TITLE: siLeDAP: easing interactions with directories
NETWORKING CONFERENCE 2007 PUBLICATIONS

Javier Masa, Cándido Rodríguez
RedIRIS, Edificio CICA, Avenida Reina Mercedes, s/n, 41012 Seville, Spain
e-mail: {javier.masa,candido.rodriguez}@rediris.es

Abstract
At this time, although directory services are still one of the main components in our technologic environments, they do not provide any significant toolkit or framework around them that make easier their interoperatibility with the applications, deployed in those environments. Also, when we want to apply more complex technologies, such as COPA, we could find more problems. siLeDAP has the main goal of developing not only an advanced browser/administrator of directory services, taking advantage of web 2.0, but also of providing a set of object classes and web services which allow to take advantage of all the power that directory services offer.

Keywords
directory services, web services, XML, REST, COPA

Introduction
Recently, RedIRIS has launched the siLeDAP [1] initiative with the aim of providing a more agile environment for directory servers to the community. Though the deployment in technological infrastructures continue growth, their main handicap for that positive growth is the inflexibility which this kind of platforms offers. Also, the tasks that will be done inside this project should make the integration of different technologies and schemas which are built around LDAP easier, such as COPA [2] and SCHAC [3].
In one of its tasks, siLeDAP pretends to offer a more transparent and agile communication and work environment with directory services, so this can be integrated with the distributed applications that we can find around it. This is possible thanks to the development of a set of APIs that can be used through web services. In this way, basic operations, such as getting an entry or update one, as well as complicated operations, such as make virtual views using COPA, are feasible for any application which makes HTTP requests to those web services.
Furthermore, when it comes to modifying our applications for integrating with directory services, interactions with them do not depend on the features that libraries of the selected programming language could offer, but the problem is solved using functions that API through the web services. Each function that it offers has a delimited action, almost atomic, such as add or delete an entry. Also, every response that they send are XML messages, so it makes interactions with any application easier and allows, for example, that web applications can take advantage of AJAX [4] for communicating with the directory server.

Exploring the siLeDAP API
As it is said, the siLeDAP API defines a set of functions for working with LDAP that can be called through web services interfaces. Initially, they have been developed using the REST format [5], because it is the simplest format and it makes integration with applications that are using AJAX easier. Anyway, more kinds of request and response formats are planned to include such as XML-RPC [6] or SOAP [7].
The structure of a REST request is, basically, an HTTP request using GET or POST methods, that sends parameters to a URL or to the body of the request, respectively. Each web service specifies in its definition which parameters expect to receive and their valid values.
For example, for searching in a directory, the documentation of siLeDAP specifies that the web service which provides that function expects to receive an HTTP request using the GET method with the following parameters:

- **dnbase**: mandatory parameter which sets the base 64 encoded DN at which the search will begin.
- **filter**: optional parameter which sets the base 64 encoded LDAP filter for the search.
- **scope**: optional parameter which sets if the search is in depth, with "sub", or in one level, with "one".
- **selAttr**: optional parameter which sets the base 64 encoded attributes list that we want to receive joined by comma.

In this way, an example request is shown below:

```
http://webserver/siLeDAP/API/searchldap.php?dnbase=ZGM9dW5pdixkYz1lcw==
                        &scope=one&selAttr=ZGMsb3U=
```

In other way, each REST request is answered by a REST response, which is an XML message which contains a set of specific elements for each web service. So each web service specifies what structure and content the response has. Considering the last example, the response is:

```
<?xml version='1.0' encoding='UTF-8' ?>
<Entries>
   <Entry dn="dc=areas,dc=univ,dc=es">
      <Attribute name="dc">
         <AttributeValue value="areas"/>
      </Attribute>
      <Attribute name="ou">
         <AttributeValue value="areas"/>
      </Attribute>
   </Entry>
   <Entry dn="dc=catalogos,dc=univ,dc=es">
      <Attribute name="dc">
         <AttributeValue value="catalogos"/>
      </Attribute>
      <Attribute name="ou">
         <AttributeValue value="catalogos"/>
      </Attribute>
   </Entry>
</Entries>
```

Currently, the siLeDAP API provides the following basic web services:

- **Searches**: as it was seen in the last example, searches can be done on the directory server.
- **Write operations**: it can be realized writing operations such as add, modify or remove entries.
- **Information about the schema**: it can be requested information about the schema such as definition of an objectClass or its attributes.
- **Get an entry**: it can be got an XML message with every attribute/value pair of the requested DN. In case the attribute has a binary value, we received it in a base 64 encoded string.
- **Import/export**: this web service has the capacity to import or export in LDIF format.

