2005

Archive Ingest and Handling Test

Michael L. Nelson
Old Dominion University

Johan Bollen
Old Dominion University

Giridhar Manepalli
Old Dominion University

Rabia Haq
Old Dominion University

Original Publication Citation

Nelson, M.L., Bollen, J., Manepalli, G., & Haq, R. (2005). Archive Ingest and Handling Test. *D-Lib Magazine, 11*(12), 1-17. doi:10.1045/december2005-nelson

D-Lib Magazine
December 2005

Volume 11 Number 12

ISSN 1082-9873

Archive Ingest and Handling Test

The Old Dominion University Approach

Michael L. Nelson, Johan Bollen, Giridhar Manepalli, and Rabia Haq

Old Dominion University
Department of Computer Science
Norfolk VA 23529
{mln, jbollen, gmanepal, rhaq}@cs.odu.edu

1.0 Introduction

The Archive Ingest and Handling Test (AIHT) was a Library of Congress (LC) sponsored research project administered by Information Systems and Support Inc. (ISS). The project featured five participants:

- Old Dominion University Computer Science Department
- Harvard University Library
- Johns Hopkins University Library
- Stanford University Library
- Library of Congress

All five participants received identical disk drives containing copies of the 911.gmu.edu web site, a collection of 9/11 materials maintained by George Mason University (GMU). The purpose of the AIHT experiment was to perform archival forensics to determine the nature of the archive, ingest it, simulate at least one of the file formats going out of scope, export a copy of the archive, and import another version of the archive. The AIHT is further described in Shirky (2005).

2.0 Research Performed

Old Dominion University (ODU) was the only non-library to participate in the AIHT. Consequently, whereas the other participants had (or were in the process of establishing) well-defined, production-level archiving systems, we did not have the capabilities or production-level burdens of an established archiving system. Instead, our focus was on alternative archiving concepts, including:

- self-archiving objects
- archive models & granularity
- archive ingest
- format conversion
- archive export & import

Another distinguishing characteristic of ODU's approach was the use of the MPEG-21 Digital Item Declaration Language (DIDL) complex object format. MPEG-21 DIDL is a complex object metadata format similar to the Metadata Encoding and Transmission Standard (McDonough, 2006), the format used by the other AIHT participants. The Los Alamos National Laboratory (LANL) first introduced digital library (DL) use of MPEG-21 DIDL (Bekaert et al., 2003). ODU has several collaborations with LANL, so we were eager to build our archives based on this format.

2.1 Buckets

Our original focus was to be on extending buckets (Nelson & Maly, 2001) to be true "self-archiving" digital objects. Unfortunately, we spent most of our time working on the archive models and MPEG-21 DIDL representations. Also, the display for buckets was previously optimized for small numbers of objects and not the 50k+ objects in the 9/11 GMU archive. We did perform considerable work on the bucket methods to adapt them to work on large archives, although interactive access to the individual resources does not really match the intended use of the bucket display. API-level access (admin method) to the bucket as an archival storage facility is the only thing that really makes sense (Figure 1). We did switch from a DOM to SAX parser for the display methods so that interactive display of the 9/11 GMU archive was possible.

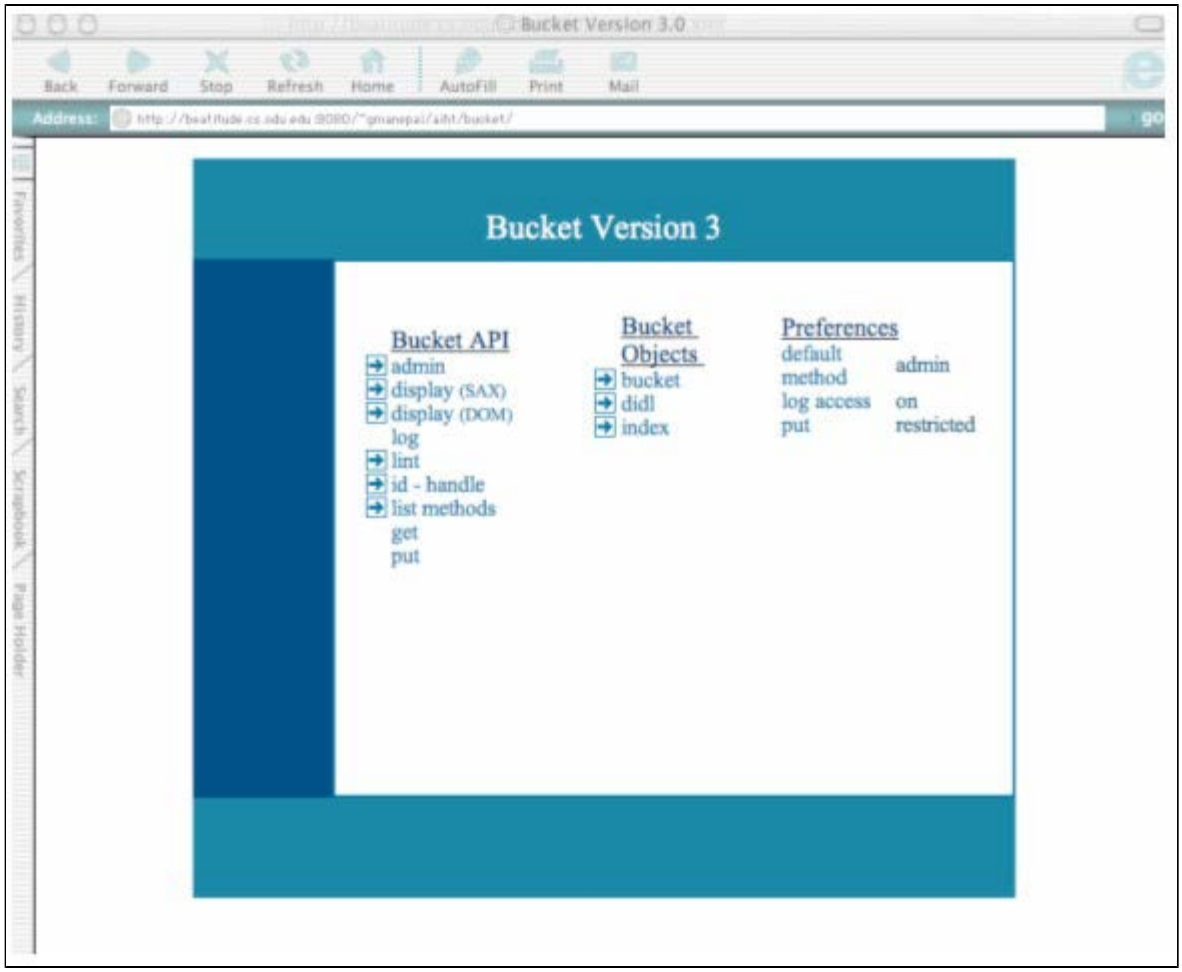

Figure 1. Bucket Methods Exposed as an Interactive Service.

