ESUP-Portail: a pure WebDAV-based Network Attached Storage

Pierre Gambarotto *, Pascal Aubry †

* ENSEEIHT, National Polytechnic Institute of Toulouse, France
pierre.gambarotto@enseeiht.fr

† IFSIC, University of Rennes 1, France
pascal.aubry@univ-rennes1.fr

Abstract

ESUP-Portail ESUP-Portail is a consortium of French Universities. Its goal is to provide a complete and open solution to Universities and University-level colleges who wish to offer integrated access to their services and information for their students and staff. This includes user data storage and many applications (such as CMS) also needing storage.

We present the characteristics needed for such a system, in terms of access, security, ease of use and functionalities. The WebDAV protocol and its extensions are overviewed, with pro and cons detailed. We finally show the architecture of the planned system.

Keywords: distributed storage, WebDAV, access control.

1 Why a new storage solution?

The ESUP-Portail [1] software, when completed, will offer:

- Basic web applications (such as webmail, forums, agenda, CMS, etc.);
- More specific ones (mark reports, accounting, registration, etc.);
- Data storage for users.

At the end of the project, each user will be able to access his storage area from anywhere (school, home, cybercafés, etc.). As it is also accessed by many applications of the ESUP-Portail digital workspace, the storage area is a pivot of the project.

1.1 Which protocol for data sharing?

One way widely adopted is to set up VPNs (Virtual Private Networks), and continue to use existing protocols (NFS, CIFS, Netware, etc.) for data sharing. When this solution is acceptable when dealing with some limited populations, it is absolutely unconceivable when dealing with tens of thousands students.

New protocols must be considered, and one comes out of the crowd: based on HTTP, WebDAV [2] is naturally well-suited for internet roaming users. It is an official standard, broadly implemented.

1.2 Requirements for the ESUP-Portail storage area

1.2.1 Clients

Users will use several clients to access their data from anywhere on the internet:

- CGI applications, possibly integrated into a portal (uPortal is the base entry-point of the ESUP-Portail workspace);
- Heavyweight interfaces, such as java/swing applets;
- Graphical file-managers, integrated into an operating system (Windows explorer, Mac OS X, Unix/Gnome Nautilus, etc.);
- Any other application.

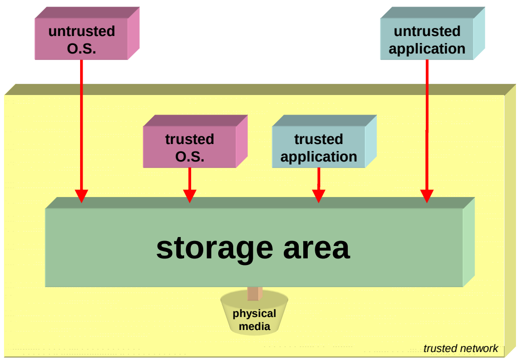


Figure 1: The clients of the storage area

These clients, depending on the way they are trusted, will be imposed different security strategies.

1.2.2 Basic features

The storage area must at least offer the following features:

- Each user must be able to **access his/her own documents**;
- Each user must be able to **set access conditions**, to permit/forbid others to access his/her documents. Access rules can depend on an external service, in order to define groups of people;
- The access protocol for users must be **WebDAV**. WebDAV should also be used by all the clients of the digital workspace accessing the storage area, i.e. applications and operating systems (trusted or not trusted).

1.2.3 Extended features

A document should not just consist of a file (data), but also of a set of **properties** (owner, access rules, author, last date of modification, version, etc.). The properties should be easily accessed and modified by users, through as many clients as possible. Moreover, the set of properties must be easily extended; additional administrator-defined properties can have unlimited usages, such as:

- One of the ESUP-Portail is INJAC [3], a Content Management System, which transforms documents to display them; the way they are displayed could depend on a specific property, e.g. the type of the documents.
- Properties can represent use-by dates, after which documents should not be readable anymore.

