Authentication and Authorisation for Widget-based Applications in a Loosely-coupled Web eLearning Environment[*]

Jean-Noël Colin and Minh Tien Hoang
PReCISE Research Center, University of Namur, Namur, Belgium

Keywords: Widgets, eLearning, Authentication, oAuth, OpenId.

Abstract: In this paper, we discuss the mechanisms used for authentication and authorisation of users in a widget-based web environment that integrates multiple components from multiple providers running at multiple locations. Users are typically teachers and pupils who access our platform either directly or through their usual Learning Management System (LMS). Although the technologies used themselves are not new, we believe that the approach we follow is innovative enough in their integration and use. Our approach is based on a survey ran among european teachers about their practices in terms of user credentials usage and sharing.

1 INTRODUCTION

The last few years have seen the emergence of a new approach to building Web applications, in the form of mashups. Mashups are applications that reuse and combine data and services available on the web (Aghaee and Pautasso, 2012). In the rest of the paper, we will follow the definition of mashup and related concepts as defined in (Zibuschka et al., 2010). Web mashups usually combine data and services that are available on the Internet– freely, commercially, or through other partnership agreements. In this perspective, we differentiate between the mashup platform (the system that integrates remote content) and the remote content itself, usually integrated through some standard protocol or approach like W3C Widgets [2].

In terms of security, one of the main issues encountered when dealing with complex software web architecture is that the traditional browser security model dictates that content from different origins cannot interact with each other, while content from the same origin can interact without constraint (De Keukelaere et al., 2008). While solutions exists (like using a proxy server or making use of `script` tags), we have to find a generic way to overcome this limitation and allow content from mixed origins to interact with each other.

A key element to this integration is the proper management of users, i.e. know who is who, and who is allowed to do what: authentication and authorisation. Users may have an account with the mashup platform or with the provider of the remote service or content; moreover, users may want to use credentials they already have with other actors on the Internet (think of the 'Sign-In with your *xxx* account' button you find on many websites). Access policies may be defined centrally, while others may be specified on the remote side. This is what we call a loosely-coupled environment: independent components interacting together through identified interfaces, with little to no knowledge about remote components and policies. The challenges we face in this work are the following:

- how do we solve user authentication in such multi-layer loosely-coupled environment?
- how do we ensure proper access to mashup services and remote content, based on a decentralised policy
- how do we propagate authentication and authorisation information through the various layers of the environment?

This research takes place in the framework of the

[*]The work presented in this paper is partially supported by the European Commission's FP7 programme –project iTEC: Innovative Technologies for an Engaging Classroom (Grant agreement N°257566). The content of this paper is the sole responsibility of the authors and it does not represent the opinion of the European Commission and the Commission is not responsible for any use that might be made of information contained herein.

[2]http://www.w3.org/TR/2012/REC-widgets-20121127/#widgets-family-of-specifications

Colin J. and Tien Hoang M..
Authentication and Authorisation for Widget-based Applications in a Loosely-coupled Web eLearning Environment.
DOI: 10.5220/0004795700170026
In *Proceedings of the 10th International Conference on Web Information Systems and Technologies* (WEBIST-2014), pages 17-26
ISBN: 978-989-758-023-9
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

the iTEC project[3], a 4 years EU-funded project focused on the design and evaluation of the future classroom in Europe.

To solve the various issues and to ensure interoperability, we have investigated several open standards and solution, and have selected oAuth and OpenID as building blocks of our solution, in an approach similar to the one presented in (Govaerts et al., 2011). We will discuss this choice further in the paper, as well as give a complete description of the solution we put in place. The rest of the paper is organised as follows: section 2 presents the conclusions of a survey run with teachers to identify use of Learning Management Systems and practices regarding use and re-use of credentials. The results of the survey guided the rest of the work. Section 3 describes the overall project architecture and requirements in terms of user management and access control. Section 4 then presents the solution design and implementation. Section 5 gives some concrete examples. Finally, section 6 provides some conclusions and ideas for future work.

The paper does not include a related work section, because although it relies on solid and proven components and approaches, we could find no publication relating to a similar approach of building a centrally-managed environment for authentication and authorising access to services in a loosely-coupled web environment while accepting external authenticators. This may be linked to the highly decoupled and distributed approach which is inherent of large European research projects.