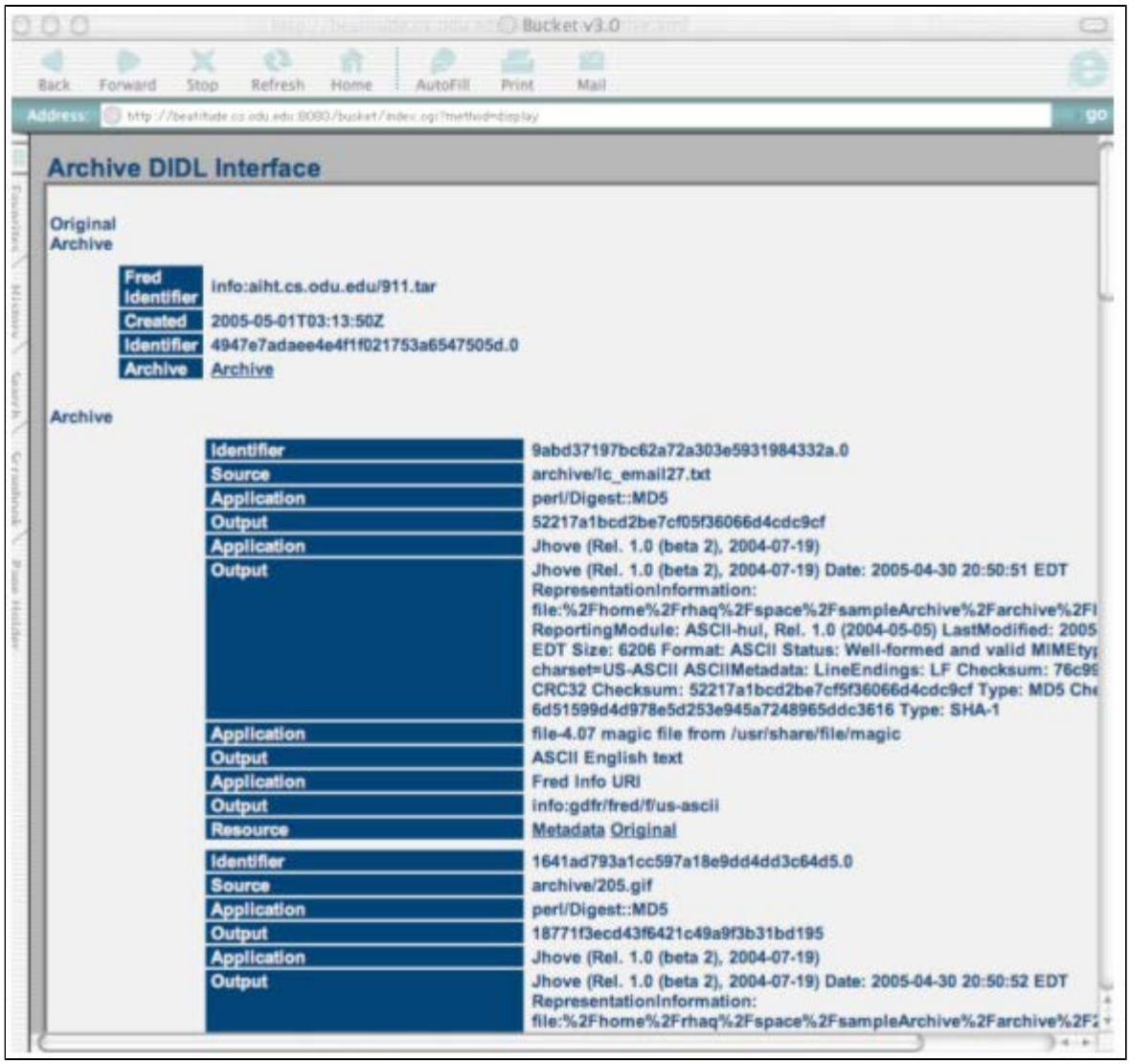

Figure 2. Interactive Bucket Display of the Test MPEG-21 DIDL.

If the user clicks on the resource tag "original", they will see the resource displayed. (Figure 3 shows one of the images from the test archive.) Clicking "metadata" will display the results shown in Appendix 3. This bucket can be further explored at: <http://beatitude.cs.odu.edu:8080/bucket/index.cgi>.

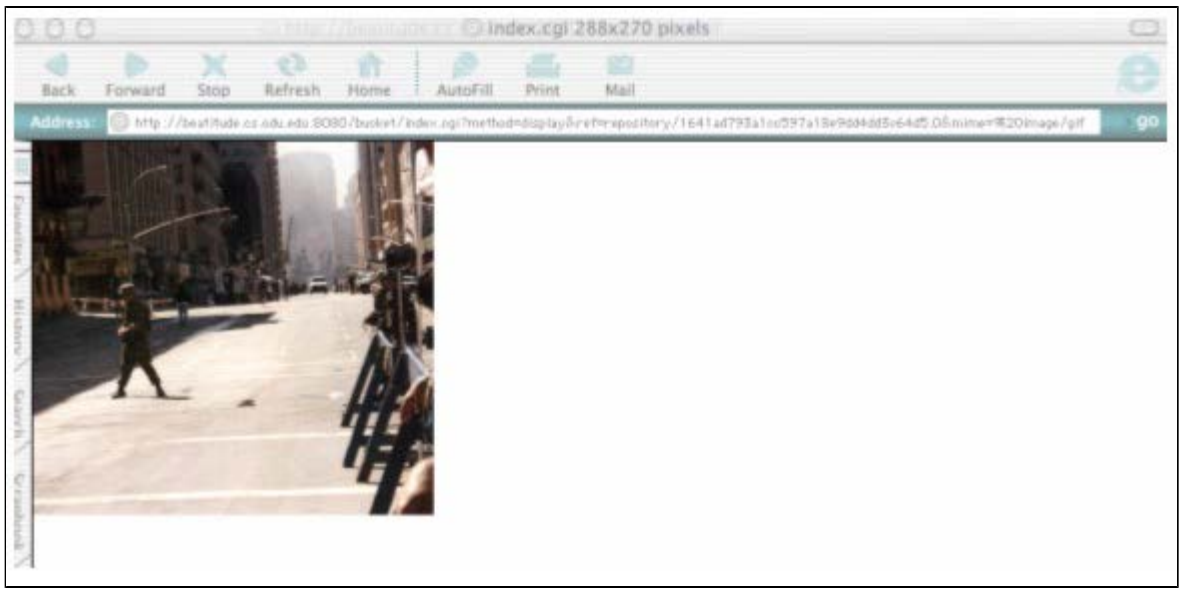

Figure 3. Display of an MPEG-21 Resource via a Bucket Display Method.

