AssetStore

`needsupdate`

Contents

Standards-based AIP Asset Store

In the current architecture, all metadata is in a relational database, and all content bitstreams are in the file system on the server. This makes certain preservation-related activities complex, including backups, auditing, and replication/distribution. The proposed new asset store stores metadata and content together as standards-based Archival Information Packages OAIS terminology. This does not replace the current relational database in DSpace. Although these AIPs are the authoritative version of information in the system, the relational database and search indices are still used for performant access; however, these are considered caches of the information in the AIPs.

There may be many different implementations of the asset store available. A simple file system based implementation could be the default. Other implementations based on Grid-style or other storage systems will be possible. The asset store is itself a DSpace module with a defined interface, and as such the implementation can be changed. However, the rest of the system needs to know what behaviour to expect from the asset store, and needs to have a conceptual model of what's in it.

The main reasons for moving to this asset store model is to ease various preservation tasks:

- Disaster recovery - If your DSpace software, or any component part, fails, you still have your data as long as you have access to where the AIPs are stored. With DSpace right now, for example, if your database becomes corrupted, the bitstream store because a big jumble of randomly-named files.
- Backup - you can get a backup of your data that is reliably self-consistent by locking just one AIP at a time, instead of your whole system (DB + bitstream store)
- Replication - since the content is natively in a self-contained standards-based form, it is very easy to replicate/mirror
- Auditing - The metadata (including the manifest of files, technical metadata etc) can also be checksummed and/or digitally signed. This is difficult to do with database table rows.

AIP Format

Initially, only items will be AIPs. Later on, AIPs for communities, collections and possibly bitstream formats can be added.

An AIP consists of:

- A core metadata METS document , conforming to the DSpace AIP METS profile. This contains the basic metadata for the object (qualified Dublin Core, the manifest of files and the technical metadata).
- Zero or more bitstreams. These may be content files (which are in the file manifest of the METS document), or additional metadata (referred to by the METS document).
- A checksum for the METS.

For example, an item AIP stored in a simple filesystem-based asset store might look like this:

```
aip-identifier/
```

```
  mets.xml          core metadata serialisation in METS
```

```
  bitstream1
```

```
  bitstream2
```

```
checksum          plain text file, containing checksum of METS document
```

(It may appear differently in other storage mechanisms e.g. Grid-style)

The proposed intention is that communities and collections are also AIPs, which contain references to the items within them. Communities and collections aren't really just containers – they have their own metadata and are archival objects in their own right. Then, from a disaster recovery point of view, your archive is resistant if the top three layers of the architecture are blown away; the asset store contains everything you need to reconstruct the archive. Also it means you can mirror/replicate a DSpace's content just by grabbing the AIPs.

In other words, the intention is that the AIPs are really independent of DSpace – you do not need a DSpace instance to be able to make use of them. The asset store is the archive; the rest of DSpace is really just a tool for managing, allowing deposit of and retrieval of objects in that asset store. AIPs are intended in a sense to transcend the DSpace application, and indeed an organisation.

One thing that is **not** in the AIPs in the asset store is stuff like information about e-people. That information is specific to a particular instance of DSpace. Say I have an AIP at MIT, and Cornell university is going to mirror it. It doesn't make sense to have e-people records referenced by that AIP since they only make sense in an MIT context. What it **does** make sense to include is some sort of declaritive policy expression for access control. This can be **interpreted** by different DSpace instances (or other systems) to decide which e-people have which exact permissions on an AIP. However, this can be developed later.

Metadata serialisation format

The AIP metadata serialisations should all be in one basic format. This lowers so many bars for interoperability, and scaling up via the asset store sharing mechanisms we've been talking about. Having DSpace support arbitrary AIP formats seems to be creating too many problems.

METS and DIDL are two potential options. For DSpace requirements, both are adequate. METS is proposed since it is the best-understood by the DSpace community.

