OFBiz Entity Engine LDAP Mapping Notes

by David E. Jones, last updated 16 May 2005

Introduction

This document details the results of an analysis for the support of LDAP in OFBiz. The initial recommendation and effort is based on the approach of using the Entity Engine as the channel for this LDAP support. This will involve configuration for structure mapping and tools based on that configuration to map queries and results from one to the other. The tools created in the scope of this effort will include accessing LDAP information through the Entity Engine (LDAP client) and Entity Engine information through LDAP (LDAP server).

Data Structure Mapping

LDAP Schema and LDIF

LDAP structure is defined by "object class" definitions. These definitions are typically found in a schema file, and there are various standard schemas available. Object class definitions consist of:

1. OID - Object ID
2. Name
3. SUP - parent object
4. Type - STRUCTURAL, AUXILIARY, ABSTRACT
5. MUST attributes - required attributes
6. MAY attributes - optional attributes

LDIF is the "LDAP Interchange Format," and is defined in RFC 2849. It is a text file representing LDAP entries and consists of a collection of entries separated by a blank line and for each entry a list of attribute name/value pairs and a collection of directives that instruct the parser about how to process the information.

Example of Standard LDAP Structure, LDIF Snippet: OpenLDAP core.schema

The first attribute in each entry in an LDIF file is typically a "dn" entry which represents the distinguishing name of the entry. The following lines are the name/value pairs of the attributes of the entry. Here is an example of an LDIF file based on the "person" object class from the core.schema file available on the OpenLDAP site:

```
dn: ou=people,dc=plainjoe,dc=org
objectClass: organizationalUnit
ou: people

dn: cn=gcarter,ou=people,dc=plainjoe,dc=org
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: gcarter
sn: Carter
givenName: Gerald
telephoneNumber: 555-555-1234
```

```
userPassword: {SHA}Fo0B4r...
```

Here is an example of the posixAccount type defined in the nis.schema file:

```
dn: cn=gcarter,ou=people,dc=plainjoe,dc=org
objectClass: posixAccount
cn: gcarter
uid: gcarter
uidNumber: 104
gidNumber: 101
homeDirectory: /home/gcarter
userPassword: {MD5}B4Rf0o...
```

XML Format for Mapping LDAP Schemas to Entity Engine Definitions

There are various things that need to be defined to create a mapping between LDAP schemas and Entity Engine entity definitions, and then based on that mapping to automatically generate entity engine operations from LDAP queries and vice-versa.

The proposal presented here will include the following elements and attributes for defining this mapping:

1. **ee-ldap-mapping** (root element)
2. **ldap-server**
    - 2.1. *name*
    - 2.2. *url*
    - 2.3. *auth-cn*
    - 2.4. *auth-password*
    - 2.5. **root-rdn** (1 to many)
        - 2.5.1. *name*
        - 2.5.2. *value*
3. **entity-map**
    - 3.1. *entity-name*
    - 3.2. *ldap-server-name*
    - 3.3. *mode* - “ldap-primary” or “entity-primary” or “ldap-only”
    - 3.4. *default-object-class*
    - 3.5. **entry-rdn** (1 to many)
        - 3.5.1. *name*
        - 3.5.2. *value* or *field-name*
    - 3.6. **field-map** (1 to many)
        - 3.6.1. *field-name*
        - 3.6.2. *attribute-name*
        - 3.6.3. *object-class* (optional)

The order of the various *-rdn elements is important and they will be linked together to form the DN. I decided to have them structured like this instead of as a big single string to avoid parsing issues because when mapping queries we'll have to form some sort of a tree of these in memory, kind of a mini-directory that is used to determine how the different nodes are mapped to the database.

The different modes are different options for reconciling between the data in the standard Entity Engine datasource, and the LDAP directory. There are basically two high level options: just use LDAP (the ldap-only option), or reconcile the data between LDAP and the EE datasource. When reconciling the data between the EE datasource and the LDAP directory there are two options on how to handle changes:

- ldap-primary: if there is a difference, update the EE datasource based on what is in LDAP

- entity-primary: if there is a difference, update LDAP based on what is in the EE datasource

Data reconciliation will be done on a "lazy" basis. In other words, when there is a query or any change operation (either through the Entity Engine or the EE external LDAP interface), depending on the LDAP setup mode, it will lookup data from one or both sources, and if applicable and if data differs it will update the data in the configured place based on the data from the other.

This is something that came up as we were talking about LDAP support a long time ago, and something that after going through this analysis seems more important because chances are you won't be able to map every field in all desired entities to a standard LDAP schema, so some additional information may still need to be tracked in the database.

This XML would go in a separate file so it can stay independent of the entity definitions. This should make it easier to customize or configure.