Work on the bucket API using MPEG-21 DIDLs is incomplete, and we plan continue this work even though the AIHT project has concluded.

2.2 Archive Models & Granularity

Since we had no archive model dictated by an existing software system or institutional procedure, we evaluated several models of representing the archive in a DIDL. In this article, we first begin with a short review of MPEG-21 DIDL terminology and its abstract data model.

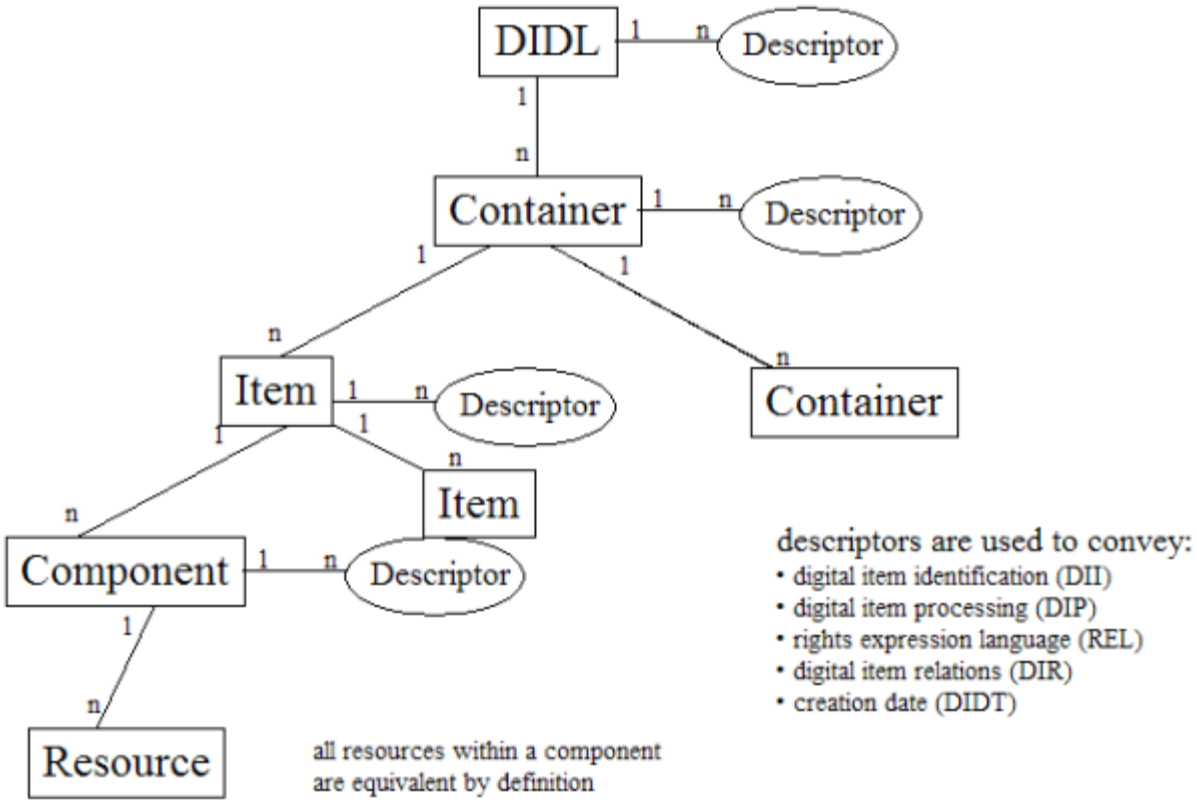

Figure 4. The MPEG-21 DIDL Abstract Data Model.

Unlike METS, MPEG-21 has an abstract data model that does not have specific semantics encoded in its declaration language. There are many nuances, but the most important concept is the definition of Containers, Items, Components and Resources. As Figure 4 shows, a DIDL contains at least one container and containers can be recursively defined. Containers eventually hold one or more Items, and Items hold one or more Components or Items. Components hold one or more Resources. Resources are the leaf nodes in the data model; they either contain URIs (by-reference representation) for data objects (PDFs, MPEGs, HTML, etc.) or contain the actual data objects in base64 encoded XML (by-value representation). Resources are the ultimate "thing" that we wish to convey, and the additional infrastructure allows the expression of the hierarchy and relationships between multiple data objects. Although a Component can contain multiple Resources, by definition the Components are considered to be equivalent representations; multiple Resources are generally specified in order to have by-reference or by-value representations, or possibly different encodings (e.g., .zip vs. .gz) of the same data object.

The other important consideration for understanding MPEG-21 DIDL is that every level in the hierarchy except for Resources can have extensible Descriptor elements (multiple Resources are bound together in a single Component, and the Component's Descriptors apply equally across all the Resources in the Component). Descriptors are simply wrapper elements; they can contain any XML encoded data. Some of the standard Descriptors that are defined by

MPEG-21 include digital item identifiers (DII), digital item processing (DIP), rights expression language (REL), digital item relations (DIR), and digital item creation date (DIDT).

We considered several different granularity models before settling on one file from the original .tar file equals one Component in the DIDL.

2.2.1 One archive = One Component

We considered not separating the untarring of the original .tar file and simply storing changes in the .tar file as a list of operations to apply to the original .tar file in different Components. While this would have been simpler from a bucket-point-of-view, it would not have resulted in an easily accessible archive. Even though the AIHT had no provisions for actually using the original archive (or reconstituting it as a web site), we did not believe that storing a largely unprocessed .tar file would be useful.

2.2.2 One archive = One Container

An almost opposite viewpoint to the above option was to represent each version of the archive (original, version at t_0, version at t_1, etc.) as a separate Container. This approach is optimized for access to different versions of the archive, and might be appropriate for browsing and retrieving different versions at different timestamps. We also considered an optimization where we would keep the original version of the archive, the current version, and a list of operations to perform to reproduce any intermediate versions. Again, although no access model was suggested in the AIHT parameters, we felt this approach would incur excessive overhead that did not match our anticipated access model.

