DSpace 2.0 Requirements and Issues

DSpace+2.0 is hoped to solve a number of issues that are prevalent in DSpace 1.5 and earlier. Please feel free to add to this page any issues you consider important in our consideration in DSpace 2.0

Better Metadata Attachment to all DSpaceObjects

DSpace 1.5 and earlier only allow limited metadata to be attached to any object other than an Item. DSpace+2.0 should allow Any DSpace object to have metadata attahed to it.

More Flexible Metadata

Metadata should not be a strict list of fields and values. Hierarchical values should be allowed. For example, an author field should be able to contain foaf RDF data, rather then just a single (dc.creator) value. A citation field should be able to contain MODS data.

All metadata (at any level of hierarchy) should available for indexing by the search/browse system.

Comments:

- _ Ryan, Excellent Point, what we are seeing today in both LoC and DCMI communities is further adoption of RDF to provide representations of complex structured metadata. So for instance, your FoaF "value" would actually be modeled as a set of explicit RDF Statements. This is where applying the concepts behind RDF Models/Graphs allows us to make such attachments. See also: DSpace+2.0-Modelling_DSpace_Objects. Taking this a example a step further to express how this would be supported, we would have a strong example of modeling and embedding Structured metadata in RDF... --Mark Diggory 14:40, 23 September 2008 (EDT)_

Technical Metadata

DSpace needs to improve the way it represents file formats, to interoperate better with other useful services like PRONOM and JHOVE (and especially JHOVE2).
See the BitstreamFormat Renovation prototype for details.

Bitstreams also need their own individual last-modified date, ideally echoing the date of the ingested file. This would be solved by the flexible metadata requirement, but perhaps it is worth imposing as one of the fixed fields baked into the data model, like size.

Cleaner Permissions/Policies and Roles

DSpace 1.5 and earlier have conflation of Policies, Groups and Roles such that there a number of critical problems:

Group inflation

There is an inflation of Groups required to manage the community collection hierarchy, a new group is created for each and every Policy defined.

- Users/Groups can be attached to Policies directly and we should either
    - allow more than one User/Group per Policy.
    - allow more than one Policy per Action type (we already do but the UI needs improvement here, the UI should look for Policies that are of a specific type and edit their membership rather than delegating to the group editor.).
- Groups should only be created as needed by the Administrator rather than auto-generated by the application.

Role / Policy Conflation

Role and Policies are conflated in the DSpace workflow. A better approach would be to recognize WorspaceItem and WorkflowItem as "Collections" of Items for which Permissions May be attached, thus...

- A DSpace Collection may have simple Policies on its contents (Access, Create, Mutate, Delete)
- A DSpace Workflow Stage may have simple Policies on its contents (Access, Create, Mutate, Delete)
- A DSpace Workspace may have simple Policies on its contents (Access, Create, Mutate, Delete)

External Identifiers

DSpace 1.5 hardcodes Handle resolution mechanisms into the core codebase. 2.0 is working to separate out the Handle service and establish a separate representation for Identifiers that may be managed externally in Metadata/Identifer registries/services rather than internally in the Application. One of these Providers will/may be designated as the default for identifier utilized by the DSpace Content Management Service when creating new DSpace Objects.

- _ I think this should be taken a step further in that its the responsibility of the underlying implementation of a Content Management Service to address the canonical identifier it uses in the generation of its Content objects. This means that the underlying CM Service implementation is also an ObjectIdentifer service capable of minting new identifiers when required when generating objects in the repository. To give key examples (a

JCR-170 store may randomly generate unique ids used to identify its contents, a Fedora store may generate info:fedora/..., urn:uuid:..., or other identifiers that represent its contents, and our default implementation will definitely be generating urn:uuid:... identifiers. I currently see in the trunk that DSpaceObject has a "setIdentifier" method exposed on it, I think this should never be the case and that DSO retrieved from the Content Repository have this set and fixed in their construction as part of the Objects creation before being delivered tot he user. Identifier objects should be immutable._ --Mark Diggory 16:02, 24 September 2008 (EDT)

- *Also consider how to preserve the persistent identifiers that are already associated with a content object at the time it is ingested – for example, Items transferred from or replicated for another repository. It's fine to assign new identifiers too, especially if they are only for internal (to the DSpace site) use, but the must also be findable through its old persistent identifiers.* LarryStone 02:38, 8 October 2008 (EDT)