2 LESSONS FROM THE TEACHERS SURVEY

A survey was run during the second year of the project (2012) to ask teachers about their current practices regarding the use of Learning Management Systems, the type of credentials they use, their familiarity with third parties that provide authentication services (Google, Facebook, Microsoft Hotmail, Yahoo) as well as their inclination towards re-using credentials among various websites.

We received a total of 269 responses from 17 countries. Distribution across countries is unbalanced, but since the survey is only indicative, this is not considered as a problem.

Results showed that Moodle is the most widely used LMS, but quite a long list of other systems are also in use. The LMS is installed either locally to the school, or shared among different schools, although almost half of the respondents did not really know.

More importantly, responses showed that although a majority of users have local credentials to login to their LMS, a significant number of them use external credentials, which can be provided by an education authority (school, regional or national authority) or by third parties (such as Google, Facebook, etc.)

Finally, the survey showed that the vast majority of teachers (and students) have accounts with one of the major Identity Providers on the Web (Facebook, Google, Yahoo) and that almost 70% of them are willing to re-use those credentials to access their school services.

However, it is important to note that integration with external Identity Providers is not always an option, due to technical restrictions defined at the school level (site-blocking firewall).

Our conclusion is that if we want to add new services to existing LMS already made available to schools, we shall have to integrate them without adding an extra authentication burden on users. It means that new services will have to extend existing infrastructure and offer the possibility of re-using credentials that users may already possess. However, because some users are concerned that re-using credentials might constitute a security risk, it is important to propose a mixed approach (i.e. use both siloed and re-used credentials).

Complete results of the survey are presented in (Colin and Simon, 2012)

3 PROJECT DESCRIPTION AND ARCHITECTURE

The iTEC project (*i*nnovative *T*echnologies for an *E*ngaging *C*lassroom) is a four-year project funded by the EU; it is focused on the design of the future classroom in Europe. The project, which involves 27 partners among which 15 Ministries of Education from across Europe, brings together teachers, policymakers, pedagogical experts - representatives from each stage of the educational processes - to introduce innovative teaching practices.

The goal of iTEC is to pilot learning and teaching scenarios using integrated technology solutions in over 1,000 classrooms in 15 countries, making it by some margin the largest pan-European validation of ICT in schools yet undertaken. The scope of the project ranges from pedagogical scenario design, customisation to local technical environments, development of required technologies and school piloting. In this paper, we will focus on the technical aspects only.

[3]http://itec.eun.org/

3.1 Basic Concepts

The idea behind iTEC technical implementation is to pull pedagogical content and tools together into a common interface in a meaningful way, i.e. so that they can support a specific pedagogical scenario. The basic building blocks of the iTEC ecosystem (aka the iTEC Cloud) are the following:

- a *shell* is the host platform (in mashup terminology, this is the Mashup-providing platform) used to integrate the various forms of content and tools; it acts as an empty 'nest' for those remote components. It is also the first user-facing component. No particular assumption is made on the shell technology, beyond the fact that it must be able to display widgets (Wilson et al., 2011). Examples of shell platforms are Learning Management Systems (LMS) like Moodle[4], mobile platform like Android[5] or Interactive Whiteboard software like OpenSankore[6]
- a *Web application* is an autonomous piece of code that offers a set of a self-contained services. It takes care of its own rendering and embeds all necessary services to serve end-users and fulfill expected functionalities. It may include user management functionalities or rely on third-party for that purpose.
- A *widget* is an elementary piece of application or content (image, video...); it embeds some functionality that is of interest to the users and is displayed through a shell. A widget does not operate on its own, but rather, requires a widget engine or runtime to create and manage widget instances and a shell to be made available to users. Several kinds of widget engines exist, like Yahoo Konfabulator, Apple's Dashboard, Opera Widgets runtime, Apache's Wookie server. A self-contained widget is a widget that embeds all its functionalities, like a calculator widget. A WebService-backed widget is a widget that relies on other components to function. Such component may take the form of backend services reached through a REST webservices interface. This would be the case for a chat widget that stores chat messages on a separate distant datastore. Widgets may offer collaboration services, like chat or videoconferencing, access to pool of resources, geolocation functionalities or administrative functions (user management...)
- A *backend service* is a software component with no user interface, that is accessed by other components through a web service interface. In iTEC, most components are built in such a way that the user interface is implemented by a widget while the business logic and data layer are implemented by a backend service, although in some cases, components are developed as a standalone web application, with or without access to a backend service.

