The browser is sent to the home organization's Web site running a Shibboleth Identity Provider. The User now sees the login of his/her home organization, where he/she enters the username and password. The Identity Provider sends the browser back to the Service Provider.

The User has been recognized by the Identity Provider as one of its users, and the Service Provider has been notified of this. The process is complete when the Service Provider requests additional information (attributes) about the User to the Identity Provider. These attributes do not necessary involve User's identity, although it could be requested as well. The Service Provider uses these attributes to decide the access policy for the User (e.g. the functionalities granted, the length of the session).

The previous process can be decoupled into two independent stages, depending on the results obtained at each one. On the one hand, the first stage ends when the User has logged in at his/her home institution by means of the WAYF. At this step, the Shibboleth system has achieved a delegated *authentication* of the User. On the other hand, the second stage involves the request and sending of attributes of the User to determine what he/she is allowed to do. Hence, the second stage provides delegated *authorizations*.

An important missing part in Shibboleth is the complementary process of single sign-on, generally known as *single log-out* (The difficulties of single sign on, 2009). At the time of this writing not only single log-out is not addressed by Shibboleth, but the requirements are not even clear. There are two possible behaviours expected when the User clicks the Log-out button: logging out from the concrete Service Provider, or logging out from all the Service Providers of the federation. Each approach has its pros and its cons, and there is no general consensus about which one should be implemented in future versions of Shibboleth. The only way to ensure that a User's session has been terminated is for the User to close all browser tabs and windows and quit the browser. The absence of single log-out has the important consequence that it is not easy to know whether the User is still logged at the federation, or his/her session has been terminated.

3.4. *OpenSSO*

OpenSSO (OpenSSO Web site, 2009) is the Sun Microsystem's (Sun Microsystems Web site, 2009) approach to the problems of managing federations with external partners and securely federating legacy applications that do not support federation. To do so, OpenSSO provides a middleware consisting of core identity services for the implementation of single sign-on in heterogeneous environments.

As Shibboleth, OpenSSO is based on the request and sending of attributes concerning the user between an Identity Provider and a Service Provider by the use of SAML 2.0. In

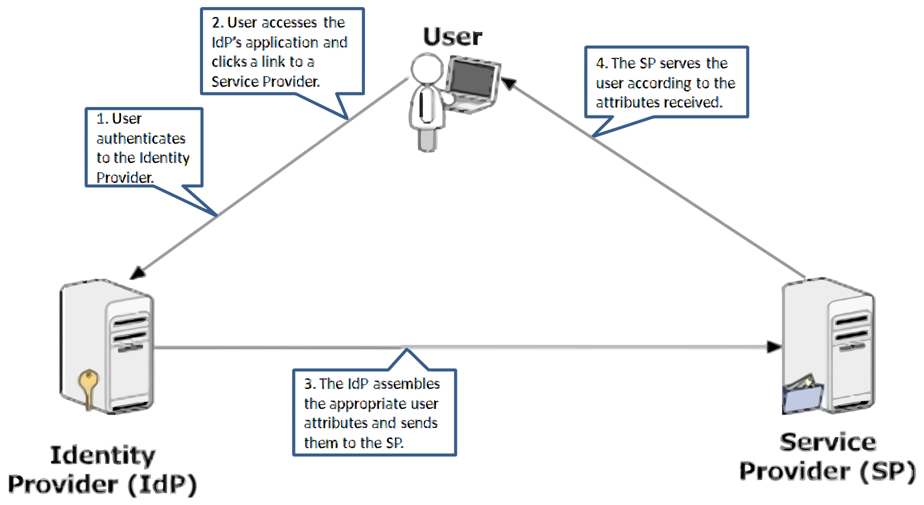

User
2. User accesses the IdP's application and clicks a link to a Service Provider.
1. User authenticates to the Identity Provider.
4. The SP serves the user according to the attributes received.
3. The IdP assembles the appropriate user attributes and sends them to the SP.
Identity Provider (IdP)
Service Provider (SP)

Fig. 5 – Message exchange in OpenSSO.

general terms OpenSSO deals with the adaptation of identity information transfers from the legacy application into SAML messages, that can be understood and processed by all the applications of a federation.

A summarized description of the operation of OpenSSO is shown in Fig. 5, where three actors can be seen:

- ***Identity Provider***: home institution of the User within the federation. The Identity Provider runs an OpenSSO instance that encapsulates information about the User (e.g. authentication, profile attributes) and sends them to the Service Provider.
- ***Service Provider***: institution of the same federation of the Identity Provider running another OpenSSO instance. The Service Provider receives the SAML identity information about the User, checks it, and uses it to grant the User access to protected resources according to an access policy.
- ***User***: person with a user account at the Identity Provider, who wants to access a protected resource at the Service Provider.

