Updating DAITSS - Transitioning to a web service architecture

Randall Fischer, Carol Chou, Franco Lazzarino

Florida Center for Library Automation
5830 NW 39th Avenue
Gainesville, FL 32605, USA
rf@ufl.edu, cchou@ufl.edu, flaz@ufl.edu

Abstract

The Florida Digital Archive (FDA) is a long-term preservation repository for the use of the libraries of the public universities of Florida. The FDA uses locally-developed software called DAITSS, which was designed to perform the major functions of Ingest, Archival Storage, Data Management and Dissemination in the OAIS reference model. A DAITSS 2 project is in process to re-write the application based on a distributed, Web services model. This paper describes the major changes in store for DAITSS 2.0, the rationale behind them, and the issues involved in their design and implementation. These changes include: moving from a monolithic to distributed processing environment; implementation of modular RESTful services; incorporation of existing tools, services, and registries; and revising the internal data model to be more conformant with the PREMIS data.
.

Introduction

The Florida Digital Archive (FDA) is a long-term preservation repository for the use of the libraries of the public universities of Florida. It has been in operation since late 2005, and as of July 1, 2008 has archived 52,000 information packages comprising 3.6 million files (10.4TB). Nine universities have agreements with the FDA to archive their submissions, which are being ingested at an average rate of 30-60 GB per day.

The FDA uses locally-developed software called DAITSS, which was designed to perform the major functions of Ingest, Archival Storage, Data Management and Dissemination in the OAIS reference model. DAITSS implements format-specific preservation strategies including normalization, migration and localization. ([Caplan 2007])

DAITSS was a pioneering digital preservation system. When it was designed and developed, there were few models of true preservation repositories and few external tools available for performing specific functions such as format validation and metadata extraction. It is somewhat remarkable that in three years of FDA operations, no major functional flaws have been discovered and few enhancements to functionality are pressing. The architecture of the application, however, requires major redesign. DAITSS was coded as a monolithic, self-contained system. A DAITSS 2 project is in process to re-write the entire system based on a distributed, Web services model.

The fundamental principles governing the original design of DAITSS have not changed. These include:

- strict conformance to the OAIS functional model;
- a requirement that the archived data store be self-defining, so that if the DAITSS system were lost, all known information about archived objects could be recovered from the data store itself;
- data once written to archival storage cannot be altered; modified objects are in effect new objects;
- original versions of archived files must be retained unaltered.

In conformance with these principles, files are modified only during the Ingest process as the SIP is transformed into the AIP. DAITSS relies upon format normalization and migration as preservation strategies, and these are implemented as part of Ingest. All files in the SIP as originally submitted are retained unaltered in perpetuity, but other versions may be derived and added to the AIP.

The basic unit of storage and processing is an Information Package. Each Information Package consists of an XML descriptor and all of the content files required to assemble one (and possibly more) representations of an information object. The Information Package is the only unit of input and output; that is, even if only a single file in an AIP is needed, the entire IP must be disseminated.

Because many years may pass between the time a file is ingested and when it requires some preservation treatment, dissemination requests are filled by a three-step process. In the first step, the AIP is exported from the repository and placed in the Ingest queue as a SIP. In the second step, the AIP-cum-SIP is re-ingested, and undergoes file identification, validation, and transformation processing according to the current version of the software. In the final step, the resulting AIP is reformatted into a DIP and delivered to the requestor.

This model will be retained in DAITSS 2. It has worked well in practice and in fact has beneficial side-effects. For example, the ingest model makes updates extremely simple, and the dissemination model allows the FDA to implement migration on request or mass migration depending on the circumstances.

Another governing principle was to use standard formats and metadata schemes whenever possible. However, at the time DAITSS was initially developed, there were few

applicable standards to chose from. METS is used as the format for SIP, AIP and DIP descriptors, and within the METS document standard schema are used for format-specific technical metadata for the few formats for which such schema exist. These include the Audio Engineering Society's draft AES schema for audio, the Metadata for Images in XML schema (MIX) for raster images, and the TextMD schema maintained by the Library of Congress for text. The Preservation Metadata: Implementation Strategies (PREMIS) Working Group was meeting as a committee while DAITSS 1.0 was being coded, but the PREMIS Data Dictionary had not yet been issued, so DAITSS 1 is only partially PREMIS compliant.