The storage engine (serving the physical storage system) should provide **indexing and searching** features, on the whole storage area, or just a part of it.

To summarize, the storage engine must allow controlled read/write access to files (file data) and properties. The access must be controlled by the authentication of the client user, and depend on his identification (who is Mr Doe?) and his profile (what can Mr Doe do?).

At least, the storage engine should be able to use external services, especially for authentication and access control.

1.3 Any existing solution?

Some hardware constructors (EMC, Network Appliance) already offer ready-to-use multi-protocol (NFS, CIFS, even WebDAV) NAS solutions, but none is adapted to the ESUP-Portail project. Indeed, WebDAV support is always limited (to RFC 2518, sometimes with authentication against LDAP or Active Directory), when HTTP extensions are needed. Moreover, commercial solutions are not opened enough to accept external plug-ins (especially for authentication and authorizations).

This observation led us to develop our own storage solution, which we present in this paper.

At first, we will recall the basics of WebDAV, and its interesting extensions. Then we will describe the software architecture of the ESUP-Portail storage system.

2 The WebDAV protocol

2.1 Generalities

WebDAV is an extension to HTTP/1.1, which initial goal was to permit remote editing through HTTP. To do so, WebDAV adds the following concepts:

- Documents are no longer data, but also metadata, called **properties**. The value of these properties can be controlled by the server, or enforced by clients' requests. A property can be used to store information about the associated resource, for instance the author or the last date of access. Technically, a property is just a (name, value) pair linked to a resource. One can use properties to manage and search a set of resources. Properties can also be seen as an extension to HTTP Headers, also expressed as a (name, value) pair, but as explained below are expressed in an XML form.
- A resource can be locked by users. Access to resources can be restricted by **locks**.
- Resources can be gathered into **collections**, much like files are gathered into directories within file systems. A collection is itself a resource, and thus can be moved, copied, deleted like another resource.

With the proper configuration, a WebDAV server can mimic a regular file system, thus providing true remote editing of resource, much like other distributed file systems, like NFS or CIFS.

Security and authentication are handled by the inherited HTTP. For instance, Basic or Digest authentications schemas can be used two schemas of authentication are available: Basic and Digest. WebDAV nevertheless recommends that Basic authentication be used together with SSL/TSL, to prevent password sniffing.

2.2 HTTP add-ons

The add-ons to HTTP to handle the new concepts described in the previous paragraph are new methods, headers, and error and status codes.

The most visible change is the introduction of an XML format to encode request and response bodies. With HTTP/1.1, information is passed using headers. XML has several advantages on HTTP headers:

- With an XML block, things concerning properties can be separated things concerning the request/response (still located in the headers.
- XML support multiple character sets and thus is adapted to internationalization.
- XML namespaces (RFC 2396) are used in WebDAV to achieve uniqueness of property names. Two properties can thus have the same name in two different namespaces.

New methods are:

- **PROPFIND**/**PROPMATCH**: respectively to get/set a property on a resource. Arguments, as well as return values, are presented into an XML block.

  Here is an example of a PROPFIND request:

```
PROPFIND /file HTTP/1.1
Host: doc.domain.org
Content-type: text/xml; charset="utf8"
Depth: 0
Content-Length: xxxx
```

```
<?xml version="1.0" encoding="utf-8"?>
<D:propfind xmlns:D="DAV:">
  <D:prop xmlns:R="http://doc.domain.org/collec/">
    <R:author/>
    <R:title/>
  </D:prop>
</D:propfind>
```

Let us remark that the targeted resource is a collection. In this case, the Depth header sets the range of the request. Possible values are 0 for the resource collection itself, 1 for the collection and its immediate children, and infinity for the collection and all its children.