The process is very similar to Shibboleth with the difference that there is no equivalent to the WAYF. A User authenticates to his/her Identity Provider application. Then, he/she clicks a link that points to a service provided by another application in a different domain (Service Provider's). The Identity Provider's OpenSSO instance assembles the appropriate SAML user attributes (authentication and user data), and encodes, signs and sends them to the Service Provider using SSL. The Service Provider's local OpenSSO instance decodes the attributes and sends them to the Service Provider's application, which compares the attributes against some access policy. Finally, the application serves the user according to such policy.

3.5. *Comparative analysis*

Although it is clear from previous sections that both OAuth, Delegation Permits, Shibboleth and OpenSSO deal with the problem of granting authorizations to access third-party systems, their scope is slightly different from ours. In this section we analyze these four technologies against the requirements of Section 2.3, and point out some considerations. One of them (the non-fulfillment of Requirement 4) is especially important as it immediately discards the use of OAuth and Delegation Permits for our scenario.

Table 1 shows a comparison of these technologies against our requirements. In order to indicate the fulfillment or not of each requirement we use the symbols "✓" (the technology

Table 1 – Comparison between OAuth, Delegation Permits, Shibboleth and OpenSSO against the requirements.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
OAuth	✓	✓	✓	×	×	✓	×	×	×	×	×	✓	✓	✓
DP	✓	✓	✓	×	✓	×	×	✓	×	×	×	×	×	✓
Shibboleth	✓	×	×	✓	✓	×	×	✓	×	✓	✓	✓	✓	✓
OpenSSO	✓	✓	×	✓	✓	×	×	✓	×	✓	✓	✓	✓	✓

fulfills the requirement) and "×" (the technology does not fulfill the requirement, or fulfills it poorly). According to this table it is important to notice the following issues:

- The lack of a single log-out mechanism in Shibboleth implies that it is not easy to know whether or not the User is still logged in the federation. Consequently the User has to authenticate himself/herself again every time he/she wants to access a protected resource. Hence, Shibboleth does not satisfy Requirement 2 (Access Transparency). As for OAuth and Delegation Permits, they do not require additional authentication processes to access the protected resource, thus satisfying the requirement.
- One of the most promoted features of the Shibboleth system is the anonymity of its users. This is based on the sending of attributes regarding the users, rather than their real identities. However, the concrete attributes sent are not limited by any means by the Shibboleth specification, but instead are part of the agreement of the federation. It could be possible that the attributes referred to personal data of the users. Hence, although Shibboleth provides support to keep users anonymous, their privacy is not guaranteed. The same issue appears in OpenSSO, as it follows the same attribute-based approach to single sign-on. OAuth and Delegation Permits, on the other hand, do not need any sensitive information to grant transparent access to their users. This implies that OAuth and Delegation Permits satisfy Requirement 3 (Privacy) but Shibboleth and OpenSSO do not.
- As mentioned, the most important aspect to be noticed is that the differences between the architecture described in Section 2.1 and those assumed in OAuth and Delegation Permits automatically discard the direct use of these technologies. Indeed, in these technologies a person authorizes a service to access another service. In our case, however, we want a service to authorize a person to access another service. This fact has the important implication that in both OAuth and Delegation Permits the person is given no freedom to choose a Web tool. OAuth does not consider the case where the Consumer tells (actively) the User which tool he/she wants to use. OAuth specification forces the User to tell the Consumer the tool that will be accessed. The case of Delegation Permits is similar: the purpose of mashups is the combination of information gathered from predefined Back-end Services, which is later presented to the user in a brand-new fashion. Since Delegation Permits has been designed for mashups, there is no point in allowing the User to choose which Back-end Service the Mashup accesses. On the other hand Shibboleth and OpenSSO fit better in our business model, as they have been thought to provide an infrastructure to allow (human) users to access to the resources of a federation. This implies that unlike Shibboleth and OpenSSO, both OAuth and Delegation Permit go against Requirement 4 (Choosability).
- OAuth does not provide any means to access specific protected resources, but instead follows an "all or nothing" approach, by means of which a Consumer can gain access to every protected resource hosted at the Service Provider. Although this approach may be interesting in some scenarios, in our system we need students to access specific resources. Delegation Permits, Shibboleth and OpenSSO, on the other hand, do consider the access to specific resources with specific permissions. Hence these technologies satisfy Requirement 5 (Granularity), but OAuth does not.
- Delegation Permits does not fulfill Requirement 6 (Simplicity), as it involves five entities instead of only the user, the LMS and the tool. On top of that, the Permit Grant Service is a complex service which provides the core functionality of Delegation Permits, and hence supposes a potential bottleneck and single point of failure. Similarly, the correct working of a federation running Shibboleth relies on a single WAYF service. The mandatory use of SSL in OpenSSO implies extra traffic and computational load, and a heavy infrastructure to work, and makes this technology fail to fulfill Requirement 6. On the contrary OAuth keeps a very simple architecture as it only involves these three entities, whose computational load is well balanced between User, Consumer and Service Provider. Thus, OAuth fulfills Requirement 6, but not Delegation Permits, Shibboleth and OpenSSO.
- Requirement 7 (Dynamic Reconfiguration) is not supported by any of the protocols under analysis. The reason is that these protocols have not been designed for scenarios where the authorizations are tied to plannings that may be prone to change, which is typically the case in e-learning environments. Among the four protocols, Shibboleth and OpenSSO are the ones that provide the highest level of versatility when configuring the properties of a delegated authorization, due to its attribute-based approach. However, both Shibboleth, OpenSSO, OAuth and Delegation Permits carry out the configuration process only at the beginning when the authorization is being negotiated, without any possibilities to modify the properties of an ongoing authorization.
- Delegation Permits includes as a parameter the expiration time of the authorization. Similarly, the expiration time can be sent as an attribute in Shibboleth and OpenSSO. However, in the case of OAuth the expiration time is imposed by the Service Provider and cannot be established or controlled by the User. Hence Delegation Permits, Shibboleth and OpenSSO satisfy Requirement 8 (Expiry), but not OAuth.
- Given that OAuth and Delegation Permits are based on a different business model, the provision of feedback from one software system (the Web tool) to another (the LMS) has not been considered. In addition Shibboleth and OpenSSO, even posing a similar business model, do not consider any communication between the Identity Provider and the different Service Providers after the initial negotiation phase, which excludes any possibility of sending any feedback. Consequently OAuth, Delegation Permits, Shibboleth and OpenSSO do not fulfill Requirement 9 (Awareness).
- Neither OAuth nor Delegation Permits provide any means by which a tool can anonymously identify its students. Hence these technologies do not fulfill Requirement 10 (Pseudonimity). Shibboleth and OpenSSO, however, can support pseudonyms if they are sent as attributes, provided that they have been agreed as accepted attributes by the federation.
- Finally, OAuth provides means to verify the integrity of the messages exchanged and the authenticity of the actors involved (Requirements 12 and 13), as opposed to Delegation Permits which does not take these aspects into account. In any case, neither OAuth nor Delegation Permits provide confidentiality mechanisms (Requirement 11). As for

