Grouper Subject API Report

GFIVO Project Team
The University of Newcastle upon Tyne
Claremont Tower, Newcastle upon Tyne, NE1 7RU, England

September 2007

Contents

1 Introduction

The GFIVO (Grouper for Federated Identity management in Virtual Organisations) project, recently undertaken by staff at Newcastle University, aims to utilise the Grouper system to manage group access to web based resources (such as Blogs/Wikis). This report aims to summarise the project team's findings on the Grouper Subject API, from experiences with both Quickstart (v1.2.0 RC4) and Production (v1.2.0) versions of Grouper. In particular, it will focus on the potential of the Subject API for institutional use and its compatibility with existing data sources and architectures.

2 Experiences with the Subject API

Here are the findings, collated from different members of the team, gained from using the Grouper system to populate subjects/groups and from using the UI to login, search for and browse these subjects/groups (through the User Interface (UI)/Grouper Shell (GSH) and raw database editing).

2.1 Compatibility with Existing Architectures

Newcastle University's web presence uses MySQL 5 databases; by default, MySQL 5 is not supported by Grouper but, with a few modifications to configuration files, (`grouper.hibernate.properties`, mentioned here, and the `sources.xml` - see Section 2.3) the Grouper system can be customised to interact with a MySQL backend.

The `grouper.hibernate.properties` file simply requires a section to be added for MySQL database support which reads:

```
# MySQL
hibernate.dialect = net.sf.hibernate.dialect.MySQLDialect
hibernate.connection.driver_class = com.mysql.jdbc.Driver
hibernate.connection.url = jdbc:mysql://<HOST>/<DATABASENAME>
```

After this, the MySQL driver needs to be placed in the `/lib` and `/dist/lib` directories of the Grouper API. After this, by following the standard setup instructions for Grouper, a MySQL database can be used with the Grouper system.

Despite using a MySQL 5 database, we could not use the MySQL5 dialect of Hibernate as this is only included in v3 and the Grouper system currently uses v2.2. We tried to implement a work-around, by downloading the source of the MySQL5 dialect and compiling it into the Hibernate 2.2 jar file, but the differences between the Hibernate versions made this impossible and the standard MySQL dialect seems adequate for the task.

2.2 Performance

We had heard of other institutions having performance issues with the Grouper system when dealing with large numbers of users/groups. Our first experiences with the Quickstart seemed to confirm these but our problems were eventually fixed by altering the search query in `sources.xml` (see Section 2.3.1). Once we started using the Production release of Grouper, we approached the task of importing "real" users/groups (using the GSH and UI) and interacting with the system as a standard user would.

2.2.1 Populating Groups

We manually inserted our 28,000 strong user base into the grouper Subject and SubjectAttribute tables and then went about the process of adding them into sample groups using GSH scripts. Adding 121,000 group memberships for our user base took 62 minutes (roughly 2000 a minute, or 30 a second). This is impressive performance and the GSH was very usable (given our teams' skill set).

The skill set required to use the GSH effectively is very much that of a "sysadmin". The GSH is comparable to other shells (BASH etc) and it's use of Java method calls requires an amount of programming knowledge (for example,

understanding why "strings" are in quotes where as other string-like values don't require them.

e.g.

```
grantPriv(‘‘ncl:Central Services:iss:Customer Services:Web Services’’,
‘‘ncf17@ncl.ac.uk’’, AccessPrivilege.ADMIN)
```

not

```
grantPriv(‘‘ncl:Central Services:iss:Customer Services:Web Services’’,
‘‘ncf17@ncl.ac.uk’’, ‘‘AccessPrivilege.ADMIN’’).
```

This kind of "mass import" would, realistically, perhaps only be carried out during the initial setup of the Grouper system in an institution and be performed by users with appropriate skill sets. Later, additional groups would be added on an "ad-hock" basis, by standard users, using the UI and, although we have tested this functionality, we don't have metrics on this yet other than to say there are no noticeable performance issues highlighted.

