D-Lib Magazine
December 2005

Volume 11 Number 12

ISSN 1082-9873

The Archive Ingest and Handling Test

The Johns Hopkins University Report

Tim DiLauro, Mark Patton, David Reynolds, and G. Sayeed Choudhury
The Johns Hopkins University
{timmo, mpatton, davidr}sayeed@jhu.edu

Introduction

From very early in its existence, the Digital Knowledge Center (DKC) in the Sheridan Libraries at Johns Hopkins University (JHU) has focused on using automated and semi-automated processes to create workflows for the creation and ingestion of digital objects. What was missing was a place to put these objects, a standard way to put them there, and a way to preserve them. This has begun to change over the past two years.

After participating in a series of workshops and meetings related to the Library of Congress' National Digital Information Infrastructure and Preservations Program (NDIIPP), we noted an emphasis on the aforementioned missing elements. When the Library of Congress (LC) announced the Archive Ingest and Handling Test (AIHT) as part of the NDIIPP, JHU saw participation as an opportunity to pursue several areas of interest.

Primary among these was the evaluation of content repositories as platforms for digital preservation. We have been concerned for some time that many in the digital library community conflate the storage of digital objects in a repository with the preservation of those objects. Participating in this test would give us an opportunity to experiment with repositories and digital preservation. JHU became especially interested in validating a level of activity that could be described as a necessary, minimal level of digital preservation.

JHU was already experimenting with Fedora and had tested ingestion of content into DSpace. The opportunity to get more hands-on experience with the facilities of these two open source efforts was an additional motivator. At a higher level, though, we were even more interested in the possibility of implementing a layer of abstraction (an application programming interface or API) over existing repository applications. Such an abstraction would allow other applications to interact with at least some facilities of a repository without knowing with which repository application (e.g., DSpace, Fedora) it is interacting. The AIHT gave us an opportunity to test the feasibility of constructing such a layer and to determine the ease with which such a layer could be applied in practice.

With funding from the Andrew W. Mellon Foundation, we are conducting a technology analysis of repositories and services to continue and build on this work. More information is available on our project wiki.[1]

Phase I: Ingestion

Data Model

Because we chose a strategy of implementing an application-agnostic repository layer and performing the AIHT using two repository applications (DSpace 1.2.1 and Fedora 1.2.1), it was necessary for us to create a data model that could be supported by both of these applications. We chose to create a very simple, high-level interface consisting of container ("DataGroup" class) and item ("DataItem" class) objects. These two classes of objects were derived from an abstract DataObject class.

The DataObject class consists of three strings: an identifier, a name, and METS metadata. The DataGroup class acts as a container that can hold multiple DataGroup and DataItem objects, in addition to an identifier, name, and METS inherited from the base DataObject class. Finally, the DataItem class has, again, all the facilities of the DataObject class and, further, holds information about each version of a particular item, a list of DataItems on which it depends, and a list of DataItems that depend on it.

DataObject
Identifier
name
METS metadata

DataGroup extends DataObject
set of child DataGroup ids
set of child DataItem ids

DataItem extends DataObject
set of DataItems ids it is dependant on
set of DataItems ids that depend on it
set of Versions

Version
Mime Type
format handle
MD5 checksum
datastream content
creation date

Each directory in the archive's filesystem was treated as a collection (or container) object and was modeled as a DataGroup, while each file was modeled as a DataItem. We instantiated this model in different ways for different purposes. For content management, for example, we stored the DataObjects in a repository. For transfer and ingestion, we stored the DataObjects as multiple METS files, one per DataObject.

Metadata

A major difficulty in analyzing the supplied archive metadata was in working with such a large XML file. The file was too big to examine in a text editor, so it had to be examined in a file pager application (e.g., GNU "less" and Unix "more" commands). This process was extremely slow and cumbersome, so we will have to find a better solution when working with large XML files in the future.

Although the metadata was of varying quality, it was the only metadata available that described the individual objects. Since this metadata comprised a variety of metadata "types" (descriptive, technical, rights, and digital provenance), we used the Metadata

Encoding and Transmission Standard (METS) 1.3 as a wrapper for all forms of metadata associated with a particular digital object.

