2008

Creating Preservation-Ready Web Resources

Joan A. Smith
Old Dominion University

Michael L. Nelson
Old Dominion University

ARTICLES

D-Lib Magazine
January/February 2008

Volume 14 Number 1/2

ISSN 1082-9873

Creating Preservation-Ready Web Resources

Joan A. Smith
Old Dominion University
<jsmit@cs.odu.edu>

Michael L. Nelson
Old Dominion University
<mln@cs.odu.edu>

Abstract

There are innumerable departmental, community, and personal web sites worthy of long-term preservation but proportionally fewer archivists available to properly prepare and process such sites. We propose a simple model for such everyday web sites which takes advantage of the web server itself to help prepare the site's resources for preservation. This is accomplished by having metadata utilities analyze the resource at the time of dissemination. The web server responds to the archiving repository crawler by sending both the resource and the just-in-time generated metadata as a straight-forward XML-formatted response. We call this complex object (resource + metadata) a CRATE. In this paper we discuss modoai, the web server module we developed to support this approach, and we describe the process of harvesting preservation-ready resources using this technique.

Introduction

Preservation is an on-going challenge for digital libraries, but even more so for the World Wide Web (The Web). Even though digital libraries are often accessed as web sites, anyone involved with digital libraries can easily point out the many differences between everyday web sites and a true Digital Library (DL). The Web is an unorganized amalgamation of digital pages with little metadata and unpredictable additions, deletions, and modifications – a crawlapalooza for the web robot. The typical DL has lots of metadata and well-organized content; it often has active preservation policies, and it is characterized by structured changes as well as support for protocols like OAI-PMH.

For an archiving repository seeking to preserve web sites, the site preparation process is challenging thanks to the wide variety of resource types and content that exist on web sites. Typically, an archivist will crawl the target web site then process each resource with various metadata utilities to extract technical information. The site author may also be interviewed for additional information which is manually added to the preservation record. In a sense, the archivist is applying the discipline of a digital library to the target web site. The archivist's ideal solution would have the site automatically provide both the original resource and maximum metadata - descriptive, administrative, technical - explaining the resource in preservation-relevant terms.

While archivists may understand web sites, webmasters typically know little about preservation models, metadata, and methods. From the webmaster's point of view, the ideal solution would be a tool installed on the web server which manages itself, and which automatically provides the "extra information" (i.e., metadata) that the archiving site needs to prepare the website for preservation, and which does not impact the normal operation of the web server. In this paper we discuss an approach which is simple for webmasters to implement and which addresses some of the metadata needs of the archivist.

Motivation

We begin by observing that digital preservation:

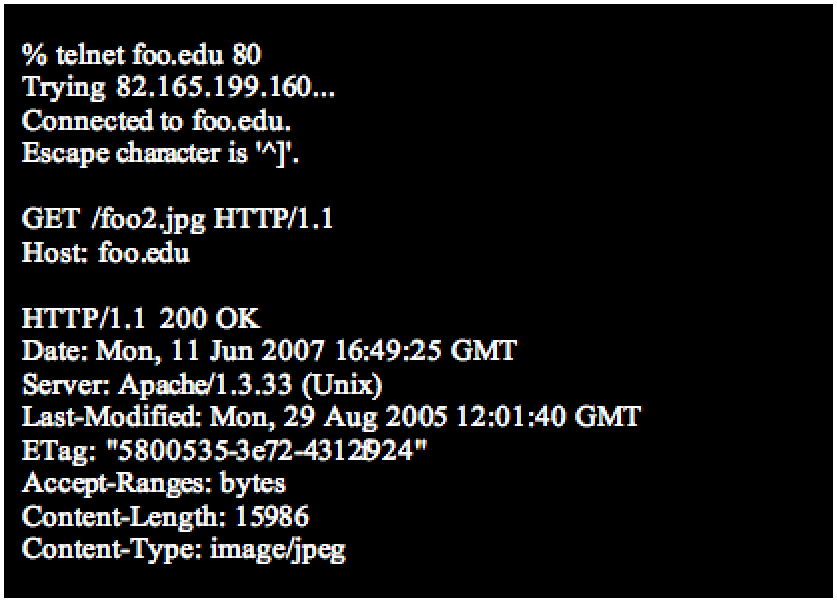
- remains in the niche of librarians and archivists
- is not sustainable as an ex post, ad hoc process
- should be congruent with practices of the general web community
- is applicable to content whose value is not always known in advance

As participants of the Archive Ingest and Handling Test (AIHT),[1] one of the lessons we learned was that preserving the GMU 911 web site was, in a sense, made more difficult because the website was not harvested directly from the web, but rather processed by site administrators and given directly to us. At first glance this would seem to be a benefit, but removing the files from the web context and preparing them for transmission actually applied an undocumented transformation, and ultimately it revealed a number of subtle but significant system incompatibilities that required manual attention. The very formulation of the AIHT made a subtle assumption that will not always be true: that a person will be available to prepare and transmit a web site for preservation. Web servers are optimized for the "here and now" and do not have the support of digital preservation as a functional design requirement. The purpose of the research reported in this article is to build a framework that allows dual access to web resources: the existing HTTP access mechanisms for conventional web agents, and a "preservation-ready" access channel that integrates the best tools of the digital preservation community into the web server.