2.2.2 Searching/Browsing Groups in the UI

Part of the mass import of groups mentioned above involved replicating all mailing lists held at Newcastle University into groups in the Grouper system. As a result, the "emaillists" stem contains 1600 groups. Opening this stem in the UI (as a user, to see your groups within the stem) takes 10 seconds from clicking on the stem title. By default, the Grouper system displays the first 50 groups it discovers. Clicking "next" to view groups 51-100 takes another 10 seconds - and so on until the end. This makes browsing through the groups in a such a large stem difficult (we don't, however, anticipate users being members of more than 50 groups in one stem).

Reducing the default number of groups to display from 50 to 10 (for example) does not alter the speed at which the groups are displayed. The page takes the same amount of time to display and there is no "windowing" whereby 50 results would be saved but only 10 displayed, so clicking "next" would simply load the next 10 from the results already stored. This is due to there being no windowing in the API, so all 1600 groups are returned to the UI each time the query runs; the UI just chooses to show 50, or 10, or the chosen number to be displayed.

2.2.3 Conclusions

Direct interaction with the database from Grouper (via the GSH) is powerful and relatively quick. The UI interaction is a little slower, especially where stems have many groups inside them. Regardless whether a user is a member of 1 or 100 groups, the page takes the same amount of time to load and should a user need to click "next", to see the following page of results, they will have to wait for the same time again.

Interestingly, from the Grouper "home" page, clicking "Hide stem hierarchy and show your groups only" takes less than 1 second, and shows all the group memberships for the user logged presently logged in, whereas showing all

the user's memberships within a stem takes 10 seconds (as mentioned above). Analysis of the MySQL query logs shows that showing a users entire group membership runs roughly 500 queries (and takes roughly 1 second) whereas showing the user's memberships inside a stem runs over 21,000 queries (and takes roughly 10 seconds). We have raised this issue on the Grouper mailing lists, suggesting adding a method to the API to "retrieve a user's group memberships", and it is being looked into as part of the discussions regarding the improvement of Grouper's performance.

2.3 Changes to sources.xml

As documented in the Grouper setup instructions, changes to the `sources.xml` need to be made to represent the database connection settings for your system. These are logical, although sligtly repetitious of the changes to other configuration files - `grouper.hibernate.properties` (see Section 2.1).

At Newcastle, we found the search queries in the `sources.xml` to be a little confusing, as described below in Sections 2.3.1 and 2.3.2.

2.3.1 Quickstart

In the Quickstart version we used (v1.2 RC4), search queries involved computationally complex "LEFT JOINS" on tables. Using the sample data provided with the Quickstart, this was not a problem; however, when we came to enter our own data (around 7000 users) the search queries took approximately 10 minutes and occupied 99% of CPU usage throughout this period.

Changes were made to the database structure: firstly, to separate out the "SubjectAttribute" table into three new tables "SubjectLogin", "SubjectDescription" and "SubjectName" (each with a common primary key) and then the queries in `sources.xml` were adapted to use these pre-formed tables rather than creating them "on the fly" using left joins. This change was discussed in detail on the GFIVO project blog at `http://gfivo.ncl.ac.uk/blog.php`.

This high CPU usage gave a bad impression of Grouper performance. However, after we noticed this was only a problem with the Quickstart version of the sources.xml (see Section 2.3.2), we carried on our investigations without worrying about this performance issue. Other institutions investigating the Grouper system may have been put off at this stage, however.

2.3.2 Production

In the Production (v1.2) of Grouper we are using, the search queries in `sources.xml` are much shorter than those found in the Quickstart and didn't monopolise the CPU for large periods of time as perviously encountered. However, we did find problems with the searches firstly when logging in as a user from the Subject table and then when searching for another user in this table.

The login process, by default, uses the `name` attribute of the `Subject` table (as the `searchSubjectByIdentifier` search block says "`select * from`

`Subject where (name=?)`" - which seems illogical as "searchSubjectByIdentifier" doesn't use the "subjectId" attribute). This is fine for the Grouper sample data (e.g. subjectId = "hawi", name = "hawi", description = "hawi") but when using real user data, using the "name" attribute for logins is inappropriate. This is especially true in our case as we intend to use federated identity management, in the form of Shibboleth, whereby the username is scoped i.e. user@newcastle.ac.uk.

