OpenObject as LDAP data-source

As a side-effect of Migrating to Claws I lost my OSX addressbook usage, at least for mail. In my company we use OpenERP for CRM, invoicing and other business needs. This means that the majority, if not all the email-addresses I need are in our OpenERP database.

So, it made sense to finish an effort I started earlier, which is to link OpenERP to our LDAP server and thus be able to query information from the OpenERP database through an LDAP interface and have every address available in the Claws addressbook (and the OSX addressbook too for that matter).

In an earlier version of OpenERP we used a specific module for this, which basically published an LDAP entry whenever we changed data in the OpenERP database. This worked, but was a less than ideal solution, not to mention it stopped working on an OpenERP upgrade. The solution I wanted was to have the data available in LDAP immediately. This meant making the LDAP server a "client" of the OpenERP database or, said another way, making the OpenERP database function as an SQL backend to the LDAP server.

Defining an SQL backend for LDAP Creating an SQL backend for slapd

is, albeit terse, documented. I mainly used the information at theOpenLDAP FAQ. The basic idea is that the LDAP-server connects through ODBC to the database, the OpenERP database in our case, and translates information found in relational tables to a subtree of the LDAP hierarchy.

To model this information, you have to create at least 3 tables in the database:

1. `ldap_oc_mappings`: which objectClass of LDAP is stored in what table;
2. `ldap_attr_mappings`: how attributeTypes of an objectClass are resolved from RDBMS data;
3. `ldap_entries`: what's the DN of an entry, and how the entry relates to its objectClass mapping and to its parent DN;

The FAQ mentions two other tables, which we do not need. The idea is to specify in these three tables how the LDAP server gets to the entries and what they mean. For addressbook-like entries for mail, the defacto objectClass to use for this is something referred to an 'inetOrgPerson'. The collection of these

objects will be below an objectClass 'organizationalUnit', giving the simplest 'tree-relation' we can think of.

Making the SQL backend use OpenERP

The table `ldap_oc-mappings` is queried by the LDAP server to map these object classes to tables, so the server knows in which tables to look for the attributes of these two classes. The next two statements insert two rows in that table, one for each object Class, mapping them to the tables `ldap_inetOrgPerson` and `ldap_organizationalUnit`, expecting a column `id` to contain the primary key for the objects.

```
INSERT INTO ldap_oc_mappings(name,keytbl,keycol)
    VALUES('inetOrgPerson','ldap_inetOrgPerson','id');
INSERT INTO ldap_oc_mappings(name,keytbl,keycol)
    VALUES('organizationalUnit','ldap_ organizationalUnit','id');
```

The `ldap_inetOrgPerson` is actually a view over the `res_partner_address` table in OpenERP, so it uses the data directly.

```
CREATE OR REPLACE VIEW ldap_inetorgperson AS
  SELECT
    a.id,
    btrim((COALESCE(a.firstname,'')||' ') || a.lastname) AS cn,
    btrim((COALESCE(a.firstname,'')||' ') || a.lastname) AS displayname,
    a.phone AS telephonenumber,
    a.lastname AS sn,
    a.firstname AS givenname,
    a.fax AS facsimiletelephonenumber,
    a.mobile,
    a.private_phone AS homephone,
    lower(a.email) AS mail,
    a.street,
    a.zip AS postalcode
  FROM res_partner_address a
  WHERE
    a.email  ''  AND
    a.email  '"' AND
    a.type  =  'contact';
```

This gives a dataset of all people who actually have an email-address registered in the OpenERP database. The column aliases are not needed as such, but make the construction of the `ldap_attr_mappings` table a bit easier. The second objectClass we registered in `ldap_oc_mappings`, `organizationalUnit` can be modelled with one simple row in the table `ldap_organizationalUnit`:

```
INSERT INTO ldap_organizationalUnit(name) VALUES ('addressbook');
```

With that row, we basically define one organizational unit in our simple tree

named `addressbook` under which all our objects of type `inetOrgPerson` will be placed.

So, at this point we have 2 objectClasses registered, we have created the raw data for them. What's left? Two things, first, we need to define how the attributes of the raw data relate to the object attributes. For this, the table `ldap_attr_mappings` contains a row for each attribute. For the `telephoneNumber` attribute, the data row is as follows:

```
INSERT INTO ldap_attr_mappings(
        oc_map_id,name,sel_expr,from_tbls,join_where
    )
VALUES(
    1,'telephoneNumber','telephoneNumber','ldap_inetOrgPerson','1=1'
);
```

This says basically to the ldap server that in order to get to the `telephoneNumber` attribute for `inetOrgPerson` (`oc_map_id` 1 refers to the first row in the `ldap_oc_mappings` table), it needs to look in the table `ldap_inetOrgPerson`, use the same attribute name for the column and apply no special where clause. It is basically a recipe for the server to translate an LDAP request into an SQL query.