Design goals for DAITSS 2

Papers While time has shown the principles, approach and basic functionality of DAITSS to be sound, the current generation of software has a number of problems:

- The application was in some respects over-built, anticipating problems and functional requirements which never materialized. Unnecessary logical complexity makes the software difficult to maintain and configure.
- DAITSS is written as monolithic Java application, hindering its ability to scale. Simple functions such as virus checking take a significant portion of processing time, but cannot easily be offloaded to an independent server.
- There is a high degree of coupling between components, making it hard to extend and enhance the application. Adding support for a new format, for example, requires changes to dozens of classes, database schema, and XML schema.

The second generation of DAITSS will address these flaws. It will also improve PREMIS compliance throughout, by bringing the internal data model into closer conformance with the PREMIS three-part Object model (file, representation, bit-stream), and by making extensive use of PREMIS Object and Event descriptions.

Eliminate unnecessary complexity

Two features, initially thought to be desirable, have proved problematic. The first is the concept of preservation levels. DAITSS depositors (called "affiliates") are allowed to associate each file format with any of three preservation levels to be applied to files contained in their SIPs: BIT, FULL or NONE. NONE specifies that files of a given format will not be archived at all. BIT specifies that files of a given format will be archived but not subject to format transformation. FULL indicates that files will be normalized and/or migrated as appropriate.

Although it seemed like good customer service to give FDA affiliates these options, in practice it has been confusing to affiliates and problematic for the archive. The option NONE was intended to allow an affiliate to assemble a single package for multiple purposes; for example, for archiving and for loading into a digital asset management system. An unexpected problem is that files in formats that cannot be correctly identified because of DAITSS limitations might be assigned preservation level NONE and dropped from the AIP. In DAITSS 2 we will assume that if a file is in a SIP it is intended for archiving, and affiliates will be responsible for assembling appropriate SIPs.

The distinction between BIT and FULL has also proved difficult to sustain, and there seems to be little added functionality in maintaining it. Since DAITSS always retains files from the SIP as originally submitted, if an affiliate wants to ignore a migrated version they can always do so. DAITSS 2 will eliminate the entire concept of preservation level and attempt full preservation treatment for all files.

The second issue involves "global" files and a kind of transformation called "localization." Global files are sets of files included in many packages. Commonly these are files needed to validate XML descriptors, such as DTDs and schema. Rather than storing them redundantly in thousands of AIPs, the global files are stored once in separate packages and referenced, as necessary, by links from other AIPs. Although this seemed like a good idea at the time, the maintenance of global files has added considerable complexity to the code. Analysis shows that the space savings are only about 1.6% of the archive store. DAITSS 2 will eliminate the concept of global files, and will include all required files in each AIP.

Localization is a DAITSS 1 function where a reference within an archived file to an external file (for example, a schema) is rewritten to refer to a locally archived version. This requires DAITSS to keep both the original and localized versions of the file. DAITSS 2 will skip localization at the file level, and instead modify validators to dynamically resolve references to the external file from a local cache.

Break up the beast

Two features, initially thought to be desirable, have proved problematic. DAITSS 2 will be comprised of simple, independent components that each perform one simple function. It is a requirement that each component can be tested and developed independently of any other component. This will make it simpler to modify or extend existing functions and to integrate new functions. For example, it would be possible to add a new risk assessment service to the current chain of processing without modifying any other service. Dividing DAITSS into separate components will also allow us to parallelize time-consuming tasks such as virus checking and checksum calculation.

Further, we believe that exposing each functional component as a stand alone service will allow researchers to extend the system into novel workflows.

In short, rather then providing major changes in functionality, we wish to simplify and support existing functions but with a wider scope.

Implementation

This The second generation of the DAITSS software will take a Web services approach. There are two main competing architectural styles for Web services today: a

Remote Procedure Call (RPC) style, and the Representational State Transfer (REST) style detailed by Roy Fielding ([Fielding 2000]). SOAP is an example of the RPC style, while REST is the basis for the classic view of HTTP used on the Web. The Web service APIs provided by Google, Amazon and Yahoo typically offer both styles of access to their services. However, the REST APIs are significantly more popular: Amazon has reported that REST style requests comprise 80% of their web service traffic ([Anderson 2006]).

Experience has shown that SOAP applications exhibit a high degree of coupling between services. This state of affairs results from very application-specific SOAP actions that must communicate data structures from one service, to the client, to other web servers. This has led to ever expanding sets of standards and complex frameworks to support what was, initially, a Simple Object Access Protocol.