2.2.3 One file = One Component

We settled on a model where we considered the .tar encoding simply an artifact of transmission and focused on the individual files. The file granularity is also tightly tied with our ingestion process, which is outlined below. It is important to stress that the model explained in this section represents our current thinking about archive representation in DIDL; other models are possible, and further use might lead to refinements. Although Appendix 1 gives a fully "expanded" view of the 8-Component test archive, we will walk through the archive structure here using the "+" and "-" XML display conventions of Internet Explorer to illustrate the final architecture. Figure 5 shows the top-level view of the archive, and the XML comments address the contents of the "collapsed" element immediately below it. Figure 5 shows one Container in the DIDL and two Descriptors (an identifier and creation date) for the container.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- archive.xml -->
- <didl:DIDL xmlns:didl="urn:mpeg:mpeg21:2002:01-DIDL-NS">
  - <didl:Container>
      <!-- Original Archive Identifier -->
    + <didl:Descriptor>
      <!-- Creation Date of this XML representation -->
    + <didl:Descriptor>
      <!-- Archive -->
    - <didl:Item>
        <!-- Original Archive -->
      + <didl:Item>
        <!-- Item File Name Mapping -->
      + <didl:Item>
        <!-- Archive Contents -->
      + <didl:Item>
      </didl:Item>
    </didl:Container>
</didl:DIDL>
```

Figure 5. The Top-Level View of the Archive.

Figure 5 shows one top-level Item, and that Item contains three sub-Items:

1. the contents of the original archive (unprocessed),
2. a mapping table of the file names as they were originally read from the .tar file and then mapped to the DIDL representation, and
3. the per-file contents of the archive.

Clicking on the last Item (per-file contents), one can see the file Components, Figure 6 illustrates eight separate Components, one for each file in the test archive.

Figure 6. Per-File Contents of the Archive.

Figure 7 shows a high-level view of a single component. There are four Descriptors associated with the Component, and although the test archive just has by-reference inclusion of the data objects in a single Resource, by-value and by-reference Resources are possible together or separately. The Descriptors associated with the Component reflect the introspection on the file performed at ingestion. The new file name is also described in the ingestion process.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- archive.xml -->
- <didl:DIDL xmlns:didl="urn:mpeg:mpeg21:2002:01-DIDL-NS">
  - <didl:Container>
      <!-- Original Archive Identifier -->
    + <didl:Descriptor>
      <!-- Creation Date of this XML representation -->
    + <didl:Descriptor>
      <!-- Archive -->
    - <didl:Item>
        <!-- Original Archive -->
      + <didl:Item>
        <!-- Item File Name Mapping -->
      + <didl:Item>
        <!-- Archive Contents -->
      - <didl:Item>
        - <didl:Component>
            <!-- File Identifier -->
          + <didl:Descriptor>
            <!-- MD5 Checksum of the File Contents -->
          + <didl:Descriptor>
            <!-- Output of Jhove -->
          + <didl:Descriptor>
            <!-- Output of file -->
          + <didl:Descriptor>
            <!-- Fred URI -->
          + <didl:Descriptor>
            <!-- File Resource -->
            <didl:Resource mimeType="text/plain" ref="repository/9abd37197bc62a72a303e5931984332a.0" />
          </didl:Component>
        + <didl:Component>
        + <didl:Component>
        + <didl:Component>
        + <didl:Component>
        + <didl:Component>
        + <didl:Component>
        + <didl:Component>
        </didl:Item>
```

Figure 7. A Top-Level View of a Single Component.

2.3 Archive Ingest

The purpose of our archive ingest process is to produce DIDLs. Ultimately, we intend for our DIDLs to be placed inside buckets, but where those buckets (and DIDLs) ultimately reside is a separate process. In short, we have no institutional repository for which we are ingesting; our process would likely be considered pre-ingest for traditional archiving operations. Figure 8 shows the top-level workflow diagram for the ODU ingest process.

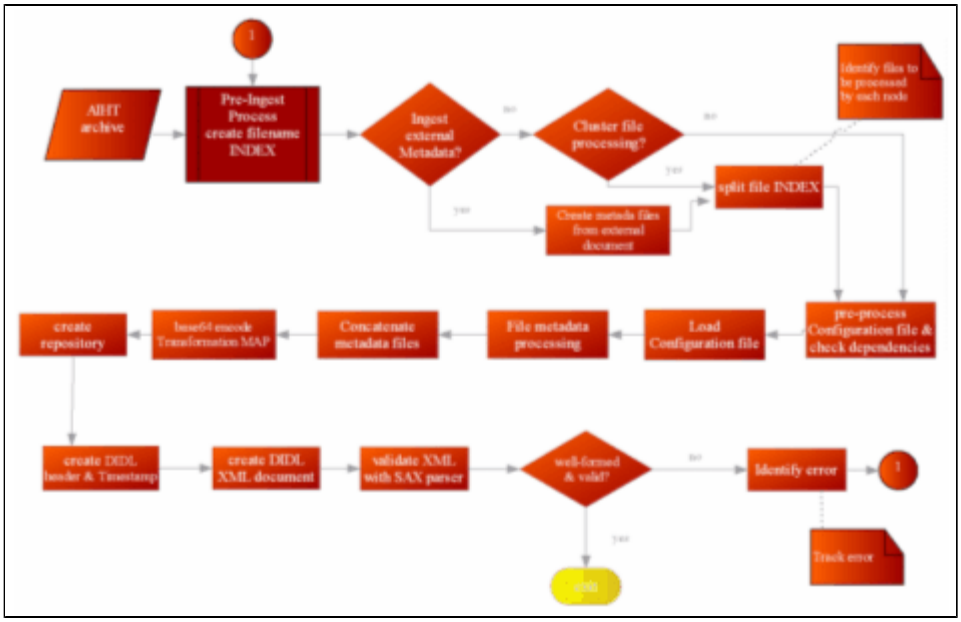

Figure 8. Archive Workflow.
(To see a larger version of Figure 8, click here.)

We parallelized our workflow process; Figure 9 shows the speed up for the AIHT source files on our 32-node Sun Solaris workstation cluster, including diminishing returns past 16 nodes.

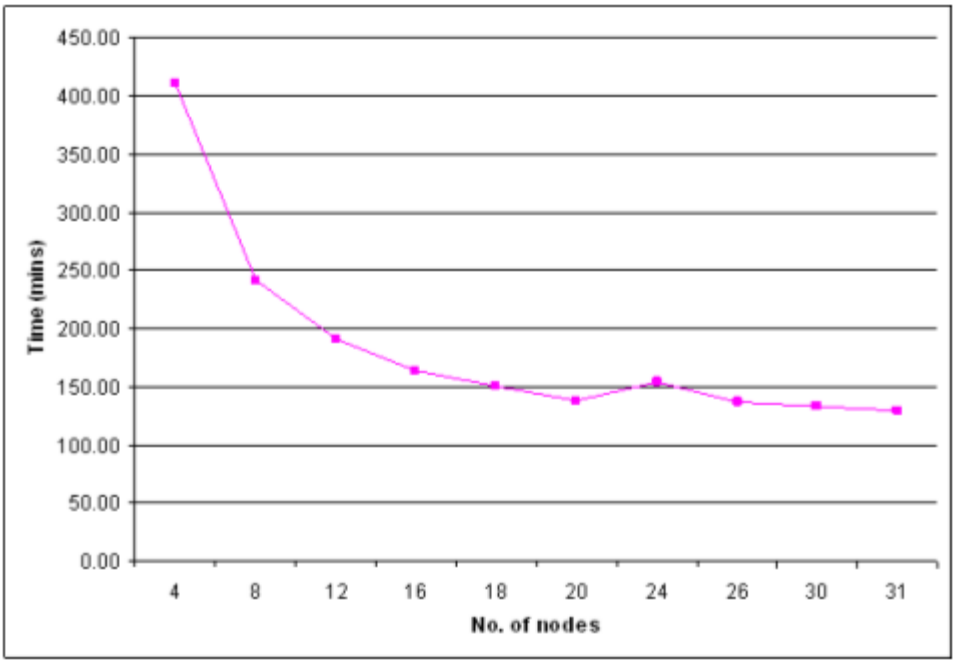

Figure 9. Speed Up for Parallel Ingest.