iTEC's approach to assemble and deliver content and tools is through the integration of widgets in a shell; the combination of widgets is derived from a pedagogical scenario, with the aim of supporting it by providing the right tools, which may be synchronous or asynchronous collaboration and communication, calculator, simulator, geolocation or content sharing. In addition to widgets, iTEC also uses web applications for some of its services.

3.2 iTEC in Action

Starting from a pedagogical scenario, widgets are assembled in a shell's space, typically a course page in a LMS or a presentation slide in an Interactive Whiteboard notebook, in a way that they can be used by end-users, a teacher and his class for example. To support this type of usecase, iTEC designed a set of components:

- the *widget store* is the main repository for widgets. It stores widget descriptions, allowing for searching, tagging, rating and commenting about widgets. It is similar in its concept to Google's PlayStore or Apple's AppStore. It is built as a widget.
- the *Composer* is used to manage pedagogical scenarios, and assemble widgets that meet the requirements of those scenarios. For instance, a scenario may include synchronous communication, picture sharing and collaborative writing, and this would translate into a chat-, a camera- and a wiki widget. This instantiation takes into account the technical settings of the target environment, i.e. the set of technologies available in a specific school. The Composer is built as a web-application.
- the *People and Event directory* is a database of people and events that may be of interest to pedagogical actors; it can be a subject expert, or a conference or similar event. Those resources are also rated, tagged and commented. This component is built as a web-application with an integrated backend service, thus supporting the development of

[4]https://moodle.org
[5]http://www.android.com
[6]http://open-sankore.org

widgets.

- the *Scenario Development Environment* is a recommendation engine, that can collects data from the above components, and provides recommendation to the Composer in its scenario instantiating role, by suggesting most appropriate resources to include in the setup. This component is developed as a pure back-end service since it is not meant to support any type of user interaction.

3.3 Authentication and Authorisation Challenges

From the description above, iTEC integrates a wide variety of components: shells, web applications, self-contained widgets, widget-based applications... This integration raises some questions in terms of user management and access control:

- user authentication may take place at the shell level, but also, some integrated services may require some form of authentication or at least be aware of the visiting user's identity. This implies the need for an authentication mechanism that can span the range of components and provide a consistent information about the user
- access control policies may be defined centrally, at the iTEC Cloud level, but these policies have to co-exist and be consistent with those defined at the shell level, or at the integrated services level, if any. Again, this requires an authorisation mechanism that integrates at the various levels of the architecture.

Our goal is thus to design a system that would meet the following requirements:

- allow user authentication at the shell level, and pass the information into sub-components (widgets and back-end services)
- allow access policies to be defined globally to the iTEC Cloud, based on a Role-Based Access Control (Ferraiolo et al., 2001) model, and have those policies propagated to the sub-components
- from the global access rules, provision local policies at the level of iTEC back-end services
- designed solution should support interoperability with major service providers

4 PROPOSED SOLUTION

The interoperability requirements led us to focus on open standards and protocols to build authentication and authorisation mechanisms. We performed a thorough study, and identified candidate protocols like SAMLv2[7], OpenID[8] and oAuth[9]. Due to their technological maturity, their relative simplicity, their support for web interactions, the availability of libraries and their wide adoption by main actors on the net, we selected oAuthv2 and OpenIDv2 as the basis for our solution. The fact that users are warned when an application wants to access protected data was also an element of choice.

