Refactoring the DSpace Domain Model

Introduction

To continue evolving towards DSpace 2.0 goals this year, @mire will continue to provide further refactorings of the DSpace legacy codebase. These refactoring efforts are focused now on resurrecting and completing the the DAO prototype work of past Hewlett Packard developer, James Rutherford. This work includes efforts to separate the legacy DSpace codebase into a separate Domain Model and Data Access Services. Once integrated into DSpace 1.8, new DSpace Services for legacy DSpace objects will provide a common API for both the DSpace Application tier and external addon modules.

(reference).

1. Naming is critical in software domain modeling. There are two qualities that you should be looking for in a good name for an entity in your model:
    a. Consistent. Business and developers should always use the same term to define a single concept.
    b. Exact. A name should exactly define the responsibilities of an entity in the domain model.
2. It should only contain POJOs. The responsibilities assigned to an entity in the domain model should be limited to only those necessary for the software domain.
3. The domain model shouldn't change because the underlying persistence implementation or the UI or any other layer changes.
4. Domain Models should not be dependent on the mechanism used to persist them.
    a. SQLException or any other storage level object or response should not be exposed in the domain.
    b. DSpaceObject classes should not be hardcoded to SQL storage calls or directly to DatabaseManager.
5. A software domain model should be easy to understand for a business domain model expert and it shouldn't have any discrepancy.
6. There are two reasons why you want to make sure that the code quality of the domain model is top-notch:
    a. The domain model is the foundation of your application. If it is bad, the whole thing can fall apart.
    b. It changes a lot. There are very few areas in your code that are going to change so many times as the domain model.

Architecture: A typical enterprise application architecture consists of the following four conceptual layers:

- User Interface (Presentation Layer): Responsible for presenting information to the user and interpreting user commands.
- Application Layer: This layer coordinates the application activity. It doesn't contain any business logic. It does not hold the state of business objects, but it can hold the state of an application task's progress.
- Domain Layer: This layer contains information about the business domain. The state of business objects is held here. Persistence of the business objects and possibly their state is delegated to the infrastructure layer.
- Infrastructure Layer: This layer acts as a supporting library for all the other layers. It provides communication between layers, implements persistence for business objects, contains supporting libraries for the user interface layer, etc.

from: http://www.infoq.com/articles/ddd-in-practice

Separating Domain Model from Data Access Services

Historically, application developers have struggled with presence of both state-full and stateless data access methods residing on the same concrete classes of the DSpace domain data model (Item, Bitstream, etc). Separation of the data access ORM implementation completely from the domain data model will allow applications to rely on those data models without being bound contractually to a specific implementation. With the application tier of DSpace no longer bound directly to the data access tier, DSpace application developers will gain an ability to provide their own customized implementation of business and data access services, reducing the need to directly alter core DSpace data model classes or resort to the ugly practice of class-path overrides.

Examples

Example	Old Form	New Form

WorkspaceItem Example	`Context context = new Context();`
	`WorkspaceItem wi = WorkspaceItem.create (context, col, true);`
	`context.commit();`
		`DSpace dspace = new DSpace();`
		`Workspace ws = dspace.getSingletonService (Workspace.class);`
		`WorkspaceItem workspaceItem = ws.create(col, true);`
		
		`// Create Item in someone elses workspace`
		`// ws.create(col, eperson, true);`

WorkflowManager Example	`Context context = new Context();`
	`WorkflowManager.start(context, wi);`
	`context.commit();`
		`DSpace dspace = new DSpace();`
		`Workflow workflow = dspace.getSingletonService`
		`(Workflow.class)`
		`workflow.start(wi);`
		
		`// Starting workflow for another user`
		`// workflow.start(wi, eperson);`

Collection Edit Example	`Context context = new Context();`
	`Collection col = Collection.find`
	`(context, id);`
	`col.setName("Some text.");`
	`col.update();`
	`context.commit();`
		`DSpace dspace = new DSpace();`
		`CollectionService cs = dspace.getSingletonService`
		`(CollectionService.class);`
		`Collection col = cs.find(id);`
		`col.setName("Some text");`
		`cs.update(col);`

End of Context?!

In the above examples, it is clear that the context object has been dropped and a new object called "DSpace" acts as an entry point into the entire Services Spring Application Context. This includes services capable of tracking and managing the current state of the transactional capabilities which the Context object originally provided. The ServiceManager Services apply to one or more levels of the Application Stack:

- Persistence Tier: Services can be provided that interact with the storage level. These services will be aggregated to form a solution for persistence of the Domain model into one or more alternative storage platforms.
- Domain Services Tier: Expose Domain Model Objects and provide Business Services that can perform important actions abstracted from both the Persistence and Presentation Tiers.
- Presentation Tier: One or more "Applications" responsible for receiving requests from clients and controlling presentation of results.