The Figure 8 node "file metadata processing" is expanded as a separate workflow process in Figure 10. This is the heart of the ingest process, and each part corresponds to the Component Descriptors shown in Figure 7. The process is designed to be expandable: both for future ingest processes and for post-archival introspection. This approach lies at the heart of our archiving philosophy: "truth" for a file is unlikely to be available, and the best we can do is to process the file in question with as many forensic utilities as possible. Each utility will provide a viewpoint for the file, and the union of these viewpoints could prove useful for future investigations. In future work, we would expand the number of such utilities, and we would include utilities capable of extracting descriptive (instead of just technical) metadata, such as "Essence" (Hardy & Schwartz, 1993).

The descriptors for storing the file metadata are explained further below, but have the general structure of importing Dublin Core (DC) semantics, with the program that is being run specified in DC.Creator and the program's output specified in DC.Description.

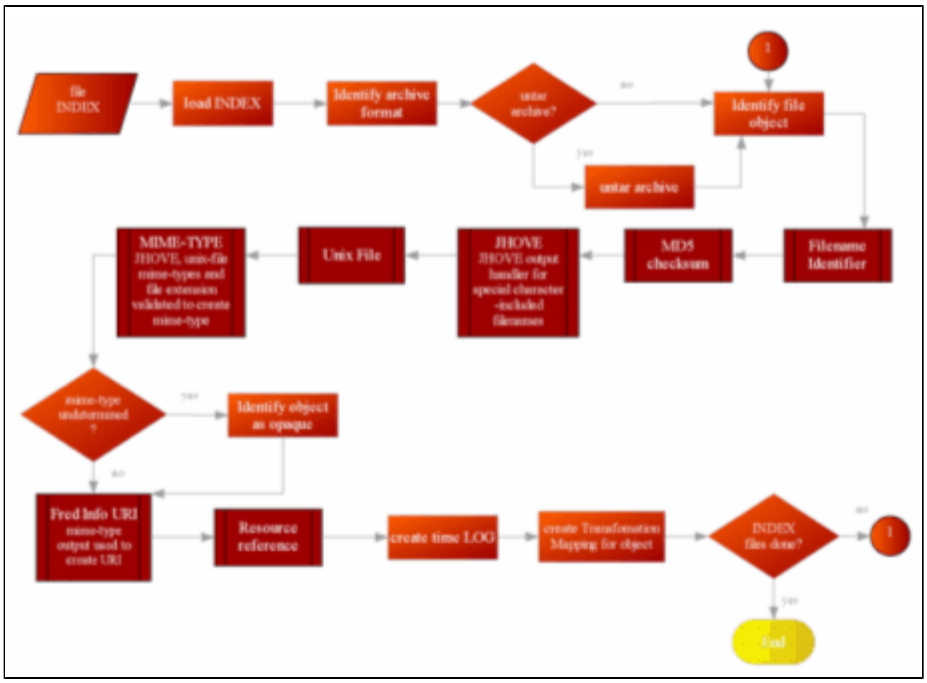

Figure 10. File Processing Workflow.
(To see a larger version of Figure 10, click here.)

2.3.1 File Identifier

We assign a new file identifier to replace the given file name in the original .tar file. It is based on the MD5 of the file name (not the file contents), appended with an integer indicating its revision level. This revision level is incremented if the file is updated (described in Section 2.4 below). The purpose of the new name is to remove any operating system unfriendly characters that might be in the file name. The corresponding XML fragment is given below:

```
<didl:Descriptor>
<didl:Statement mimeType="text/xml; charset=UTF-8">
  <dc:identifier xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">9abd37197bc62a72a303e5931984332a.0</dc:identifier>
  <dc:source xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">archive/lc_email27.txt</dc:source>
</didl:Statement>
</didl:Descriptor>
```

Although the (new, old) values are given above, a complete mapping file of all the old names to new names is created and inserted into the archive. This will aid services that later try to reconstitute the archive and need to maintain referential integrity. This Descriptor follows a slightly different format than the others, with DC.Identifier (new) and DC.Source (old) used instead of DC.Creator and DC.Description.

2.3.2 MD5 Checksum of the File Contents

A checksum of the file contents is computed so the veracity of the original file can be verified. It is important to not confuse this MD5 value (file contents) with the MD5 value specified in 2.3.1 (file name). The corresponding XML fragment is given below:

```
<didl:Descriptor>
<didl:Statement mimeType="text/xml; charset=UTF-8">
  <dc:creator xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">perl/Digest::MD5</dc:creator>
  <dc:description xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">52217a1bcd2be7cf05f36066d4cdc9cf</dc:description>
</didl:Statement>
</didl:Descriptor>
```

2.3.3 JHOVE Output

Each file was processed with JHOVE (JSTOR/ Harvard Object Validation Environment; hul.harvard.edu/jhove/) to create technical metadata about the file. JHOVE can provide voluminous technical metadata about a limited number of popular MIME types; it can be thought of as a "depth-first" approach to technical metadata. Although JHOVE can produce XML output, we used plain text output in our ingestion process. This is an artifact of trying to minimize the number of XML elements to speed up parsing when we had only a DOM-based parser. This could be changed in the ingest process configuration file. The corresponding XML fragment is given below:

```
<didl:Descriptor>
<didl:Statement mimeType="text/xml; charset=UTF-8">
  <dc:creator xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">Jhove (Rel. 1.0 (beta 2), 2004-07-19)</dc:creator>
  <dc:description xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">Jhove (Rel. 1.0 (beta 2), 2004-07-19) Date: 2005-04-
30 20:50:51 EDT RepresentationInformation:
 file:%2Fhome%2Frhaq%2Fspace%2FsampleArchive%2Farchive%2Flc_email27%2Etxt ReportingModule: ASCII-hul,
 Rel. 1.0 (2004-05-05) LastModified: 2005-04-10 20:25:35 EDT Size: 6206 Format: ASCII Status: Well-formed and valid
 MIMEtype: text/plain; charset=US-ASCII ASCIIMetadata: LineEndings: LF Checksum: 76c99b38 Type: CRC32
 Checksum: 52217a1bcd2be7cf5f36066d4cdc9cf Type: MD5 Checksum: 6d51599d4d978e5d253e945a7248965ddc3616
 Type: SHA-1</dc:description>
  </didl:Statement>
  </didl:Descriptor>
```