OpenIDv2 (Foundation, 2007) is an open and standard protocol for signing on to websites using one single set of credentials. The protocol has been developed for many years and adopted by major players on the Internet, like Google. It relies on the assumption that users have an identity defined with an Identity Provider (IdP), and want to use that identity to access various services offered by Service Providers (SP). The typical flow is a user visiting a Service Provider that requires authentication; SP prompts the user for her identity or that of her IdP. The user is then redirected to the IdP to authenticate, and if authentication succeeds, the user is sent back to the SP with the proof that successful authentication did take place. Optionally, the IdP may provide additional information about the user (this requires some protocol extensions).

oAuthv2 (Hardt, 2012) is a protocol for managing delegation of authorisation. Its main use case is a user (the *resource owner*) needing to give access to some of its resources hosted on a server (the *resource server*) to a *client*, typically another service. To avoid forcing the user to give her credentials to the client, oAuthv2 introduces a workflow where when the user is asked by the client to give access to a resource, she is sent back to an *authorisation server* where she authenticates and is then asked to grant or deny access. Upon success, the authorisation server issues an *access token* to the client that it will use to access the resource on behalf of the user. In this way, the user's credentials are never disclosed to the client. This is the protocol that Facebook or Yahoo use for granting access to their services to remote sites, after getting the agreement of the user. oAuthv2 supports various types of 'grants', to support different profiles of this protocol and accommodate different situations:

Authorisation Code Grant this is the most secure scenario, in which the client directs the resource owner to the authorisation server for authentication and access request; upon success, the authorisation server issues an *authorisation code* to

[7]http://saml.xml.org/
[8]http://openid.net/
[9]http://oauth.net/2/

the client, that the client then exchanges with the authorisation server for an *access token*, that is finally presented by the client to the resource server to get access to the resource. All interactions with the resource owner go through her user-agent (typically her browser). This scenario supports client authentication by the authorisation server before issuing an access token, and ensures that the access token never reaches the resource owner's user-agent, which could lead to token leakage.

Implicit Grant this is a simplified version of the previous scenario, in which instead of being issued an authentication code by the authorisation server, the client directly receives an access token. This scenario is targeted at clients implemented in a browser, typically in javascript. In this case, the authorisation server does not authenticate the client, and the access token is exposed to the resource owner or other applications with access to its user-agent.

Resource Owner Password Credentials Grant this scenario is built on the assumption that there exists a high degree of trust between the resource owner and the client. The resource owner provides the client with her credentials, and the client uses them to request an access token from the authorisation server. This scenario supports client authentication.

Client Credentials Grant in this scenario, the client is acting on its own behalf, not on behalf of the user. The client authenticates directly to the authorisation server and receives an access token.

It is worthwhile noting that oAuthv2 also supports extension grants that allow to extend the token request mechanism to support different types of credentials, like SAML assertions.

Because we chose to use oAuthv2 to secure widget access to back-end services, and because widgets usually involve client-side computing and get access to the user's environment, the implicit grant is the only option of choice. However, we also successfully implemented the client credentials grant to secure access to the SDE backend service. One of the drawbacks of the implicit grant is the absence of client authentication, but this can be explained by the nature of widgets, which are running client-side, making available any sensitive information to other components running in the user's environment (user-agent). It would thus not be possible to securely store client credentials at the widget level.

To integrate those protocols into the iTEC environment, we designed the UMAC (User Management and Access Control) framework, which comprises the following components:

- the *UMAC server* is responsible for user authentication, issuance of tokens, and management of user data and privileges; it plays the role of the OpenID's Identity Provider, the oAuth's authorisation server, and implements a back-end service to access, store and manage user data and privilege information.
- the *UMAC filter* is an authorisation guard that sits in front of back-end services; the back-end service represents the oAuth's Resource Server, and the UMAC filter is in charge of validating access tokens.
- the *UMAC widgets* are a collection of widgets that allow to access and manage authentication and authorisation information in the iTEC Cloud. Those widgets allow to register a new user, to update a user's details, to create sets of users, and to assign iTEC roles.
- the *UMAC library* is a JavaScript library of tools to help the widget developer to easily integrate with the UMAC framework and not care about the various protocols' implementation.

4.1 UMAC Server

The UMAC Server serves two main purposes: authenticating users and controlling access to back-end services.

To authenticate users, UMAC Server implements the OpenID Provider specification. It handles authentication requests from iTEC relying parties, typically shells or web applications, authenticates users, and responds to relying parties; UMAC Server supports SREGv1.0 and AXv1.0 OpenID extensions to provide basic information of logged in user (username, first and last names, email address, language, timezone, country). Authentication is checked against a local database of users.