In other projects we have investigated "lazy preservation"[2] and "just-in-time preservation,"[3] both of which assume the web site administrator has made no prior effort regarding preservation. In this project, we investigate the level of functionality possible if the administrator is willing to install an Apache module that will facilitate the preservation efforts of others. In short, we have created the software that we feel would have greatly simplified our tasks in the AIHT project.

A Review of HTTP Operation

A Web server automatically generates a very minimal amount of metadata. A typical HTTP response contains just enough information to enable the smooth transfer of content from web server to web client or crawler. Consider the following hypothetical request-response excerpt:



```
% telnet foo.edu 80
Trying 82.165.199.160...
Connected to foo.edu.
Escape character is '^]'.

GET /foo2.jpg HTTP/1.1
Host: foo.edu

HTTP/1.1 200 OK
Date: Mon, 11 Jun 2007 16:49:25 GMT
Server: Apache/1.3.33 (Unix)
Last-Modified: Mon, 29 Aug 2005 12:01:40 GMT
ETag: "5800535-3e72-4312f924"
Accept-Ranges: bytes
Content-Length: 15986
Content-Type: image/jpeg
```

The Last-Modified and ETag elements let the client/crawler determine if the resource content has changed since the last visit. The Content-Type element contains the MIME type of the resource if it is known. While there are other ways to determine a resource's MIME type, by default web servers rely on the filename extension to fill in the Content-Type field of the response. It is up to the client to figure out how to handle the file, and that varies greatly from system to system. Table 1 lists various MIME types, including some that most web browsers will not recognize. The type "application/vrml" in Table 1 was in wider use ten years ago but is typically not understood by today's browsers. IANA's registration[4] listing provides an interesting view of the sliding window of file-format popularity. The web server, after all, is focused on communicating today rather than tomorrow, and as new "types" come into vogue, older types tend to fall into disuse. In short, web server MIME typing has serious limitations when it comes to providing adequate preservation information about the data format of web resources.

MIME Type	Usage	Example Target Application
model/gsm	Geometric description language	Graphisoft's ArchiCAD
x-world/x-vrml	Virtual reality modeling	Alteros 3D
application/dicom	Medical imaging	MIR CTN software
image/xif	Scanning and OCR	Pagis
image/tiff	High quality images	Pagemaker; Photoshop
application/fdf	PDF forms data exchange (extension = "fdf")	Adobe Reader' Excel; Oracle
application/pdf	Print-quality documents	Adobe Reader, Foxit

Table 1: Example MIME Types. The reader is invited to test the linked resources to see how different browsers handle (or fail to handle) these types. Note that some browsers, especially Internet Explorer, "re-type" resources using various methods. The same browser may behave differently on different computers, even with the same underlying OS.

Preservation Metadata

Even if the MIME type happens to be correct, from a preservation perspective the sparse metadata available from blind crawling poses a challenge for the archivist. Consider our example resource, the "foo2.jpg" image. To preserve the image, we ought to have composition information such as the image map and NISO details, along with supporting data such as subject matter (two Foos having lunch), creator (photographer=Mr. Foo) and, possibly, origination hardware (Nikon CoolPix 500) and software (Adobe Photoshop Essentials version 2.5). Usually, archives will process each resource upon ingestion, adding the necessary metadata and analyzing the resources using tools like ExifTool or JHOVE. On a small scale, this "many-resources-to-one-archival-crawler" approach has certain advantages, particularly since skilled archivists are involved in determining the appropriate metadata for the ingested resources. In addition, coordination between archive and source site is probably mostly contemporaneous or nearly so. That is, there is relatively little delay between the crawl and archival ingestion processing.

What happens on a larger scale, though? At what point does such post-crawl processing overwhelm the ingesting archive or result in a long delay between crawl and analysis? Clearly, getting the responding server to preprocess the resource and include the results together with the original resource in one complex-object response would help both the particular archivist and the general goal of web preservation. The best time to get information about a resource is at the time of the request.

