Goal

The Graffito subproject "jcr-mapping" is an object/JCR persistence and query service. This tools lets you to persist java objects into a JCR compliant repository - including association, inheritance, polymorphism, composition, and the Java collections framework. Furthermore, this jcr-mapping allows you to express queries in Java-based Criteria, as well as in JCR query language.

In order to easily support the JCR specification, any CMS application based on an high level object model can use this framework. For example, a classic Forum application contains interfaces like "Forum", "Topic" and "Post". The object implementations are data objects that can be used with persistence tools like Hibernate, OJB, ... Now the same data objects can be managed by our JCR mapping tools to persist them into a JCR compliant repository.

How the persistence engine is working ?

The persistence engine uses a mapping file describing for each classes how to map its attributes. This mapping file contains also information on object associations, inheritance mapping strategy, lazy loading, cache strategy, ...

There are 4 attributes/fields "types" :

1. Simple fields : primitive data types and simple objects (String, Long, Double, ...) . Thoses fields are mapped into JCR properties.
2. Bean fields : One class can contain an 1..1 association to another bean. In this case, the attribute is a custom object. Those fields are mapped into JCR nodes.
3. Collection fields : One class can contain an 1..n association to a collection of beans (or Map). Those fields are mapped into a collection of JCR nodes.
4. Reference field : One good example to understand the "reference" type is the Folder concept. A folder "B" can have an attribute called "parentFolder" which is a simple field pointing to the parent folder "A" . Of course, in a JCR repository, it is a nonsense for persist this "parentFolder" attribute into a "B" subnode. Another interesting example are links made between cms objects (folders, documents, ...).

Of course, all those "mapping types" imply different mapping algorithms.

Framework Design

The JCR mapping framework is composed of the following components :

- The mapping model component

This component reads the mapping file and keeps it in memory for fast use. it uses some Java objects that basically represent the mapping xml file.

- The persistence component

It would be a quite big component which cares about searching, updating, inserting and so on. This is the core component in the framework.

    - subcomponent: object to item converter

Converts an object structure starting with the given object to a node structure while using the instructions from the mapping model.

    - subcomponent: item to object converter

Converts a node structure starting with the given node to an object structure while using the instructions from the mapping model.

    - subcomponent: persistence manager

It calls the JCR functionality with the help of the above converters. E.g. for "public void insert (String path Object anObject);" it calls the object to item converter and creates the resulting node out of "anObject". After that it is easy to insert this node with the JCR functionality. Or for "public Object findById(Object anId)" it uses the JCR functionality to find the nodes. With the item to object converter it should be possible to create the objects out of the nodes found by the JCR functionality.

    - subcomponent: atomic type converter / primary type converter : it uses to convert JCR property type into atomic java type.

- subcomponent: cache

Note:

- Maybe the search features would be defined in another components. Otherwise, the persistence component will be a little bit heavy.
- We have to check how to implement the cache component (AOP ?). Maybe the Spring module project can help.

Example

Here is the small example which explains how the tools is working.

The object graph to persist

```
public class A
{
    private String a1;
    private String a2;
    private B b;
    private Collection collection; // Collection of C
    /* Add here the getter/setter */
}
public class B
{
     private String b1;
     private String b2;
     /* Add here the getter/setter */
}
public class C
{
     private String id;
     private String name;
}
```

The mapping file

```
<graffito-jcr>
    <class-descriptor className="org.apache.portals.graffito.jcr.testmodel.A" jcrNodeType="nt:unstructured" >
         <!-- Field-descriptor is used to map simple attributes to jcr property -->
         <field-descriptor fieldName="a1" jcrName="a1" />
         <field-descriptor fieldName="a2" jcrName="a2" />
                   <bean-descriptor fieldName="b" jcrName="b" proxy="false"  />
                  <collection-descriptor fieldName="collection" jcrName="collection" proxy="false"
               fieldId="id" className="org.apache.portals.graffito.jcr.testmodel.C" />
    </class-descriptor>
    <class-descriptor className="org.apache.portals.graffito.jcr.testmodel.B"  jcrNodeType= "nt:unstructured" >
                  <field-descriptor fieldName="b1" jcrName="b1" />
         <field-descriptor fieldName="b2" jcrName="b2" />
    </class-descriptor>
  <class-descriptor className="org.apache.portals.graffito.jcr.testmodel.C"  jcrNodeType= "nt:unstructured" >
         <field-descriptor fieldName="id" jcrName="id" />
         <field-descriptor fieldName="name" jcrName="name" />
     </class-descriptor>
</graffito-jcr>
```