Shibboleth and OpenSSO, they satisfy Requirements 11, 12 and 13. However, they take advantage of the use of SSL and hence a fair comparison with OAuth and Delegation Permits (that do not require SSL) is not possible.

We can conclude that OAuth, Delegation Permits, Shibboleth and OpenSSO exhibit some desirable characteristics but they have been developed for slightly different scenarios, and therefore they do not satisfy all our requirements. The most feasible possibility is to redesign one of these technologies to take advantage of its current strengths but solving its limitations.

To that end we took OAuth as our starting point due to its openness and the strong developing community behind it (OAuth developing community, 2009), and its architectural simplicity. The result is a variant of OAuth which we named Reverse OAuth, since it is the student (Consumer) who tells the LMS (User) which tool to use, as opposite to what happens in the normal version of OAuth. The rest of this article provides a thorough description of Reverse OAuth.

4. A single sign-on solution for e-learning systems. Reverse OAuth

With Reverse OAuth we solve the abovementioned deficiencies of OAuth, Delegation Permits, Shibboleth and OpenSSO. Its operation is strongly based on OAuth, but featuring some characteristics from Delegation Permits, Shibboleth and OpenSSO.

In this section we provide an in-depth description of Reverse OAuth. In Section 4.1 we give a high-level description of the protocol, defining the actors involved and the messages they send and putting the strength on the differences between it and OAuth. Section 4.2 describes the fields and format of the messages involved. Finally, Section 4.3 studies the behaviour of Reverse OAuth against our requirements, comparing it with OAuth, Delegation Permits, Shibboleth and OpenSSO.

4.1. *General description*

We agree on using the same terminology of OAuth to emphasize the similarities of both protocols. Hence, we have:

- ***Service Provider***: Web application that hosts those protected resources that are accessed via Reverse OAuth. In our scenario, the role of Service Provider will be played by a Web tool.
- ***User***: application that has a working account at the Service Provider. In our case it will be the LMS which plays this role, having working accounts in all those Web tools offered to the students/teachers.
- ***Consumer***: person that uses Reverse OAuth to access the Service Provider on User's behalf, namely a student or a teacher of the LMS who wants to use the Web tool.

Fig. 6 describes the interaction that takes place between these three entities when the Consumer wants to access

