Google Summer of Code 2008 Fedora Integration

Title: DSpace & Fedora integration

Student: Andrius Blažinskas

Mentor: Richard Rodgers

Co-Mentors: James Rutherford, Scott Phillips, Richard Jones, ???

Abstract

Two popular digital content repositories – DSpace and Fedora are quite different in nature and have different data models. Both of the repositories have different advantages. Integration of these two repositories would allow wider digital content dissemination and management possibilities. Utilizing repositories in a separate way, digital content must be prepared and replicated for each of them. To avoid this replication a specific driver implementation, allowing one repository to access data through another repository, must be created. It is obvious that a lot of work must be done to fully achieve desired result, so my proposal is to create a working storage driver prototype for DSpace which will allow storing, accessing and managing at least basic DSpace data in Fedora repository considering its relationships and associated policy.

Development process (plan)

Work phases:

- In-depth analysis of both repositories data models and the possibilities of mapping between them. (Done)
- DSpace & Fedora model mapping design: basic mapping. (Done)
- Basic DAO & BitStore interfaces implementation. (Done)
- Integration with DSpace and resolution of problems. (Done)

- Creation of documentation. (Done)

Deliverables:

- Working Fedora interface library prototype for DSpace.
- Mapped model and code documentation.

Model mapping

The first and most essential thing that must be done is DSpace model mapping into Fedora model. Defined mapping and created library should allow not only storing basic data but also retaining the main infrastructure. That is, it should be able to preserve DSpace defined relationships and policy rules if possible.

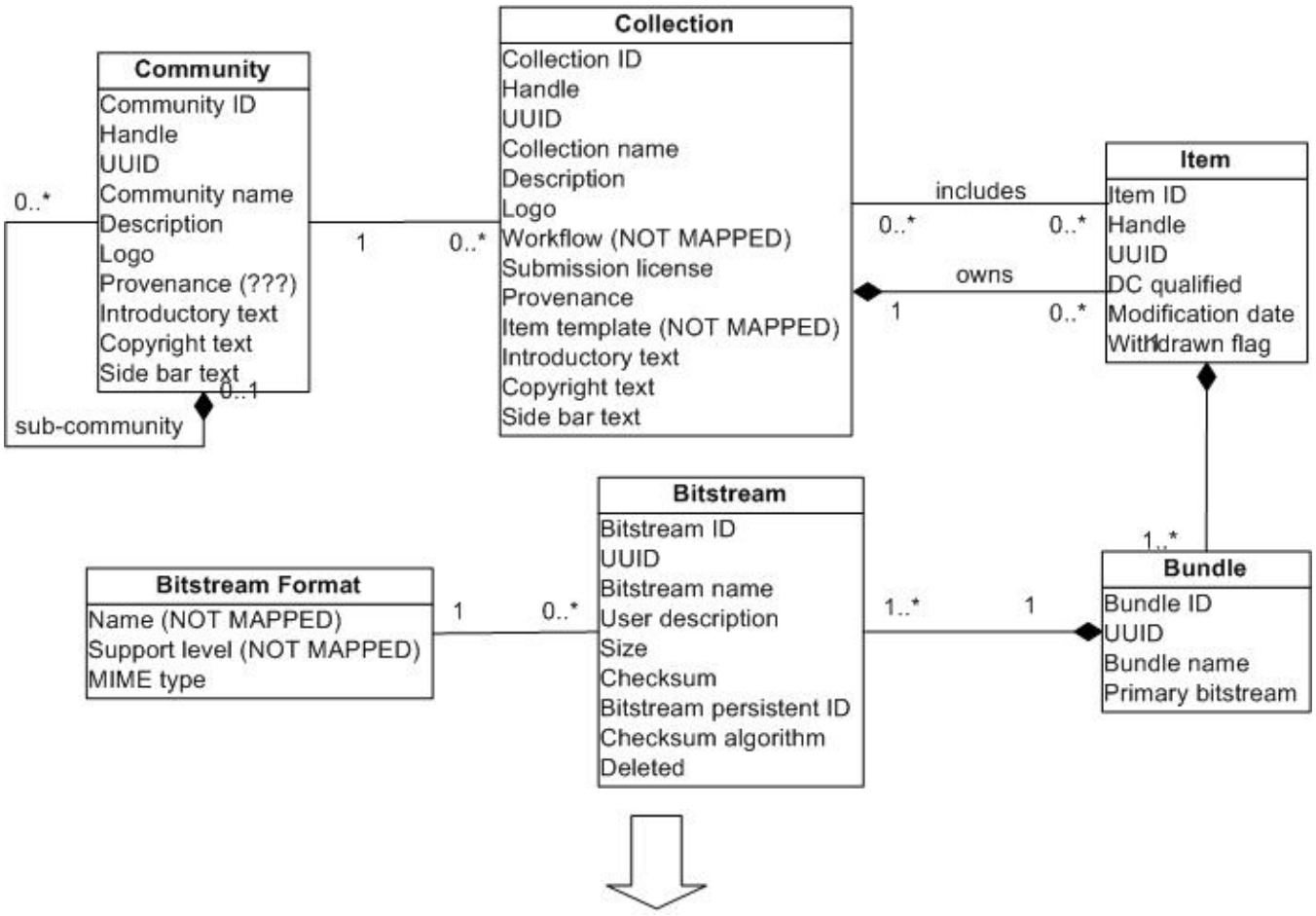
To understand the possible model mapping, Fedora object must be described. In Fedora, Fedora object is a general entity and most of the other Fedora entities are defined on its basis. Fedora object can have relationships with other Fedora objects in hierarchical manner and so in some cases it can be treated both as parent and as a child. Fedora object can contain datastreams which can be metadata or other simple files. Fedora object relationships are indicated in special *RELS-EXT* datastream. As we will see, several of DSpace entities can be mapped to Fedora object.