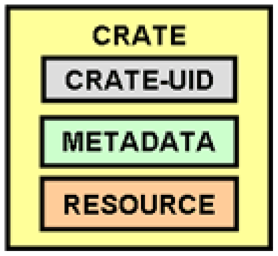
Metadata Utilities

How can metadata be derived for web resources? Several tools have been developed in recent years that can be used to analyze a web resource. The limitations of MIME typing as currently implemented by web servers has led to projects like the Global Digital Format Registry (GDFR)[5] and Pronom's[6] DROID tool, which provide a deeper introspection of the resource's format. Once the format type is known and described, additional utilities can extract information like keywords and subject matter, or derive an abstract from text content. JHOVE,[7] which arose from Harvard's JSTOR project, can identify, validate and characterize a number of file types including images (JPEG, GIF, PNG, etc.), text (HTML, XML), and PDF documents. Table 2 lists some common utilities that can be helpful for processing resources in preparation for digital preservation.

Name	Description
JHOVE	Analysis & characterization by type (img, audio, text)
Kea	Key phrase extraction
OTS	Open Text Summarizer
ExifTool	Image/video metadata extractor
PDFlib-pCOS	Extract PDF metadata (commercial tool)
MP3-TAG	Extract audio file tags
Essence	Customized information extraction
GDFR	Extended MIME file typing
MD5	Message Digest
File Magic	Type identification using special ID bits of the file
DROID	File signature analysis (internal and external)

Table 2: Some utilities for producing resource metadata.

Each of these utilities has a command-line interface, so a web resource could be analyzed by each of them in series, with the output redirected to, say, a single external file. Since utility output is typically ASCII text or XML, the content is human-readable – an advantage for long-term resource preservation. If we combine this output with a Base64-encoding of the resource, we would have a neatly packaged, pre-processed web resource ready for archive ingestion and preservation preparation. We call this complex-object model a "CRATE" (Figure 1, below).



CRATE-UID: A disambiguating identifier such as a URI or UUID.

METADATA: Analysis results from utilities like Jhove or Kea applied to the resource.

RESOURCE: The original resource, encoded as Base64.

Figure 1: CRATE Model.
A CRATE consists entirely of XML-formatted, plain ASCII (human-readable) content.

The concept calls for the disseminating web server to preprocess the resources it serves up by using metadata-generation utilities, such as those described here, and to serialize this information together with the Base64-encoded resource in a simple XML-formatted complex object response. How can the web server be configured to provide such a response? The answer lies in the Apache web server module, mod_oai.

mod_oai and the CRATE Model

We mentioned earlier that web administrators often install additional modules on web servers. They are specified in the web server's configuration file, using a simple enabling directive:

LoadModule <module-name> <path/to/module.so>

Common examples for the Apache web server are mod_perl and mod_python (Perl/Python-CGI optimizers), mod_ssl (to support secure socket layer connections), and mod_jserv (Java servlet engine). There is also the **mod_oai** module which enables the web server to process OAI-PMH-style requests,[8] [9] like "GetRecord" and "ListIdentifiers." By selecting a complex object metadata format such as MPEG-21 DIDL, the user can issue a resource request and get both the resource and associated metadata returned together in the response. As illustrated in Figure 2 below, normal user access to the site is unchanged; users who want to make OAI-PMH requests add "modoai" to the root URL, and append the OAI-PMH request string. The details are specified using the "Location" directive:

```
<Location /modoai>                     ←  If the URL beings "http://foo.edu/modoai"
    SetHandler modoai-handler          ←  ...then process the rquest using the mod_oai module
</Location /modoai>
```

Any request which begins with "http://foo.edu/modoai" will be processed by the mod_oai module, checked for proper OAI-PMH syntax, a valid identifier and parameters. The response will be in the requested metadata format.