One of the requirements drawn from the survey described in section 2 mandates that iTEC should allow users to login using third-party credentials, namely Google, Facebook or Yahoo. Thus the UMAC Server supports user authentication using any of those systems, by implementing an OpenID Relying Party (in the case of Google and Yahoo) and an oAuth client (in the case of Facebook).

Access control to iTEC services is handled by UMAC Server. Access requests may come from widgets or web applications, in which case the oAuthv2 scenario implemented is the implicit grant, but requests may also come from standalone applications, which are run in a more controlled environment, and for which the selected scenario is the client credentials grant. Thus UMAC Server implements the related sections of the oAuthv2 specification, and handles Authorisation Requests (for the implicit grant) and Access Token Requests (for the client credentials grant), issuing access tokens to widgets and controlled applications respectively. A token is a random string concatenated with a timestamp.

In addition to the authentication and authorisation functionalities, UMAC server is also used to store user information; this information is made accessible through a REST API, which is protected by the oAuthv2 protocol, just like any other iTEC back-end service. Basic CRUD functionalities are implemented to add, update, delete or get information about a user account. This API is accessed typically by the UMAC widgets.

Finally, UMAC server is used to manage user privileges; those privileges span all iTEC services, i.e. apply equally to shells, widgets or back-end services. Six levels of privileges are defined in a strictly hierarchical way: super-user, administrator, coordinator, teacher, student and guest. The level of privilege of a user is passed to the OpenID relying party upon authentication through SREG or AX extensions, where available, and they are checked by the token validation process between the UMAC filter and the UMAC server.

For a seamless user experience, UMAC authentication is propagated to the shell through a plugin mechanism which is dependent on the shell itself. In this way, once the user is authenticated, all shell components (typically widgets) can reuse the user information.

4.2 UMAC Filter

The UMAC filter is designed to be put in front of back-end services, and interacts with the UMAC server following the oAuthv2 protocol to control access to the services by ensuring that only authorised requests get served. The current implementation of the filter takes the form of a servlet filter, which makes it very easy to integrate and (de)activate and realises a separation of concerns by allowing the service developer to work independently from the access control mechanism.

In the oAuthv2 terminology, the UMAC filter acts as the protection part of the resource server. It receives requests for access in the form of REST calls (basically http requests), and for each requests, it checks that an access token is provided. If no token is present, an error is returned, and it is up to the client to obtain one. If a token is present, its validity is checked by querying the UMAC server through a secure channel, and upon success, the lifetime of the token and the userid of the token owner are returned to the filter. Based on this information, the filter then checks the local access policy that defines the rules for accessing the service. These rules are expressed using the Apache Shiro[10] system. If the rules are evaluated positively, access is granted and the request is passed to the service. Otherwise, an error is returned. For efficiency reasons, the UMAC filter caches the validated tokens for a period of time to avoid unnecessary roundtrips with the UMAC server.

4.3 UMAC Library

The UMAC library is a Javascript library of functions that aims at facilitating the development of widgets and their integration with UMAC authentication service, more precisely, the oAuth authentication endpoint's service. It hides the complexity of the protocol by providing methods to manage the whole authentication process (request for token, redirect to authentication form, token transfer to requesting component and error handling).

Figure 1 presents the UMAC components (in gray) as well as the interactions with other iTEC systems. UMAC Server is used for authentication (solid lines) either from shell, widgets or web applications like the composer. This follows the OpenID protocol. Authentication may be local (using the User DB) or rely on third-party authenticators (right-most box). Regarding authorisation (dashed lines), UMAC widgets allow to register or update user information through the UMAC REST Web Service, which is protected by the UMAC filter. Similarly, any other iTEC component may access iTEC back-end services which are protected by the UMAC filter (see bottom of the diagram).

5 EXAMPLE SCENARIOS

In this section, we provide concrete examples of the system described above.