This problem was corrected by changing the `where` clause in the "searchSubjectByIdentifier" search block to read "`where (subjectId=?)`"; this in turn created another problem. Although users could log into the Grouper system using their "subjectId" the search functionality was broken. As the search does something behind the scenes with the search query, searching for a person returned the error:

```
Unexpected error - edu.internet2.middleware.subject.SubjectNotFoundException:
subject not found:  <subjectId>,person
```

This problem was discussed on the grouper users email list and was fixed by changing the search query for both the `searchSubjectByIdentifier` and `searchSubject` search blocks' queries to:

```
select * from Subject where (name=?)  or (subjectId=?)
```

and changing the "numParameters" param-value to 2 (this is because the numParameters value does not set the number of parameters to be received by the query but, rather, sets the number of times the one parameter (?) can be used in the query).

2.3.3 Conclusions

Changes to the `sources.xml` file are a requirement for setting up the Grouper system (as described in the Grouper setup instructions). However, we found the changes in search queries between Quickstart and Production versions of the system meant that migrating to the Production system was as time consuming as originally setting up the Quickstart (as the `sources.xml` file was so markedly different).

The changes we made to the search blocks required a sysadmin's understanding of XML and databases, as well as monitoring of error and database log files to monitor system activity. This level of knowlege, however, made other problems, such as the `searchSubjectByIdentifier` not using the "subjectId" attribute (which would seem more logical on some levels) and the `numParameters` value meaning "the number of times the parameter can be used" not "the number of parameters to be passed in", harder to comprehend.

Further documentation on the `sources.xml` file and its nuances would be useful to system developers looking to customise Grouper for their own institutional needs.

3 Overall Conclusions

Our use of the Grouper system and investigation into the Subject API has introduced us to the development side of the Grouper system and engaged us with the Grouper community (using the Grouper mailing lists) while providing us with useful information to continue the project. Here, we report our institutional view of the Subject API as well as concluding on our findings from working with the Subject API and setting out our future work plans.

3.1 Our view of the Subject API

In Figure 1 (Page 7) we illustrate our institutional view of the Subject API. We intend to use the pre-existing Shibboleth identity management system at Newcastle University to inform the Grouper system. The Grouper system will, in turn, interact with the MySQL database backend to store and retrieve group information. This group information will then be used to control access to web based applications through the creation of `.htaccess` files.

Utilising the existing Shibboleth middleware system allows for an enhanced user experience in the use of Single-Sign-On. Using MySQL as a backend means the existing infrastructure within the University can be utilized and generating `.htaccess` files removes the need for potentially costly server restarts.

3.2 Final Conclusions and Future Work

The Grouper system was not designed specifically for use with MySQL databases. Due to the use of MySQL databases, in Newcastle University's web based content and applications, we have decided to make the necessary modifications to the configuration files to allow us to pursue development with a MySQL backend.

The performance of the GSH is promising, although the response time of the UI when faced with stems containing more than, or users belonging to more than, 100 groups is disappointing. This situation is, in reality, quite rare and unlikely (that one user will belong to more than 100 groups) and, at worst, is avoidable (by further dividing up stems and so reducing the load on the system).

The current level of documentation regarding the configuration files for Grouper (in particular the `sources.xml` file) is minimal. The changes in the files - particularly the search queries - between Quickstart and Production versions do not assist in the deployment of the system. Because of these factors, deploying the system at an institution requires a high level of knowledge of the existing systems in place and the ability to adapt previous knowledge/learn new skills quickly.

Having deployed the Grouper system and imported our user data, we now intend to set up self service and centrally administered groups for access to web based tools such as Wikis and Blogs. We will report on this phase early in the New Year.

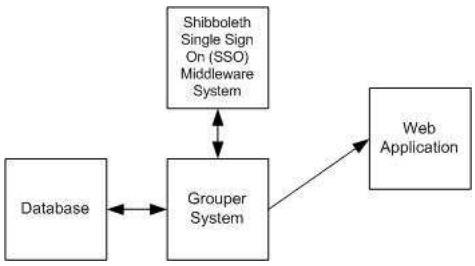

Figure 1: Our view of the Subject API