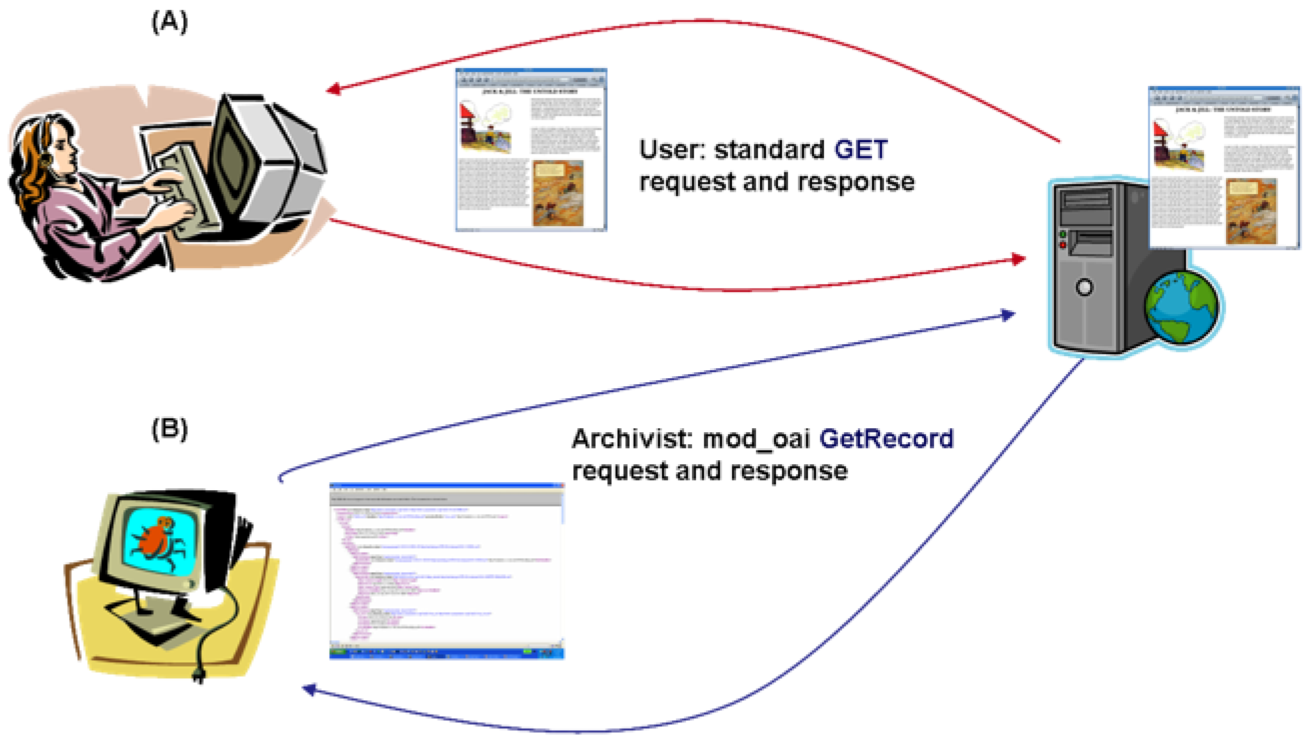

(A)
User: standard GET request and response
(B)
Archivist: mod_oai GetRecord request and response

Figure 2: (A) Normal web page request and (B) OAI-PMH request

For the example given in Figure 2-B, the mod_oai response is returned as human-readable ASCII in the DIDL XML format (called a "DID"), with the resource encoded in Base64 and with the HTTP response headers included as basic metadata (see Table 3, below).

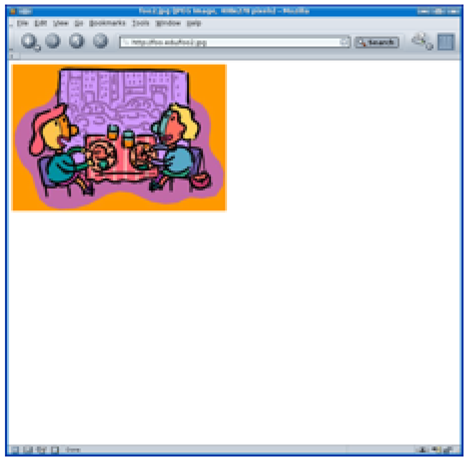
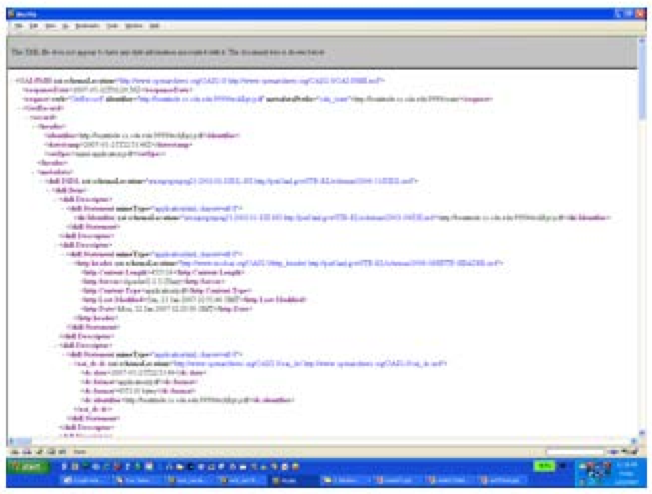
Standard Request (URL)	**mod_oai Request (URL)**
http://foo.edu/foo2.jpg	http://foo.edu/modoai?verb=GetRecord&metadataPrefix=oai_didl &identifier= http://foo.edu/foo2.jpg
The request in HTTP	**The request in HTTP**
Standard Response	**mod_oai Response**
 *The JPEG image as seen in a browser*	 *The JPEG as an MPEG-21 DIDL response(browser view)*