Object Model

Asset Store API

From an asset store API point of view, the salient points are that an AIP consists of:

- a core metadata file, conforming to the DSpace AIP profile;
- zero or more bitstreams;
- potentially a checksum

Prototypes

Various prototype APIs have been created by members of the DSpace community to try out different approaches:

- JimsAssetStorePrototype from JimDowning
- RichardsAssetStoreApi from RichardRodgers
- RobsAssetStorePrototype from RobertTansley

Requirements

- An asset store basically corresponds to the archival storage function of the OAIS reference model. It's tasked with storing AIPs.
- The asset store API must allow multiple implementations of the asset store to exist, and for this to be transparent to asset store users. It is not necessarily a 'classic' filing system behind the API.
- An AIP is essentially a conceptual object, with an identifier.
- In concrete terms, an AIP consists of:
    - a METS metadata file, conforming to the DSpace AIP METS profile;
    - zero or more bitstreams;
    - a checksum
- Each and every Community, Collection and Item has precisely one corresponding AIP in the asset store.
- Each Item AIP is basically self-contained; if for reasons of storage efficiency etc., there is normalisation, for example, two AIPs contain identical bitstreams, and the storage mechanism elects to store only one copy of this, this is transparent to the asset store user.
- In the OAIS model, Archival Information Collections are types of AIP that contain other AIPs. To simplify the storage abstraction, this containership is represented by having Community and Collection AIPs also as self-contained units, related to AIPs 'contained' within them by reference in the AIP metadata.
  An AIP should, within reason, contain everything needed for a system (DSpace or other) to make use of and audit* the digital asset within. In other words it should contain:
    - Descriptive metadata
    - Technical metadata
    - Administrative metadata
    - Provenance (history)
    - Any appropriate rights metadata/licenses
    - The constituent files (bitstreams)

- It should not contain information that unduly ties it to a particular DSpace instance or software version, for example:

Permissions relating to individual e-people record

An open issue is whether an AIP should contain provenance information. If you're talking about the provenance of the bitstreams in the AIP, that seems to work; it's part of the metadata. If the provenance of the metadata itself or of the AIP as a whole is also important, the AIP then contains its own provenance metadata which feels like a potential security/robustness hole.

- AIPs may of course make reference to other AIPs and resources; for example, a Collection AIP will contain references to Items within it, and Item AIPs may contain references to representation information, such as bitstream format registry entries and so forth.
- These references are stored within the metadata component of an AIP, which need not be understood by an asset store implementation.
- It must be possible to audit the entire contents of the asset store using the API. This includes:
    - Finding out what AIPs are in the asset store
    - Checking that all AIPs are present and correct
    - Verifying checksums
    - Verifying relationships between AIPs
    - Validating METS metadata
- Identifier schemes will vary between DSpace instances. Therefore the asset store API and implementations must make no assumptions about IDs other than:
    - AIP IDs are unique within an asset store
    - Bitstream IDs are a tricky one. The current prototype assumes only unique with an AIP; discussion further down
    - All identifiers can be stored and exchanged as UTF-8 encoded Java Strings.
    - An identifier is not necessarily URN-syntax or a 'safe' file name.
    - Identifiers are assigned elsewhere in the system, not by the asset store itself
      Note this does not exclude the possibility of a single module that is responsible for both identifiers and asset storage.
- Versioning is handled in the metadata. This allows different DSpaces to handle versioning differently, and means that asset store implementations don't need to worry about it.
- Full transaction-safe capability is quite a high bar to set for an asset store implementation. The compromise proposed is to allow transaction-safety for individual AIPs, as well as preventing concurrent updates of a single AIP, i.e.:
- It should be impossible for two modules/processes to modify the same AIP at one time.
- It should be possible to 'roll back' write operations on a single AIP should an error occur at some point during modification, to ensure that at a given point in time, the AIP is self-consistent. Otherwise users/processes accessing the AIPs (including audits) may find an inconsistent, apparently 'mangled' AIP.
- The asset store should not be used as a temporary staging area, e.g. during user submission. The asset store is for AIPs not SIPs. A caller will only use the asset store API to create an AIP when all of the relevant metadata and bitstreams are assembled and available. Otherwise the asset store may contain mid-submission AIPs (not really fully 'ingested') which would be inconsistent and cause audits etc. to fail.

Issues

- Should individual bitstreams have arbitrary IDs at the asset store API level?
- What transactional support should the asset store offer? Are single bitstream / AIP modification transactions acceptable?

Use Cases

See AssetStoreUseCases