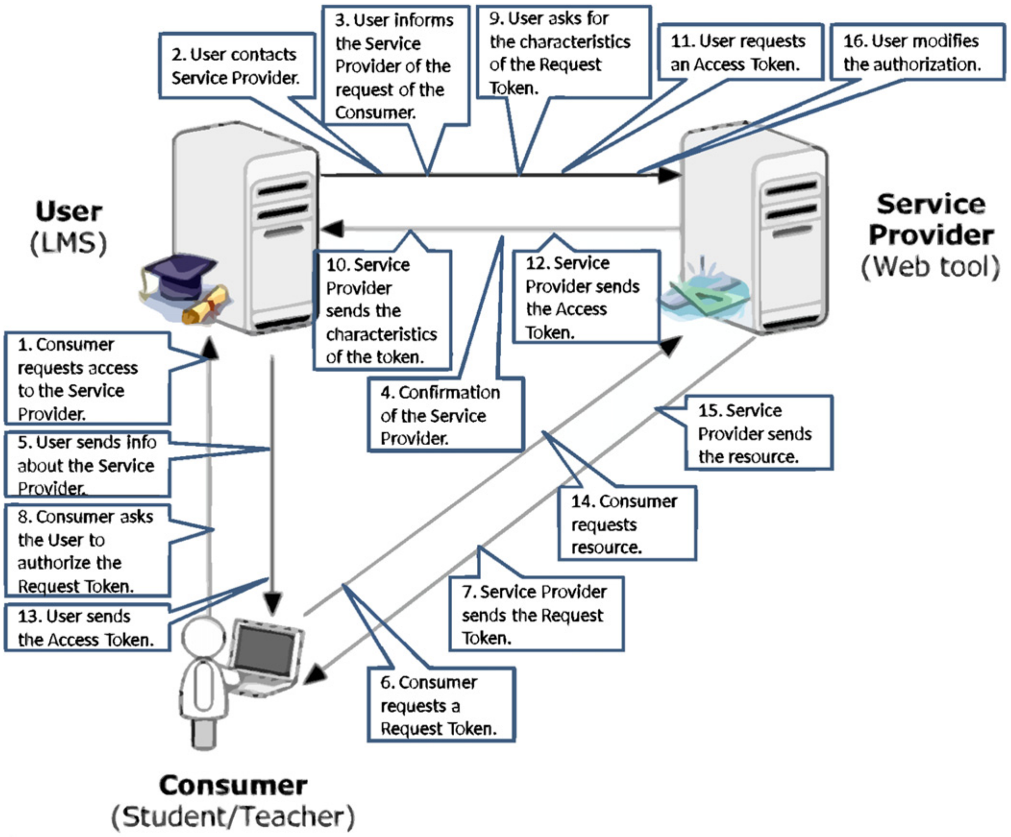

2. User contacts Service Provider.
3. User informs the Service Provider of the request of the Consumer.
9. User asks for the characteristics of the Request Token.
11. User requests an Access Token.
16. User modifies the authorization.
User (LMS)
Service Provider (Web tool)
10. Service Provider sends the characteristics of the token.
12. Service Provider sends the Access Token.
1. Consumer requests access to the Service Provider.
4. Confirmation of the Service Provider.
5. User sends info about the Service Provider.
15. Service Provider sends the resource.
14. Consumer requests resource.
8. Consumer asks the User to authorize the Request Token.
7. Service Provider sends the Request Token.
13. User sends the Access Token.
6. Consumer requests a Request Token.
Consumer (Student/Teacher)

Fig. 6 – Message exchange in Reverse OAuth.

a protected resource hosted at the Service Provider. The sequence of messages these entities send is as follows:

1. Firstly the Consumer makes a request to the User to access a protected resource hosted at the Service Provider.
2. The User contacts the Service Provider and authenticates itself.
3. The User informs the Service Provider that the Consumer will request a Request Token. With this message the Service Provider can map a Consumer with the User that grants the authorization.
4. The Service Provider sends a confirmation.
5. The User sends the Consumer information regarding the Service Provider and the desired resource. This information comprises the URL where the Consumer must make the request, the URL of the resource, the available permissions to access it (e.g. read, write, execution), the date when the authorization expires, and an alphanumeric string by which the Service Provider can identify the Consumer.
6. The Consumer contacts with the Service Provider and requests a Request Token. The message includes those parameters the User sent to the Consumer in the previous step.
7. The Service Provider sends the Request Token to the Consumer.
8. The Consumer asks the User to authorize the Request Token and trade it for an Access Token.
9. The User asks the Service Provider for the characteristics of the Request Token. The purpose of this message is that the User can check whether the characteristics of the Request Token that the Consumer requested in step 6 are the same of step 5. If this verification were not carried out it would be feasible that the Consumer changed the characteristics of the Request Token as he/she pleased.
10. The Service Provider sends the characteristics of the Request Token to the User. The User carries out the checking described in the previous step. In case it were not successful the protocol halts, which implicitly denies the Consumer the access to the protected resource. If successful, the protocol continues to the next step.
11. The User requests an Access Token from the Service Provider.
12. The Service Provider sends an Access Token to the User, which will be later used by the Consumer to request the protected resource.
13. The User sends the Access Token to the Consumer.
14. The Consumer shows the Access Token to the Service Provider.
15. The Service Provider checks the validity of the Access Token. If valid, it sends the resource.
16. If necessary, the User sends a message to the Service Provider to modify the characteristics of the authorization that has been granted to the Consumer.

4.2. *Message description*

All the messages that take place in Reverse OAuth are HTTP messages, whose parameters are encoded with the POST method. The parameters are case-sensitive, and their name must begin with the string ‘‘roauth_’’.

