Bamboo Book Model - CMIS Binding and Fedora Repository Implementation

For an overview of the context and use cases for which a Bamboo Book Model was developed, see *Interoperability of Digital Collections*.

Bamboo Book Model

The Bamboo Book Model represents a codex book as a hierarchy of typed containers and objects with a predictable structure. Each node in the hierarchy has predictable metadata attributes.

The head node, the BambooBook object, contains three nodes:

1. a pages node – a bag of page nodes, for which order is inferred from a page number attribute on each page node;
2. a contents node – a bag of contents objects; and,
3. a sources node – a bag of source objects.

Of these three, the pages branch of the hierarchy is the most fully developed. Each page node is a container that comprises a set of objects: plain text; page image; HTML text; TEI; and Morphadorned text (http://morphadorner.northwestern.edu). No manifestation of a page is required. Any manifestation present in the page node declares its type and the MIME type of its content and includes its content

We envisioned a Contents container that would include various views of the book. A table of contents would provide the traditional part, chapter, and section map onto the pages of the book. Other maps, e.g. list of illustrations, word index, etc. would also be possible. While these maps could be rendered for human readers, the function of the content maps is to make these traditional structures available to software. The content map need not be limited to traditional structures, of course. Each map would declare its type, and each map type would have a predictable structure. No content maps were modeled in Phase I.

The Book Model normalizes book content by transforming content from different content repositories with possibly disparate formats into a set of common formats. In some cases information is lost in the transformation. For example, information encoded in the typographical features of a page of text is lost in the plain text manifestation. In some cases information is added. A Morphadorned text, for example, is richly annotated.

We envisioned placing source documents, images, and so on in the Sources container so that it would be possible to inspect the state of a text prior to the normalizing transformations we applied, and so that when commonly used tools can readily be applied to the source document, as with a TEI encoded text, for example, the source document would be available. Typing and provenance metadata would be applied to the objects within the Sources container. Although source content is provided by the CI HUB, content type and provenance metadata were not modeled and consequently, those metadata for source content are not yet provided.

The Contents and Sources containers were not implemented in the CMIS binding of the Book Model. The Pages container was implemented in the CMIS binding, including the page manifestations noted above.

Implementation through Binding to CMIS

We implemented the Bamboo Book Model by binding it to the CMIS API. A software binding for a abstract model like the Bamboo Book Model is a mapping of the structural elements of the model onto the data structures and methods of a software. Often, these data structures and methods are expressed in an API. Content Management Interoperability Services (CMIS) is an API designed to represent common create, read, update, and delete operations on folder- and file or document-like objects. CMIS also provides for the definition of attributes of objects through the extension of its base object classes. Binding the Book Model to CMIS allows us to leverage software built for content management to operate on the structure and content of the codex book.

Binding details

Here is an example of extending the base CMIS class folder class to serve as the root container of the BambooObject. We extended the base class by adding four properties:

1. *dc.issued* captures when the object was obtained from a source repository by the Collections Interoperability Hub
2. *bamboo.source* and *bamboo.source-url* capture the repository from which the object was obtained and the identifier of the object in the repository
3. *bamboo.owner* captures who requested that the object be obtained through the CI Hub

The *bamboo:book* object type is an extension of *bamboo:folder*, which has properties relevant to any bamboo container: all attribute types declared in Dublin Core. This code snippet illustrates how CMIS types are declared in the Apache Chemistry OpenCMIS tool we used for our local object store. (See below.)