5.1 Authenticating a User

This is the very first step to do to consume a protected service or piece of content. A user makes an authentication request directly from a widget, from a shell or from a relying web application by clicking on

[10]http://shiro.apache.org/

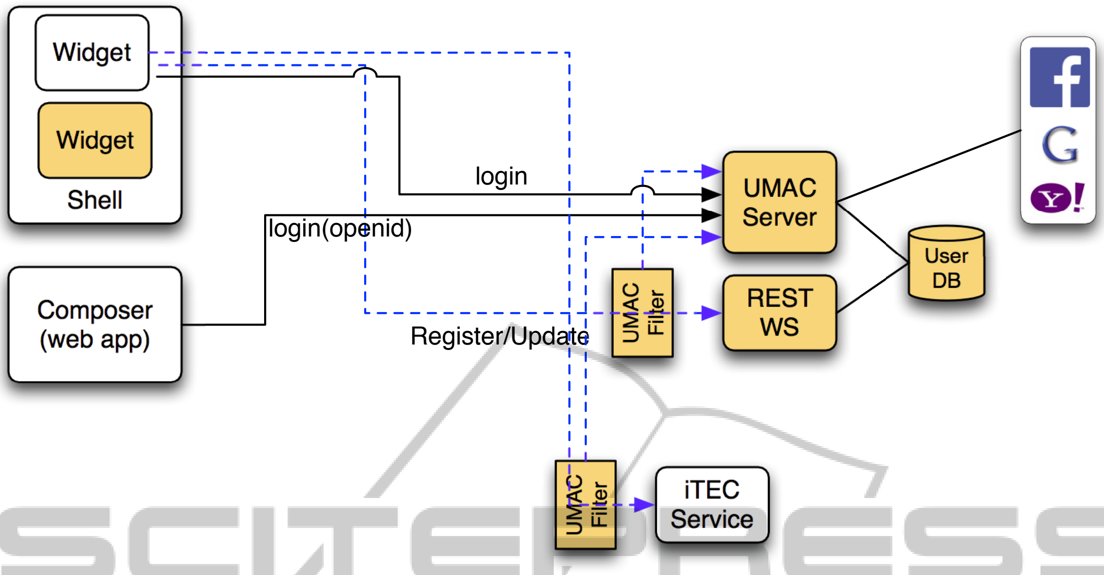
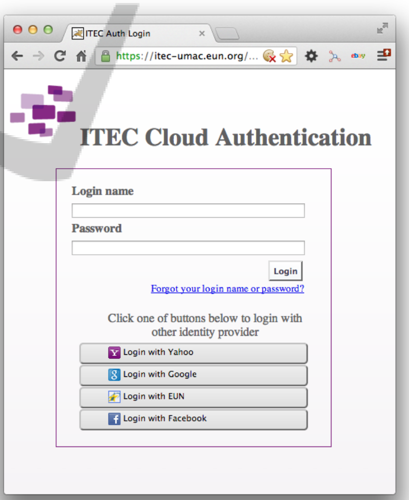


Figure 1: UMAC components.

an URL or a button. This causes redirection to the UMAC Authentication Window (see figure 2),where the user must provide her credentials to UMAC Server or use her external authentication sources such as Yahoo, Google, Facebook to proof his identity. Upon successful authentication, an authentication response (in case of OpenID flow) or an access token (in case of oAuth flow) is generated by UMAC Server and sent back to the Relying Party or the Client respectively.

It is to be noted that even though the authentication action is triggered at the widget level, authentication information is pushed up to the container level (i.e. the Mashup Providing Platform).


ITEC Auth Login
https://itec-umac.eun.org/...

ITEC Cloud Authentication

Login name

Password

Login

Forgot your login name or password?

Click one of buttons below to login with other identity provider

Login with Yahoo

Login with Google

Login with EUN

Login with Facebook

Figure 2: UMAC Login window.

5.2 User Authorisation

For fine grain and flexible authorisation, UMAC relies on both a permission-based approach and on a role-based system, in which permissions are assigned to roles rather than being assigned directly to users. Roles are then assigned to users who inherit all permissions linked to the role. At the moment, role structure is flat, i.e. we do not support hierarchical roles.