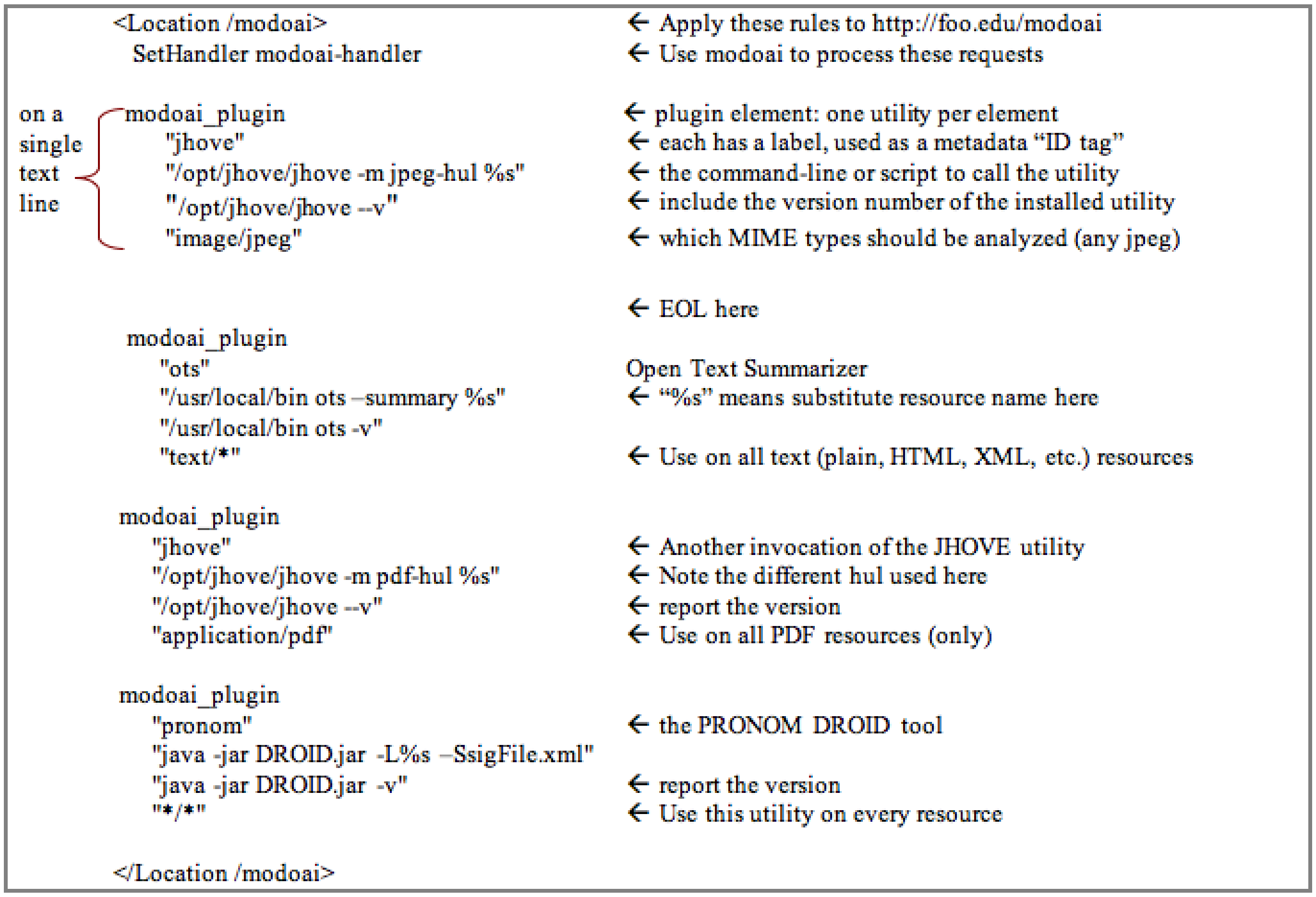
Table 3: Response to standard request vs. OAI-PMH (mod_oai) request.

Note that the HTTP protocol is operating as usual. The GET request contains the OAI-PMH request within it. If we look at the log entry, we will see:

> **"GET /modoai?verb=GetRecord&metadataPrefix=oai_didl&identifier=**
> **http://foo.edu/foo2.jpg HTTP 1.1"**

The original "oai_didl" response was formatted using the MPEG-21 DIDL standard, and contained the OAI-PMH request data, the HTTP headers from the response, and the resource encoded in Base64. No additional metadata was available.