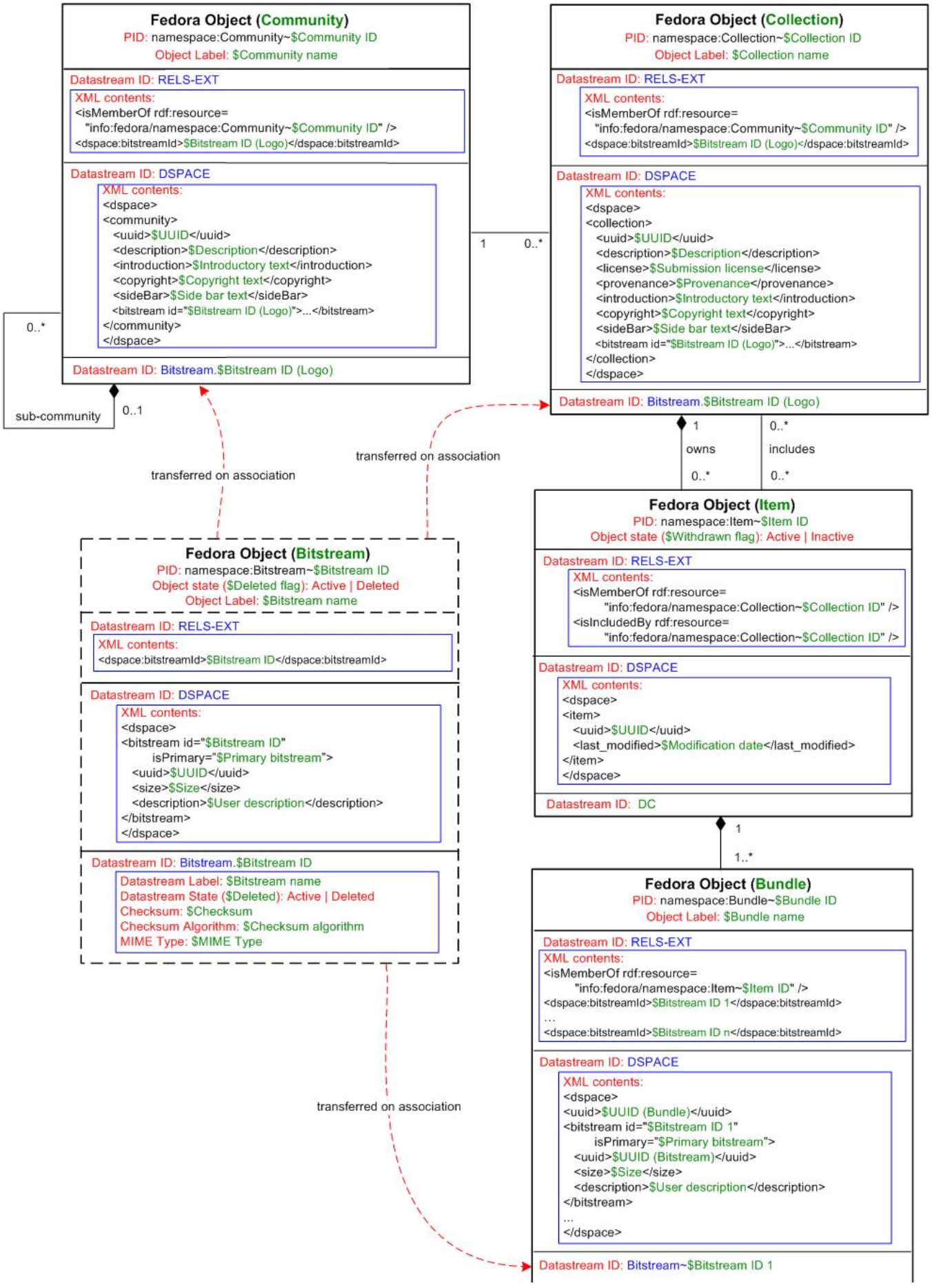
Figure 1 provides relative DSpace data model, which is a little bit extended version of basic model (http://www.dspace.org/index.php?option=com_content&task=view&id=149). Several additional fields are added considering database fields.
Possible DSpace data model mapping to Fedoras is provided in *Figure 2*. In diagram, every Fedora object representing DSpace entity has RELS-EXT datastream. Fragment of its XML contents is provided to show how relationships between objects will be implemented physically.

