Google Summer of Code 2007 Versioning

DSpace Versioning

- Student: Robert Graham (Texas A&M)
- Mentor: Mark Diggory (MIT)
- Backup Mentors: Robert Tansley (Google) and Scott Phillips (Texas A&M)

Next Steps

- When old revisions have bitstreams edited, they shouldn't affect other bitstreams

Linear Versioning of Items

As stated in the project abstract and the DSpace Architecture Recommendations (DAR) referenced below, the goal of this project is to add versioning to DSpace. The character of the versioing being implemented is already described in the DAR better than I can reproduce here, but the main idea is a sequence of linear revisions of Items.

Concessions

In an effort to scope the project down to a tractable size for the summer, I have made a few simplifying decisions:

- Bypass and Ignore Workflows

In the end, we were able to integrate with Workspaces and Workflows this summer.

- Assume a Keep Old Revisions Policy

Deletion is not a supported use case, but a revision is just and Item and thus can be 'withdrawn' from the public side of the repository.

Versioning

New Revisions

Allow and facilitate the creation of new revisions of Items. All the metadata and bundles of an Item are duplicated to create the new version. The bitstreams are not duped and no triggers are in place to keep this from being a problem currently when an old Item revision has a bitstream changed. That problem isn't too difficult to fix and should be at the top of the list.

Revision numbers

Each revision of an object will be given a unique revision number integer in sequence.

Revision's metadata

Every revision must have its own metadata. Changes to metadata may or may not result in the creation of a new version of an Item. It is left to the user's discretion.

Manifest

The system will maintain a METS manifest that details each Item revision and what its makeup is. This will be used to find and retrieve versions of an Item or its members.

Point to Latest

The default behavior of the system in looking for a versioned object (Item, bitstream, etc.) without a specified revision number is to supply the latest such revision.

API

Revise or Replace

The API for versioning must maintain the ability to create a new revision of a target object or to simply replace the latest revision of it. This may or may not be actually implemented as creating a new (linear) revision anyway.

Find

The API must be able to find a given revision of any Item or other versioned object and if not supply either an error or the latest revision of the object being sought.

Roll Backs

The API should allow for a roll back to a previous revision (set as latest). Rolls backs simply create new revisions as the system is linear and must maintain past versions. This could be sticky if it is a commonly used feature because the revision history will be a mess. As such this is NOT currently implemented.

Storage Layer

The manifest will point to the revisions and where those actually exist in the DSpace repository's storage layer as the storage layer itself will remain wholly ignorant of versioning.

UI

Create Item Revision

Allow for creating new Item revisions by:

- adding or changing constituent objects or metadata
- composing a new revision from the HEAD revision of an Item

View Item Revisions

Create a meaningful UI that visualizes the revision history of an Item and links to the relevant objects.

Design

Structural Versus Metadata

In light of the fact that DSpace will probably include versioning as a core feature, we have decided to put the versioning information in structurally and automatically reflect that in the metadata somehow. In terms of design, this will mean database changes/additions.

Development Status

This page is going to be a fast updating and sometimes chaotic dump of what I'm working on, how the current codebase works, and where I'm headed with things. If you feel you have something to add, feel free to edit and shoot me an email.

Here and Now

What the code does

ArchiveManager has been given a CLI to interact with the code I've created. It can create new revisions of items that show up in the JSP UI and are copies of the given original item. The data and metadata associated with them can then be edited in the JSP UI. When a new version of an Item is created, the Item is not placed in the archive, but into that user's workspace and after it is accepted by them goes through the whole process of workflow based on the repositories configured workflows.

Revision Creation Process & Policy

The idea is a 'deep copy' of an Item with extra versioning metadata that can then be edited to reflect the user's wishes for the new revision. A fairly slick UI can be put on top of this if the impetus is out there, or the existing one can be used if you know what is going on and are somewhat lazy.