By re-engineering mod_oai to have a plugin architecture, we were able to add utilities which would process each resource upon request when the "oai_crate" metadata prefix is specified. Any utility which has a command-line interface can be included. For utilities that are best applied to specific file types, the configuration directive can limit processing to those files. For example, JHOVE has a number of "hul" tools, each geared to analyzing a certain MIME type: the "pdf-hul" for Adobe PDF files, the "gif-hul" for GIF images, etc. Figure 3 shows a hypothetical Location directive for the new mod_oai, with utilities invoked based on the MIME type of each resource.


```
         <Location /modoai>                          ← Apply these rules to http://foo.edu/modoai
           SetHandler modoai-handler                 ← Use modoai to process these requests

on a    ┌modoai_plugin                              ← plugin element: one utility per element
single  │    "jhove"                                ← each has a label, used as a metadata "ID tag"
text   <     "/opt/jhove/jhove -m jpeg-hul %s"      ← the command-line or script to call the utility
line    │    "/opt/jhove/jhove --v"                 ← include the version number of the installed utility
        └    "image/jpeg"                           ← which MIME types should be analyzed (any jpeg)

                                                    ← EOL here
          modoai_plugin
             "ots"                                  Open Text Summarizer
             "/usr/local/bin ots –summary %s"       ← "%s" means substitute resource name here
             "/usr/local/bin ots -v"
             "text/*"                               ← Use on all text (plain, HTML, XML, etc.) resources

         modoai_plugin
            "jhove"                                 ← Another invocation of the JHOVE utility
            "/opt/jhove/jhove -m pdf-hul %s"        ← Note the different hul used here
            "/opt/jhove/jhove --v"                  ← report the version
            "application/pdf"                       ← Use on all PDF resources (only)

         modoai_plugin
            "pronom"                                ← the PRONOM DROID tool
            "java -jar DROID.jar -L%s –SsigFile.xml"
            "java -jar DROID.jar -v"                ← report the version
            "*/*"                                   ← Use this utility on every resource

         </Location /modoai>
```

Figure 3: Configuring mod_oai with the new plugin architecture.

For mod_oai, the utilities only operate on resources when the CRATE metadata format is requested:

> **http://foo.edu/modoai?verb=ListRecords&metadataPrefix=oai_crate**

The response is written in the MPEG-21 DIDL XML document type, using LANL's implementation style.[10] For our example image of two Foos having lunch, the output would look something like Figure 4, below. (Note that browsers vary greatly in how they display XML, and may suppress some or all of the content of an XML document. In this case, the output is best viewed in a plain text window).

```
<OAI-PMH xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
 http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
<responseDate>2007-08-23T12:24:31Z</responseDate>
<request verb="GetRecord" identifier=" http://foo.edu/foo2.jpg "
 metadataPrefix="oai_crate">http://foo.edu/modoai/</request>
<GetRecord>
        <record>
                <header>
                        <identifier> http:/foo.edu/foo2.jpg </identifier>
                        <datestamp>2006-08-21T22:47:43Z</datestamp>
                        <setSpec>mime:image:jpeg</setSpec>
                </header>

<metadata>
<didl:DIDL xsi:schemaLocation="urn:mpeg:mpeg21:2002:02-DIDL-NS
 http://purl.lanl.gov/STB-RL/schemas/2004-11/DIDL.xsd">
<didl:Item>
        <didl:Descriptor>
                <didl:Statement mimeType="application/xml; charset=utf-8">
                <dii:Identifier xsi:schemaLocation="urn:mpeg:mpeg21:2002:01-DII-NS
 http://purl.lanl.gov/STB-RL/schemas/2003-09/DII.xsd">
                        http:/foo.edu/foo2.jpg
                </dii:Identifier>
                </didl:Statement>
        </didl:Descriptor>

<didl:Descriptor>
    <didl:Statement mimeType="application/xml; charset=utf-8">
        <crate:plugin xsi:schemaLocation="http://www.cratemodel.org/OAI/
        2.0/crate/CRATE1a.xsd">
                <crate:PluginLabel>jhove</crate:PluginLabel>
                <crate:PluginVersion> Jhove (Rel. 1.1, 2006-06-05)
</crate:PluginVersion>
                <crate:PluginContent>
Jhove (Rel. 1.1, 2006-06-05)
 Date: 2007-08-23 08:24:31 EDT
 RepresentationInformation: /var/www/html/foo2.jpg
  ReportingModule: JPEG-hul, Rel. 1.2 (2005-08-22)
  LastModified: 2006-08-21 18:47:43 EDT
  Size: 64609     Format: JPEG     Version: 1.00     Status: Well-Formed and valid
  SignatureMatches: JPEG-hul  MIMEtype: image/jpeg Profile: JFIF
  JPEGMetadata:
        .......
                </crate: PluginContent>
        </crate:plugin>
     </didl:Statement>
</didl:Descriptor>

<didl:Descriptor>
    <didl:Statement mimeType="application/xml; charset=utf-8">
        <crate:plugin xsi:schemaLocation="http://www.cratemodel.org/OAI/
        2.0/crate/CRATE1a.xsd">
                <crate:PluginLabel>md5</crate:PluginLabel>
                <crate:PluginVersion> MD5-Sum Version </crate:PluginVersion>
                <crate:PluginContent>
                        44b34c4ce42509188ecbd9d08530b363        foo2.jpg
                </crate: PluginContent>
        </crate:plugin>
     </didl:Statement>
</didl:Descriptor>

<didl:Descriptor>
    <didl:Statement mimeType="application/xml; charset=utf-8">
        <crate:plugin xsi:schemaLocation="http://www.cratemodel.org/OAI/
        2.0/crate/CRATE1a.xsd">
                <crate:PluginLabel>file</crate:PluginLabel>
                <crate:PluginVersion>File Magice Version 1.1.1</crate:PluginVersion>
                <crate:PluginContent>
                foo2.jpg: JPEG image data, JFIF standard 1.00, resolution (DPI),
"LEAD Technologies Inc. V1.01", 36 x 27
                </crate: PluginContent>
        </crate:plugin>
     </didl:Statement>
  </didl:Descriptor>
</metadata>

<didl:Component>
```

```
        <didl:Resource mimeType="image/jpeg" encoding="base64">
           /9j/4AAQSkZJRgABAAEAJAAbAAD//gAfTEVBRCBUZWNobm9sb2dpZXMgSW5jLiBWMS4wMQD
           .....
           iigAooooAKKKKACiiigD//2Q==
</didl:Resource>
<didl:Resource mimeType="image/jpeg" ref="http://foo.edu/foo2.jpg"/>
</didl:Component>
</didl:Item>
</didl:DIDL>
</metadata>
</record>
</GetRecord>
</OAI-PMH>
```