The Service Provider must supply documentation including the URLs where User and Consumer have to sent their requests, as well as the supported signature methods and Reverse OAuth versions. It is the Service Provider’s responsibility to bring the User a pair User Key–User Secret. This pair is used by the User to authenticate itself against the Service Provider.

Authentication is possible due to the use of digital signatures. The User uses the User Secret to sign the messages it sends to the Service Provider. Using a specific signature algorithm would imply a security breach in Reverse OAuth when the algorithm is broken. Hence, Reverse OAuth is agnostic with regards to the signature algorithm, allowing its use with solutions such as MD5, SHA-1, RSA, DSA, etc.

The concrete fields of each message of Reverse OAuth are summarized in Table 3.

4.3. *Fulfillment of the requirements with Reverse OAuth*

From the previous description some differences can be appreciated between the technologies of Section 3 on the one hand, and Reverse OAuth on the other hand. As described

Table 2 – Fulfillment of the requirements by Reverse OAuth.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
OAuth	✓	✓	✓	×	×	✓	×	×	×	×	×	✓	✓	✓
DP	✓	✓	✓	×	✓	×	×	✓	×	×	×	×	×	✓
Shibboleth	✓	×	×	✓	✓	×	×	✓	×	✓	✓	✓	✓	✓
OpenSSO	✓	✓	×	✓	✓	×	×	✓	×	✓	✓	✓	✓	✓
R. OAuth	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓

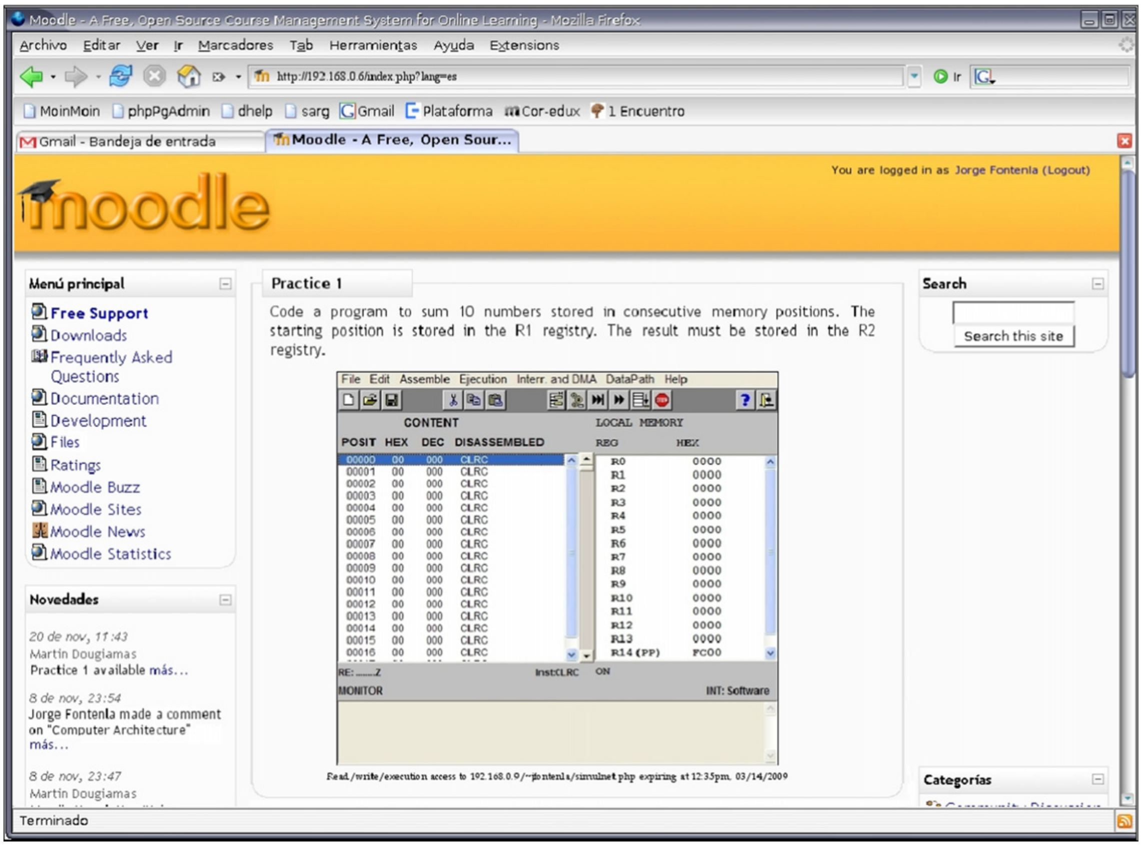
Table 3 – Message summary. The fields marked with the "*" symbol are optional.