In contrast, the REST approach is centered around resources. In HTTP, the most successful example of a RESTful architecture, there are only six operations and each of them are atomic. PUT creates a named resource, GET retrieves it, POST modifies it, and DELETE removes it. HEAD retrieves simple metadata for the resource.

The state of a RESTful application is maintained as a set of external resources. A client program effects the progress of the application by performing incremental changes using defined operations on externally stored resources. Such limitations allow, counter-intuitively, far greater flexibility on the part of client-based applications, illustrating the key design strategy in software engineering of using the least powerful language to accomplish a task ([W3C 2006]).

In its purest form, the state of an application is driven by resources that contain links to other services, the so-called Hypertext As The Engine Of Application State (HATEOAS). In DAITSS this is illustrated by the Action Plan service described below. Briefly, this service is given data identifying and characterizing a format, and returns the location of an appropriate transformation service that can effect format migration and normalization. The archival policy of the FDA is thus driven by a very simple service which publishes links to other services.

The DAITSS Storage Service

Rather than implementing a wholly new version of DAITSS at some time in the future, our plan is to gradually morph DAITSS 1 into DAITSS 2 by pulling out pieces of the code and replacing them with newly written Web services that perform the same function. Our first Web service has already been incorporated into the production version of DAITSS used by the Florida Digital Archive: a simple storage service loosely based on the Amazon S3 Web service. The implementation of this storage service resulted in a significant performance increase for the FDA.

Each AIP is assigned an intellectual entity identifier (IEID) and its constituent files and descriptors packaged together as a GNU tar file. The MD5 checksum of the tar file is computed as well as the checksums of the individual files it contains. The assembled package is then submitted to two geographically isolated servers using SAN-attached file systems as long term storage. The package-level checksum is used to ensure that the initial transmission completed successfully, and is also retained for subsequent fixity checking on the stored AIP.

A typical HTTP conversation for the initial store is shown for an AIP that has been assigned the IEID E20080715_AAACAZ; the client stores the AIP using the HTTP PUT function.

Request:

```
PUT /silo003/E20080715_AAACAZ HTTP/1.1
User-Agent: DAITSS v1.5
Host: storage.fcla.edu:3000
Content-MD5: 2thsYe6iN5MvIBAJ5UMWCQ==
Content-Type: application/octet-stream
Content-Length: 32044941
[ ... inline data ... ]
```

Response:

```
HTTP/1.1 201 Created
Connection: close
Date: Mon, 11 Aug 2008 16:08:42 GMT
Content-Length: 0
```

Possible success and error conditions with the associated response status codes include:

Success
- 201 The resource was created

Client Error
- 400 Missing resource name in PUT request
- 403 Duplicate package name
- 405 Storage location is full
- 409 Checksum error
- 411 Invalid request headers

Server Error
- 500 Specific server error message included

The DAITSS 2 Service Architecture

We next describe the entirely services-based architecture planned for the second generation of DAITSS. The current monolithic application will be decomposed into a set of relatively simple Web services, some of which are described below. The composition of each service into a complete Ingest process will require preservation events to be recorded as they occur, and later assembled into a complete record of the archiving process. Therefore each function will create an event description expressed in PREMIS XML, which will ultimately be assembled into the AIP descriptor. Main components of the Ingest Process are shown in Figure 1.

The Description Service

File format identification and validation is a central function of DAITSS. In DAITSS 2, each data file is sent to the Description Service for identification, validation and characterization. The service uses DROID for a preliminary identification of the file format, which is used to select the appropriate validator. If DROID returns the information that the file is identified as multiple formats associated with different validators, the

most appropriate validator is selected by the service. For the formats most commonly presented to the Florida Digital Archive, a modified version of JHOVE is used as the validator, and the preliminary format is used to select the initial JHOVE validation module. JHOVE may include in its output the information that the file is actually described by multiple formats; if so the most appropriate format is selected by the service. The result of JHOVE validation and characterization is then parsed and mapped into PREMIS, and the JHOVE format information is converted back to a PRONOM format identifier.