In this way, applications which are using this API has the following architecture:

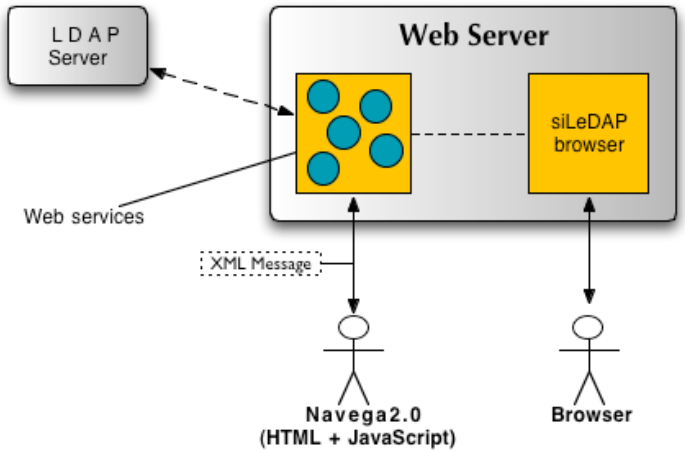

Figure 1: A general siLeDAP architecture

As it can be seen, web services are hosted in a web server which PHP capabilities, as they are programmed in that technology. These web services are not only reached through GET or POST

methods of HTTP protocol using XML messages but also they can be reached using the set of classes which defines their functionality. So, in this way, the application "Navega2.0", which is developed in HTML and JavaScript, can communicate with a LDAP server through the REST interface of the API and the browser of siLeDAP, as it is a PHP application, uses that set of classes without any HTTP request.

COPA 2.0

COPA is a coding schema that allows effective searches in hierarchical structures, at one level or recursively, and do not impose a significative complication to existing storage and retrieval systems (databases, directories,...) . The system is able to perform searches using metadata that is included in the pre-existing structure.

The basic idea is the creation of string identifiers made of the concatenation of special sets of symbols, that incorporate hierarchy position data in them. These identifiers are added to the data available for a certain element (attributes in a directory object, for example).

COPA coding uses a letter (`l`) and several numbers (`n`) per each hierarchy level, so a COPA code has an aspect like this: `lnlnlnlnln`. The letter identifies a certain level within the hierarchy and the numbers conform the identifier for an element at that level. For example, the code `a2b5c1` refers to the second node (`2`) at the first level (`a`), the fifth node (`5`) at the second level (`b`), and the first node (`1`) at the third level (`c`). This regular structure simplifies the definition of both positive and negative filters, making very easy the expression of complex searches inside the virtual hierarchy defined by the codes.

The COPA coding scheme can be applied to any hierarchical naming structure, using the COPA code at the "leaf " part of the name. Let's consider the case of uniform resource names (URNs): the above example can be part of an URN (using the prefixes assigned to RedIRIS) like this:

```
urn:mace:rediris.es:copa:classificationID:a02b05c01
```

One of the main features of COPA is that it allows the decoupling of the actual representation of the information from the view that is offered at a given time. A hierarchical classification in knowledge areas can be maintained on a flat structure or on a structure based on organizational criteria. The knowledge area hierarchy is kept by the metainformation using COPA codes. This way, the (possibly multiple) affiliation of an object to a certain area is represented in the object itself, not by means of its position in the hierarchy (as it is the common approach up to now).

A flat structure in the information tree is key for maintaining the independence of entries with respect to the position of the objects related to them, so their position(s) within the structure may be changed without actually moving them in the supporting system. A change in the corresponding attribute(s) is enough.

Let's assume a COPA-based hierarchical structure of knowledge areas, established in terms of different levels, more specific as the depth in the structure increases. The following examples shows the codes corresponding to a path from a node at the top level in the structure (`a1`) to the branch at `a1b1c8d6`:

Code	Name
a1	Sciences
a1b1	Biology
a2b1c8	Entomology
a2b1c8d6	Insect Taxonomy

Let's assume that this structure is stored in an LDAP server where all the entries corresponding to the area definitions are under `dc=areas,dc=org,dc=es`. Each entry has two attributes: the COPA code and the name of the knowledge area.

It is simple to build a program able to navigate the directory showing the hierarchical structure defined by the COPA codes: it just has to search for the areas at the same level and show them.

The first level can be covered by an LDAP filter including those entries with a code attribute starting with `a` but without `b`:

```
(&(code=a*)(!(code=a*b*)))
```

Once the entries at the first level are shown, if the user selects Sciences (`code=a1`), the filter to be applied must include those entries with a code starting with `a1b` but without a `c`, that is:

```
(&(code=a1b*)(!(code=a1b*c*)))
```

It is easy to imagine the process for the rest of levels. It is also possible to build different (and orthogonal) virtual views by means of other COPA-coded attributes representing different hierarchies.
COPA schema version 2 includes new objectClasses and attributes for LDAP client or applications to do automatic virtual navigations in directory services. The new objectClasses are:

- **copaVirtualViewNav**: it defines which virtual views can be used.
- **copaVirtualViewSpec**: it saves the configuration of a virtual view, such as where areas are stored or which attribute it is used for associating an entry with an area.
- **copaArea**: a generic object for defining an area.
- **copaResource**: a generic object for defining a resource.