Figure 4: Example CRATE response from mod_oai.

To save space, the utilities' output has been abbreviated here. An actual example of the analysis of this "foo2.jpg" image, along with output from additional utilities, can be seen in the Appendix to this article. The utilities enabled on this server are:

- **File** - Looks at the "magic bytes" to determine MIME type
- **MD5** - Provides the MD5 hash value of the file
- **JHOVE** - File analyzed using the HUL appropriate to specific file type
- **DROID** - Pronom's DROID utility which evaluates MIME type
- **ExifTool** - Phil Harvey's Perl script which analyzes images

Best-effort Metadata

The approach described here is a best-effort approach to metadata. The information produced is:

- Unverified
    - Utility results are not cross-checked for validity.
    - Output of analyses is mapped directly into the XML response "as is".
    - If two utilities produce conflicting analyses, both are reported without preference or alteration.
- Undifferentiated
    - The plugin utility output is not categorized into types like "administrative" or "technical" metadata.
    - Both the resource (encoded as Base64 for ASCII readability and long-term viability) and the utilities' metadata are contained in the CRATE response.
- Extemporaneous
    - The information is generated at the time of dissemination.
    - The information is applied to each resource according to specifications in the web server configuration file. New utilities or other modifications to the configuration may produce a different response.

Some utilities have optional configurations that allow the output to be written in XML rather than in plain text. The JHOVE analysis in Figure 4 uses the plain-text output option. In the Appendix to this article, we have changed the "jhove.conf" configuration file to use XML. Even so, no XML-level validation occurs during the CRATE response. Instead, the JHOVE output is wrapped in [CDATA] tags to protect the surrounding oai_crate XML content from being "broken" by the utility's embedded XML.

Given the complexity of web server administration in general, it is perhaps inevitable that a utility will be incompletely installed or updated, or will otherwise behave unexpectedly. What happens when there is a problem with the utility? The error details are treated as utility output and encapsulated within the CRATE response. Again using JHOVE as an example, if it is installed, but a configuration file has not been defined, the following error message might be displayed:

```
edu.harvard.hul.ois.jhove.JhoveException: Configuration file not found or not readable;
 use -c to specify
 at edu.harvard.hul.ois.jhove.JhoveBase.init(Unknown Source)
      at Jhove.main(Unknown Source)
```

Figure 5. Error message generated in response to the "version" command when the configuration file has not been defined.