For each of the columns in the `ldap_inetOrgPerson` view, such a row needs to be present in the `ldap_attr_mappings` table.

Still here? The final step is to create the third meta table `ldap_entries`. This table is basically the lookup table to map ldap-index values to rdbms-index values. I have defined `ldap_entries` as a view on the raw data as follows:

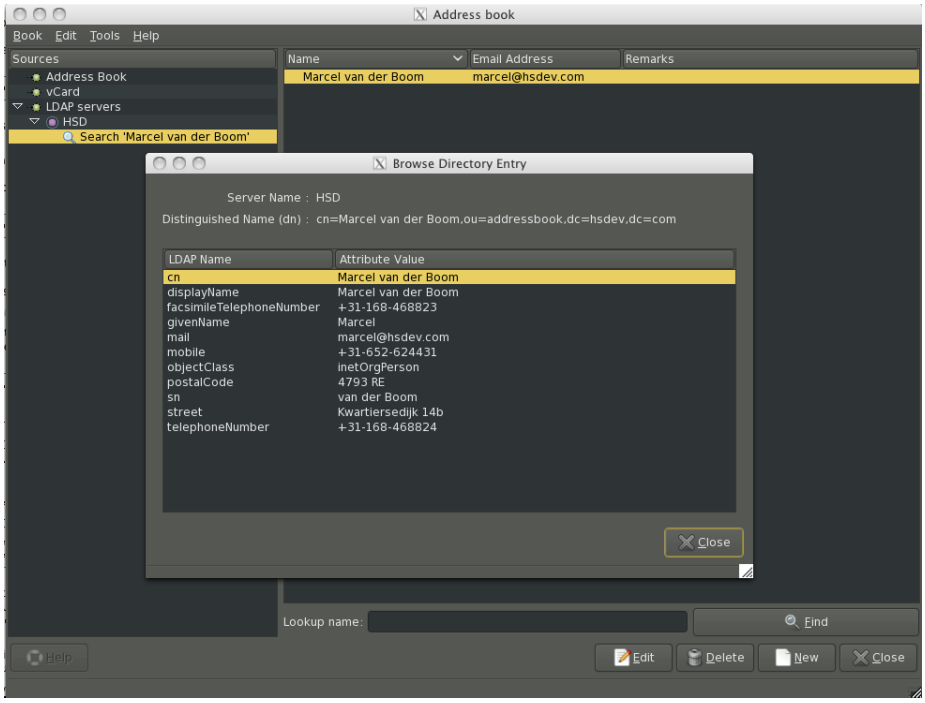
```
CREATE OR REPLACE VIEW ldap_entries AS
  SELECT
    0 AS id,
    'ou=addressbook,dc=hsdev,dc=com' AS dn,
    2 AS oc_map_id,
    0 AS parent,
    0 AS keyval
   UNION
  SELECT
    ldap_inetorgperson.id,
    ('cn=' || ldap_inetorgperson.cn) ||
    ',ou=addressbook,dc=hsdev,dc=com' AS dn,
    1 AS oc_map_id,
    0 AS parent,
    ldap_inetorgperson.id AS keyval
  FROM ldap_inetorgperson;
```

This does 2 things, it refers the ldap addres `ou=addressbook,dc=hsdev,dc=com` as the organizational unit (`oc_map_id` = 2) and assigns that ID 0. The second

part of the UNION then formats each of the inetOrgPerson addresses as an LDAP address in the constructed tree (mine would be: `cn=Marcel van der Boom,ou=addressbook,dc=hsdev,dc=com`) and maps it to the ID in the raw data table (`keyval`) and puts the organizational unit as its parent.

With the above a minimal proof of concept can be constructed so that each partner address which has an email-address shows up in LDAP. After this, it's a matter of configuring the email-client using the `ou=addressbook,dc=hsdev,dc=com` as a search base.

Here's a picture of my record in the claws address book:



The same concept can be applied to other data in OpenERP (the partner records themselves come to mind or user accounts). It would not be that hard to wrap the above into an OpenERP module to manage this. Once the LDAP server has a configuration to use an SQL backend, all configuration can be done in OpenERP itself, modelling access using the meta tables. Perhaps I'll do that at some point, if some of my customers would benefit from this too.