JHU focused on developing the supplied metadata into a useful package, rather than trying to extract and analyze technical metadata from the digital objects themselves. We felt that the metadata would be most useful to us and to our partners if it were converted from its idiosyncratic original format to more standard schemes. To accomplish this, we created crosswalks from the original format to more appropriate standard formats, converting the various supplied and derived metadata to appropriate METS extension schemas such as the Metadata Object Description Schema (MODS) for the descriptive and source metadata sections, Digital Production and Provenance Metadata Extension Schema (DIGIPROVMD) for the digital provenance section, and Rights Declaration Metadata Extension Schema (METSRights) for the rights section. Whenever possible, we used displayLabel attributes that matched the field names from the original GMU 9/11 Archive database. A spreadsheet describing this activity is available.[2]

Generate Submission Information Package (SIP)

We began by generating the Submission Information Package (SIP), which consisted of METS files based on the metadata provided with the archive. JHU chose to use the metadata from the MySQL database tables for this process, though alternatively we could have used the XML or Access versions of the metadata. We chose this approach because it was easier to integrate than the other two formats.

To explore the utility of our data model, we parsed HTML files, extracting dependencies to other files in the archive and storing them in the SIP. Ideally, such a mechanism would be generalized so that similar dependency checking would be facilitated across a variety of content types.

Ingestion Process

Once the SIP was created, we were ready to ingest the content into the repository. As mentioned above, the SIP contained dependency information for a subset of the files. Within the archive, files referenced each other with absolute or relative URLs, which are correlated to the archive identifier in the provided metadata. When we moved these objects into the repository, however, we wanted to be able to manage these relationships at the repository layer, instead of through the archive's identifiers.

To accomplish this, the ingestion process required us to iterate over each DataObject twice. The first iteration reserved a repository identifier for each DataObject and constructed a mapping between this ID and the object's original identifier within the archive. The second iteration used this mapping to update references between DataObjects. The new repository identifier-based relationships were stored back into the DataObjects before they were stored into the repository. Finally, the DataObject and associated content were stored into the repository. When the ingestion finished, the repository identifier for the root DataGroup of the ingested tree of digital objects was returned.

Ingestion Issues

- In both Fedora and DSpace, bulk ingestion was extremely slow, initially taking days to complete. As the amount of content in the repository grew, the ingest time stretched to a week or more. We tracked down and resolved several database problems in Fedora, reducing ingestion time to about a day – still a relatively long time. We also found database problems in DSpace, but we did not have time to track them down until the very end of the project. Our recommendations were adopted and integrated into both applications.

- Both DSpace and Fedora imposed constraints on the ingestion process. DSpace required that the bulk ingest process, which uses the DSpace Java API, has filesystem access to the DSpace assetstore. Fedora provided an easy to use SOAP interface, but required that all ingested datastreams be fetched from an http scheme URL, so the contents of the archive had to be placed behind a web server to facilitate ingestion.
- Because of the two-phase process and the database performance problems, ingestion was time-consuming. Often, coding errors (especially in the early stages), local memory limits, and server resource issues caused crashes. Having to restart from the beginning caused long delays. To improve overall performance, we implemented a checkpoint mechanism that allowed restart from any time after the first phase (or iteration) of ingestion had completed. The ingestion state (the mapping of archive ids to repository ids and the list of ids that have already been ingested) was saved to a file. If the ingestion terminated, then it could be restarted from this checkpoint.
- During ingestion, we also discovered that some of the pathnames in the metadata database were wrong, causing the ingestion to fail. We modified the ingestion process to be more fault tolerant and to log problems for easier resolution and tracking. While evaluating these errors, we additionally discovered that there were inconsistencies between at least the MySQL and XML versions of the metadata.
- The size of the collection was also a factor. The overall ingestion process had memory issues until we rewrote the METS loading code and ingestion code to stream DataObjects. Initially, we kept all objects in memory to build the mapping from archive to repository ids.
- During ingest, when a METS file was loaded and turned into a DataObject, the METS file was stored as a string in the DataObject. Since the METS stored with each DataObject is treated as the canonical representation, the METS must be rewritten to update pointers when a DataObject is exported (the METS must refer to the local datastreams) or migrated. We would design this differently, were we to do it again.

Phase II: Archive Export & Re-import

The goal of AIHT Phase II was for each of the four participants to export the contents, in a format of their own choosing, from the system into which they ingested in Phase I. After all participants completed their exports, each participant selected one of the other participants' exports to ingest anew. JHU chose to export to Stanford and to import from Harvard.