XML namespaces are used. Notice the "DAV" namespace, used for all properties specific to WebDAV. Two properties are specifically requested in this example, "author" and "title". It is also possible to request all the properties of a resource, by requesting the special "DAV:allprop" property:

```
<D:prop xmlns:R="http://doc.domain.org/collec/">
  <D:allprop/>
</D:prop>
```

The response could be, for instance:

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml;
charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>
      http://doc.domain.org/collec/
    </D:href>
    <D:propstat>
      <D:prop xmlns:R="http://doc.domain.org/collec/">
        <R:author>
          <R:Name>Toto</R:Name>
        </R:author>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
    <D:propstat>
      <D:prop>
        <R:title/>
      </D:prop>
      <D:status>HTTP/1.1 403 Forbidden</D:status>
      <D:responsedescription>
        The user does not have access to
        the DingALing property.
      </D:responsedescription>
    </D:propstat>
  </D:response>
  <D:responsedescription>
    Access violation error.
  </D:responsedescription>
</D:multistatus>
```

Notice the 207 status code, called "Multi-Status". This code permits to response several status codes in the same message. Real status codes are embedded into the XML block. In our example, the first property requested by the client is successfully returned, while the second one is somehow protected, thus its access is denied.

LOCK/UNLOCK: respectively set/unset a lock on a single resource or a set of resource. More precisely, when a LOCK is requested on a resource, a lock token, unique in time, is generated and returned to the requester. This lock token must be resent in each request to the locked resource, through the If header. A lock is a temporary item and can disappear at any time, and is neither stored nor restored in case of a server crash or reboot. A lock can also have a validity, as indicated by the "DAV:timeout" property. If a client requests a LOCK method with the If Header containing a valid lock token, then the LOCK is refreshed and the timeout reset. The UNLOCK method request must have the Lock-Token header present, with the correct lock token for the lock to be deleted. A lock token can be linked to several resources, but the UNLOCK method destroys the lock token, and thus the lock on all the resources locked by this token.

Notice the possibility of locking a non existent resource, that can be used to forbid the creation of a resource at a locked URL.

MKCOL: to create a collection.

RFC2518 also defines the effects of those new methods on the resources, as well as the effects of pure HTTP methods on locks, properties and collections.

2.3 Access Control

To control access to a resource, a WebDAV server relies on two services:

An authentication service, that must answer the question "*Is the requester really who he claims to be?*"

An authorization service, used on successful authentication, that must answer the question "*Can the authenticated requester perform a specific action on the target resource?*"

2.3.1 Authentication

As already said, WebDAV completely relies on HTTP authentication, as described in RFC 2068 and 2069. To sum up, if a non authenticated request is received by an HTTP (WebDAV or not) server (arrow 1 of figure 2), the server sends back a 401 status code (arrow 2). The requester agent, typically a web browser, then asks its user login/password information (arrows 3), and resends the initial request with an extra "Authorization" header containing authentication data (arrow 4).