- Bundles are copied/duped, Bitstreams are linked to the copied bundles
- revision is set to the HEAD revision number plus one
- you can ask for a new revision of an older revision and it will copy that older item into the new revision, but the new revision becomes the HEAD revision and its previous_revision points at HEAD

- creating a WorkspaceItem on creation of a new version of an existing Item and putting the new version of the Item through the Workspace and Workflow systems
- currently giving each item revision its own (handle) default pid - the HEAD Revision should get the original external id and the old revision gets a new one

DB Schema Changes

Added columns to the Item table itself:

- revision - an integer revision number
- previous_item_id - an integer item_id for the previous revision of this item
- original_item_id - an integer that is unique to a string of revisions of a logical item, facilitates finding the HEAD in one SQL query and such

External IDs

The functionality is described above and this is one of the more dynamic areas of the project. The underlying DAO prototype is not even completely commited in its approach to EIDs.

Extras

- Bitstreams can now clone themselves
- Unlink a Bitstream from a revision by simply using Bundle.removeBitstream

Using the CLI on ArchiveManager

- Add {dspace.dir}/bin to your path or cd {dspace.dir}/bin
- dsrun org.dspace.core.ArchiveManager opts

Options

- -i item id Allows for the specification of an item by id
- -u email or id of eperson Allows for the use of Authorization and specifying a user

- -p print the given item's (-i id) item_id, revision, previous_revision
- -a print the same item data as -p for all items
- -m print the given item's metadata
- -z print the given item's persistent identifiers
- -r create a new revision of the given item using the given user (-u email or id)

DAO+Prototype

James Rutherford has started a project that deals with the object relational mappings in DSpace. It looks as though this work is a part of the future of DSpace and the decision was made to attack versioning in a parallel effort. This is intended to allow the effort here to have more value to DSpace going forward. The repercussions of that decision, however, put the versioning project on a more experimental or prototype footing with a constantly shifting codebase to adapt to as the summer went along.

Architecture Review

In early 2007 a long set of meetings established the roadmap for the next several DSpace releases called the Architecture Review. This project's aim were derived largely from the goals put forward in those meetings. The particular implementation still had a few issues to iron out and backward compatibility had to be balanced against future utility. The tradeoffs and decisions involved in this project were a large part of the effort over and above the lines of code written. The project was developed with the risks ironing out the issues posed by iteratively adding functionality and analyzing decisions as we went. Often, planning and discussion produced the final design before we committed anything to code, but here and there our iteratively development paid off by allowing us to react quickly after seeing decisons in action.

Structural Metadata

One decision was whether or not to put the metadata that described versions and revisions into structural metadata or descriptive metadata. Structural meant metadata changes and descriptive is more lightweight. The decision to go with structural was made because versioning is likely to become an important feature of DSpace and users are free to modify the descriptive metadata schema and even change standards which made that approach untenable in the long term.

EIDs and URL Spaces

External Identifiers and the URL space where users find items in the DSpace repository are a tegent to versioning because the handling of identifiers and the location of objects in an URL space are complicated by Items being able to have multiple revisions that may all be relevant to different audiences of the repository. The decision here was to follow the recommendations of the Architecture Review for the URL space and to give each revision its own EID with a special EID reserved for the HEAD or most recent revision of an Item.

Workflows and Workspaces

The project came to a point where new revisions of Item were nearly exact duplicates of the old Item, but free to be edited by the user. This type of policy was removed in favor of one where the new revision was created as an Item in the Workspace of the user and had to go through all the processes and workflows that are involved in the creation of a normal Item. The first approach was really a concession to getting a functional prototype fast. The need for these revisions to go through the same processes and reuse the same interfaces as used in Item creation is obvious now and is a great reuse of the existing code. It also reduces the need for extra UI elements and pages to be created to support versioning in DSpace.

Bitstreams and Bundles

The DSpace data model is undergoing some changes as DSpace itself matures. These changes are things that we're aware of in the versioning code, but we also had need to create something for the here and now. In the clear need to be as efficient as possible with disk space, the bitstreams for an Item are not duplicated for a new revision. The metadata is, but the storage footprint there is much smaller and the likelyhood of change higher. This doesn't mean that any metadata change precipitates a new version. Some of the details fo when to make new versions are yet to be set, but the flexibility for those decision to be made later is built into the versioning API.

See Also

- Project Abstract: http://code.google.com/soc/dspace/appinfo.html?csaid=19480D851769941
- "Toward the next generation: Recommendations for the next DSpace Architecture". DSpace architecture review group, John Mark Ockerbloom, chair. 24 January 2007.

- "Versioning Proposal". John Mark Ockerbloom. 11 October 2006.
- ArchReviewNotes
- Codehaus XFire Versioning