Not all DSpace entities or fields can be directly mapped to Fedoras, so additional datastream with identifier *DSPACE* is added to Community, Collection, Item, Bundle and Bitstream Fedora objects. This datastream is sort of specific DSpace metadata XML file containing all unmapped text fields or other data. Each XML file contents example is provided in corresponding entity (*Figure 2*).

Item entity in DSpace can be marked as "withdrawn". Fedora does provide similar feature for its objects and datastreams, by allowing setting object or datastream state. Possible state values are: "active", "inactive" and "deleted". State "inactive" is chosen for withdrawn Items here, but "deleted" could be easily used too.

DSpace Item entity contains associated Dublin Core qualified metadata XML file. Fedora does provide default datastream with DC identifier for Dublin Core metadata in every object, so it can be used to contain these fields.


Community
Community ID
Handle
UUID
Community name
Description
Logo
Provenance (???)
Introductory text
Copyright text
Side bar text
0..*
0..1
sub-community
1
0..*
Collection
Collection ID
Handle
UUID
Collection name
Description
Logo
Workflow (NOT MAPPED)
Submission license
Provenance
Item template (NOT MAPPED)
Introductory text
Copyright text
Side bar text
includes
0..*
0..*
owns
1
0..*
Item
Item ID
Handle
UUID
DC qualified
Modification date
Withdrawn flag
1
1..*
Bundle
Bundle ID
UUID
Bundle name
Primary bitstream
1..*
1
Bitstream
Bitstream ID
UUID
Bitstream name
User description
Size
Checksum
Bitstream persistent ID
Checksum algorithm
Deleted
1
0..*
Bitstream Format
Name (NOT MAPPED)
Support level (NOT MAPPED)
MIME type


