Hibernate prototype

This work is based on DAO-Prototype, made by James Rutherford, and it's now syncronized with the current code in trunk of version 1.5. The code is at this link.
The idea is to improve the architecture, solving some problems in the implementation.

Problems

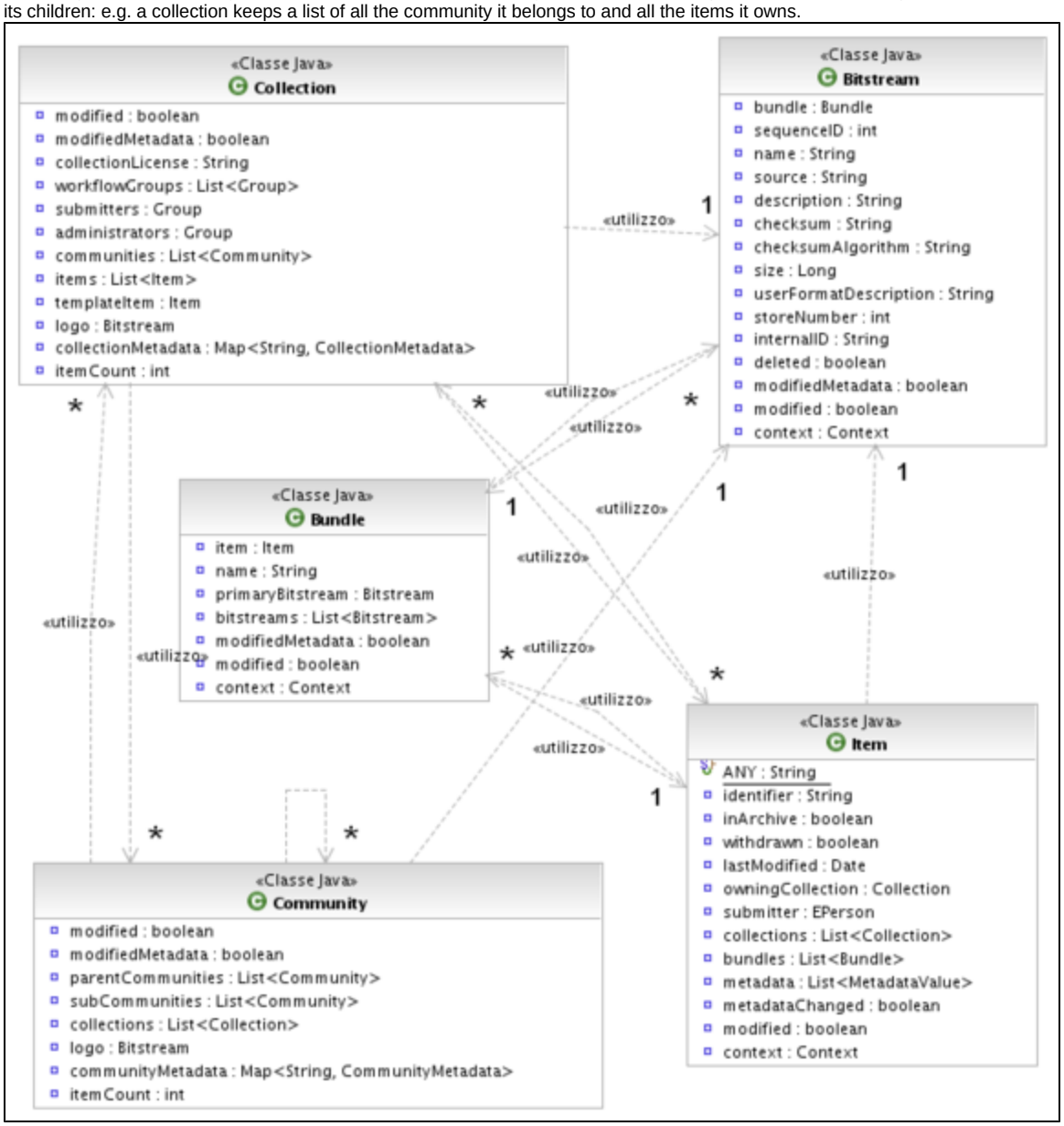
The highest-level problem is the high coupling between Business Logic Layer and Storage Layer: DAO objects, that are part of the persistence level, perform logic and are massively used by model objects, so the Business Level is very much dependent from the Storage Level.
This problem can be separated in different problems, moreover there are several other architectural problems.