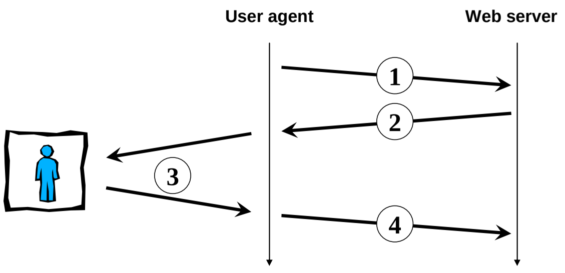


Figure 2: Basic HTTP authentication

If the authentication is correct, the access is granted. Otherwise, another 401 response is sent back to the user.

2.3.2 Authorization

WebDAV (or HTTP for that matter) is just a protocol, and doesn't specify the authentication source, i.e. where the server searches for the correct authentication of a user.

This question also stands for authorization. Once authenticated, the server has to get the privileges associated to that particular identity in order to answer the question: "*Has the authenticated entity the ability to perform a particular method on the resource?*"

An internet draft has been approved on last November (2003) as a wannabe RFC on WebDAV Access Control Protocol [4] (called thereafter ACP).

All possible requesters, human or computational agent, are called principals in this proposal. A WebDAV server supporting those recommendations has to store a representation of each principal as a WebDAV resource. A principal resource is then identified by one or several URIs, but one of them must be an URL that is called "DAV:principal-URL". Among the other URIs (grouped in DAV:alternate-URI-SET) you can find link to external services, like "ldap://". A principal can also be a set of principals, hence forming a group. As groups can be members of other groups, the notion of group is recursive, which builds a hierarchy of privileges. The members of a group and the groups a principal is belonging to can be known (resp. changed) by retrieving (resp. updating) the "DAV:group-member-set" and "DAV:group-membership" properties.

Privileges needed to perform a method on a resource are expressed with XML properties. The most representative ones are "DAV:read", "DAV:write", "DAV:read-acl", "DAV:write-acl". Each privilege is then defined with regard to a particular set of HTTP/WebDAV method. For instance, the "DAV:read" privilege restricts the GET and PROPFIND methods, but "DAV:read-acl" restricts PROFIND only with regards to access control related properties.

PROPFIND/PROMATCH are user to get/set Access Control Property (henceforth ACP) on a resource, just as it is done for any other property. "DAV:owner" and "DAV:group" ACPs respectively reflect the principal that "owns" the resource, and the group of the resource.

The most useful ACP is "DAV:acl". ACL stands for Access Control List. An ACL is a list of Access Control Entity (ACE). Each ACE represents a relation controlled on the resource between a principal and a privilege. The relation can be to grant or to deny to the principal the use of the privilege.

To sum up, if you want to restrict the use of a method for a particular resource and a specific set of principals, you have to:

Find the correct privilege associated with the method you want to restrict, for instance "DAV:unlock" privilege for the UNLOCK method.

Define the principals to which the restriction will apply. You can use a reference (URL) to a principal, or use a property that itself define a reference to a principal. For instance, the "DAV:owner" and "DAV:group" properties define a reference to a principal. It is also possible to define an inverted selection, meaning "all the principals who are not …"

Build an ACE with the previous arguments. Grant or deny relation must be chosen.

Add the ACE to the ACL of the resource.

Set the "DAV:acl" property of the resource with the ACL built in the previous stage.

For instance, if you want to deny anybody but the owner of a resource to PUT or PROPPATH the resource, you should update the "DAV:write" privilege. This can be done with such a request:

```
PROPPATCH /path/to/resource HTTP/1.1
Host: doc.domain.org
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:propertyupdate xmlns:D="DAV:">
  <D:set>
    <D:prop>
      <D:acl>
        <D:ace>
          <D:principal>
            <D:property><D:owner/></D:property>
          </D:principal>
          <D:grant>
            <D:privilege><D:write/></D:privilege>
         </D:grant>
       </D:ace>
      </D:acl>
    </D:prop>
   </D:set>
 </D:propertyupdate>
```

3 Architecture

From a user point of view, the feeling of being in a workspace is essentially due to:

Having a unique entry point;

Authenticating only once.

Having a unique entry point is achieved by the ESUP-Portail web portal (uPortal [5]). This portal gives access to all the resources of the workspace thanks to embedded applications (uPortal channels) and hypertext links to stand-alone applications.

Authenticating only once is achieved within the ESUP-Portail workspace by its Single Sign-On, i.e. CAS [6, 7], (Central Authentication Service).

We describe in this part the global architecture of the storage area. This architecture can seem complex to newbie users, because modularity has been voluntarily emphasized to ease the integration into existing environments.

As shown on figure 3, the storage component is made up of:

A **protocol layer**, implementing a complete WebDAV server with ACP understanding;

A **file system abstraction**, which allows writing to the physical storage as to a high-level file-device;

- An **authentication module**, which controls the access to storage area at a global level;
- A **granting module**, which allows or denies actions at an operation level.

The storage area relies on other components of the digital workspace:

- The **SSO service**;
- The **user database**;
- A **grouping service**.

The interactions of the WebDAV server with these services are explained below.

3.1 The filesystem abstraction

The filesystem abstraction allows writing to the physical storage media as to a high-level file-device. Moreover, the access of the storage system to the media is supposed to be full (read/write) and exclusive (the storage system is the only one to access the media).

In practice, the media can be:

- A local filesystem, of which the formatting type is understood by the operating system running the WebDAV server;
- A distant filesystem located on a SAN (Storage Area Network), mounted by any well-known file-sharing protocol (NFS, CIFS, etc.).

The only reason of this abstraction layer is portability: we want system administrators to be able to rely on any existing media.

3.2 The authentication module

At first, we shortly recall the main principles of CAS. Then we show how the storage system can be integrated into the SSO area of the ESUP-Portail digital workspace.

3.2.1 CAS Single Sign-On

CAS, developed by Yale University, was chosen by the ESUP-Portail consortium for its SSO mechanism for the following reasons:

- It is an **open-source and free** product;
- Its **security level** is very satisfying;
- It allows **multi-tier installations** without propagating any password;
- A CAS server is very **easy to set up** and configure;
- Web applications are **very easy to CAS-ify** (to set CAS-compliant).

Single Sign-On is achieved by CAS using the following techniques:

- **Authentication is centralized** to a single server, the only machine receiving user credentials, always through an encrypted tunnel;