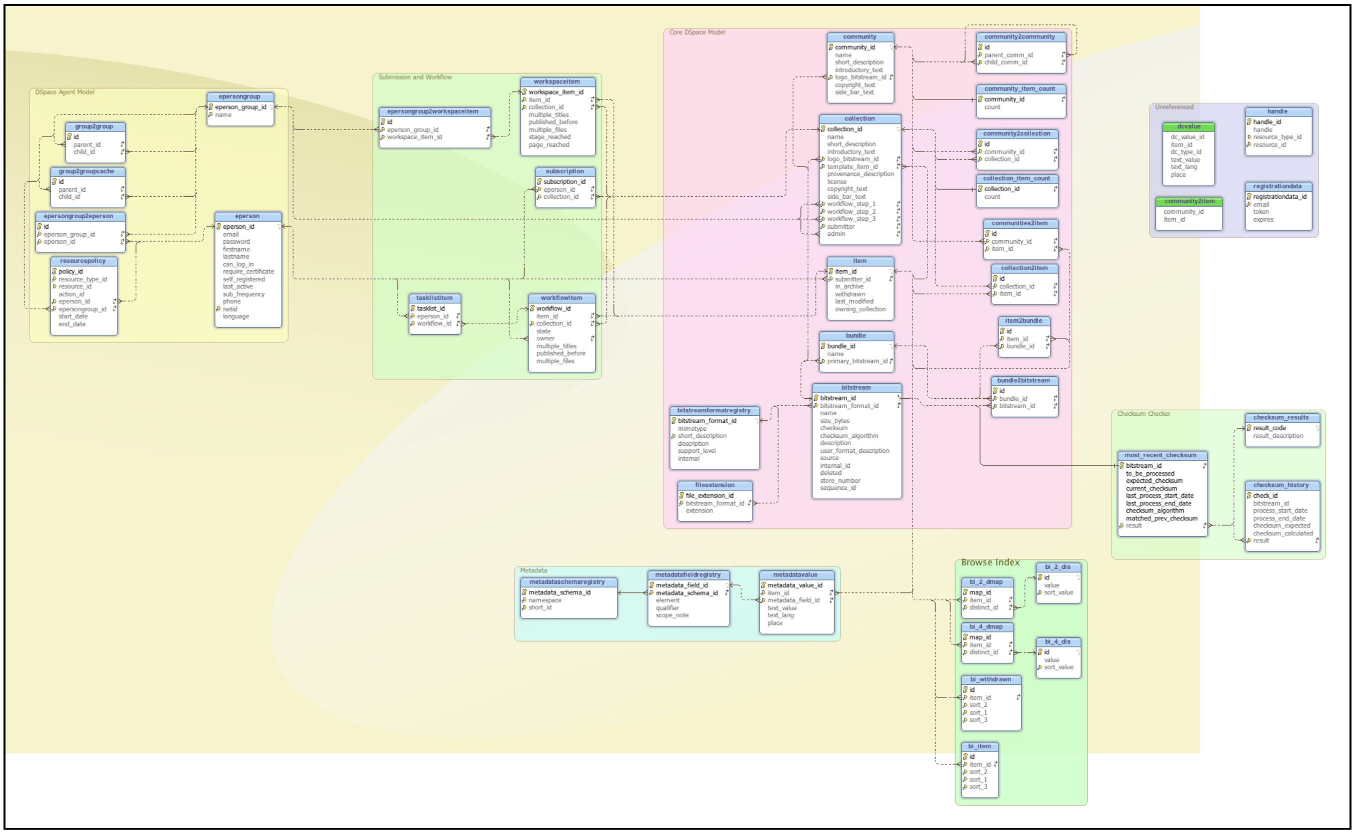
Data Model Should be expressed as Interfaces

DSpace 1.5 has a Data Model that is hardcoded Concrete Objects. These should be interfaces that are implemented by the service provider such that critical properties and methods cannot be accessed and the object state broken.

Database Related Issues, Recommendations and Analysis

Database Overview Graphic

This Graphic was generated with DbSchema and I've worked to organize it so that its divided into Group representing Various Conceptual Units of DSpace.



Database Schema Groups

- DSpace Core Model
- DSpace Metadata Model
- DSpace Agent Model
- DSpace Browse Index
- Submission and Workflow
- Checksum Checker
- Unreferenced Tables

Concerns

I have the following concerns about the DB Schema

1. Overuse of Sequential Keys on tables that do not need them
2. Lack of use of Composite Primary Keys to assure uniqueness on tables like Collection2Item

3. Foreign Key dependencies in the core on the Agent Model
4. Unnormalized workflow step fields on Collection database.
5. submitter and admin fields on collection table
6. submitter id on Item

True Archival Information Packages

DSpace needs a way to encode and transfer digital objects without *any* loss. All of the current methods (batch import, packager, etc) lose *some* information when copying an object from one DSpace site to another. A true *Archival Information Package* (AIP) includes every piece of information about the object, including the structure of Bundles and Bitstreams, rights metadata (i.e. the relevant policies), technical metadata, and *all* descriptive metadata. The main use of AIPs is to copy objects between repositories for migration, custody transfer, and replication for preservation.

Storing AIPs locally can also keep a self-contained record of each object as a fine-grained backup of the information that is otherwise only in the RDBMS. Given a true AIP, you can make a completely accurate copy of a DSpace Item at another site, and then use it to restore the original site if it is lost. An archive could be rebuilt or cloned by re-ingesting
a set of its own AIPs.