The Legacy DSpace Context is now internalized, stored within the DSpace request and provided via a Legacy Context Service when applications still require direct acces to it. It is no longer the responsibility of the Application developer to track the context and its state; any open context will be committed or aborted at the end of the cycle for the current application request.

However, if the application developer does need access to the context for legacy purposes, it will be available via the Service manager.

```
public interface IContextService {

     public IContext getCurrentContext();

}
```

```
DSpace dspace = new DSpace();
Context context = dspace.getSingletonService(IContextService.class).getCurrentContext();
```

API Contracts for the DSpace Domain Model

Extraction of the DSpace Data Model as an API from the DSpace core libraries provides a contract for application developers to rely on when developing enduser applications for DSpace. Additional API on core DSpace business and data access services (workflow and data access) will assure that applications can rely on these services contractually while the underlying implementations are improved and alternate implementations begin to emerge. Past prototyping in the DSpace 2.0 and GSoC projects have verified that, once properly restructured, integration with alternative storage implementations including options such as Fedora, JCR Repositories, DuraCloud and Semantic Storage systems such as Tupelo will be greatly eased. An initial Prototype of this API now resides in the dspace-core project and we will be providing JIRA tasks and patches to trunk that will reflect what are some relatively minor changes to the org.dspace.content and core packages to support it.

Stage 1: DSpace 1.8

The DSpace Domain Model API differs from the DSpace 2.0 Entity model in that it continues to express the buisness model of the DSpace 1.x application. To continue to utilize the DSpace 1.x DSpaceObject content model as the Legacy implementation of persistence, the first draft of the Domain Model does not attempt to deviate from the original. Instead, we only seek to separate the "Interface" for the instances of model objects from the static methods that provide for its persistence.

In DSpace 1.8 the following tasks will be completed:

1. Application code that uses the Domain Model will be refactored to utilize the new API where possible.
2. Services will be created to support update.
3. The folowing Classes will be migrated to dspace-model:
    a. org.dspace.authenticate.AuthenticationMethod.java
    b. org.dspace.authorize.AuthorizeException.java
    c. org.dspace.core.Constants.java
    d. org.dspace.event.Consumer.java
4. DCValue and MetadataValue will get API (public fields in Java classes are inappropriate).
5.

Stage 2: DSpace 1.9

In DSpace 1.9 we will seek to complete two tasks against the new Domain Model:

1. We will move all calls to static DSpaceObject methods to use DAO Services delivered by the ServiceManager, thus DSpaceObjects will become POJOs (Plain Old Java Objects).
2. We will refactor all applications to use the new Interfaces when working with the object model rather than direct.

Preparing DSpace for Integration with Fedora

Given that these changes will open the door for DSpace to be capable of supporting multiple alternate underlying implementations, @mires work in refactoring DSpace in this area will set the stage for integration with Fedora.

Based on recent work by Google Summer of Code students in prototyping integration with Fedora, the tractability of combining the applications has been shown. Initial implementations of data access services with Fedora will be capable of mapping between the DSpace and Fedora domain models, and where appropriate, this mapping will be customizable. Ultimately, it will be possible to persist DSpace Items within Fedora while making little to no changes to the features of DSpace end user applications.

Roadmap

Based on the strategies outlined above, a roadmap for the next two years of DSpace development can be charted. Continued refactorings to separate out the Data Model and Access Services will best be accomplished over two major revisions of DSpace. A first iteration (1.8) will establish the API and Services on existing DSpace core classes while leaving all implementation in place, with deprecations to alert all projects which have altered these methods that they should prepare for significant changes on the way in the next version. A second iteration (1.9) will finally move those deprecated methods out of the core classes and push them into the data access service implementations.

Achieving DSpace 2.0

By continuing this path of development, we continue to show that the original funded DSpace 2.0 Initiative continues to supply solutions and recommendations on the roadmap towards DSpace 2.0. With the Services framework and the first significant refactoring of DSpace core classes in place, the DSpace developer community can now begin to schedule changes that should happen as stepping stones to prepare the community for the release of a DSpace version 2.0.

Welcoming Collaboration

@mire realizes that such work on core DSpace libraries requires consensus and coordination into the community. As such we welcome feedback, critique and community participation on the initiative.