A permission is basically a tuple that expresses what action can be executed by a service on a specific data scope. Our model follows a Discretionary Access Control approach (Saltzer and Schroeder, 1975) where each piece of data is owned by exactly one user. Permissions are defined according to the specifications below (expressed in Augmented Backus-Naur Form (Crocker, 2008)):

```
permission = action ":" service-name
    *(":" data-scope)
action = (atomic-action *("," atomic-
    action)) / "*"
atomic-action = "create" / "read" / "
    update" / "delete"
service-name = 1*(DIGIT / ALPHA / "-"
    /"\_")
data-scope = 1*(DIGIT / ALPHA) /
    special-data-scope / (data-scope
    *("," data-scope))
special-data-scope = "me" / "mine" /
    "*"
```

The special-data-scope element can take one of three special values:

- "*me*": this data-scope represents the user currently logged in, i.e. the user who issued the access request; this is typically used for the user management to allow a user to update his own data, but others'
- "*mine*": this data-scope includes all data that is owned by the user currently logged in;
- "*": this data-scope includes all data; it does not need to be specified, e.g.: the permission `action:service-name:*` has exactly the same meaning with `action:service-name`

The next paragraphs provide some concrete examples of expressing UMAC permissions.

The rule "any user can read and update his information and unregister his account" can be decomposed as follows:

- The actions are "*update*", "*read*" (for update and read information) and "*delete*" (for unregistering account).
- The service in used is "*users*".
- The id of the target is the identifier of the requesting user himself, which can be the real id of user as recorded in UMAC, or the special id "*me*".

The rule would then translate to: `update,read,delete:users:me`.

In UMAC this is the default permission assigned to all registered users.

Role-based rules follow a similar approach: for instance, the rule "UMAC admin Role can read and update any user information as well as grant and revoke iTEC Coordinator of Belgium role to a user" includes two different sets of permissions:

- read and update information of any user, which translates to `update,read:users:*` or more briefly `update,read:users`
- grant and revoke belgian coordinator role:, which translates to `read,delete:roles:coordinator:be` (the last `be` data scope represent the belgian record)

Permission to role assignments are stored in the UMAC server database, and are managed through the UMAC widgets; this operation can performed at runtime, which means that the role can be defined dynamically. Moreover, all modification of permission/-role assignments takes effect immediately for all users to whom the role is assigned.

5.3 Securing a Webservice

As stated in section 4.2, backend services are protected by the UMAC filter. Implementation of the filter takes the form of a servlet filter because most iTEC technologies use the Java language and because it is easy to integrate as a proxy in a non-Java environment. As a consequence, the protected web service provider only needs to include the declaration of the UMAC filter in the servlet mapping section of the web.xml configuration file.

In iTEC, the UMAC filter is currently deployed at the People and Event Directory (P&E Directory)[11]. With this deployment, P&E Directory developers only focus on their functional development, all authorisation rules being activated by a simple declaration in its configuration file. Authorisation rules of the filter can be defined dynamically, as shown in section (5.4)

The UMAC filter is also used to protect the UMAC Backend Service (REST WS box on figure 1)

5.4 Central Policy Management

Permissions, roles, role-permissions assignment and user-role assignments are managed by the UMAC Server; they can be updated through the UMAC widgets or any other component that access the UMAC REST WS, provided the appropriate level of privilege is granted.

UMAC supports the `super-admin` role, a static built-in role which is assigned to the UMAC system administrator once it is installed. Since the authentication and authorisation engine is based on Apache Shiro framework(Team, 2013), most of authentication authorisation information are defined in a "shiro.ini" file, including: built-in super admin accounts, default permissions granted to all authenticated user, mapping table between service names and REST service contexts which require authorisation and information for initialising authentication realms (local realm to login locally to UMAC, OpenID realm to login remotely with Yahoo, Google...).

The UMAC filter (section 4.2) has a similar implementation except that there is no authentication database since it relies entirely on the authentication service of UMAC server. It only maintains an authorisation database which can also be configured through REST services. The UMAC filter administrator is defined in "shiro.ini" at filter side; it has similar function to `super-user` at UMAC server but is completely independent. This design helps the filter benefit from authentication service of UMAC server while keeping a totally independent authorisation lattice under the control of the filter administrator.

The UMAC authorisation widgets are developed to manipulate all information related to authorisation

[11]http://itec.eun.org/web/guest/people-and-events

in the iTEC Cloud. The widgets also allow an authorised user to register a backend service provider and assign authorisation rules to that service provider, thus providing a central place to manage all authorisation at the iTEC components level. This allows for a more consistent and secure approach.

