Johns Hopkins University Archive Ingest and Handling Test (AIHT) Final Report

I) Introduction

A) Project Description

The Archive Ingest and Handling Test (AIHT) was funded by the Library of Congress (LC) as a component of its National Digital Information Infrastructure and Preservation Program (NDIIPP). The purpose of this test was to evaluate in an unconstrained manner how participants would manage the task of preserving an archive provided in a format that was not predetermined by the participants. The September 11 ("911") Digital Archive serviced from the Center for History and New Media at George Mason University was selected for this purpose.

B) Personnel

Sayeed Choudhury, Project Manager, provided overall administrative oversight of the project, including personnel issues, project reporting, and resolution of any issues that arose during the term of the contract.

Tim DiLauro, Technical Lead, provided overall technical oversight, including overall data architecture goals.

Jaquelyn Gourley, Project Coordinator, tracked project activity and worked with Choudhury and DiLauro to satisfy both project update and financial reporting requirements.

Johnny Graettinger, Student Programmer, analyzed performance problems in DSpace and Fedora.

Ying Gu, Student Programmer, examined the archive metadata for consistency and investigated image manipulation in Java.

Mark Patton, Software Developer, developed software and Java class design, supported the project wiki, and supervised student programmers on the project.

David Reynolds, Metadata Specialist, supported the metadata development aspects of the project.

Jason Riesa, Student Programmer, wrote a tool to validate our METS format.

C) Motivation

Johns Hopkins University (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 quite 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.

We were also interested in evaluating the possibility of implementing an API over existing repository applications to determine the feasibility of constructing such a layer and the ease with which such a layer could be applied in practice.

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 also a motivator.

Project Manager Choudhury and Technical Lead DiLauro had participated in a number of the NDIIPP planning meetings that lead to the conceptualization of and call for AIHT proposals. Based on this participation, Hopkins became especially interested in validating a level of activity, which could be described as a necessary, minimal level digital preservation.

D) Initial Conditions

Project participants were provided with a USB drive containing the following:

* A short README.txt file with a brief description of the contents of the drive.
* A tar archive of the content files and a file containing a checksum for the tar file.
* Metadata associated with the archive in XML, MySQL (tables & dump), and Microsoft Access formats.

E) Strategy

JHU chose a strategy of implementing an application-agnostic repository layer and performing the AIHT using both DSpace 1.2.1 and Fedora 1.2.1. Because of this approach, it was also necessary for us to create a data model that could be supported by both of these repository applications. We chose to create a very simple high-level interface consisting of container ("DataGroup") and item ("DataItem") objects.

Although this metadata was of varying quality, it was the only metadata available that described the individual objects. Since this metadata comprises descriptive metadata, technical metadata, rights metadata, and digital provenance metadata, 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. We converted the various supplied and extracted 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 911archive database.

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 decided that the metadata would be most useful to us and to our partners if it were converted from its idiosyncratic original format to the widely supported standards mentioned in the preceding paragraph. To accomplish this, we created a crosswalk between the original format and MODS, METSRights and DIGIPROVMD. Additional metadata was captured from the digital objects and their file structure.

One major difficulty in analyzing the supplied 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 chunks using the Less for Windows file pager program. 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.

In order to keep the size of the METS objects relatively small, we created a separate METS object for each digital object and for each collection in the archive. We also considered creating METS objects at the archive collection level but decided against it. We defined a METS object as a discrete entry in the OBJECT table of the original archive. We then mapped all the associated metadata from OBJECT and other tables into the METS object.

II) Phase I: Ingest & Markup

The first phase was designed to track the ingestion and handling of a real-world archive, the George Mason 911 Archive. The objective for each participant was to transfer and ingest the archive, preserving the integrity of the content within the archive, and backup and restore the entire archive.

As mentioned earlier, we chose a model consisting of a collection of containers (DataGroup class) and the items they contained (DataItem class). These two classes of objects were derived from an abstract DataObject class.

DataObject:

* name
* id
* arbitrary string

DataGroup extends DataObject:

* set of DataGroup ids
* set of 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

For the 911 archive, we modeled each collection (<row> element within the single <COLLECTIONS> element) as a DataGroup. Each file was modeled as a DataItem. The collections within the archive were directories in the archive. We instantiated this model in different ways for different purposes. For management, we stored the DataObjects in a repository.