- Bad distribution of responsabilities: DAO objects implement business logic, (e.g. they create model objects). A DAO should only retrieve information from the db to present data in the Storage Layer, and shouln't be an active object which partecipates in the implementation of business logic.
- Bond between Java objects and db rows: when creating a model object, there are 2 queries in the db, a create to have an empty entry line in the db, and then, after setting all the properties, an update to save effectively the properties of the object in the db. This is bad because we need two queries when only one would be tightly necessary, we keep the db connection busy between the object creation and the object settings (that could require extra time), we can't effort db structure with not null constraints.
- Fragmented logic: business logic is implemented by DAOs, by model objects and by managers (e.g. ArchiveManager). We want to centralize the business logic to have a central point to offer API.
- Visibility of Storage Layer: every module of Application Layer can see DAOs. This is bad, every level should see only the underlying level, Application Layer should use only Dspace Public API, and not objects of Storage Layer. Moreover, Business Logic Layer shouldn't know the structure of Storage Layer, DAO object could be transparent and their operations wrapped.

The proposed solution follows a Domain-Driven approach: Business Logic Layer and Storage Layer have been revised.

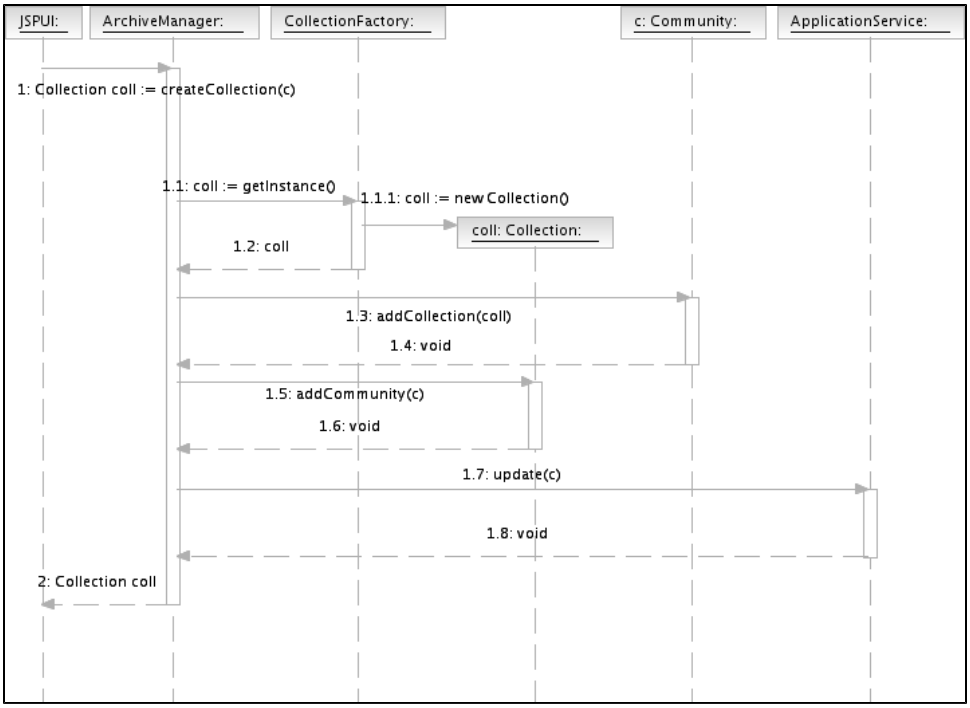
Business Logic Layer

To revise this layer two J2EE patterns have been used: Session Facade and Application Service.
Model objects have now only attributes, and no behaviour: the only methods are setter/getter ones. A model object knows its relations with its fathers and its children: e.g. a collection keeps a list of all the community it belongs to and all the items it owns.