2.3.4 File Output

If JHOVE is used for depth-first technical metadata, then the Unix command "file" is used for breadth-first technical metadata. "file" knows very little about a wide variety of formats and would be a useful insight as to a file's format and purpose if JHOVE was unaware of the file type. The corresponding XML fragment is given below:

```
<didl:Descriptor>
<didl:Statement mimeType="text/xml; charset=UTF-8">
  <dc:creator xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">file-4.07 magic file from
 /usr/share/file/magic</dc:creator>
  <dc:description xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">ASCII English text</dc:description>
```

```
</didl:Statement>
</didl:Descriptor>
```

2.3.5 Fred URI

As a further attempt to classify the file in question, we perform a lookup in the Format Registry Demonstrator (Fred) to get an Info: URI to classify the file type. Fred is a precursor to the Global Digital Format Registry (GDFR) (Abrams & Seaman, 2003). The reasoning behind Fred and GDFR is that MIME types are biased toward engineering efficiency around a temporal bubble of today and are insufficient in a historical context for specifying the "type" of a file. For example, the MIME type "application/pdf" does not specify what version of PDF the file is, and different PDF versions have very different features and behaviors. The corresponding XML fragment is given below:

```
<didl:Descriptor>
<didl:Statement mimeType="text/xml; charset=UTF-8">
  <dc:creator xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">Fred Info URI</dc:creator>
  <dc:description xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">info:gdfr/fred/f/us-ascii</dc:description>
</didl:Statement>
</didl:Descriptor>
```

2.3.6 Resource

While not a Descriptor, the actual Resource for the file should be discussed. In particular, despite the presence of JHOVE, "file" and Fred, we ultimately assign the MIME type using the Perl module File::MMagic. This is because JHOVE is not good at guessing the MIME type of files that it does not know, "file" does not strictly return MIME types (cf. "ASCII English Text" in section 2.3.4) and few applications know how to process a Fred Info URI. Despite better alternatives, MIME types will be with us for some time. The corresponding XML fragment is given below:

```
<didl:Resource mimeType="text/plain" ref="repository/9abd37197bc62a72a303e5931984332a.0" />
```

2.4 Format Conversion

Format conversion is handled as a batch, post-archive process. We have a script that "walks" a DIDL and looks for MIME types matching the input parameter to convert to the specified output parameter. We have an additive, file-granularity model of additions to the archive. Figure 11 illustrates the process we leave the original file in place, but increment the integer suffix on the file name. We also add a new Descriptor to the previous version that contains an XPath pointer to the new version.

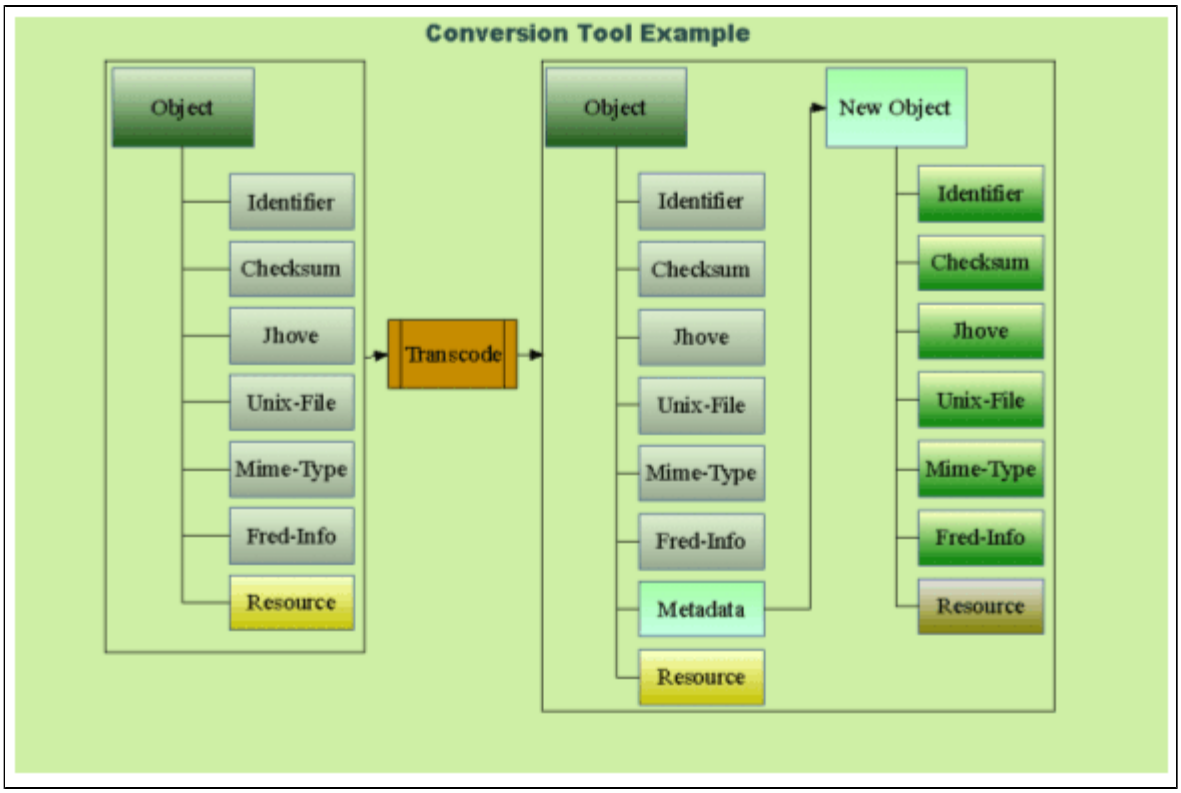

Figure 11. Conversion Process.

If the previous version of the Resource was specified as:

```
<didl:Resource mimeType="image/jpeg" ref="repository/9abd37197bc62a72a303e5931984332a.0" />
```

then the new version of the resource is specified as:

```
<didl:Resource mimeType="image/png" ref="repository/9abd37197bc62a72a303e5931984332a.1" />
```

We investigated converting PDF to Scalable Vector Graphics (SVG), an XML-based image format. But ultimately, the tools for SVG were not mature enough to make this feasible. We eventually used "transcode" (www.transcoding.org) to convert AVI to the DVD profile of MPEG-2 (VOB). "transcode" successfully converted approximately half of the AVIs, and the resulting VOBs were a factor of 5 larger. The results of "transcode" are shown in Figure 12.