For transfer and ingestion we stored the DataObjects as METS files, one file per object. Each DataObject's METS file contains descriptive, administrative, and technical metadata -- derived from the 911 archive metadata -- and the other information required by the data model. When a METS file is loaded and turned into a DataObject, the METS file is stored as a string in the DataObject. The METS content was stored in order to associate the administrative and technical metadata in the METS file with the DataObject. This self-reference has some implications for migration and export. Since the METS stored with each DataObject is treated as the canonical representation, the METS must be rewritten when exported (the METS must refer to the local datastreams) and when a DataObject is migrated. We would design this differently, were we to do it again.

Within a DataItem, the actual archive file is referenced using URLs. Fedora, unfortunately, requires an "http" scheme URL for performing ingestion and could not read from a "file" schemeURL or be given datastream bytes inline. Therefore, we had to make the contents available via a web server to support ingestion.

A) Ingestion Process

Generate Submission Information Package (SIP)

We begin by generating the SIP, consisting of METS files based on the metadata provided with the archive. JHU chose to use the MySQL database tables for this process, though we also 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 and storing links to other files in the archive. Ideally, such a mechanism would be generalized so that similar dependency checking would be facilitated across a variety of content types.

During this process we discovered that there were some errors in the database. While evaluating these errors, we additionally discovered that there were inconsistencies between at least the MySQL and XML versions of the metadata.

Ingestion into Repository

** Ingestion process

Once we have the SIP format created, the content can be ingested into the repository. Within the archive, files reference each other using absolute or relative URLs, which are correlated to the archive identifier in the provided metadata. When we move these objects into the repository, however, we want to be able to manage these relationships at the repository layer.

To accomplish this, the ingestion process requires us to iterate over each DataObject twice. The first iteration reserves a repository identifier for each DataObject and constructs a mapping between this repository ID and the object's archive ID. The second iteration uses this mapping to handle references between DataObjects. The new repository identifier based relationships are stored back into the DataObjects before they are ingested into the repository.

Finally, the DataObject and associated content are stored into the repository. When the ingestion finishes, the root handle is returned. This is the repository identifier of the root DataGroup of the ingested tree of digital objects.

** Issues performing the ingestion

In both Fedora and DSpace, bulk ingestion was extremely slow. Ingestion initially took days to complete. As the amount of content in the repository grew, the time for an ingest stretched to a week or more. Late in the project, we tracked down and resolved several database problems in Fedora. After the fixes, Fedora ingestion took about a day, still a relatively long time. We also found database problems in DSpace, but did not have time to track them down until the very end of the project. We have reported these problems to the DSpace developer team.

In addition to their performance problems, both DSpace and Fedora imposed additional 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 fairly time-consuming. Often during ingestion, coding errors (especially in the early stages), local memory, and server resource issues caused abnormal termination. With the general slowness of the process and the need to restart the ingestion from the beginning, minor problems had the potential to -- and actually did -- cause long delays.

To improve overall performance, we implemented a checkpoint mechanism that would allow a restart at 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) is saved to a file. If the ingestion terminates, then it can 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.

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.

III) Phase II: Archive Export & Re-import

The goal of this phase of the project was for each participant to export its archive in a format of its own choosing. After all participants completed their exports, each participant selected one of the other participants' archives to ingest anew.

Text from the SOW: This phase will examine the issues and protocol requirements for the passing of whole archives -- with accompanying metadata -- between institutions operating different preservation regimes. Participants in the AIHT will develop strategies for this handoff through experience gained during the first phase.

Counter parties for the archive transfer will be chosen at random before the transfer is to occur. Each participant will export an archive to one other participant and will receive a copy of the archive from one other participant. Participants will work out shared assumptions between themselves leading to the development of a minimum set of broadly applicable standards.

A) Export

Export was a simple matter of starting with the root DataGroup object and recursing over the connected objects in the repository. The METS metadata and content datastream(s) associated with each object were written to disk as the object was traversed.

We needed to modify our METS format slightly to be able to load an exported archive. Our original format required http URLs. Now we also needed to support loading files from the same directory. We did this by allowing a DataItem hold an array of bytes or a URL. The METS format indicated data was available locally with a handle. The last issue was dealing with Fedora's requirement for http URLs. We had to modify the Fedora layer so that during ingest the bytes would be written out to a file accessible via a webserver.

B) Stanford Ingest of JHU Export

Stanford commented after their ingest of our archive that they would have expected one METS object for the entire archive. Because our approach used many METS files -- on the order of the number of items in the archive -- the Stanford ingest programs experienced out of memory conditions. This will be an area that we will look at for future archive ingest projects, though this situation may have been ameliorated, had they used the reference code provided by JHU.