Field name	Messages	Description
roauth_accept	13	Acceptance or not of the Request Token by the Service Provider.
roauth_access_mode	5, 6, 16	Permissions over the protected resource.
roauth_access_token	12, 13, 14	Alphanumeric string that allows its owner to request the protected resource.
roauth_action	3, 9, 11, 16	Action the recipient of the message has to carry out with some of the fields.
roauth_consumer_identity	3, 5, 6, 8, 14	Pseudonym of the Consumer.
roauth_consumer_ip	10	IP address of the machine that requests a Request Token to the Service Provider (namely, the Consumer's IP).
roauth_control_channel	12, 16	Identifier of the channel that allows the User to dynamically reconfigure the authorization.
roauth_error*	4, 10, 12	Error code, if any.
roauth_expiration_time	5, 6, 10, 16	The expiration time of the authorization.
roauth_nonce	5, 6, 9, 16	One-use number.
roauth_request_token	7, 8, 9, 11	Alphanumeric string that allows its owner to request an Access Token.
roauth_requested_URL	5, 10	The URL of the protected resource the Consumer wants to get access to.
roauth_service_provider_URL	5	The URL where the Consumer can request a Request Token.
roauth_signature	3, 9, 11, 16	Signature of the message.
roauth_signature_method	3, 9, 11, 16	The algorithm employed to generate the signature.
roauth_timestamp	5, 6, 9, 16	Temporal mark of the message.
roauth_user_key	3, 9, 11, 16	The username of the User at the Service Provider.
roauth_user_URL	6	The URL of the User.
roauth_version*	3, 4, 5, 6, 9, 10, 11, 12, 16	Version of Reverse OAuth, for compatibility purposes.

below these differences are the cause of the better behaviour of Reverse OAuth against our requirements, summarized in Table 2. However, all of them arise from the fact that in Reverse OAuth the Consumer is a person (namely a student or a teacher) and the User a software system (typically the LMS). As a consequence, the rest of the differences follows. Notice the following issues:

- In Reverse OAuth the Service Provider is chosen by the Consumer (the student) and not by the User (the LMS). The Consumer has the ability to choose a Service Provider according to his/her needs and preferences. In other words, Reverse OAuth fulfills Requirement 4 (Choosability).
- Reverse OAuth allows one to specify the concrete resource the student is granted access to (field roauth_requested_URL). Hence, Requirement 5 (Granularity) is fulfilled.
- The architectural design of Reverse OAuth keeps the simplicity of OAuth, involving just three actors. Consequently, Reverse OAuth fulfills Requirement 6 (Simplicity).
- None of the technologies of Section 3 fulfill Requirement 7 (Dynamic Reconfiguration). Once the authorization has been granted there is no way it can be modified or revoked. In Reverse OAuth the User can modify the characteristics of the session of the Consumer at the Service Provider in runtime (step 16 of the protocol) in order to adapt them to the current needs.
- Reverse OAuth, like Delegation Permits, allows to specify the validity period of an authorization (field roauth_expiration_time). Hence, Requirement 8 (Expiry) is fulfilled.
- Neither OAuth nor Delegation Permits nor Shibboleth provide any mechanism to send feedback from the Web tool to the LMS. Reverse OAuth includes information in its messages (fields roauth_consumer_id-entity and roauth_control_channel) that can be used by the User to track the activities of the Consumer at the Service Provider. This implies the fulfillment of Requirement 9 (Awareness).
- In step 3 of Reverse OAuth, the student identifies himself/herself against the tool by means of a pseudonym (field roauth_consumer_identity). This pseudonym is used during the rest of the protocol instead of the real identity of the student. Hence, Reverse OAuth fulfill Requirement 10 (Pseudonimity).
- The use of signatures in the messages of Reverse OAuth (field roauth_signature) allows one to detect both modifications in the messages and impersonation of the actors involved in the protocol. Hence, Requirements 12 and 13 are fulfilled.
- Once an authorization has expired (or the User has revoked it), the corresponding Request Token and Access Token is no longer valid, which ensures that authorizations are single-use. An attacker could try to reuse these expired tokens to predict the behaviour of the token generation process at the Service Provider, and obtain fake tokens. However, Reverse OAuth does not require a specific algorithm to generate Request Tokens and Access Tokens. Hence, Reverse OAuth is prone to using high entropy processes in order to ensure unpredictibility in the generation of authorizations. All these facts together ensure that single-use authorizations are considered, and secure to use. Hence, Reverse OAuth fulfills Requirement 14 (Single-use Authorizations).

It is worth mentioning that Reverse OAuth does not incorporate any mechanisms to ensure the confidentiality of the messages exchanged (non-fulfillment of Requirement 11). The reason is that the provision of confidentiality would require a complex architecture for key and certificate distribution, which would increase dramatically the complexity of the protocol. Hence we have decided to provide a protocol which is agnostic in confidentiality technologies. However, Reverse OAuth can be used in combination with other technologies such as TLS if confidentiality were required.


