Rationale and Initial Experience

The drivers for the implementation of Grouper at the University of Memphis are the implementation of a new ERP and Exchange. We cutover to our Grouper-based group registry in October 2007.

Our old 'group registry' was an LDAP directory with a vendor provided web UI to manage groups. These groups were used primarily for email, and to a lesser extent authorization.

The need for an improved group registry first became apparent with the arrival of Exchange. Initially we cron'd Perl scripts to synchronize our LDAP groups to Active Directory. The most significant shortcoming was that membership was difficult to maintain consistently because not all of the members of a group in the group registry had accounts in AD, primarily due to licensing costs for AD related software. After rolling out a provisioning system and a person registry in Java to replace most of our Perl scripts, it was time for a new group registry.

In general, our groups have three membership types: members, owners, and senders. Each membership type may consist of people, other groups, or email addresses. We have three types of groups: manual, managed, and courses. Manual groups are maintained solely by people. Managed groups are based on business logic, largely roles within the University. Courses are based on student course data. Automatic processes maintain managed and course groups, with some exceptions.

Ideally, our notion of a group would support both positive and negative membership assertions. For example, the membership of the group "All Students" would be maintained automatically based on affiliation with the University, but we may need to add a few members who fall outside of the automatic process (such as a VP who wants a copy of all email), or remove them (a student who dropped out but still shows up in our source systems for whatever reason).

I attempted to implement both positive and negative membership assertions using Grouper's custom lists, but I didn't understand Grouper enough to be successful. Apparently, parent-child membership relationships are not supported for custom lists as they are with the default 'members' list. I then looked at using group math, based on the 1.0 composite group tutorial, which meant creating 6 or so groups for every group: manual-adds, managed-adds, manual-removes, includes, excludes, and finally, the resulting composite. With our three types of membership (members, owners, and senders) we would have 18 Grouper groups per group! The compromise for now was to toss out negative assertions. We support manually adding members to a managed group using a custom 'includes' list. When a member is 'included', they are added to the 'members' list. When a member is removed from a group, membership in the 'includes' list is checked and the member can only be removed if they are not a member of the 'includes' list. This way we support adding exceptions to our automatically maintained groups.

We have integrated Grouper into our provisioning system, which currently targets LDAP directories, including AD. We may soon provision groups to Luminis (the front end to our Banner ERP) using their API. We decided to provision groups to target LDAP directories in a 'flat' structure instead of hierarchically, meaning that we don't include other groups as members of group. Some applications support nested groups, but others don't, so we opted for flat groups.

Many of our most important large email groups were dynamic LDAP groups representing affiliation with the university, e.g. All Students. To implement those groups in Grouper, we use an external process to update membership. Our person registry includes the notion of a 'role', such as 'student', whose logic is encoded as Java code in an XML file interpreted by BeanShell. Whenever a person's data changes in a source system, we calculate roles based on the changed attributes, update Grouper, and trigger our provisioning system to update provisioned resources (such as AD) appropriately.

Regarding code, we opted to wrap the Grouper Group class with our own

```
public class edu.memphis.idm.Group {
private edu.internet2.middleware.grouper.Group group;
...
```

to provide a group API matching our business rules while leveraging Grouper as a group repository.

We opted not to use the Grouper UI because we thought it was too complicated for most of our clients (stem - what's a stem?). So, we have kept our old web based UI, and whenever a group is changed in our LDAP directory we update Grouper. We are working on a simple UI using Rails and JRuby; when ready, groups will be sourced in Grouper.

Our infrastructure consists of approximately 1,000 manually maintained groups (many of which are quite stale, e.g. tjb-test), 6,000 course groups, and 2,000 managed groups representing university affiliation. We maintain ~25,000 active user accounts. We're running Grouper on a Sun V490 with postgres. We are running 1.2.1, which we found is much faster than 1.2.0.