This points to a broader issue observed during the various import processes during this phase. Three of the four non-LC participants, including JHU, used METS as part of the 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. Establishing this type of agreement early in the project likely would have improved the efficiency and success of this component.

C) JHU Ingest of Harvard Export

Adapting the Harvard archive to our repository layer was relatively easy. We simply had to convert the Harvard archive to our data model and then write a small amount of code to export it nicely. We chose to use a single root DataGroup that contained a DataItem for each file in the Harvard archive.

Our Harvard ingestion required two steps in order to keep memory usage down. (The two steps could be avoided by using a streaming parser, looping through the xml twice, and creating temporary files.) First, all of the metadata is dumped from the Harvard export.xml file to a directory with aith.tool.GenerateHarvard. Then aiht.tool.FedoraIngest can read from that directory and do the ingestion. We made the Harvard archive files available on a web server to avoid making another copy of the content and keep the ingestion efficient.

The Harvard export format is a directory structure organized in a hierarchy similar to the original 911 archive. For example, the file aiht/data/2004/12/21/29/49987.txt in the 911 archive maps to the following in the Harvard archive:
  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 - other info like mimetype and checksum for each version in a simple text format

We easily could have stored Harvard's export.xml in the repository as XML metadata of the root DataGroup, but we did not. Also note that we did not write a reader for the 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 archive.

IV) 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 of the project was to implement a flexible mechanism that would allow systematic migration of content that met specific criteria. We anticipate that the expertise of others in the area of 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.

A) Design

For the 911 archive we chose to migrate JPEGs to TIFFs and add metadata about the conversion to the TIFFs. The JPEGs are not deleted; we just add a new version to JPEG DataItems.

The migration state is saved to a file so it can be restarted. The metadata is NISO MIX XML stored in the ImageDescription field of the TIFF. We encoded information for the NISO MIX child elements of ChangeHistory, including DateTimeProcessed, ProcessingAgency, ProcessingSoftware, and ProcessingActions. We used the Java ImageIO library to do the conversion.

B) Migration process

Unfortunately the library appears to leak memory. The migration process eventually runs out of memory and stops. Additionally, the library also catastrophically fails on some input. We worked around this problem by automatically restarting the migration process if it fails. To speed up restarting, ids of migrated DataObjects are stored. The process would run out of memory 5 times during each migration.

There are about 12500 jpgs in the 911 archive. Of those 182 failed to convert. The Java ImageIO library throws null pointer exceptions and occasionally runs out of memory on some jpegs.

We eventually conducted an ingest into DSpace, migration, and export with only one DataItem failing. During migration we ran out of diskspace. Since the migration was not a transaction, the DataObject was left in an indeterminate state and could not be exported. (The underlying repository would have to roll back on errors in order for the repository layer to make the migration a transaction.)

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 that METS as well as add a new version to a DataItem. This complicated the repository implementations a somewhat, as the repositories now had to support the additional functionality of setting a string for each DataObject.

V) Lessons Learned and Recommendations

A) Lessons Learned

Technical Issues

Memory consumption was a big issue, even with what is, in the grand scheme of things, a relatively small archive, in terms of both absolute size and number of objects. Therefore, when processing objects, stream through them and write intermediate results to disk, instead of keeping everything in memory.

We made the export needlessly complicated by storing the METS 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 info 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 a migration and why.

After a bulk operation there should be an easy way to rollback the changes.

Be prepared for tools not to work perfectly (e.g., TIFF Java ImageIO library problems), especially when you have no control over their development.

Organizational Issues

Problems with contract negotiations significantly delayed the start of work. Additionally, in our case, this caused extra delays in hardware procurement because of a conflict with Apple Computer's product release cycle.

Because the project timeline did not correspond to the academic schedule, it was difficult to attract students. Most students had commitments by the time the project started.

Because the project involved only a single source archive, it allowed participants to over 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 see how that solution performs with another archive.

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. Basically, 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.

B) 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 that was difficult to work with. 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 are incorporating our recommendations into newer releases of Fedora.

DSpace

DSpace could only be manipulated by a local process which uses the DSpace java API to access storage. At the time of this writing, the DSpace@Cambridge project is completing work on a set of web services that would allow this process to be performed remotely. More information about this work is available on DSpace wiki at .

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 do this again, we would need to align our datastream naming conventions with the bitstream naming conventions of DSpace. This would be 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 at .

VI) Attached Appendices

A. Metadata mapping spreadsheet - AIHTtoMETSmapRev5-05.xls
B. DSpace and Fedora performance reports - DSpace_AIHT.rtf and Fedora_AIHT.rtf
C. Source code - aiht-distrib.tar.gz