Moodle - A Free, Open Source Course Management System for Online Learning - Mozilla Firefox
Archivo Editar Ver Ir Marcadores Tab Herramientas Ayuda Extensions
http://192.168.0.6/index.php?lang=es
MoinMoin phpPgAdmin dhelp sarg Gmail Plataforma Cor-edux 1 Encuentro
Gmail - Bandeja de entrada
Moodle - A Free, Open Sour...
moodle
Menú principal
Free Support
Downloads
Frequently Asked Questions
Documentation
Development
Files
Ratings
Moodle Buzz
Moodle Sites
Moodle News
Moodle Statistics
Novedades
Search
Search this site
Categorías
Practice 1
File Edit Assemble Ejecution Interr. and DMA DataPath Help
Terminado

You are logged in as Jorge Fontenla (Logout)

Code a program to sum 10 numbers stored in consecutive memory positions. The starting position is stored in the R1 registry. The result must be stored in the R2 registry.

```
              CONTENT                      LOCAL  MEMORY
POSIT  HEX  DEC  DISASSEMBLED              REG         HEX
00000  00   000  CLRC                      R0          0000
00001  00   000  CLRC                      R1          0000
00002  00   000  CLRC                      R2          0000
00003  00   000  CLRC                      R3          0000
00004  00   000  CLRC                      R4          0000
00005  00   000  CLRC                      R5          0000
00006  00   000  CLRC                      R6          0000
00007  00   000  CLRC                      R7          0000
00008  00   000  CLRC                      R8          0000
00009  00   000  CLRC                      R9          0000
00010  00   000  CLRC                      R10         0000
00011  00   000  CLRC                      R11         0000
00012  00   000  CLRC                      R12         0000
00013  00   000  CLRC                      R13         0000
00014  00   000  CLRC                      R14 (PP)    FC00
00015  00   000  CLRC
00016  00   000  CLRC
RE: .......Z                  Inst:CLRC    ON
MONITOR                                    INT: Software
```

Read/write/execution access to 192.168.0.9/~jfontenla/simulnet.php expiring at 12:35pm. 03/14/2009

20 de nov, 11:43
Martin Dougiamas
Practice 1 available más...

8 de nov, 23:54
Jorge Fontenla made a comment on "Computer Architecture" más...

8 de nov, 23:47
Martin Dougiamas

Fig. 7 – Access to SimulNet via Reverse OAuth.

5. Proof of concept

To demonstrate the feasibility of our approach we built a prototype in our corporative network consisting on an existing Web-based application called SimulNet (Anido et al., 2001), used in the University of Vigo in the subject *Computer Architecture*. SimulNet provides a virtual laboratory environment where students can put into practice their knowledge about low-level programming. Among its features we can find code edition, compiling and debugging.

We adapted SimulNet to support a basic login–password authentication mechanism which grants access to the full functionalities of the tool. The credentials are sent as plaintext. For working purposes we created an only working account at SimulNet, which was assigned to the LMS.

Acting as an LMS we deployed an instance of Moodle at a different subnet from SimulNet's. The choice of Moodle responds to its important presence in online universities, as well as to its opensourceness that allowed us to make the required modifications in its source code.

When a student directs his/her browser to the URL where Moodle is running he/she is displayed a Web page where he/she can authenticate himself/herself. After this authentication stage the main page of the course is displayed to the student. We created a subject named ''Computer Architecture'', and a task for this subject named ''Practice 1''. When the students goes to the page of Practice 1 he/she is provided a hyperlink to SimulNet. At the moment the student clicks on the link the process described in Section 4 starts.

The result is showed in Fig. 7. The steps of Reverse OAuth are mostly transparent to the student. However, when he/she gains access to the tool Moodle provides him/her the details of the authorization. In our case, the student is remembered by means of a verbatim message (placed under SimulNet's frame) that he/she will enjoy read, write and execution permissions over SimulNet (located at the URL 192.168.0.9/~jfontenla/simulnet.php) until 12:35 pm, 03/14/2009 (the URL of the protected resource, expiration time, permissions, etc.).

The usability of the design was evaluated by several members of the staff of the University of Vigo, some of whom with affiliation in the field of software design and some others with a main e-learning background. All of them found the prototype usable and appropriate.

6. Conclusions

Current LMSs are playing a key role at giving access to learning resources, avoiding spatial and temporal barriers. However, their possibilities are limited by their growing complexity and the ''one size does not fit all'' problem. These limitations suggest splitting LMSs themselves from learning tools. This approach not only implies the design of new e-learning systems, but an entirely new business model where the development of LMSs and learning tools can go separate (although complementary) ways. The need of single sign-on in such scenario has been our starting point. Reverse OAuth provides a solution to this problem by means of a delegated authorization mechanism.

It has been the focus of our efforts to achieve a lightweight, open and interoperable protocol. Reverse OAuth is agnostic about programming languages as it is based in a ''neutral'' protocol like HTTP. Hence, our implementation of Reverse OAuth in Moodle can be easily ported to other LMSs.

Reverse OAuth is not meant to be exclusive of the e-learning domain. Other scenarios where the granting of authorizations is useful are possible, but Reverse OAuth has been designed with e-learning specific requirements in mind. Hence it may not adapt well to such scenarios without prior revision. However, it is an open and freely available protocol, and can be modified as necessary. Reverse OAuth itself is an example of adaptation of a previously existing protocol (OAuth) to a domain with specific requirements.

