AssetStoreClippings

DevelopmentAreas > AssetStore >

Clippings from the old asset store pages

Object Model

What is the object model? Community, Collection, Item, Bitstream Format? May others be defined by DSpaces, and may the 'extended types' be 'peers' of the above, or must they be sub-classes?

Echoing RichardJones comments on the mailing list, it seems sensible to define the types of entity in the object model at a higher level, and not to build knowledge of them into the asset store. JimDowning

---

Perhaps a way forward here is to think in object-oriented terms. The asset store could deal only with a base class of AIP, which could have some basic common attributes/properties/fields. E.g. these could be simple things like identifier, creation date, modification date. These would be the only things that the asset store itself needs to understand (though I could envisage smarter asset stores that understand more). Sub-classes of this (Community, Collection, Item etc.) would, in addition to these, have extra attributes/properties/fields.

In other words, the basic AIP serialisation (base DSpace AIP class) must contain some basic information about the AIP, such as (not trying to say this is the complete list):

- Creation and last modified dates
- What class it is (see below)

DSpace object classes (or types; I'll use 'classes' here though since I want to borrow some OO concepts) have a particular profile within that serialisation format. For example, collections have the IDs of items that are within them, items have some sort of structure map (a la METS etc).

The asset store API just understands and talks about AIPs in terms of the basic serialisation (the base class). This means that you can extend the object model by subclassing the AIP base class and the asset store API will still work.
Of course your asset store manager may understand more about specific DSpace AIP classes, and act on those, but the asset store API itself does not, meaning that implementations are not forced to deal with that unless they have to

In general you just get AIP metadata serialisations and bitstreams out of the asset store API. You don't get objects you can do much procedural stuff with. The content management API provides classes which understand particular AIP classes (profiles) and provides procedural methods.

To extend the DSpace object model, you define a new AIP class (profile). This can sub-class from an existing one or from the base class in standard OO fashion. You can then implement a procedural interface to this by adding to the CM API.

Rather than the CM API being a single API implemented by a single module, maybe the way to go is to have each class that deals with a particular AIP class as a separate API in the DSpace JVM. Then, if you have a DSpace module that understands a particular class of AIP other than one of the standard 4 (community, collection, item, bitstream format), you can have your module depend on the CM API module that supports that class. Then you know that the DSpace JVM you are in 'understands' that class.

Note there should be no restriction on implementations/instances adding extra metadata to an AIP.

Versioning

A potential simple way of extending this to include versioning would be simply to have different metadata.xml's for each revision. Since bitstreams are referenced by checksum, if the same bitstream is in multiple revisions it still need only be stored once. Whether to keep around bitstreams only referenced by previous revisions can be a policy decision taken by a particular DSpace instance.

```
aip-identifier/
  metadata.xml.1      first revision of metadata serialisation
  metadata.xml.2      later revisision of metadata serialisation
  metadata.xml        current (latest) metadata serialisation
  184BE84F293342      bitstream 1 (filename = checksum)
  3F9AD0389CB821      bitstream 2
  330F925A1D0386      bitstream 3
  checksum            checksum of AIP
```

I think this simple approach is fine for DSpace, dealing with reference information - I really don't think we should get into worrying about diffs or trying to implement CVS-style functionality.