5.5 Implementing Widgets and Services

Integrating a new service into UMAC is a very simple process that consists of instantiating the UMAC filter in front of the new service. As stated above, the filter takes the form of a Java servlet, that intercepts all service requests, authenticates them and checks authorisation together with the central policy. A mapping between the centrally-managed iTEC roles and the service-specific roles is defined and translated into the Shire rules.

Widgets access services through the XMLHttpRequest API that allows to connect to remote sites and services. To make the integration with UMAC as easy as possible for widget developer, calls to services are wrapped through calls of the UMAC library that hide the logic of the underlying protocols. From the widget perspective, invoking a service is simply a matter of issuing a request to the service. Authentication and token issuance is handled under the control of the library.

6 CONCLUSION & FUTURE WORK

In this paper, we described a model and its implementation that supports centralised authentication and authorisation in a loosely-coupled multi-layered web application in the eLearning area. The model is open to third-party authenticators, following the conclusion of a survey run among potential users of the platform. Although the individual components of the approach themselves are not particularly innovative, yet state-of-the-art, we advocate that the integration approach we propose, that takes into account requirements and behaviours of end-users, and the complexity of the environment to which it applies constitute the core of the research. It could easily be transposed to other similar environments.

The whole iTEC ecosystem has been used through various cycles of piloting, involving over 2,000 classrooms across 19 European countries, and the UMAC component, although almost invisible to the vast majority of users, did work properly, securing the services from 4 main providers access through a wide variety of widgets.

In the future, we intend to work on a stronger and easier integration between shells and UMAC components, which would allow further customisation of the shell based on information received from the UMAC system. We will explore the IMS Learning Tools Interoperability specifications(Wilson et al., 2011)[12] as it seems to be a potential candidate for supporting tools integration.

A scope should be added to tokens, to reduce the granularity of authorisation rules and limit the potential impact of token interception, and client authentication should be added to the protocol to ensure that only authenticated clients can obtain access tokens from UMAC server.

REFERENCES

Aghaee, S. and Pautasso, C. (2012). An evaluation of mashup tools based on support for heterogeneous mashup components. In Harth, A. and Koch, N., editors, *Current Trends in Web Engineering*, volume 7059 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg.

Colin, J.-N. and Simon, B. (2012). D7.2: Second generation of iTEC shells and composer. Project deliverable 7.2, University of Namur.

Crocker, D. (2008). Augmented BNF for Syntax Specifications: ABNF. RFC 5234, RFC Editor.

De Keukelaere, F., Bhola, S., Steiner, M., Chari, S., and Yoshihama, S. (2008). Smash: secure component model for cross-domain mashups on unmodified browsers. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 535–544, New York, NY, USA. ACM.

Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., and Chandramouli, R. (2001). Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274.

Foundation, O. (2007). Openid authentication 2.0. `http://openid.net/developers/specs/`.

Govaerts, S., Verbert, K., Dahrendorf, D., Ullrich, C., Schmidt, M., Werkle, M., Chatterjee, A., Nussbaumer, A., Renzel, D., Scheffel, M., Friedrich, M., Santos, J. L., Duval, E., and Law, E. L.-C. (2011). Towards responsive open learning environments: the ROLE interoperability framework. In *Proceedings of the 6th European conference on Technology enhanced learning: towards ubiquitous learning*, EC-TEL'11, pages 125–138, Berlin, Heidelberg. Springer-Verlag.

Hardt, D. (2012). The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor.

Saltzer, J. and Schroeder, M. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308.

[12]http://www.imsglobal.org/toolsinteroperability2.cfm

Team, A. S. (2013). Apache shiro reference documentation. `http://shiro.apache.org/reference.html`.

Wilson, S., Sharples, P., Griffiths, D., and Popat, K. (2011). Augmenting the vle using widget technologies. *Int. J. Technol. Enhanc. Learn.*, 3(1):4–20.

Zibuschka, J., Herbert, M., and Roßnagel, H. (2010). Towards privacy-enhancing identity management in mashup-providing platforms. In *Proceedings of the 24th annual IFIP WG 11.3 working conference on Data and applications security and privacy*, DB-Sec'10, pages 273–286, Berlin, Heidelberg. Springer-Verlag.