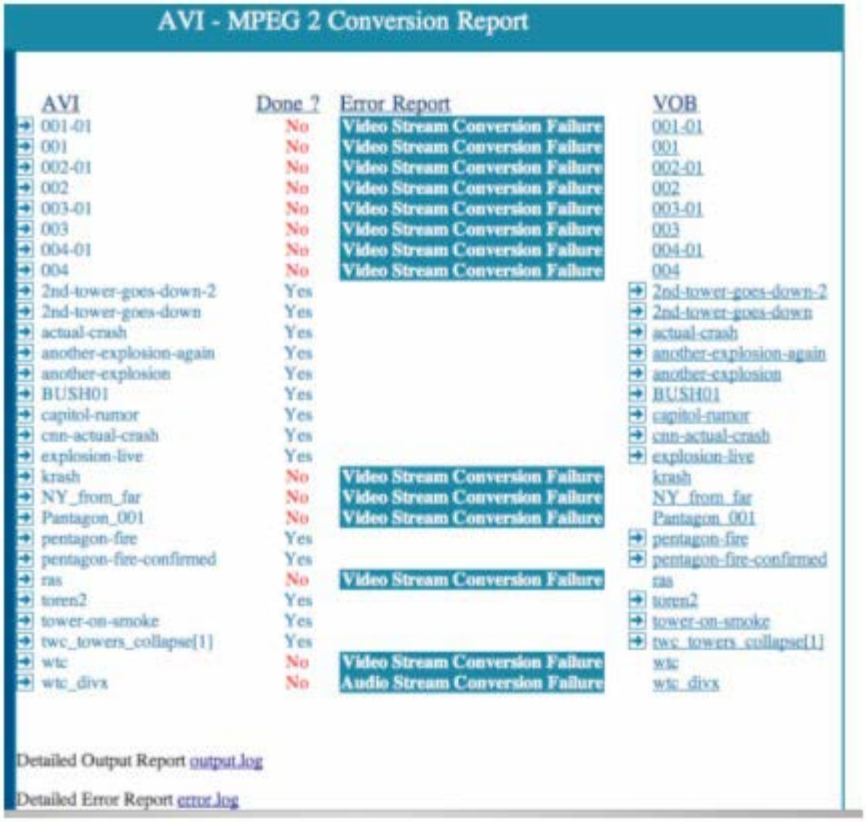

Figure 12. Transcode AVI -> VOB Conversion Results.

Our conversion script also used Image Magick to convert between common graphics formats. The results of JPEG -> PNG conversion (no errors) are available at: <http://beatitude.cs.odu.edu:8080/~gmanepal/aiht/convert/images/>.

2.5 Archive Export and Import

In OAIS terminology, MPEG-21 DIDL was both our Archive Information Package (AIP) and our Dissemination Information Package (DIP). Archive export for us is as simple as removing the DIDL from the bucket using the bucket API: <http://beatitude.cs.odu.edu:8080/bucket/index.cgi?method=get&id=didl>.

Our treatment of importing archives (Submission Information Packages (SIPs)) is both simple-minded and paranoid. Operating on a file-level granularity, we take whatever technical and descriptive metadata that comes with the file, save it into Descriptors, and then process the file as per the regular ingest method. Whatever metadata is attached with the file is stored, but not explicitly trusted. That is, if technical metadata was produced by JHOVE, we would still run our version of JHOVE on the new file. If we repeatedly imported our own DIDL, it would have the same files, but accumulate an increasing number of descriptors.

We chose the Harvard export METS profile because it appeared most similar to ours in terms of file granularity. Our DIDL structure remains unchanged; it simply has more Descriptors. We make no attempt to understand the external metadata we import; parsing and provenance are problems for future users of the archive. The following XML fragment shows how we encapsulated Harvard's metadata into Descriptors:

```
<didl:Descriptor>
<didl:Statement mimeType="text/xml; charset=UTF-8">
  <dc:creator xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://purl.org/dc/elements/1.1/
 http://dublincore.org/schemas/xmls/simpledc20021212.xsd">External Metadata</dc:creator>
  <dc:description xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:aes="http://www.aes.org/audioObject" xmlns:app="http://hul.harvard.edu/ois/xml/ns/drs/app"
```

```
 xmlns:mix="http://www.loc.gov/mix/" xmlns:tcf="http://www.aes.org/tcf"
 xmlns:txt="http://www.loc.gov/METS/text/" xmlns:xlink="http://www.w3.org/TR/xlink"
 xsi:schemaLocation="http://purl.org/dc/elements/1.1/ http://dublincore.org/schemas/xmls/simpledc20021212.xsd">
<file ID="F1" MIMETYPE="image/jpeg" SEQ="1" SIZE="194914" ADMID="T1"
 CHECKSUM="a7969810684c468525313b8282501405" CHECKSUMTYPE="MD5"
 OWNERID="aiht/websites/chnm/september11/REPOSITORY/CONTRIBUTORS/1199_photos/wtc_web/wetc5.jpg">
  <FLocat LOCTYPE="URL" xlink:type="simple" xlink:href="file:///aiht/data/2004/12/17/0/122.jpg" />
  </file>

<mix:mix>
<mix:BasicImageParameters>
<mix:Format>
  <mix:MIMEType>image/jpeg</mix:MIMEType>
  </mix:Format>

<mix:Compression>
  <mix:CompressionType>6</mix:CompressionType>
  </mix:Compression>

<mix:PhotometricInterpretation>
  <mix:ColorSpace>6</mix:ColorSpace>
  </mix:PhotometricInterpretation>

<mix:File>
  <mix:Orientation>1</mix:Orientation>
  </mix:File>

  </mix:BasicImageParameters>

<mix:ImageCreation>
<mix:DigitalCameraCapture>
  <mix:DigitalCameraModel>Canon Canon EOS D30</mix:DigitalCameraModel>
  </mix:DigitalCameraCapture>

  </mix:ImageCreation>

<mix:ImagingPerformanceAssessment>
<mix:SpatialMetrics>
  <mix:SamplingFrequencyUnit>2</mix:SamplingFrequencyUnit>
  <mix:ImageWidth>540</mix:ImageWidth>
  <mix:ImageLength>360</mix:ImageLength>
  </mix:SpatialMetrics>

<mix:Energetics>
  <mix:BitsPerSample>8 8 8</mix:BitsPerSample>
  </mix:Energetics>

  </mix:ImagingPerformanceAssessment>

  </mix:mix>
  </dc:description>
</didl:Statement>
</didl:Descriptor>
```

We also experimented with exporting our archive as a METS document. Appendix 2 shows the conversion of test archive (Appendix 1) into METS.

3.0 Lessons Learned

3.1 Large XML Files

We began this process rather stubbornly regarding large XML files. The current state of most XML enforces file size limits (most parsers are DOM-based and consume memory greedily). We considered these limitations to be an artifact of the here-and-now, and we tried to resist any engineering approach to subdivide the files into manageable chunks. We believed that the physical structure should more closely resemble the logical structure, and in a few

years the concept of what constitutes a "large" XML file will likely be very different.