The error message is reported any time that JHOVE is invoked without a proper configuration file reference. Since a configuration file error would impact both the "version" command and the resource-analysis command, the error would appear in both sections of the response for that plugin:

```
<didl:Descriptor>
    <didl:Statement mimeType="application/xml; charset=utf-8">
        <crate:plugin
xsi:schemaLocation="http://www.modoai.org/OAI/2.0/crate/CRATE1a.xsd">
                <crate:PluginLabel>jhove</crate:PluginLabel>
        <crate:PluginVersion>edu.harvard.hul.ois.jhove.JhoveException: Configuration
file
        not found or not readable;
                use -c to specify
                at edu.harvard.hul.ois.jhove.JhoveBase.init(Unknown Source)
                at Jhove.main(Unknown Source)
        </crate:PluginVersion>
<crate:PluginContent>
                edu.harvard.hul.ois.jhove.JhoveException: Configuration file not found
or not readable;
                use -c to specify
                at edu.harvard.hul.ois.jhove.JhoveBase.init(Unknown Source)
                at Jhove.main(Unknown Source)
</crate: PluginContent>
        </crate:plugin>
        </didl:Statement>
</didl:Descriptor>
```

Figure 6. Error message generated in response to the resource-analysis command when the configuration file has not been defined.

Conclusions

From the archiving crawler's perspective, it is clearly more efficient to have the servers at 100 sites each pre-analyze its own resources than it is for the archiving server to analyze the resources of 100 sites. How practical is it for the disseminating server to perform this metadata analysis? Does this "extra" effort overwhelm the web server to the point that users requesting regular pages are frustrated by slow response time? We are currently collecting metrics on the impact to servers, particularly as metadata utility performance affects server responsiveness to regular (browser) clients. However, sites can make an agreement with an archiving site to (a) arrange a specific crawling timeframe and/or (b) only allow selected crawlers to perform this kind of enhanced crawl, either by user/password agreement between the parties or by other common access restrictions (crawler's host IP, for example).

Another consequence of this approach that we are investigating is the content-length of the full response. XML files can grow very large, very quickly, and many utilities produce extremely verbose output, particularly where images are concerned. There are several mechanisms for dealing with this issue. Resumption Tokens are used by OAI-PMH servers to control the number of records returned to a

ListRecords query. HTTP has the "Transfer-Encoding: Chunked" option which can alleviate the problem. Finally, the crawler's XML parser can be designed to filter the incoming stream to meet the repository's storage limitations. All of these can be combined to produce an acceptable archiving solution. In summary, we believe this approach offers important metadata opportunities for repositories tasked with large-scale archiving of everyday websites.

Acknowledgements

This research is supported by the Library of Congress under the National Digital Information Infrastructure Preservation Program (NDIIPP).

Further information on mod_oai, including beta software and demos, can be found at http://www.modoai.org

Appendix

Notes

1. M.L. Nelson, J. Bollen, G. Manepalli, and R. Haq. Archive Ingest and Handling Test: The Old Dominion University Approach. *D-Lib Magazine* (11)12, December 2005. <doi:10.1045/december2005-nelson>.

2. F. McCown, J.A. Smith, M.L. Nelson, and J. Bollen. Lazy Preservation: Reconstructing Websites by Crawling the Crawlers. *8th ACM International Workshop on Web Information and Data Management (WIDM 2006)*. November 10, 2006. Arlington, Virginia, USA. <doi:10.1145/1183550.1183564>.

3. F. McCown, and M.L. Nelson. Evaluation of Crawling Policies for a Web-Repository Crawler. *17th ACM Conference on Hypertext and Hypermedia (HYPERTEXT 2006)*. August 23-25, 2006. Odense, Denmark. p. 157-168. <doi:10.1145/1149941.1149972>.

4. Internet Assigned Numbers Authority (IANA). MIME Media Types. <http://www.iana.org/assignments/media-types/>.

5. S. L. Abrams and D. Seaman. Towards a global digital format registry. *World Library and Information Congress: 69th IFLA General Conference and Council*, pages 1–9, August 2003.

6. J. Darlington. PRONOM: A Practical online compendium of file formats. *RLG Diginews*, 7(5), 2003.

7. JHOVE: JSTORE Harvard Object Validation Environment. <http://hul.harvard.edu/jhove/>.

8. M. L. Nelson, J. A. Smith, I. Garcia del Campo, H. Van de Sompel, and X. Liu. Efficient, automatic web resource harvesting. In *Proceedings of ACM WIDM 2006*, pages 43–50, 2006. <http://doi.acm.org/10.1145/1183550.1183560>.

9. H. Van de Sompel, M. L. Nelson, C. Lagoze, and S. Warner. Resource harvesting within the OAI-PMH framework. *D-Lib Magazine*, 10(12), December 2004. <doi:10.1045/december2004-vandesompel>.

10. LANL DIDL Schema Document. <http://purl.lanl.gov/STB-RL/schemas/2004-11/DIDL.xsd>.

Top | Contents
Search | Author Index | Title Index | Back Issues

Previous Article | Next Article
Home | E-mail the Editor

doi:10.1045/january2008-smith