Hibernate and data layer updates

Update to Hibernate 3 Instructions

This is now committed to grouper HEAD. Here are the steps to start using Hibernate3:

1. Sync and get the new jars: hibernate3.2.6, i2micommon, asms, cglib, c3p0, ehcache, backport-util-concurrent. Remove old jars (e.g. hibernate.3.2.5)

2. Change the grouper.properties to use hib3 dao factory

```
#dao.factory=edu.internet2.middleware.grouper.internal.dao.hibernate.HibernateDAOFactory
dao.factory=edu.internet2.middleware.grouper.internal.dao.hib3.Hib3DAOFactory
```

3. Change the grouper.hibernate.properties to use the hib3 ehcache, and hib3 dialect (in this example, mysql, but change this to whatever DB driver type you are using)

```
#hibernate.dialect                = net.sf.hibernate.dialect.MySQLDialect
hibernate.dialect                = org.hibernate.dialect.MySQL5Dialect

#hibernate.cache.provider_class       = net.sf.hibernate.cache.EhCacheProvider
hibernate.cache.provider_class       = org.hibernate.cache.EhCacheProvider
```

4. You must start using c3p0 database pooling (this is the only one we unit test with grouper with). This means changing the grouper.hibernate.properties (feel free to set the c3p0 pool settings as you see fit. Below is a safe version which should perform fine, but you can tune it to err on the side of performance if you like:

```
# Use DBCP connection pooling
#hibernate.dbcp.maxActive             = 16
#hibernate.dbcp.maxIdle               = 16
#hibernate.dbcp.maxWait               = -1
#hibernate.dbcp.whenExhaustedAction    = 1

# Use c3p0 connection pooling (since dbcp not supported in hibernate anymore)
# http://www.hibernate.org/214.html, http://www.hibernate.org/hib_docs/reference/en/html/session-configuration.html
hibernate.c3p0.max_size 16
hibernate.c3p0.min_size 0
#seconds
hibernate.c3p0.timeout 100
hibernate.c3p0.max_statements 0
hibernate.c3p0.idle_test_period 100
hibernate.c3p0.acquire_increment 1
hibernate.c3p0.validate false
```

5. Check your log4j.properties, if you have TRACE log on hibernate, change to ERROR. If you have net.sf.hibernate, might want to change to org.hibernate. Otherwise ignore.

```
log4j.logger.org.hibernate                          = ERROR, grouper_error
```

Proposed objectives

1. Upgrade to the latest version of hibernate, which is now 3.2.5
2. Add consistency to how hibernate and db resources are handled
3. Add support for transactions

Hibernate upgrade

- Blair has already added support for hibernate. When I ran the unit tests with hib3, and half of them failed, but maybe after a little work, many would be fixed
- In order to take advantage of hib3, the properties files needs to be changed regarding: dao.factory, hibernate.cache.provider_class, hibernate.dialect
- Hib2 will be experimental after the switch to hib3, but it is not recommended. It will not support transactions. (GrouperTransaction)
- Any add-ons which use hibernate might need to be tweaked

Hibernate / DB resources

- This goes hand in hand with the transaction support, so let's discuss this too
- Error handling in JDBC and hibernate is repetitive, tedious, and easy to get wrong. There are cases in grouper where the error handling is not done optimally or consistently
- Here is an example

```
public Map findAllAttributesByGroup(String uuid)
    throws  GrouperDAOException
  {
    Map attrs = new HashMap();
    try {
      Session hs  = Hib3DAO.getSession();
      Query   qry = hs.createQuery("from Hib3AttributeDAO as a where a.groupUuid = :uuid");
      qry.setCacheable(false);
      qry.setCacheRegion(KLASS + ".FindAllAttributesByGroup");
      qry.setString("uuid", uuid);
      Hib3AttributeDAO a;
      Iterator              it = qry.iterate();
      while (it.hasNext()) {
        a = (Hib3AttributeDAO) it.next();
        attrs.put( a.getAttrName(), a.getValue() );
      }
      hs.close();
    }
    catch (HibernateException eH) {
      throw new GrouperDAOException( eH.getMessage(), eH );
    }
    return attrs;
  }
```