A PREMIS XML document for that file is returned by the Description Service to guide further Ingest processing. The returned PREMIS document has three sections: an object section that includes a single PRONOM format identifier and technical metadata according to an extension schema appropriate to that format; an event section that describes the outcome of the validation, including any anomalies found; and an agent section that identifies the service used. An abbreviated version of an example document is shown below**.**

```
<object xsi:type="file">
    <objectIdentifier>
        <objectIdentifierType>
         DAITSS2</objectIdentifierType>
        <objectIdentifierValue>
        E20080715_AAACAZ/florida.tif
         </objectIdentifierValue>
   </objectIdentifier>
   <objectCharacteristics>
       <compositionLevel>0</compositionLevel>
       <size>3001452</size>
       <format>
           <formatDesignation>
               <formatName>TIFF</formatName>
               <formatVersion>4.0</formatVersion>
           </formatDesignation>
           <formatRegistry>
               <formatRegistryName>
               PRONOM</formatRegistryName>
               <formatRegistryKey>fmt/8
               </formatRegistryKey>
            </formatRegistry>
       </format>
       <objectCharacteristicsExtension>
           <mix:mix xmlns:xsi="http://www.w3.org/
           2001/XMLSchema-instance" >
            ...
            </mix:mix>
       </objectCharacteristicsExtension>
   </objectCharacteristics>
</object>
<event>
    <eventIdentifier>
        <eventIdentifierType>DAITSS2
        </eventIdentifierType>
        <eventIdentifierValue>1</eventIdentifierValue>
    </eventIdentifier>
    <eventType>Format Description</eventType>
    <eventDateTime>2008-07-17T12:32:50
   </eventDateTime>
    <eventOutcomeInformation>
       <eventOutcome>Well-Formed and valid
       </eventOutcome>
       <eventOutcomeDetail>
          <eventOutcomeDetailExtension/>
       </eventOutcomeDetail>
    </eventOutcomeInformation>
</event>
<agent>
    <agentIdentifier>
        <agentIdentifierType>uri</agentIdentifierType>
        <agentIdentifierValue>
         http://daitss.fcla.edu/describe
         </agentIdentifierValue>
    </agentIdentifier>
    <agentName>Format Description Service
    </agentName>
    <agentType>Web Service</agentType>
 </agent>
```

The Action Plan Service

The Action Plan Service is sent the PREMIS document produced by the Description Service and returns a simple XML document containing one or more links to services to be used to transform (migrate or normalize) the associated file. If DAITSS is not capable of transforming a given format, or if a particular file contains too many anomalies to be reliably transformed, the document will contain, instead of links, a stanza noting the limitation.

The Action Plan service succinctly specifies the migration and normalization policy of an installation of DAITSS. The service illustrates a key feature of the RESTful approach, which is to let links drive the process of ingest. An example of a document returned by the action plan service follows.

```
   <instructions>
      <normalization>
          <transformation>
           http://daitss.fcla.edu/transform/wave_norm
          </transformation>
      </normalization>
      <migration>
          <limitation>codec not supported
          </limitation>
      </migration>
   </instructions>
```

The Action Plan Service is driven by a set of XML documents that serve a dual function: they are used internally to specify the transformation services to be applied, and they are published externally to document our archival policy:

```
<action-plan format="WAVE" date="2008-07-02"
author="Andrea Goethals, FCLA">
    <processing>
       <normalization>Each audio stream in the WAVE
       file will be normalized into an uncompressed
       PCM(LPCM) audio stream with sample size of 16
       bits/sample.
       <transformation>
       http://daitss.fcla.edu/transform/wave_norm
        </transformation>
```

```
        <limitations>
          <supported-codec>PCM</supported-codec>
          <supported-codec>MP3</supported-codec>
        </limitations>
      </normalization>
    </processing>
    <strategy>
      <original>Migrate to newer WAVE versions or to
      an open, standardized and well supported audio file
      format that is to be a good successor to WAVE.
      </original>
      <normalized> Migrate to an open, standardized and
      well supported audio stream format that is losslessly
      compressed.
      </normalized>
    </strategy>
    <timetable>
      <item action="review" date="2009-07-02"/>
      <item action="revise" date="2009-07-02"/>
      <item action="short-term" date="2009-07-02">
       Write or locate a converter which converts WAVE
       files with data in one of audio encoding formats
       listed in 3.1 to WAVE files in LPCM format.
      </item>
    </timetable>
</action-plan>
```

The Transformation Service

The current version of DAITSS provides both normalization and migration of data files. The second generation of DAITSS will support these transformations via a collection of Transformation Services. A file is submitted to the appropriate Transformation Service as specified by the Action Plan service; the transformed file is returned via HTTP. It is possible for multiple files to be produced as output from a single submission. For instance, DAITSS may normalize a PDF file into a collection of TIFFs, one per page. For cases like these, the Transformation Service returns a composite document using the MIME multipart/mixed standard.