As it is seen in the next figure, using COPA a virtual structure can be created in our LDAP using the real one:

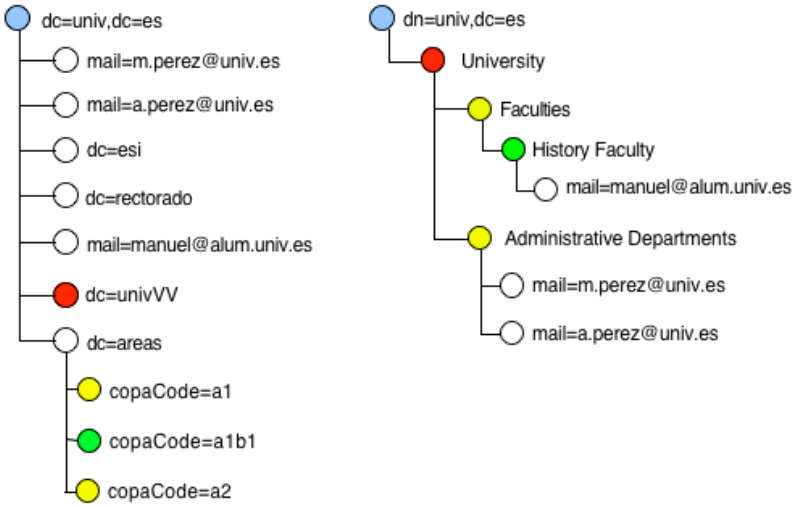

Figure 2: COPA & LDAP scenario

The siLeDAP API provides a web service which manages all operations that we need to work with the COPA technology. In fact, this is very useful, since most of them are complicated and it is not necessary to implement them in every programming language. The most relevant COPA operations that can be reached through this web service are:

- **Inits a new virtual view navigation**, which specifies that in a specific entry in LDAP there is available a new virtual view navigation.
- **Gets a list of available virtual views**, where it is specified which virtual view should be used initially.
- **Gets a list of areas**, and optionally specifying the parent area.
- **Gets data from an specific area**.
- **Gets the list of resources** (entries) that are associated for an area of a virtual view.

XML Enabled Directory

On 16th November 2006, the Internet Engineering Task Force (IETF) published some Internet-Drafts about XML Enabled Directory (XED) [8], an initiative for working in XML with directory services. The communication protocol that XED will use is XML Lightweight Directory Access Protocol (XLDAP) [9], which a first version that is semantically equivalent to LDAP version 3 [10]. XED defines two transport methods for XLDAP messages: over TCP/IP and over SOAP.
This draft in the IETF specifies the kind of operations that can be done using XLDAP messages. These are codified through Robust XML Encoding Rules (RXER) [11], which is basically an XML-based definition of the Abstract Syntax Notation One (ASN.1). Thanks to this, the ASN.1-based definition of the version three of LDAP, and all of its messages, can be defined in XML, as it can be showed in the draft of XLDAP.
At this moment, the siLeDAP team is developing a web service for the API which implements the XLDAP over SOAP protocol. So every directory service will have the capacity to communicate using

this new standard, without having to update the software or to wait for this feature.

Conclusions

Although the development of the siLeDAP project is still in an initial stage, we're already getting some positive results that they allow us to be optimistic with this initiative. The development of the web applications Navega2.0, which can navigate in a directory using only HTML and JavaScript technologies, and Navega2.0-copa, which is a more complex version and can navigate through virtual views in a directory, show us that we can programme a complete LDAP navigator requiring only a few hundred lines of code. This is possible thanks to the REST interface of web services and the API not only offers atomic and basic operations but it can be requested complex ones.

About XLDAP, we're working in a web service which implements XLDAP over SOAP 1.1, so any LDAP server can offer this technology. But, after analyzing the document presented at the IETF, we think this is not enough for making integration between applications in an heterogeneous environment and directories easier. In fact, it only defines the same messages that we can find in LDAP version 3, and they are not easy to manage and we need a set of messages for doing a basic operation.

References

[1] "siLeDAP project," https://forja.rediris.es/projects/siledap/.
[2] "COPA 2.0 schema," http://www.rediris.es/ldap/esquemas/copa.schema.
[3] "TF-EMC2 SCHAC project," http://www.terena.nl/activities/tf-emc2/schac.html.
[4] Jesse James Garret, Ajax: A New Approach to Web Applications. Available from http://www.adaptivepath.com/publications/essays/archives/000385.php
[5] R. Fielding, "Architectural Styles and the Design of Network-based software Architectures," http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
[6] "XML-RPC Specification," http://www.xmlrpc.com/spec.
[7] M. Gudgin, M. Hadley, N. Mendelsohn, J.J. Moreau, H.F. Nielsen, "Simple Object Access Protocol (SOAP) 1.2" http://www.w3.org/TR/soap12.
[8] "The XML Enabled Directory" http://www.ietf.org/internet-drafts/draft-legg-xed-roadmap-05.txt.
[9] "The XML Enabled Directory: Protocols" http://www.ietf.org/internet-drafts/draft-legg-xed-protocols-04.txt.
[10] IETF. RFC2251: Lightweight Directory Access Protocol v3, December 1997
[11] "Robust XML Encoding Rules (RXER) for Abstract Syntax Notation One (ASN.1)" http://www.ietf.org/internet-drafts/draft-legg-xed-rxer-07.txt