Fedora Object (Community)
PID: namespace:Community~$Community ID
Object Label: $Community name
Datastream ID: RELS-EXT
XML contents:
<isMemberOf rdf:resource=
"info:fedora/namespace:Community~$Community ID" />
<dspace:bitstreamId>$Bitstream ID (Logo)</dspace:bitstreamId>
Datastream ID: DSPACE
XML contents:
<dspace>
<community>
<uuid>$UUID</uuid>
<description>$Description</description>
<introduction>$Introductory text</introduction>
<copyright>$Copyright text</copyright>
<sideBar>$Side bar text</sideBar>
<bitstream id="$Bitstream ID (Logo)">...</bitstream>
</community>
</dspace>
Datastream ID: Bitstream.$Bitstream ID (Logo)
0..*
sub-community
0..1
Fedora Object (Collection)
PID: namespace:Collection~$Collection ID
Object Label: $Collection name
Datastream ID: RELS-EXT
XML contents:
<isMemberOf rdf:resource=
"info:fedora/namespace:Community~$Community ID" />
<dspace:bitstreamId>$Bitstream ID (Logo)</dspace:bitstreamId>
Datastream ID: DSPACE
XML contents:
<dspace>
<collection>
<uuid>$UUID</uuid>
<description>$Description</description>
<license>$Submission license</license>
<provenance>$Provenance</provenance>
<introduction>$Introductory text</introduction>
<copyright>$Copyright text</copyright>
<sideBar>$Side bar text</sideBar>
<bitstream id="$Bitstream ID (Logo)">...</bitstream>
</collection>
</dspace>
Datastream ID: Bitstream.$Bitstream ID (Logo)
1
0..*
owns
includes
0..*
0..*
transferred on association
transferred on association
Fedora Object (Item)
PID: namespace:Item~$Item ID
Object state ($Withdrawn flag): Active | Inactive
Datastream ID: RELS-EXT
XML contents:
<isMemberOf rdf:resource=
"info:fedora/namespace:Collection~$Collection ID" />
<isIncludedBy rdf:resource=
"info:fedora/namespace:Collection~$Collection ID" />
Datastream ID: DSPACE
XML contents:
<dspace>
<item>
<uuid>$UUID</uuid>
<last_modified>$Modification date</last_modified>
</item>
</dspace>
Datastream ID: DC
Fedora Object (Bitstream)
PID: namespace:Bitstream~$Bitstream ID
Object state ($Deleted flag): Active | Deleted
Object Label: $Bitstream name
Datastream ID: RELS-EXT
XML contents:
<dspace:bitstreamId>$Bitstream ID</dspace:bitstreamId>
Datastream ID: DSPACE
XML contents:
<dspace>
<bitstream id="$Bitstream ID"
isPrimary="$Primary bitstream">
<uuid>$UUID</uuid>
<size>$Size</size>
<description>$User description</description>
</bitstream>
</dspace>
Datastream ID: Bitstream.$Bitstream ID
Datastream Label: $Bitstream name
Datastream State ($Deleted): Active | Deleted
Checksum: $Checksum
Checksum Algorithm: $Checksum algorithm
MIME Type: $MIME Type
1
1..*
Fedora Object (Bundle)
PID: namespace:Bundle~$Bundle ID
Object Label: $Bundle name
Datastream ID: RELS-EXT
XML contents:
<isMemberOf rdf:resource=
"info:fedora/namespace:Item~$Item ID" />
<dspace:bitstreamId>$Bitstream ID 1</dspace:bitstreamId>
...
<dspace:bitstreamId>$Bitstream ID n</dspace:bitstreamId>
Datastream ID: DSPACE
XML contents:
<dspace>
<uuid>$UUID (Bundle)</uuid>
<bitstream id="$Bitstream ID 1"
isPrimary="$Primary bitstream">
<uuid>$UUID (Bitstream)</uuid>
<size>$Size</size>
<description>$User description</description>
</bitstream>
...
</dspace>
Datastream ID: Bitstream~$Bitstream ID 1
transferred on association