In some cases the Transformation Service is a locally developed program. In many cases a Transformation Service is simply an HTTP wrapper around an external, probably open source, program such as Ghostscript, ffmpeg, mencoder, or libquicktime. For these cases, a simple specification of the action of the program suffices to build the service.

```
<transformations>
  <transformation ID='WAVE_NORM'>
    <instruction> ffmpeg -i #INPUT_FILE# -sameq –a
    codec pcm_s16le #OUTPUT_FILE#
    </instruction>
    <extension>.wav</extension>
    <software>FFmpeg version SVN-r9102
    </software>
    <configuration> --prefix=/opt/local--
    prefix=/opt/local --disable-vhook--
    mandir=/opt/local/share/man  --enable-shared --
    enable-pthreads --disable-mmx
    </configuration>
    <dependency>libavutil version: 49.4.0
      libavcodec version: 51.40.4
      libavformat version: 51.12.1
    </dependency>
  </transformation>
  <transformation ID='AVI_NORM'>
    <instruction>mencoder #INPUT_FILE# -oac pcm –
     ovc lavc -lavcopts vcodec=mjpeg --o
     #OUTPUT_FILE#
     </instruction>
    <extension>.avi</extension>
  </transformation>
  <transformation ID='MOV_NORM'>
    <instruction>lqt_transcode -ac rawaudio -vc mjpa
    #INPUT_FILE# #OUTPUT_FILE#
    </instruction>
    <extension>.mov</extension>
  </transformation>
  <transformation ID='PDF_NORM'>
    <instruction>gs -sDEVICE=tiff12nc
    -sOutputFile=#OUTPUT_FILE# -r150 –dBATCH
    -dNOPAUSE #INPUT_FILE#
    </instruction>
    <extension>page%d.tif</extension>
  </transformation>
  ....
</transformations>
```

The AIP Service and subsequent processing

At this point both the original file and any derived versions are submitted to an AIP Service, which acts as a holding area for this intellectual entity. The PREMIS object and event descriptions are also saved. When the last file in the SIP has been fully processed, a complete AIP descriptor is assembled combining information from the original SIP descriptor with the saved object and event information. Finally, the entire package is sent to the Storage Service, which, as noted above, distributes the AIP to multiple locations.

Conclusion

As noted above, we believe that dividing complex services into simple, well understood components will allow the creation of novel preservation workflows. One new function under consideration is a risk assessment service, which will accept information extracted from an AIP descriptor and return the preservation risk associated with the packages.

However, the architecture has other advantages. For one thing, it will make it possible for the Florida Digital Archive to share services with other preservation repositories. Several institutions and projects are developing Web services based systems or components, including (but not limited to) The National Archives (UK), the California Digital Library, PLANETS and PRESERV. The FDA (and other DAITSS users) will be technically capable of integrating externally-written services if rights and organizational issues allow.

In addition, while the first generation of DAITSS is actively maintained and distributed as open source software, we have made little effort to promote its use in the community, as our experience has been that DAITSS

is overly complex to configure and difficult to maintain. The Florida Center for Library Automation has neither the resources nor the mandate to exert significant effort supporting external sites. We expect that DAITSS 2 will be much easier to configure and operate, and that other institutions would find it attractive to implement the system or some of its component services. The architecture is particularly advantageous to local sites, which could customize the distribution version of DAITSS by supplying their own action plans and services as needed.

References

Anderson, T. 2006. *WS-* vs the REST*. The Register April 29, 2006. http://www.theregister.co.uk/2006/04/29/oreilly_amazon

Caplan, P. 2007. *The Florida Digital Archive and DAITSS: A Working Preservation Repository Based on Format Migration*. International Journal on Digital Libraries, 20 March 2007, doi: 10.1007/s00799-007-0009-6. Available at http://www.springerlink.com and http://www.fcla.edu/digitalArchive/pdfs/IJDL_article.pdf

Fielding, R. 2000. *Architectural Styles and the Design of Network-based Software Architectures.* University of California, Irvine http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

W3C, 23 February 2006. *The Rule of Least Power.* W3C Technical Architecture Group. http://www.w3.org/2001/tag/doc/leastPower.html