CMIS Binding of the BambooBook

```
<?xml version="1.0" encoding="UTF-8"?>
<cmisra:type xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/"
        xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="cmis:cmisTypeDocumentDefinitionType"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:dcterms="http://purl.org/dc/terms/"
        xmlns:bamboo="http://bamboo.at.northwestern.edu/types">
  <cmis:id>bamboo:book</cmis:id>
  <cmis:parentId>bamboo:folder</cmis:parentId>
  <cmis:displayName>Book</cmis:displayName>
  <cmis:queryName>BOOK</cmis:queryName>
  <cmis:description>Folder containing all content relating to a specific book or text</cmis:description>
  <cmis:baseId>cmis:folder</cmis:baseId>
  <cmis:creatable>true</cmis:creatable>
  <cmis:fileable>true</cmis:fileable>
  <cmis:queryable>true</cmis:queryable>
  <cmis:fulltextindexed>false</cmis:fulltextindexed>
  <cmis:includedInSupertypeQuery>true</cmis:includedInSupertypeQuery>
  <cmis:versionable>false</cmis:versionable>

  <cmis:propertyStringDefinition>
    <cmis:id>dc:issued</cmis:id>
    <cmis:localName>issued</cmis:localName>
    <cmis:displayName>issued</cmis:displayName>
    <cmis:queryName>issued</cmis:queryName>
    <cmis:description>Dublin Core Issued date</cmis:description>
    <cmis:propertyType>string</cmis:propertyType>
    <cmis:cardinality>single</cmis:cardinality>
    <cmis:updatability>readwrite</cmis:updatability>
    <cmis:inherited>false</cmis:inherited>
    <cmis:required>true</cmis:required>
    <cmis:queryable>true</cmis:queryable>
    <cmis:orderable>false</cmis:orderable>
  </cmis:propertyStringDefinition>

  <cmis:propertyStringDefinition>
    <cmis:id>bamboo:source</cmis:id>
    <cmis:localName>source</cmis:localName>
    <cmis:displayName>Source Repository</cmis:displayName>
    <cmis:queryName>source</cmis:queryName>
    <cmis:description>Bamboo Source Repository</cmis:description>
    <cmis:propertyType>string</cmis:propertyType>
    <cmis:cardinality>single</cmis:cardinality>
    <cmis:updatability>readwrite</cmis:updatability>
    <cmis:inherited>false</cmis:inherited>
    <cmis:required>true</cmis:required>
    <cmis:queryable>true</cmis:queryable>
    <cmis:orderable>false</cmis:orderable>
  </cmis:propertyStringDefinition>

  <cmis:propertyStringDefinition>
    <cmis:id>bamboo:source-url</cmis:id>
    <cmis:localName>source-url</cmis:localName>
    <cmis:displayName>Source URL</cmis:displayName>
    <cmis:queryName>source-url</cmis:queryName>
    <cmis:description>Bamboo Source URL</cmis:description>
    <cmis:propertyType>string</cmis:propertyType>
    <cmis:cardinality>single</cmis:cardinality>
    <cmis:updatability>readwrite</cmis:updatability>
    <cmis:inherited>false</cmis:inherited>
    <cmis:required>true</cmis:required>
    <cmis:queryable>true</cmis:queryable>
    <cmis:orderable>false</cmis:orderable>
  </cmis:propertyStringDefinition>
  <cmis:propertyStringDefinition>
        <cmis:id>bamboo:owner</cmis:id>
        <cmis:localName>owner</cmis:localName>
        <cmis:displayName>Owner</cmis:displayName>
        <cmis:queryName>owner</cmis:queryName>
        <cmis:description>Owner</cmis:description>
        <cmis:propertyType>string</cmis:propertyType>
        <cmis:cardinality>single</cmis:cardinality>
        <cmis:updatability>readonly</cmis:updatability>
        <cmis:inherited>true</cmis:inherited>
        <cmis:required>true</cmis:required>
        <cmis:queryable>true</cmis:queryable>
        <cmis:orderable>true</cmis:orderable>
  </cmis:propertyStringDefinition>

</cmisra:type>
```

A number of object types are declared:

Object type	Extends	Properties	Contains
bamboo:book	bamboo:folder	• dc.issued • bamboo:source • bamboo:source-url • bamboo:owner	0..n bamboo:pages
bamboo:page	bamboo:folder	• dc:issued • bamboo:seq • bamboo:page_name • bamboo:label	0..n bamboo:page-documents
bamboo:folder	cmis:folder	• dc:identifier • dc:date • dc:contributor • dc:coverage • dc:creator • dc:description • dc:format • dc:language • dc:publisher • dc:rights • dc:source • dc:subject • dc:title • dc:relation • dc:type	
bamboo:page-document	bamboo:document	• dc:issued • bamboo:seq • bamboo:page_name • bamboo:label	
bamboo:page-image	bamboo:page-document		
bamboo:page-image-jp2	bamboo:page-document		
bamboo: page-plaintext	bamboo:page-document		
bamboo:page-xhtml	bamboo:page-document		
bamboo:page-morphadorned	bamboo:page-document		
bamboo:page-tei	bamboo:page-document		
bamboo:page-thumb150	bamboo:page-document		

bamboo:document	cmis:document	• dc:identifier • dc:date • dc:contributor • dc:coverage • dc:creator • dc:description • dc:format • dc:language • dc:publisher • dc:rights • dc:source • dc:subject • dc:title • dc:relation • dc:type	

This table reports on the actual implementation as it stood in March 2012. The code contains provisional classes for source documents and a logical contents view, but from the perspective of March 2013, these seem to be thoughts not fully formed. Similarly, we might have expected to see a *bamboo:pages* object type contained by *bamboo:book* and that would serve as a container for *bamboo:page* objects. Implementation lagged behind the evolving ideas around how to model BambooBooks.

Alternate bindings

Binding the Book Model to CMIS makes sense if

1. the source book object is obtained in pieces through distinct calls to some API, and
2. the user wishes to browse, inspect, or otherwise operate on parts of a book without necessarily committing to retrieving the whole book.

As it turns out, when we first began working with the HathiTrust API, we laboriously collected each book one page at a time. This generated a large number of hits on the HathiTrust API, which HathiTrust interpreted as harvesting. They recommended that we not retrieve books piecemeal, but as a whole packaged in a single archive. So, the first condition was not met for HathiTrust. Curiously, the first condition continued to hold for TEI encoded texts for which we obtained page images from a commercial source. The TEI for a book was retrieved as a single file, but the page images were not. There were no page images for Perseus texts; pages were created from a single text object based a convention for treated certain divisions in the text as "pages", which might give one the impression that we did not know much about classical texts. Of course, we were squeezing the Perseus classical texts into the pages construct because our client object viewer expected texts chunked as pages. That is, we assumed that second condition above always held.