Logic is now owned only by managers, no more logic owned by DAOs. The old managers have been invested with more responsabilities, and new managers with specific purposes have been created.
These are some important changes:

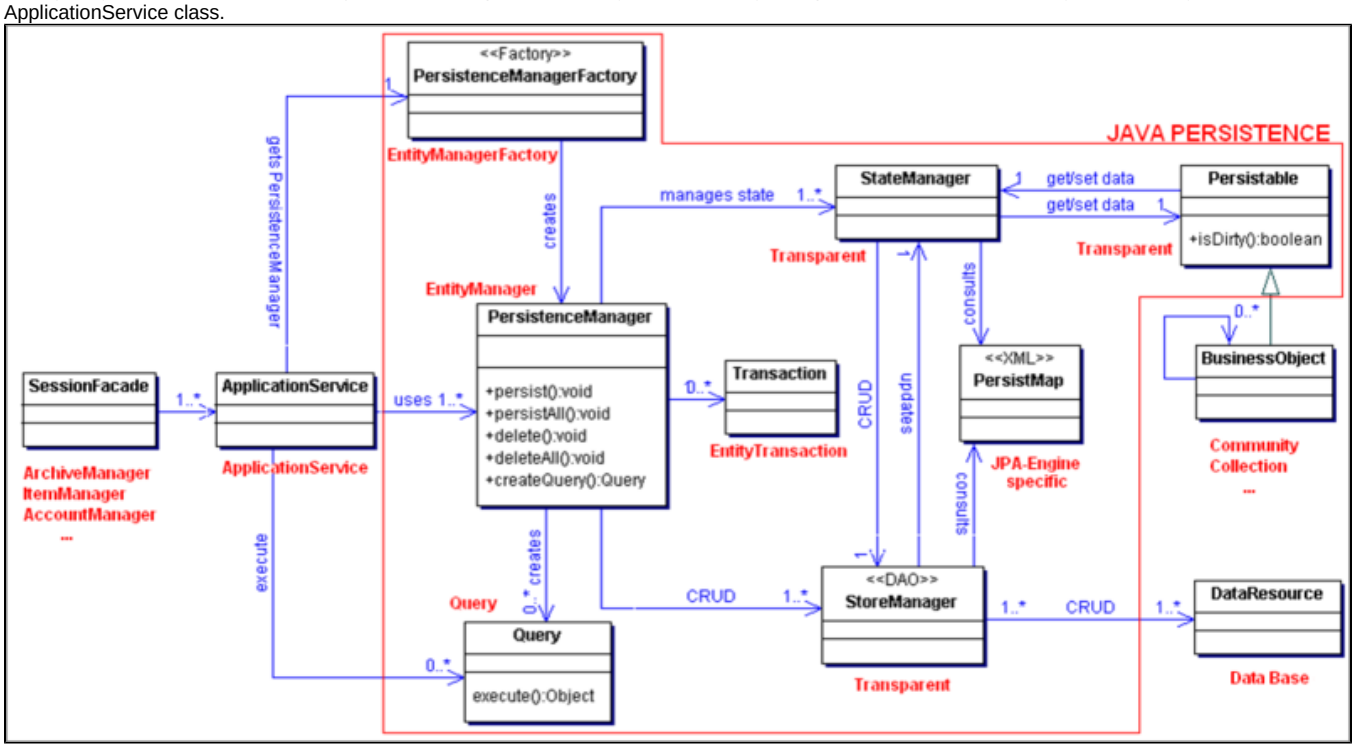
- ArchiveManager has now the responsability of managing Community, Collection and Item: it manages links between these objects, creates and removes them. It keeps all the old functionalities.
- ItemManager is a new manager: it manages the internal part of an item, bundles and bitstreams.
- AccountManager manages EPerson and Group: creations and removals, plus some method for InProgressSubmission objects. Old responsabilities are kept.
  The Dspace Public API are now only methods of the managers.
  To create model objects, factory classes have been introduced: to create, eg, a Collection, ArchiveManager invokes CollectionFactory (no more *dao.create*)



It's important to notice the separation between the creation of a Java object and its persistent representation in the DB: managers and factories works only on the object, and do not know anything about persistent fields. The Business Logic Layer doesn't know how CRUD operations against the db are performed.
To manage persistence, in the managers' point of view, there is an appropriate class, ApplicationService. ApplicationService is a manager that performs persistence operations, it offers "find" and "findAll" methods, and other ones to perform CRUD operations. When a manager needs to save (persist) an object, it will call the *save()* method of ApplicationService, from the point of view of the Business Logic Layer, DAO objects do not exist, there's only a manager for persistence. ApplicationService is therefore an interface between Business Logic Layer and Storage Layer, and its methods represent the Storage API.