Export to Stanford

We accomplished the export to Stanford by walking the tree of DataObjects and converting each one into a set of files. A DataGroup became a single METS file. A DataItem became a METS file and a set of data files, one for each version. The exported archive was a directory consisting of METS files and data files.

In order to further test our tools, we made some changes so that exported archives could re-ingested into a repository. The METS format was extended to support linking to data files in the same directory. DataItem versions were changed to hold either an HTTP URL or an array of bytes. The Fedora ingest process was modified to work around Fedora's requirement for HTTP URLs.

Stanford Ingest of JHU Export

Stanford commented after their ingest of the JHU archive that they had expected one

METS object for the entire archive. Because our approach resulted in many METS files – on the order of the number of items in the archive – the Stanford ingest programs experienced out-of-memory conditions. This situation may have been ameliorated had they used the reference code provided by JHU; however, this will be an area that we will look into for future archive ingest projects.

This matter points to a broader issue observed during the various import processes of this phase. Though three of the four non-LC participants (including JHU) used METS as part of their dissemination packages, each of our approaches was different. Clearly there would be some advantage to working toward at least some common elements for these processes. Establishing this type of agreement early in the project likely would have improved the efficiency and success of the export/import component of AIHT.

JHU Ingest of Harvard Export

The Harvard export stored metadata in a single exports.xml file and stored data in a directory structure similar to the original GMU 9/11 Archive. We modeled the Harvard export as a root DataGroup containing a DataItem for each data file.

Ingestion required preprocessing the Harvard export in order to keep memory usage down. The exports.xml file was too large to keep in memory and ingestion required random access. The preprocessing step solved this problem by creating a separate file for each piece of metadata contained in exports.xml.

The ingested Harvard export was exported in yet another format. The directory hierarchy of the original Harvard export was recreated. Each data file became three files. For example, the file aiht/data/2004/12/21/29/49987.txt became:

- aiht/data/2004/12/21/29/49987.txt.0 – the data for version 0
- aiht/data/2004/12/21/29/49987.txt.xml – the technical metadata
- aiht/data/2004/12/21/29/49987.txt.info – info such as mimetype and checksum

The export.xml could have been stored in the repository as XML metadata of the root DataGroup, but was not, because it would have added immensely to the size of the archive. It should also be noted that we did not write a reader for our Harvard export format. Ideally, we would have had the intermediate ingest format be the same as the export format, which is what we did for our own version of the 9/11 Archive.

Phase III: Format Transformation

While some other participants focused on the selection of most appropriate formats and transformation tools for preservation quality, JHU's goal for this phase was to implement a flexible mechanism that would allow systematic migration of content that met specific criteria. We anticipate that the expertise of others with appropriate preservation formats and software tools will eventually be captured in format registries. While it would be ideal to have a generalized mechanism for doing this, we chose to filter this operation based on the MimeType.

We chose to migrate JPEGs to TIFFs and to add metadata about this conversion to the TIFF itself, in addition to the DataItem metadata. We used NISO MIX XML and stored it in the ImageDescription field of the TIFF header. We encoded information for the NISO MIX child elements of ChangeHistory, including DateTimeProcessed, ProcessingAgency, ProcessingSoftware, and ProcessingActions.

The original JPEGs were not deleted; we simply added a new version datastream to DataItems that had a JPEG MimeType. As was the case with ingestion, we implemented checkpointing so that the process could be restarted in case it was interrupted.

We used the Java ImageIO library to perform the conversion. Unfortunately, the library appeared to leak memory. The migration process eventually ran out of memory and stopped. Additionally, the library also failed catastrophically on some input. There were about 12,500 jpegs in the 9/11 Archive, of which 182 failed to convert. The Java ImageIO library threw null pointer exceptions and occasionally ran out of memory on some jpegs. We worked around this problem by automatically restarting the migration process when it failed. To speed up restarting, ids of migrated DataObjects were stored. The process ran out of memory five times during each migration.

Because the METS stored with each DataObject was treated by the export process as the canonical representation of that DataObject, the migration process had to modify it, in addition to adding a new version to a DataItem.

Lessons Learned and Recommendations

Technical Issues

Memory consumption was a big issue, even with what was a relatively small archive, in terms of both absolute size and number of objects. When processing the objects, we found it necessary to access objects one at a time and write any intermediate results to disk.