But, suppose the second condition doesn't hold typically. Then, one might imagine a less chatty binding in which a book is modeled in the manifest and metadata of a content package.

The need for a client object store

Having retrieved objects through a Bamboo Collections Interoperability Hub, we assumed that scholars would need to store them locally in the client Research Environment (RE). The point of Collections Interoperability is interoperability from the perspective of the tools that operate on book objects. We assumed that tools would consume BambooBook objects in some binding or another. By representing BambooBook objects in a CMIS binding in the client (local) RE, we hoped to gain flexibility. A scholar could operate on local or remote objects by pointing her tools to the local CMIS repository or the remote CI Hub interface; there would no difference the interfaces from a tool's perspective. For tools that do not consume a CMIS binding, we would extract components of the book and deliver them to the tool by some other means, e.g. on the file system or through a web service.

Fedora Repository as a persistence layer for a CMIS Repository

We chose Fedora Repository as the persistence layer for the client RE's CMIS Repository because of its flexible object and metadata capabilities and because several of the universities charged with developing Bamboo Research Environments used Fedora in their digital collections infrastructure. A component's BambooBook type and its associated metadata are stored as metadata streams on Fedora objects. Its content is stored as a content stream. Finally, containment relationships are modeled as object relations between Fedora objects.

Implementing the CMIS Repository

We used the Apache Chemistry (http://chemistry.apache.org) project's OpenCMIS Java server implementation (release 0.5.0) to implement the CMIS Repository we deployed in our prototype Bamboo Research Environment. We subclassed the Chemistry AbstractCmisService class, replacing the implementation of CMIS methods a file system data store with Fedora service calls.

CMIS permissions

The 0.5.0 OpenCMIS server did not implement permissions beyond a rudimentary repository level permissions to read or to read and write objects. Permissions were associated with the account that had authenticated to the repository Web services interface using Basic AUTH. Clearly, finer grained permissions were needed.

The driving use case in our thinking about permissions was scholarly collaboration around operations on objects. A group of scholars wishes to work collaboratively on curating a text object. Bamboo had adopted groups as a means of expressing who can collaborate using the functionality Bamboo affords (see *Group Service API* and *Authorization and Policy*). The owner of the object, that is, the person who originally created the instance of it in the repository, has permission to read, update, and delete the object as well the permission to grant permissions to others, including individuals and groups.

We constructed BambooAcl as an implementation of the java.security.acl.Acl interface. A BambooAcl enumerates permissions assigned to Principals. A BambooAcl can be assigned to any object within a BambooBook object and the BambooBook object itself. Given the hierarchical structure of the BambooBook object, if when an operation on an object within the book is requested no BambooAcl is present for that object, we assume that the object inherits permissions from its containing object. To evaluate whether the requested operation is permitted for the Principal making the request, we recurse the hierarchy upward until a BambooAcl is found. The recursion continues all the way up to the root of the repository. If no BambooAcl is found at the root, we accept the repository default permissions.

The BambooAcl assigned to an object is serialized as JSON and stored as a data stream on the object in the Fedora repository.

When an operation on a object is requested through the CMIS interface, we evaluate permissions for all of the declared Principals associated with the request. Zero, one, or many Principals can be associated with a request. In practice, the Principals include the user's identifier and the identifiers of any groups to which the user belongs, which membership is asserted for the purposes of the request.

Obtaining user and principal names

As noted above, the OpenCMIS provided only a simple approach to AuthNZ. We retained this approach for basic authentication, but treated the calling CMIS client – a Bamboo Research Environment (or "Work Space application" in terms current during phase one of the Bamboo Technology Project) – as the authenticating agent. For the purposes of evaluating permissions based who ultimately had authenticated to the Bamboo RE as a user, we provided a means capturing the user's ID and a list principal names passed in HTTP headers with the RESTful request. These are extracted and placed in the security context by BambooContextHandler, a customer security context handler class.

Our approach to passing in principal names was an expedient hack. It pushed the task of resolving principal name back to the client application. This has the consequence that although we expose a CMIS service that can be used to access BambooBook modeled content stored in a Fedora repository, only specialized clients that are trusted to assert user identity and associated principal names can use the service so long as permissions are in play. See *Authentication - Current Limitations and Future Direction*.

Deployment of the Bamboo CMIS-Fedora Server

To deploy the OpenCMIS server and configure the Fedora Repository it uses as a persistence layer, see the wiki page *Deploying the Bamboo CMIS-Fedora Server*.

Implementation code

Source code for the CMIS binding can be found at http://svn.code.sf.net/p/projectbamboo/code/work-space-repository/ (Javadoc at http://javadoc.projectbamboo.org/fedora-cmis/)