- If there is an exception thrown before the hs.close(), then the session will not be closed, which could leak DB connections or leave resources locked on the database
- If we use inverse of control (like how Spring might do it), then we can handle this centrally. Here is another block we can change:

```
public void addType(GroupDTO _g, GroupTypeDTO _gt)
    throws  GrouperDAOException
  {
    try {
      Session     hs  = Hib3DAO.getSession();
      Transaction tx  = hs.beginTransaction();
      try {
        hs.save(  // new group-type tuple
          new Hib3GroupTypeTupleDAO()
            .setGroupUuid( _g.getUuid() )
            .setTypeUuid( _gt.getUuid() )
        );
        hs.saveOrUpdate( Rosetta.getDAO(_g) ); // modified group
        tx.commit();
      }
      catch (HibernateException eH) {
        tx.rollback();
        throw eH;
      }
      finally {
        hs.close();
      }
    }
    catch (HibernateException eH) {
      throw new GrouperDAOException( eH.getMessage(), eH );
    }
  }
```

Here is how it would look with inverse of control

```
public void addType(final GroupDTO _g, final GroupTypeDTO _gt)
    throws  GrouperDAOException {

    HibernateSession.callbackHibernateSession(HibernateTransactionType.READ_WRITE_OR_USE_EXISTING,
        new HibernateHandler() {

          public Object callback(HibernateSession hibernateSession) {
            Session hs = hibernateSession.getSession();
            hs.save(  // new group-type tuple
                new Hib3GroupTypeTupleDAO()
                  .setGroupUuid( _g.getUuid() )
                  .setTypeUuid( _gt.getUuid() )
              );
            hs.saveOrUpdate( Rosetta.getDAO(_g) ); // modified group
            //let HibernateSession commit or rollback depending on if problem or enclosing transaction
            return null;
          }

    });
  }
```

- Note that the session does not need to be opened or closed, it is in a callback, which is provided in an anonymous inner class. So the proper exception handling and resource handling will be done everywhere
- Here is a readonly example:

```
public GroupDTO findByName(String name)
    throws  GrouperDAOException,
            GroupNotFoundException
  {
    try {
      Session hs  = Hib3DAO.getSession();
      //TODO CH 20080209 Change this to be one query, not two to attribute then group
      Query   qry = hs.createQuery("from Hib3AttributeDAO as a where a.attrName = 'name' and a.value = :value");
      qry.setCacheable(false);
      qry.setCacheRegion(KLASS + ".FindByName");
      qry.setString("value", name);
      Hib3AttributeDAO a = (Hib3AttributeDAO) qry.uniqueResult();
      hs.close();
      if (a == null) {
        throw new GroupNotFoundException("Cannot find group with name: '" + name + "'");
      }
      return this.findByUuid( a.getGroupUuid() );
    }
    catch (HibernateException eH) {
      throw new GrouperDAOException( eH.getMessage(), eH );
    }
  }
```

Here is how the code would look with inverse of control (notice how data is passed in and out of the anonymous block (final params, and a return object)

```
public GroupDTO findByName(final String name)
    throws  GrouperDAOException,
            GroupNotFoundException
  {
    Hib3AttributeDAO hib3AttributeDAO = (Hib3AttributeDAO)HibernateSession.callbackHibernateSession(
           HibernateTransactionType.READONLY_OR_USE_EXISTING,
        new HibernateHandler() {

          public Object callback(HibernateSession hibernateSession) {
            Session hs  = hibernateSession.getSession();
            //TODO CH 20080209 Change this to be one query, not two to attribute then group
            Query   qry = hs.createQuery("from Hib3AttributeDAO as a where a.attrName = 'name' and a.value = :
value");
            qry.setCacheable(false);
            qry.setCacheRegion(KLASS + ".FindByName");
            qry.setString("value", name);
            Hib3AttributeDAO a = (Hib3AttributeDAO) qry.uniqueResult();
            return a;
          }
    });
    //this throws exception, keep out of
    GroupDTO groupDTO =  hib3AttributeDAO == null ? null : Hib3GroupDAO.this.findByUuid( hib3AttributeDAO.
getGroupUuid() );
    //handle exceptions out of data access method...
    if (groupDTO == null) {
      throw new GroupNotFoundException("Cannot find group with name: '" + name + "'");
    }
    return groupDTO;
  }
```