Example of Mapping of core.schema:person to OFBiz UserLogin/Party

```
<ee-ldap-mapping>
    <ldap-server name=”auth-server” url=”localhost” auth-cn=”admin” auth-
password=”ofbiz”>
        <root-rdn name=”dc” value=”org”/>
        <root-rdn name=”dc” value=”plainjoe”/>
    </ldap-server>
    <entity-map entity-name=”UserLogin” ldap-server-name=”auth-server”
mode=”ldap-primary” default-object-class=”person”>
        <entry-rdn name=”ou” value=”people”/>
        <entry-rdn name=”cn” field-name=”userLoginId”/>
        <field-map field-name=”currentPassword” attribute-
name=”userPassword”/>
        <field-map field-name=”partyId” attribute-name=”cn”/>
    </entity-map>
    <entity-map entity-name=”Person” ldap-server-name=”auth-server”
mode=”ldap-primary” default-object-class=”person”>
        <entry-rdn name=”ou” value=”people”/>
        <entry-rdn name=”cn” field-name=”partyId”/>
        <field-map field-name=”lastName” attribute-name=”sn”/>
        <field-map field-name=”firstName” attribute-name=”givenName” object-
class=”inetOrgPerson”/>
   </entity-map>
</ee-ldap-mapping>
```

Query Mapping

LDAP Query Definitions

LDAP Queries use a LISP-like syntax with lots of parenthesis, as defined in RFC 2254. Each condition is in parenthesis and is either a filter condition, or combination condition. The various special operators: filter operators: =,~=,<=,>=; wildcard: *; combine operators: &,|,!

```
(attribute filer-op value)
(combine-op (condition1)(condition2))
(cn=*carter)
(&(|(sn=smith)(sn=jones))(cn=john*))
```

When doing a search there are various things passed in, and the filter is just one of them:

1. Search Base (DN to search below)
2. Search Scope (base only, one level, all sub-levels)
3. Search Filter (as described above)
4. Attributes to Retrieve (can use "*" for wild card)
5. Types Only (return just attributes names, or names and values)

Adding and Modifying LDAP Entries

When an LDAP Entry is added the full DN plus the set of attributes to be associated with it are sent to the LDAP server. This is in essence the same information that is shown in the LDIF examples above. LDIF files can often be imported right into an LDAP server, though through the Java API these are split up into various objects. For JLDAP these objects include LDAPAttribute, LDAPEntry, etc.

API/Protocol Support

Use of the JLDAP Java LDAP Client (http://www.openldap.org/jldap/)

The JLDAP library is an open source Java LDAP client available from Novell. It can be downloaded and various documentation is available here: http://developer.novell.com/ndk/jldap.htm.

In addition to the option of using a more classic LDAP client such as this, we could also consider accessing the LDAP server through one of the LDAP JDBC drivers, either from OctetString or Novell. For more control over how it works, an LDAP client would be best so the only possible reason to use the LDAP JDBC driver would be to simplify things, but if it does not handle certain requirements it just won't work anyway.

As JNDI is a sub-set of LDAP, for this particular tool it means that we may not be able to access the entire directory or all of the features of a LDAP directory through JNDI. So, rather than using JNDI to access the directory we will use a Java LDAP client as described here.

Use of the Apache Directory Project (http://directory.apache.org)

The Apache Directory Project may eventually have more in it, but for now it is primarily an LDAP/JNDI server written in Java. With its built-in extension mechanism it is well suited to providing the infrastructure needed to create the tool that will map LDAP operations to the Entity Engine, or in other words allow an Entity Engine driven LDAP server.

ApacheDS LDAP Server API

To create a custom operation handler for the ApacheDS server you simply extend the org.apache.ldap.server.ContextPartition class. There is brief documentation about this at: http://wiki.apache.org/directory/ApacheDirectoryServer.

Conclusion and Estimates

This document presents various concepts to be used for the mapping between an LDAP server and the OFBiz Entity Engine, and going in both directions for that. There are various open source tools that can be leveraged for this work, and the tools appear to be somewhat mature. These tools are really what will make this effort possible.

The actual implementation of this can be done in 2 parts, though some of the code will be shared between them (especially model classes for the ee-ldap-mapping XML files). These 2 parts would be: 1. using an LDAP server to replicate or replace the data stored in a database for certain fields on certain entities (EE to LDAP mapping, recommend using JLDAP client), and 2. using the EE to store data and act as the back-end for an LDAP server (LDAP to EE mapping, recommend using ApacheDS server).

The Entity Engine API as it is now would not change, and using LDAP entries in addition to SQL records would be managed through the configuration described here. In a way there would be some additional interfaces added, namely those that allow access to Entity Engine data through an LDAP server interface.

Based on the analysis done that was the basis for this document, my estimate is that a competent engineer should be able to implement both of these tools in approximately 60-80 hours. This would include implementation of the tools, creation of test mappings and performance of tests to validate the tool in a couple of scenarios.

References

LDAP System Administration by *Gerald Carter* (O'Reilly 2003)

Apache Directory Project: http://directory.apache.org

Novell JLDAP Java LDAP Client: http://www.openldap.org/jldap/

Novell JLDAP Documentation (great for general concepts): http://developer.novell.com/ndk/doc/jldap/pdfdoc/jldapenu/jldapenu.pdf

LDAP JDBC Driver: http://www.octetstring.com/products/BridgeDriver.php

Novell LDAP-JDBC Driver: http://developer.novell.com/ndk/ldapjdbc.htm