We believe that Reverse OAuth is a useful model to achieve single sign-on in e-learning systems and, given the wide-spread use of Learning Management Systems, we expect its progressive adoption by the research community.

Acknowledgements

This work has been funded by the Spanish Ministerio de Educación y Ciencia under grant TIN2007-68125-C02-02, and by the Galician Consellería de Innovación e Industria under grant PGIDIT06PXIB32 2270PR.

REFERENCES

Anido L, Llamas M, Caeiro M, Santos J, Rodriguez J, Fernandez MJ. An update on the SimulNet educational platform. Towards standards-driven e-learning. IEEE Transactions on Education 2001;44(2).

Fontenla J, Caeiro M, Llamas M. Towards a generalized architecture for the integration of tools in LMSs. In: International symposium on information and communication and technologies applied to education (SIIE); 2008.

Hasan R, Conlan R, Slesinsky B, Ramani N, Winslett M. Please permit me: stateless delegated authorization in mashups. In: Annual computer security applications conference (ACSAC); 2008.

IMS Tools Interoperability Specification. Last accessed on January, 2009 at http://www.imsglobal.org/ti/index.html.

Internet2 home page. Last accessed on June, 2009 at http://www.internet2.edu.

Introduction to single sign-on. Last accessed on June, 2009 at http://www.opengroup.org/security/sso/sso_intro.htm.

Klobucar T. Requirements collection and analysis with focus on privacy and security issues. Prolearn: European Commission Sixth Framework Project (IST-507310). Last accessed on March, 2008 at: http://isotc.iso.org/livelink/livelink?func=ll&objId=5906025&objAction=browse.

NetVibes Web site. Last accessed in June, 2008 at http://www.netvibes.com.

OASIS Web site. Last accessed on June, 2009 at http://www.oasis-open.org/home/index.php.

OAuth developing community. Last accessed on June, 2009 at: http://wiki.oauth.net.

OAuth first summit. Last accessed on June, 2009 at http://www.hueniverse.com/hueniverse/2008/07/i-can-had-open.html.

OAuth specification. Last accessed on June, 2009 at http://oauth.net/core/1.0.

OAuth Web site. Last accessed in June, 2008 at http://oauth.net.

OpenID Web site. Last accessed in June, 2009 at http://openid.net.

OpenSSO Web site. Last accessed on June, 2009 at: https://opensso.dev.java.net.

SAML Web site. Last accessed on June, 2009 at http://saml.xml.org.

Shibboleth enabled applications and services. Last accessed on June, 2009 at https://spaces.internet2.edu/display/SHIB2/ShibEnabled.

Shibboleth Web site. Last accessed on June, 2009 at http://shibboleth.internet2.edu.

Sun Microsystems Web site. Last accessed on June, 2009 at: http://www.sun.com.

The difficulties of single sign on. Last accessed on June, 2009 at https://spaces.internet2.edu/display/SHIB2/SLOIssues.

Vogten H, Martens H, Nadolski R, Tattersall C, van Rosmalen P, Koper R. CopperCore service integration – integrating IMS learning design and IMS question and test interoperability. In: Proceedings of the sixth international conference on advanced learning technologies (ICALT'06); 2006.

Web site of the Blackboard project. Last accessed on June, 2009 at http://www.blackboard.com/us/index.bbb.

Web site of the dotLRN project. Last accessed on June, 2009 at http://dotlrn.org.

Web site of the Moodle project. Last accessed on June, 2009 at http://moodle.org.

Web site of the Sakai project. Last accessed on June, 2009 at http://sakaiproject.org.

Yan N. Build your mashup with web services. In: IEEE international conference on web services; 2007.

Jorge Fontenla is a Telecommunications Engineer and a PhD Student in the University of Vigo. He is currently Assistant Teacher at the Department of Telematic Engineering, University of Vigo.

Manuel Caeiro received his PhD in Telecommunications Engineering from the University of Vigo in 2007. He is currently Assistant Teacher at the Department of Telematic Engineering, University of Vigo. He has received several awards by the W3C, NAE CASEE new faculty fellows and the IEEE Spanish Chapter of the Education Society.

Martín Llamas received his Engineering degree (1986) and his PhD degree (1994) from the Polytechnic University of Madrid. From 1994 to 1997 he was Vice Dean of the Higher Technical School of Telecommunication Engineers. From 1999 to 2003 he was the head of the ICT Area of the University of Vigo. He is member of ACM, IEEE and IFIP WG3.6 (Distance Education). He has received several awards by the W3C and IEEE.

Luis Anido Rifón has a Telecommunication Engineering degree with honours (1997) in the Telematics and Communication branches and a Telecommunication Engineering PhD with honours (2001) by the University of Vigo. Currently, he is Associate Professor in the Telematics Engineering Department of the University of Vigo. He has received several awards by the W3C, the Royal Academy of Sciences and the Official Spanish Telecommunication Association.