The java code that show the API

```
A a = new A();
a.setA1("a1");
a.setA2("a2");
B b = new B();
b.setB1("b1");
b.setB2("b2");
a.setB(b);

C c1 = new C();
c1.setId("first");
c1.setName("First Element");
C c2 = new C();
c2.setId("second");
c2.setName("second Element");


Collection collection = new ArrayList();
collection.add(c1);
collection.add(c2);

a.setCollection(collection);


// Create a new object (assigned to the "/test" path)
persistenceManager.insert("/test", a);

// Get the object associated to the "/test" path)

a = (A) persistenceManager.getObject(A.class, "/test");



// Update the object

a.setA1("new value");
B newB = new B();
newB.setB1("new B1");
 newB.setB2("new B2");
a.setB(newB);

persistenceManager.update("/test", a);
```

The Mapping File structure

Our prototypes could help us to review and finalize the DTD/Schema. If needed, this structure can be changed.

```
<!ELEMENT graffito-jcr (class-descriptor*)>

<!--

    Class descriptor - Each class descriptor describes the mapping strategy used for one a java class

    *className : the className

    * jcrNodeType : the primary jcr node type, it can be nt:unstructured

-->

<!ELEMENT class-descriptor (field-descriptor*, bean-descriptor*, collection-descriptor*)>

<!ATTLIST class-descriptor

                className CDATA #REQUIRED

                jcrNodeType CDATA #IMPLIED >

<!--

    Field descriptor - A field descriptor maps one atomic object attribute (primitive types, String, Long, ...)  into a JCR property

    * fieldName : the field/attribute name

    * jcrName : the jcr property name (optional). If it is not defined, fieldname is used to specify the jcr property name

-->

<!ELEMENT field-descriptor EMPTY>

<!ATTLIST field-descriptor

                fieldName CDATA #REQUIRED

                jcrName CDATA #IMPLIED

>

<!--

    Bean descriptor - A bean descriptor maps one object attribute (primitive types, String, Long, ...)  into a JCR node.

    * fieldName : the field/attribute name

    * jcrName : the jcr node name (optional). If it is not defined, fieldname is used to specify the jcr node name

    * proxy : Use lazy loading or not. if true, this attributes is not loaded when the main object is retrieved. it will be loaded when the get method is called.

-->

<!ELEMENT bean-descriptor EMPTY>

<!ATTLIST bean-descriptor

                fieldName CDATA #REQUIRED

                jcrName CDATA #IMPLIED

                proxy (true | false) "false"

>

<!--

    Collection descriptor - A collection descriptor maps one object attribute based on a collection (or a map)  into a series of JCR nodes.

    * fieldName : the field/attribute name

    * jcrName : the jcr property name (optional). If it is not defined, fieldname is used to specify the jcr node name

    * proxy : Use lazy loading or not. if true, this attributes is not loaded when the main object is retrieve. it will be loaded when the get method is called.


-->

<!ELEMENT collection-descriptor EMPTY>

<!ATTLIST collection-descriptor

                fieldName CDATA #REQUIRED

                jcrName CDATA #IMPLIED

                proxy (true | false) "false"

                clasName CDATA #REQUIRED

>
```

External tools

The following tools could be very useful for some users but are not mandatory. They should be external applications that can be used in other projects. They can be executed from ant, maven or a more advanced application.

- The JCR Node Type Registration Tools :
  - Registers new JCR node types from an xml file.
  - Registers the classes as node types while using the mapping model.
- The Class Generation Tools : Generates classes out of given node types while using the mapping model to know how the classes should be generated.

JCR Node Type Registration Tools

Since the node type management is out of the current JCR specification scope, we have to support different implementations (one for each supported JCR server).

Each implementation can call the API proposed by the JCR server and import an xml file containing the new node types to register. This xml file can look like the "custom_nodetyes.xml" proposed by the Jackrabbit project.

So, there are 2 steps to register node types :

- Create a custom_nodetypes.xml from our mapping file (and the Java introspection).

- Use the appropriate implementation to import them into the jcr server.

Open issues - Planning

- Review the mapping file structure.
- Review the component interfaces & API. We are almost agree on the component design. It should be nice to review together the component API.
- Support for secondary node type.
- Component frameworks ? Spring ?
- Support for : interfaces, inheritance, the complete collection framework.
- Lazy loading
- Versionning
- Search
- Security - Fine grained access control
- cache (2 level ?, tx mangement)
- Transaction management (with/without the cache)