Also it should be noted, that Item entity has two types of relation with Collection entity. In Fedora, simple relations between objects are expressed in *RELS-EXT* using *isMemberOf* relation type. However, custom relations can easily be introduced, so additional relation *isIncludedBy* is added here to emphasize inclusion rather than ownership. Not really sure if it is good to use custom relation, but it works.

Relations between mapped DSpace entities in Fedora can be found by searching resource index with ITQL queries. Such a query example:

```
select $object from <#ri>
where  $object <fedora-rels-ext:isMemberOf>
   <info:fedora/demo:Collection~123.456-789>
```

Example query result in SPARQL:

```
<?xml version="1.0" encoding="UTF-8" ?>
<sparql xmlns="http://www.w3.org/2001/sw/DataAccess/rf1/result">
<head>
  <variable name="object" />
</head>
<results>
  <result>
    <object uri="info:fedora/demo:Item~213.456-789" />
  </result>
  <result>
    <object uri="info:fedora/demo:Item~223.456-789" />
  </result>
</results>
</sparql>
```

The same way can be formed query for included Items:

```
select $object from <#ri>
where  $object <fedora-rels-ext:isIncludedBy>
   <info:fedora/demo:Collection~123.456-789>
```

More tricky situation is with DSpace Bitstreams. Basically, they are mapped to Fedora datastreams. When ingested, every Bitstream is put into separate temporary Fedora object. Later, when Bitstream is associated with any entity (Bundle, etc), it is transferred to this entity object as Fedora datastream. In some special cases, when bitstream is linked to several entities, Bitstream in Fedora is moved and kept in separate dedicated object, with relations in RELS-EXT to other parent entities.
This separate Bitstream object scenario also satisfies the case, when Fedora is used only to store Bitstreams and small associated metadata set, without preserving full model structure (only FedoraBitStore functionality). In this case, this object is not temporary but always permanent. However, the idea of one datastream (Bitstream) per one object still isn't that attractive...

Every entity, containing any associated Bitstreams, has bitstreamIDs in RELS-EXT. This gives possibility to directly find any Bitstream by its ID simply querying resource index, without going through all Fedora objects.

Sample RDF fragment Fedora object must contain in RELS-EXT for Bitstreams to be findable:

```
<rdf:RDF xmlns:dspace="http://www.dspace.org/elements/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#
         ... ">
  <rdf:Description rdf:about="info:fedora/demo:Bundle~1">
    <dspace:bitstreamId>9</dspace:bitstreamId>
    ...
  </rdf:Description>
</rdf:RDF>
```

Sample query to select Fedora object containing specified bitstream:

```
select $object from <#ri>
where  $object <http://www.dspace.org/elements/bitstreamID> '9'
```

Most of Bitstreams are contained in associated Bundle Fedora object, except LOGOs, they reside in associated entity object (Community or Collection).

Most of the Bitstream metadata fields can be directly mapped to similar Fedora datastream fields. Other fields like size, description etc, will be located in separate corresponding Bitstream section in *DSPACE* datastream.

Fedora datastream state can be set to "deleted" the same way corresponding Bitstream in DSpace can be marked "deleted".

Bitstream format does not have direct mapping and currently only MIME Type is mapped.

Identifiers

Fedora objects (which represent DSpace entities) PIDs can possibly be formed using general pattern: **<Fedora namespace ID>:<DSpace entity type>~<DSpace entity ID>**. At the moment, DSpace entities IDs are internal DSpace identifiers (method getID() is used). Examples of IDs provided in *Table 1*.