For simple database actions (mainly single actions), there is a helper framework in HibernateSession to remove the need for inverse of control:

Here is the same block (note, transaction level can be configured, but there is an intelligent default):

```
public GroupDTO findByName(final String name)
    throws  GrouperDAOException,
            GroupNotFoundException {

    Hib3AttributeDAO hib3AttributeDAO = HibernateSession.byHqlStatic()
      .createQuery("from Hib3AttributeDAO as a where a.attrName = 'name' and a.value = :value")
      .setCacheable(false)
      .setCacheRegion(KLASS + ".FindByName")
      .setString("value", name).uniqueResult(Hib3AttributeDAO.class);

    //this throws exception, keep out of
    GroupDTO groupDTO =  hib3AttributeDAO == null ? null : Hib3GroupDAO.this.findByUuid( hib3AttributeDAO.
getGroupUuid() );
    //handle exceptions out of data access method...
    if (groupDTO == null) {
      throw new GroupNotFoundException("Cannot find group with name: '" + name + "'");
    }
    return groupDTO;
  }
```

Similarly for object based queries, there are helper classes/methods. Here is an example:

```
public String create(GrouperSessionDTO _s)
    throws  GrouperDAOException
  {
    try {
      Session       hs  = Hib3DAO.getSession();
      Transaction   tx  = hs.beginTransaction();
      Hib3DAO  dao = (Hib3DAO) Rosetta.getDAO(_s);
      try {
        hs.save(dao);
        tx.commit();
      }
      catch (HibernateException eH) {
        tx.rollback();
        throw eH;
      }
      finally {
        hs.close();
      }
      return dao.getId();
    }
    catch (HibernateException eH) {
      throw new GrouperDAOException( eH.getMessage(), eH );
    }
  }
```

This code will look like this:

```
public String create(GrouperSessionDTO _s)
    throws  GrouperDAOException {

    Hib3DAO  dao = (Hib3DAO) Rosetta.getDAO(_s);
    HibernateSession.byObjectStatic().save(dao);
    return dao.getId();
  }
```

- I suggest that this is the only way to get a Session object so that we ensure it is done correctly
- The inputs to the Hib3SessionHandler describe the session returned. If it is readonly the performance can be dramatically increased (Penn has seen this, I can measure to make sure Grouper will reap the benefits also)
- The syntax and tricks to anonymous inner classes can be new and different to programmers who have not used them, but I think with proper documentation and examples they are easy to get the hang of (all the Penn Java developers use them now just fine)