Storage Layer

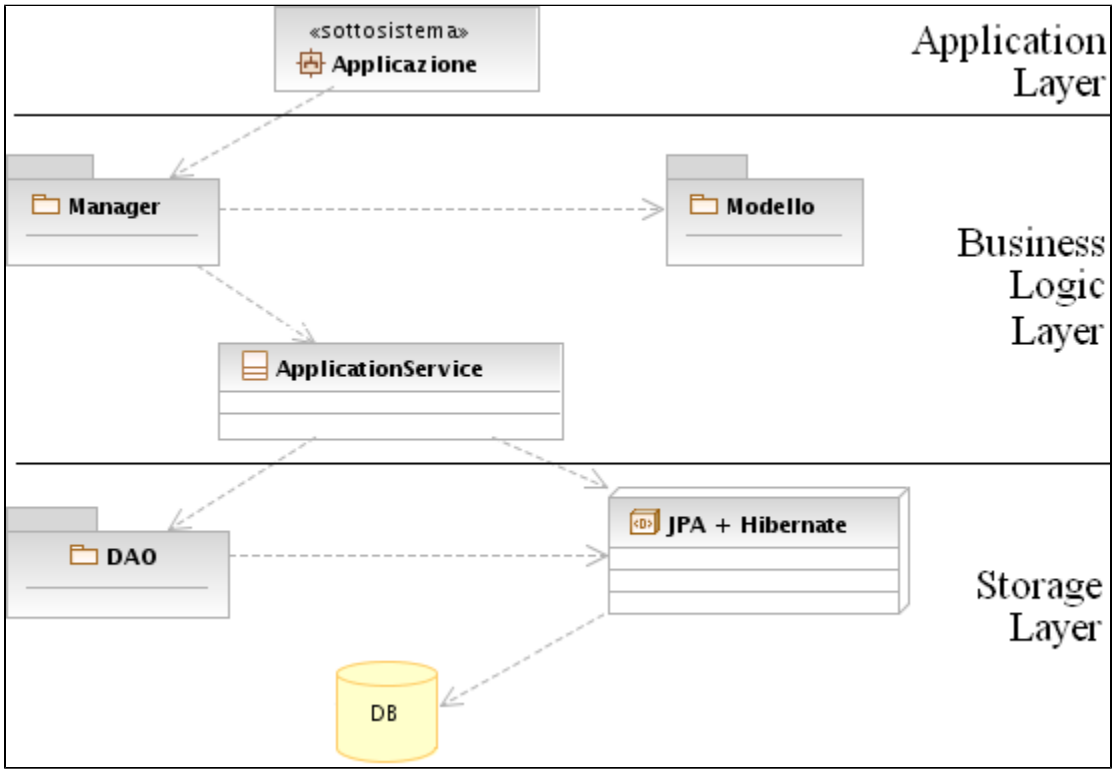
Storage Layer have been revised using Domain Store pattern. This layer is now totally transparent, because the other layers know only its interface, ApplicationService class.



To implement Domain Store, Java Persistence (JPA) has been used, with Hibernate as JPA Engine. All model objects have been mapped with JPA Annotations: grounding on these mappings, the DB Schema is automatically generated, there are no DBMS-specifical directives. Only standard JPA annotations have been used, so it's really easy to change JPA-Engine (e.g. TopLink).
Simple CRUD operations are responsability of JPA's EntityManager, while research operations and complex queries are performed by DAOs: DAO objects have now no logic, they perform only queries to "access the data".
The methods of ApplicationService just call EntityManager or a particular DAO.

New architecture

What follows is a high-level view of the architecture of the prototype:



Current status

- There's still some DAOs that implement logic: in particular, Browsing DAOs (e.g. BrowseDAO) and URI DAOs (e.g. ObjectIdentifierDAO). That logic should be moved away from these DAOs, maybe creating an appropriate manager.
- Many lines of code about Logging and Authentication have been commented, actually these two services do not work: it is necessary to reintroduce both.
- JSPUI has been adapted to the new API offered, but its working has not been tested yet, because it's necessary to have logic out of all DAOs before.
- JSPUI is the only interface "supported" right now.
- In general, code is "unclean", many comments are missing and many parts are commented and should be removed. Many operations that have been removed should be re-introduced, marked as deprecated.
- Most of tests are missing: many of the complex queries of the DAO have not been tested, so are many tests about managers' methods.