Fedora PID must satisfy pattern:

```
*'(**\[A-Za-z0-9|A-Za-z0-9\]**\|*{*}-\|\.)-*:-( *{-}+(+{-}{*}{*}{-}+\[A-Za-z0-9|A-Za-z0-9\\]+{-}{*}{*}{-}+)\|+{-}{*}+\|\.\|~\|_\|(%++\[0-9A-F|0-9A-F\]++\{2\}))+'
```

So it does not allow some special characters like slash ("/"), which is used in DSpace handles. These characters must be escaped or replaced. Currently I have replaced "/" by "-".

Bundle identifier is formed combining parent Item handle and DSpace Bundle ID (possibly from database), separated by underscore symbol "_".

Fedora datastream, representing Bitstream, ID can be formed in similar way by using pattern: **Bitstream.<Bitstream ID>**, since symbol "~" is not allowed in datastreams IDs.

It is also possible to use **Bitstream~313.456-789_7_24** as ID, but since part **313.456-789_7** will already be included in Fedora object (Bundle) ID, there is no need for replication.

Table 1: Identifiers

Fedora entity representing DSpace entity	ID pattern	ID example
Fedora Object (Community)	<Fedora namespace ID>: Community~<Community ID>	demo: Community~1
Fedora Object (Collection)	<Fedora namespace ID>: Collection~<Collection ID>	demo:Collection~1
Fedora Object (Item)	<Fedora namespace ID>: Item~<Item ID>	demo:Item~1
Fedora Object (Bundle)	<Fedora namespace ID>: Bundle~<Bundle ID>	demo:Bundle~1
Fedora Datastream (Bitstream)	Bitstream.<Bitstream ID>	Bitstream.1

Other entities: users, groups and policy

There are several other essential entities, which are not part of the data model. These entities are: users and groups.

In Fedora, all information about users is held in one *fedora-users.xml* file. Fedora currently does not provide any API for users' management. However this could be organized in custom way by implementing additional webservice under Tomcat for *fedora-users.xml* file modification.

Roles in Fedora are somewhat equivalent to groups in DSpace and are also indicated in *fedora-users.xml* file.

Mapping and creating DSpace users and groups in Fedora is just a first step. Much more interesting and tricky is implementation of policy. This is planned for a later time, so at the moment I leave this question open.

Open questions

- Mapping of any other fields (Workflow, Item template, Bitstream format name and support level)?
- Original DSpace identifiers in *DSPACE* datastream or even in *RELS-EXT* (automatically indexed and searchable)?
- ...?

Implementation details

I propose to create a driver prototype which will provide DSpace the possibility to access Fedora repository as a primary storage to store bitstreams and metadata. Driver classes will have the same method interfaces as current DSpace "org.dspace.storage" package classes and will be accessed in the same manner. Driver will communicate directly with Fedora repository using its SOAP API (API-A and API-M).

To prevent software defects, all written code will be tested using JUnit. I will also provide code documentation.