However, we deviated from this approach slightly. Table 1 shows the various DIDLs we uploaded to the AIHT drop box. The first two were completely by-value inclusions of all files. The others retreated slightly to by-reference inclusion of the files, having a flat directory of md5 named files (tarred for uploading to the drop box). As Table 1 shows, the by-reference files are much smaller, but still quite large for commonly available tools.

Table 1. XML File Sizes.

Name	XML File Size (bytes)	Notes
DIDL.xml	15382712841 (15 GB)	By-Value. This first upload did not contain the files "outside" of the original .tar file
DIDL2.xml	35633037513 (35 GB)	By-Value. This upload contained all files (.tar file + database files)
DIDL3.xml	322653621 (322 MB)	By-Reference. All files. file size does not include .tar file (26 GB).
DIDL4.xml	407093487 (407 MB)	By-Reference. Harvard Import. File size does not include. tar file (24 GB).

Our ingestion process used a SAX-based parser, so large XML files were not a problem. But buckets originally used a DOM-based parser and needed to be converted to a SAX-based parser in order to process the files. More importantly, the display method for buckets was biased toward interactive access to the bucket contents. This does not match well with archives of any size, so "overview" adjustments needed to be made to insure that archive display was even possible. In the future we need to create a display method that will generally handle large files so that interactive display of contents (or at least a highly summarized view of the contents) is possible. We also plan to add support for more efficient file storage mechanisms, such as Internet Archive ARC files (Liu et al., 2005).

3.2 Acquisition Models

Perhaps the most important thing we learned during the AIHT dealt with the archive acquisition model. To some extent, the AIHT was an exercise in file forensics, and it served that purpose well. But there was a mismatch between a web crawler acquisition model and an institutional repository model.

With an institutional repository, there is controlled input and an internal data model. The content presumably is ultimately manifested in a web site, but any archive export would derive from the internal data model, not the web site representation. This is in contrast to a web crawling model, where the web site is traversed and the relationship between pages and links is used to recreate a model of the web site, if not the internal data model.

However, what we received was neither fish nor fowl. We received a file-system-based representation that had both web crawls and bits of an internal data model strewn throughout the file system. Although we did not have time to verify this, we believe it would have been ultimately easier to crawl the 911.gmu.edu web site ourselves to get the data in a web normalized form. The AIHT .tar file might have represented the most general case (one cannot focus on just web sites that are still active), but it probably represented the most difficult non-Byzantine case.

If we could assume cooperation on the part of the web site maintainer, then the easiest thing for them to do is to install mod_oai, an Apache module that streams out web site contents using the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) and complex object formats such as MPEG-21 DIDL (Nelson et al., 2005). mod_oai essentially converts web sites into OAIS DIPs. We cannot assume that all web sites will have mod_oai installed, but those web sites that would like to be archived by third parties could certainly use mod_oai to facilitate archiving. It would have been interesting to investigate GMU's use of mod_oai to see if it would make their web site dissemination task easier and the other AIHT participants' tasks easier as well.

Finally, this leads to our experiences with the Transfer Metadata (TMD). We did not use the TMD at any point in our ingest process. The only reason we would have used it would be for verifying the checksums of individual files. However, rather than further developing a TMD format, along with all the creeping featurisms that would inevitably result, we would like to see future attempts use complex object formats, such as METS or MPEG-21 DIDL. In essence, these formats are both the archived object and the TMD. If a TMD surrogate is needed, it can be computed from the complex object format.

Conclusions

For the AIHT, we encoded the content to be archived as an MPEG-21 DIDL complex object with the bucket API providing web access to the content. We created an archival data model that used internal identifiers in place of the original identifiers and does not delete the original files when they are migrated to new formats. Each individual file to be archived is mapped to an MEPG-21 DIDL "Component" element. We believe that metadata generation tools will continue to evolve, so we process the original file through an arbitrary number of introspective programs and store their respective output in separate MPEG-21 DIDL "Descriptor" elements. We make no attempt to evaluate or test the output of the various metadata generation tools – we only record their output for future forensic efforts.

Throughout the project, we took a decidedly "data-centric" view of archiving. This is a result of our belief that the repository itself is a preservation threat the data must survive. To this end, we aggressively sought to aggregate the content into a single file, even at the expense of creating XML files that were too large for some current tools to manipulate. Similarly, we believe using MPEG-21 as both AIP and DIP will result in easier future migrations.

References

Abrams, S. L. & Seaman, D. (2003). Towards a Global Digital Format Registry. *Proceedings of World Library and Information Congress: 69th IFLA General Conference and Council, Berlin Germany*. Available at: <http://www.ifla.org/IV/ifla69/papers/128e-Abrams_Seaman.pdf>.

Bekaert, J., Hochstenbach, P. & Van de Sompel, H. (2003). "Using MPEG-21 DIDL to Represent Complex Digital Objects in the Los Alamos National Laboratory Digital Library", *D-Lib Magazine*, 9(11). Available at: <doi:10.1045/september2003-bekaert>.

Liu, X., Balakireva, L. Hochstenbach, P. & Van de Sompel, H. (2005). File-based storage of Digital Objects and constituent datastreams: XMLtapes and Internet Archive ARC files, *Proceedings of ECDL 2005*, pp. 254-265. Available at: <http://arxiv.org/abs/cs.DL/0503016>.

Hardy, D. R. & Schwartz, M. F. (1993). Essence: A Resource Discovery System Based on Semantic File Indexing. *Proceedings of Winter Usenix*, pp. 361-373. Available at: <http://citeseer.ist.psu.edu/hardy93essence.html>.

Nelson, M. L. & Maly, K. (2001). Smart Objects and Open Archives. D-Lib Magazine 7(2). Available at <doi:10.1045/february2001-nelson>.

Nelson, M. L., Van de Sompel, H., Liu, X. Harrison, T. L., & McFarland, N. (2005). *mod_oai: An Apache Module for Metadata Harvesting*. arXiv Technical Report cs.DL/0503069. Available at: <http://arxiv.org/abs/cs.DL/0503069>.

McDonough, J. P. (2006). METS: Standardized Encoding for Digital Library Objects. *International Journal of Digital Libraries* (in press).

Shirky, C. (2005). AIHT: Conceptual Issues from Practical Tests. *D-Lib Magazine*, 11(12). Available at: <doi:10.1045/december2005-shirky>.

Appendix 1. Sample MPEG-21 DIDL (8 File Archive)

Appendix 2. METS Version of Test Archive Specified in Appendix 1

Top | Contents
Search | Author Index | Title Index | Back Issues
Previous Article | Next article
Home | E-mail the Editor

D-Lib Magazine Access Terms and Conditions

doi:10.1045/december2005-nelson