We made the export needlessly complicated by storing the METS in each DataObject. The METS should have been assembled from the content instead of stored and then reassembled during export.

Having separate command line applications for every DSpace and Fedora function was cumbersome. The configuration information for the repositories should have been stored in one config file and referred to symbolically. This will become easier as we develop a more systematic approach to layering repository APIs over various implementations.

The log files produced during ingest, export, and migration are important. They should be produced in some structured way so they can be used easily later. For example, it should be possible to easily figure out what items failed on migration and why. After a bulk operation there should be an easy way to rollback the changes. It is important to be prepared for tools not working perfectly (e.g., TIFF Java ImageIO library problems), especially when the ingesting or exporting organization has no control over the development of those tools.

Organizational Issues

Problems with contract negotiations significantly delayed the start of work. The project timeline did not correspond to the academic schedule, so it was difficult to hire students on a timely basis.

The metadata provided to the project participants was inconsistent. The contents of the MySQL database did not match that encoded in the XML. We did not evaluate the contents of the Microsoft Access database. While this might seem like a technical problem, we feel that it is much more related to process. There should be only one source for each data element. Derivatives and aggregations can be produced from these to provide content in whatever form is necessary. While these issues may seem specific to AIHT, they represent a class of non-technical issues that can affect the overall effectiveness of a technical effort.

Since the project involved a single source archive, participants were able to optimize their solutions for this particular archive. A future project might ask participants to develop a more generalized solution, perhaps based on an initial reference archive, and then assess how that solution performs with another archive.

Observations

Format Registries

Format registries will form the basis for future automated processes that support ingestion into and ongoing management (including format migration) of content already in repositories. We anticipate that these facilities will eventually become the depositories for expert analysis and tools related generally to formats, with specific emphasis on the preservation of those formats. More effort should be focused on developing this critical piece of infrastructure.

Fedora

Fedora has a SOAP interface with which it was difficult to work. The biggest problem was a lack of documentation. The Fedora implementation represented each DataObject with a Fedora Object. A Fedora Object contained one datastream with inline XML data about the DataObject. Another inline XML datastream contained a base 64 encoded string (the JHU METS format). The other datastreams of a FedoraObject each stored a DataItem version.

The Fedora ingest performance problems occurred because certain columns were not indexed. We have communicated our results to the Fedora development team, and they have incorporated our recommendations into newer releases of Fedora. More detailed information regarding these problems is available on our wiki.[3]

DSpace

DSpace could only be manipulated by a local process which uses the DSpace API to access storage. The DSpace@Cambridge project is developing a set of web services that would allow this process to be performed remotely. More information about that work is available on the DSpace wiki.[4]

The DSpace implementation encodes each DataObject as a DSpace Item. Each Item uses its DC metadata to encode information about the DataObject. DataItem versions are stored in a Bundle.

The DSpace GUI does not handle content created this way very well. If we were to go through the archive ingest process again, we would need to align our datastream naming conventions with the bitstream naming conventions of DSpace. This would be relatively easy to accomplish.

We also had a problem with DSpace performance. These issues have been reported back to the DSpace community. We have written a report, which is currently available on the DSpace wiki and on our own wiki.[5]

Acknowledgments

We thank the Library of Congress for their support of the Archive Ingest Handling Test. We also thank Jacquelyn Gourley who helped with logistics and project management; Johnny Graettinger who analyzed performance problems with DSpace and Fedora; Ying Gu who examined the metadata archive for consistency and investigated image manipulation in Java; and Jason Riesa, who wrote a tool to validate our METS format.

Notes

1. See the project wiki at <https://wiki.library.jhu.edu/display/RepoAnalysis/ProjectRepository>.

2. The spreadsheet is available at <https://wiki.library.jhu.edu/download/attachments/641/AIHTtoMETSmapRev5-05.xls>.

3. For details about our recommendations for Fedora, see our wiki at <https://wiki.library.jhu.edu/display/TechReports/FedoraPerformance> or <https://wiki.library.jhu.edu/x/uw>.

4. The DSpace wiki is at <http://wiki.dspace.org>.

5. The report we have written about DSpace performance, may be seen at <https://wiki.library.jhu.edu/display/TechReports/DSpacePerformance> or <https://wiki.library.jhu.edu/x/uQ>.

doi:10.1045/december2005-choudhury