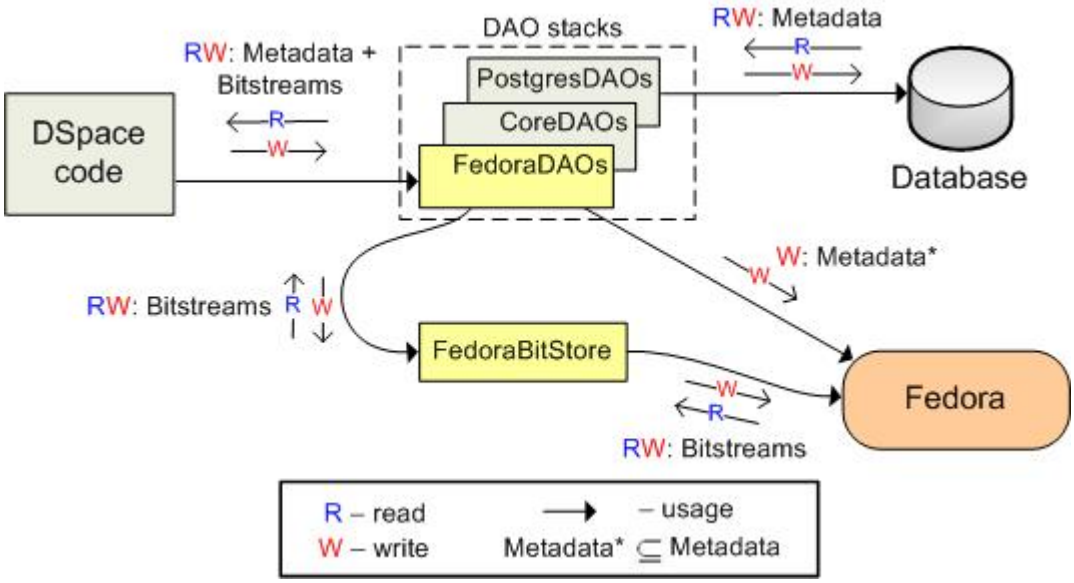
Comments (RLR):
The programmatic way DSpace accesses bitstreams and metadata is very different. Bitstreams are treated as opaque simple objects (although a few additional properties are required like a checksum). There is already some preliminary work on creating a clean abstraction to the underlying storage system (see http://wiki.dspace.org/index.php/PluggableStorage). I would recommend starting with this 'Bitstore' interface, since it will be incorporated into DSpace+1.6, and already supports several storage back-ends: filesystem, Storage Resource Broker, Amazon S3, and Sun's HoneyComb. The last 2 are essentially http client calls, so they already resemble using the Fedora SOAP API.

But the metadata is another story - DSpace does very little to abstract away from direct JDBC/SQL calls into a RDBMS. I think here the question of a 'driver' is less obvious, and you might want to explore a few designs before committing a lot of work. For example: could the metadata be placed in a bitstream and stored through the other driver? This is not a functional apping, but would satisfy e.g. a replication scenario. Should you attempt a high level metadata abstraction that bypasses current DSpace (but could be retrofitted into it)? Etc. I am just throwing out thoughts to elicit additional discussion here. |

After initial analysis, the decision was made to start work from interface library (driver), which will allow managing basic DSpace model entities (Community, Collection, Item, etc.) in Fedora repository. This library will be independent from DSpace itself.

<...>

Currently implemented driver actually is a combined DAOs (http://wiki.dspace.org/index.php/DAO+Prototype) and BitStore (http://wiki.dspace.org/index.php/PluggableStorage) interfaces implementation. It can be used as both: DAO implementation or much more simplier standalone BitStore implementation. Actually, FedoraDAOs directly utilizes FedoraBitStore, bypassing BitstreamStorageManager.



Driver allows store and retrieve Bitstreams, while metadata is only stored in Fedora. Relations are also preserved between Fedora objects using RELS-EXT.

As not all metadata is pushed into Fedora and it is not used back by DSpace, in parallel with FedoraDAOs, default DSpace DAOs (CoreDAOs and PosgresDAOs) are running to ensure that DSpace has access to all data fields in database, which it needs to function correctly.

TODO

- Fedora APIA and APIM connection pool, since currently these are loaded everythere on demand.
- Direct communication with Fedora from DAOs through SOAP is very slow, so it lags the WEB-UI (JSPUI or XMLUI). Some sort of asynchronous JobManager must implemented.
- Currently used internal DSpace identifiers for Fedora entity identification is very straightforward solution. There should be utilized Fedora system generated identifiers (or where possible maybe handles?)
- Additional service allowing to manage users and groups in Fedora should be created. This service will be used by DFI (DSpace & Fedora integration) driver (by EPersonDAO and GroupDAO?).
- Policy mapping implementation (user management service must be created to associate policy with users and groups?).