- **HTTP redirections are used**, from applications to the authentication server for unauthenticated users, and back to applications when authenticated;
- **Information is passed** by the authentication server to applications during the redirections, thanks to tickets (cookies and/or CGI parameters).

Readers interested in using CAS may refer to [X] for a complete explanation on the way CAS is used within the ESUP-Portail software.

Even if tickets can be used at system-level by some PAM-compliant services (as shown in [X]), CAS is a web-based only SSO.

Now, neither human beings nor operating systems can, at the time we write this article, meet CAS requirements (get tickets with HTTP requests, perform 30x HTTP redirections, etc.).

In practice, the storage area will have to perform multi-

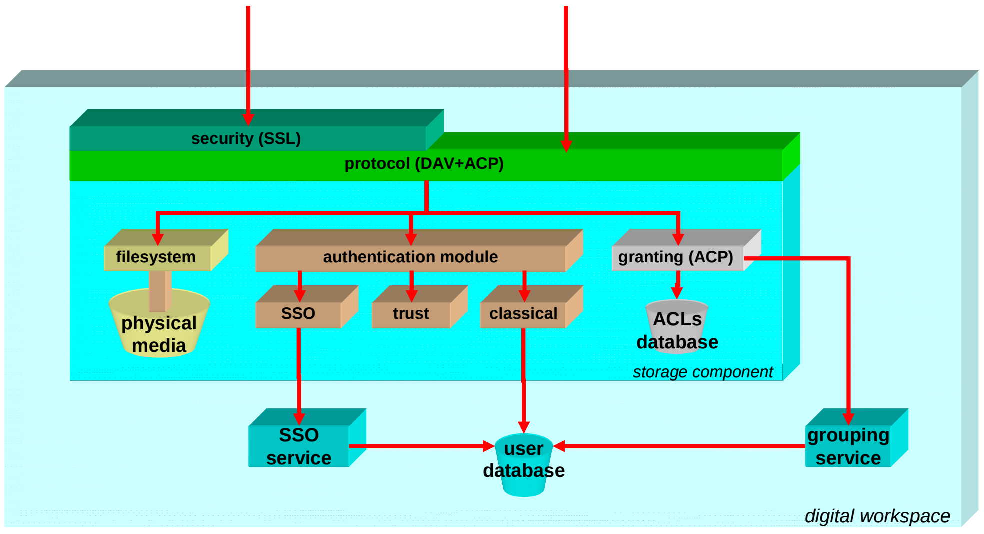


Figure 3: Software architecture

purposes authentication:

- SSO authentication for web applications;
- Classical authentication for non-web applications and operating systems.

Furthermore, some applications, part of the ESUP-Portail software suite, will be considered as trusted, and thus will need no authentication (the storage area will always trust the identity provided by such applications):

3.2.2 *SSO authentication*

The integration of CAS is quite easy because CAS is distributed with many client libraries, of which a Java library.

The only problem is to detect correctly the kind of the agent (client) accessing the storage area. Indeed, only capable web browser should be redirected to the CAS server when not authenticated. Others, unable to perform redirections like operating systems, should use a classical authentication (see below).

3.2.3 *Classical authentication*

By “classical”, we mean:

- Directly relying on the user database of the digital workspace;
- Accepting authentication methods commonly-used in web programming, such as HTTP basic authentication (realms), or even personal X509 certificates trusting policies.

This authentication mode will always be forced for clients that are not able to perform redirections, such as operating system. In this case, non-authenticated clients receive a *401 Unauthorized* error code.

For instance, classical authentication could behave exactly as the well-known mod_auth_ldap Apache module.

3.3 The granting module

This module is in charge of implementing access control.

For each request, the granting module knows:

- The authenticated user;
- The name of the target resource;
- The action to be performed.

In order to decide whether the action can be performed or not, the module queries the ACL database, and the grouping service if the ACL corresponding to the resource uses a group specification.

4 Current implementation and perspectives

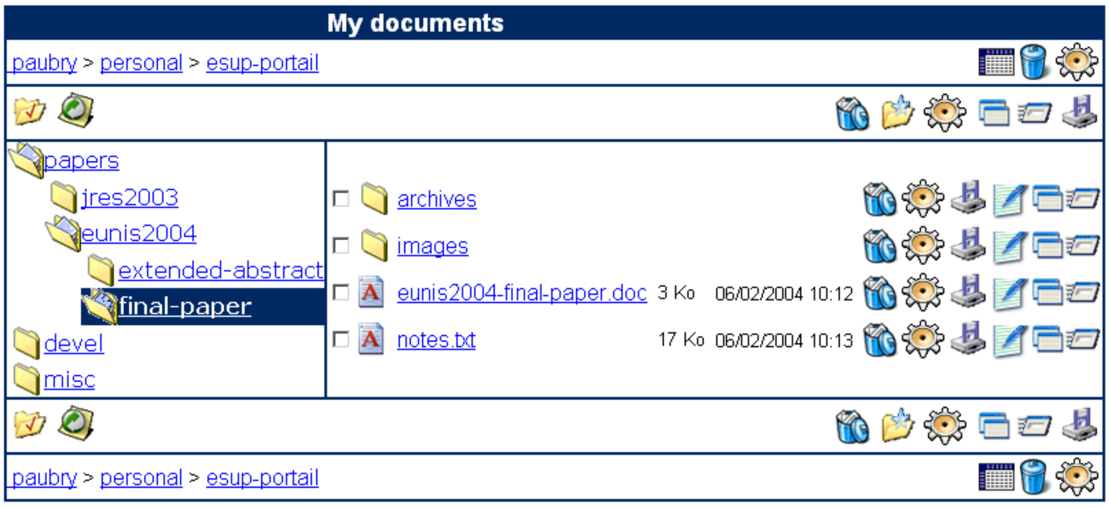
At the time we write this article, ESUP-Portail software v1 is distributed with restricted features. Full-featured v2 is planned for late 2004.

4.1 The WebDAV server

With ESUP-Portail software v1, the storage area is for personal data only, since data sharing between users is not implemented yet. Each user has full access to his own area, but no more.

This offers nevertheless a basic answer to roaming users that want to access personal data from anywhere. It is achieved by using:

- Apache web server [8];
- Apache mod_dav, to add WebDAV capabilities [9];
- Apache mod_auth_ldap, to add classical authentication [10].
- CAS mod_cas, to add SSO authentication.



My documents

paubry > personal > esup-portail

papers
- jres2003
- eunis2004
  - extended-abstract
  - final-paper