Add support for transactions

- Lets take the use case of web services in batch mode, where the caller wants to add a few new groups, and if one fails, they should all fail. This is not currently possible
- To support transactions we can use the inverse of control described above
- We can support four modes: GrouperTransactionType.READONLY_OR_USE_EXISTING, READONLY_NEW, READ_WRITE_OR_USE_EXISTING, READ_WRITE_NEW
    - READONLY_OR_USE_EXISTING: even if in the middle of a transaction, create a new read/write autonomous nested transaction. If this block is exited normally it will always commit. If exception is thrown, it will always rollback.
    - READONLY_NEW: even if in the middle of a transaction, create a new readonly autonomous nested transaction. Code in this state cannot commit or rollback.
    - READ_WRITE_OR_USE_EXISTING: use the current transaction if one exists. If there is a current transaction, it MUST be read/write or there will be an exception. If there isnt a transaction in scope, then create a new read/write one. If you do not commit at the end, and there is a normal return (no exception), then the transaction will be committed if new, and not if reusing an existing one. If there is an exception, and the tx is new, it will be rolledback. If there is an exception and the tx is reused, the tx will not be touched, and the exception will propagate.
    - READ_WRITE_NEW: even if in the middle of a transaction, create a new read/write autonomous nested transaction. If this block is exited normally it will always commit. If exception is thrown, it will always rollback.
- Note that you can nest transactions to any level (actually there is a hard stop at 15 since I cant picture actually wanting to nest that deep, and I want to try to detect if the ThreadLocals are acting up)
- When we know the ThreadLocals should be empty, we should clear them. e.g. at the start of an HTTPRequest in web services or UI: HibernateSession.resetAllThreadLocals()
- To code the above use case, it would look something like this (note there is an implicit complete if no exception, and rollback if there is an exception, but you can control this if you like with GrouperTransaction methods):

```
/**
   * run multiple logic together
   * @param grouperSession
   * @param groupName
   * @param groupName2
   * @param displayExtension
   * @param groupDescription
   */
  public void runLogic(GrouperSession grouperSession, String groupName,
      String groupName2, String displayExtension, String groupDescription) {
    try {

      //insert a group
      Group.saveGroup(grouperSession, groupDescription, displayExtension, groupName,
          null, SaveMode.INSERT, false);

      //insert another group
      Group.saveGroup(grouperSession, groupDescription, displayExtension, groupName2,
          null, SaveMode.INSERT, false);
    } catch (StemNotFoundException e) {
      throw new RuntimeException("Stem wasnt found", e);
    } catch (Exception e) {
      throw new RuntimeException(e);
    }

  }

  /**
   * show simple transaction
   * @throws Exception if problem
   */
  public void testTransaction() {

    final GrouperSession rootSession = SessionHelper.getRootSession();
    final String displayExtension = "testing123 display";
    final String groupDescription = "description";
    final String groupName = "i2:a:testing123";
    final String groupName2 = "i2:b:testing124";

    try {
      R.populateRegistry(2, 2, 0);

      GrouperTest.deleteGroupIfExists(rootSession, groupName);
      GrouperTest.deleteGroupIfExists(rootSession, groupName2);
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
    //demonstrate passing back data with final array
    final Integer[] someInt = new Integer[1];

    //you can pass back one object from return
    String anythingString = (String) GrouperTransaction
        .callbackGrouperTransaction(new GrouperTransactionHandler() {

          public Object callback(GrouperTransaction grouperTransaction)
              throws GrouperDAOException {

            //everything in here will run in one transaction, note how to access "this"
            TestGroup0.this.runLogic(rootSession, groupName, groupName2,
                displayExtension, groupDescription);

            //pass data back from final array (if need more than just return value)
            someInt[0] = 5;

            //if return with no exception, then it will auto-commit.
            //if exception was thrown it will rollback
            //this can be controlled manually with grouperTransaction.commit()
            //but it would be rare to have to do that

            //pass data back from return value
            return "anything";
          }

        });
    System.out.println(anythingString + ", " + someInt[0]);
  }
```

- In this case it is not readonly (defaults to READ_WRITE_OR_USE_EXISTING, but can be controlled), and a transaction wrapper is passed to the callback (GrouperTransaction). There is a method called "GrouperTransaction.commit()" (takes param which can be only if new or always) which will determine if the transaction originated in this callback, or is from a ThreadLocal from an outer transaction. If it originated here, it will commit. Else not.
- If you can picture the code in the addGroup() method, it will look similar... read/write transaction, with USE_EXISTING, and "commitIfNewTx()" which when called from here will not commit.
- Since we are supporting nested transactions, there can be a ThreadLocal stack of hibernate sessions which can be accessed from anywhere and not need to be passed around to all method signatures.
- All 87 places that use Hib3DAO.getSession() were refactored. It will be tedious but should be low impact and should improve the quality of the code
- The higher impact changes to the code relate to how hibernate is handled:
    1. Since the Session (from Hibernate) can only deal with one object by key at a time, each time an object is retrieved from hibernate it must be "evicted" from the session. The ByObjectStatic and ByHqlStatic helper methods will do this, and if "Query.list()" for example is called outside of the helpers, then the results must be evicted. The side effect is that no hibernate dirty checking will be available. This is no problem since that is more pain than it is worth
    2. If a transaction type is a NEW transaction type, then at the end of the HibernateSesssion callback, it will be committed or rolled back (and Session object is discarded, and state is synced with the DB). However, if it is a "USE_EXISTING" transaction type, then at the end of the HibernateSession, the Session (from Hibernate) will start bulking up. So HibernateSession will flush() (write all queries to the wire), and clear() (take all objects in the Session out). This will make the transactions work