- devel
- misc

archives			
images			
eunis2004-final-paper.doc	3 Ko	06/02/2004 10:12	
notes.txt	17 Ko	06/02/2004 10:13	

paubry > personal > esup-portail

Figure 4: A snapshot of the uPortal WebDAV channel

Using Apache was for us the simplest way to get very quickly basic features with a reliable solution. However, we knew from the beginning that it was not scalable, and that performance issue will appear with intensive use.

The storage system proposed by ESUP-Portail software v2 will be:

A complete rewriting of v1, in Java;

Based on Slide [11] libraries;

Supporting ACP and thus implementing data sharing.

4.2 WebDAV clients

Brought with classical (LDAP) authentication, the WebDAV server included in ESUP-Portail software v1 can be accessed with any traditional WebDAV client.

The storage area can be accessed by web browsers (in read-only mode), once they are CAS-authenticated (thanks to the mod_cas Apache module).

It is also possible for users to use a web browser and manage their personal storage area within uPortal (the ESUP-Portail web portal), thanks to a WebDAV uPortal channel.

The channel is trusted by the WebDAV server (a secret is shared between the server and the channel, and both run on a private VLAN).

This uPortal channel will be improved in v2 to bring facilities to manipulate ACLs, thus allowing data sharing. It will probably be the most used client, at least until other clients such as operating systems are ACP-compliant.

References

[1] The ESUP-Portail project, http://www.esup-portail.org

[2] WebDAV, Web-based Distributed Authoring and Versioning, http://www.webdav.org

[3] INJAC : de l'utilisation de Cocoon et J2EE pour la gestion du cycle de vie de documents web, in French, P. Gambarotto & B. Sor, JRES2003, Lille (France), November 2003, http://www.jres.org.

[4] WebDAV Access Control Protocol, http://www.webdav.org/acl

[5] uPortal, a free and shared web portal, http://mis105.mis.udel.edu/ja-sig/uportal/

[6] Central Authentication Service, http://www.yale.edu/tp/auth/

[7] ESUP-Portail: open-source Single Sign-On with CAS (Central Authentication Service), P. Aubry, V. Mathieu & J. Marchal, EUNIS2004, Ljubljana (Slovenia), July 2004, http://eunis.fri.uni-lj.si/EUNIS2004/

[8] The Apache HTTPD server project, http://httpd.apache.org

[9] Mod_dav: a DAV module for Apache, http://www.webdav.org/mod_dav/

[10] LDAP authentication module for Apache, http://www.muquit.com/muquit/software/mod_auth_ldap/mod_auth_ldap_apache2.html

[11] The Jakarta Slide project, http://jakarta.apache.org/slide/

Acknowledgements

The members of the JA-SIG group for their work on uPortal;

The ESUP-Portail project for their feedback and contributions;

Vincent Mathieu and Julien Marchal (University of Nancy 2) for their precious help with the first release of the storage area and the WebDAV uPortal channel.

Vitae

Pierre Gambarotto issued a Ph.D. degree in Computer Science (Artificial Intelligence) in 2003. He is currently working as system administrator at ENSEEIHT, Toulouse, France. He is the leader of the ESUP-Portail storage group, also involved in CMS.

Pascal Aubry played with real-time systems at ECP until 1993. In the succeeding years, he worked at IRISA on the distribution of synchronous programs and received his Ph.D. in Computer Science in 1997. Currently at IFSIC, University of Rennes 1, he manages web-projects. He has been part of the ESUP-Portail project since its beginning in late 2002, working on web security (SSO, authorizations) and data storage.