ASSET MANAGEMENT SYSTEM FOR COOLSTATELA

A Thesis
Presented to
The Faculty of the Department of
Computer Science
California State University, Los Angeles

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
Farrukh Shakil
March 2008
ACKNOWLEDGEMENTS

First of all I would like to thank my project advisor Dr. Russ Abbott who guided me throughout the project and he was always there to discuss and help with any problems we encountered. I would also like to thank Dr. Raj Pamula for helping me choose the right project for myself. This project wouldn't be possible without the skills learned from Dr. Chengyu Sun. I am thankful to him for that and also to all other faculty members who equipped me with skills needed for this project. I would like to thank other group members in this project for their support and specially Prof. Jon Beaupre for his understanding, patience and believing in us. My parents who always supported me and encouraged me. Finally, thanks to the greatest of all God who always listens and helps me.
ABSTRACT

Asset Management System for CoolStateLA

By

Farrukh Shakil

Any system that deals with many different media types like images, audio, video and text and also needs to represent complex media objects composed of different kind of medias, like a News consists of images, text and audio requires a very efficient and powerful Asset Manager. This thesis proposes an Asset Management System that supports tasks and decisions surrounding ingesting, defining, cataloging, organizing, storage and retrieval of CoolStateLA's digital contents. In addition to that, this thesis suggests a mechanism to deliver the contents in a way that can help effectively communicate the intellectual work reflected by those contents to others.
TABLE OF CONTENTS

ACKNOWLEDGMENTS .......................................................................................................................... iii

ABSTRACT ................................................................................................................................................ iv

TABLE OF CONTENTS ........................................................................................................................... v

LIST OF TABLES ........................................................................................................................................ x

LIST OF FIGURES .................................................................................................................................... xi

Chapter 1. Introduction .......................................................................................................................... 1

1.1 Project CoolStateLA ........................................................................................................................ 1

1.2 Asset Management System ........................................................................................................... 1

1.3 Scope of the Project .......................................................................................................................... 2

1.4 Intended Audience .......................................................................................................................... 3

Chapter 2. Architecture .......................................................................................................................... 4

2.1 Architecture Of CoolStateLA ......................................................................................................... 4

2.1.1 Content Management System .................................................................................................. 5

2.1.2 Workflow Management System ................................................................................................ 6

2.1.3 Asset Management System ..................................................................................................... 6

2.1.4 STORI Manager ....................................................................................................................... 6

2.1.5 RSS Feed Filters ....................................................................................................................... 7

2.1.6 Wiki and Bulletin Board .......................................................................................................... 7

2.2 Architecture of Asset Management System ................................................................................... 7

2.2.1 Fedora Repository .................................................................................................................... 9
2.2.2 Fedora Client ........................................................................... 9
2.2.3 Fedora Client API ..................................................................... 9
2.2.4 CoolStateLA's Fedora Client/Controller ............................... 10
2.2.5 Integration Interface ............................................................... 10

Chapter 3. Digital Repositories for A·S......................................................... 11
3.1 Digital Repositories ........................................................................... 11
3.2 Selection of Right Repository for CoolStateLA's AMS ..................... 12
3.2.1 Factors to Consider ................................................................. 12
3.2.1.1 Cost /Budget ................................................................ 12
3.2.1.2 Functionality ............................................................... 13
3.2.1.3 Reliability .................................................................. 13
3.2.1.4 Security .................................................................. 13
3.2.1.5 Documentation and Support ......................................... 13
3.2.1.6 Activities ................................................................ 14
3.2.2 Shortlisting .............................................................................. 14

Chapter 4. Fedora Repository ........................................................................ 17
4.1 Configurations and Environment ...................................................... 17
4.2 Fedora Objects ............................................................................... 19
4.3 Datastreams and Disseminators ......................................................... 20
4.4 Storage and Retrieval Mechanism ..................................................... 23
4.4.1 Ingesting Objects .................................................................. 23
4.4.1.1 Meta-data ................................................................ 23
LIST OF TABLES

Table 3.1-DSPACE vs FEDORA ................................................................................................. 15
Table 4.1-Datastream Attributes ............................................................................................ 27
Table 4.2- APIA-Methods ......................................................................................................... 32
Table 4.3-API-A Methods (REST) .......................................................................................... 34
Table 4.4- API-M Methods (REST) ........................................................................................ 35
Table 6.1-Fedora Content Model Objects ............................................................................... 65
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>CoolStateLA's Architecture</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>AMS Architecture</td>
<td>8</td>
</tr>
<tr>
<td>4.1</td>
<td>APIM Testing-Roundtrip Time Per Operation</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>APIM Operation Per Second</td>
<td>19</td>
</tr>
<tr>
<td>4.3</td>
<td>Fedora Object</td>
<td>22</td>
</tr>
<tr>
<td>5.1</td>
<td>Item and Collection Objects</td>
<td>42</td>
</tr>
<tr>
<td>5.2</td>
<td>Batch Ingest Model</td>
<td>45</td>
</tr>
<tr>
<td>6.1</td>
<td>Compound Content Model</td>
<td>61</td>
</tr>
<tr>
<td>6.2</td>
<td>Atomistic Content Model</td>
<td>64</td>
</tr>
<tr>
<td>6.3</td>
<td>Fedora Content Model</td>
<td>66</td>
</tr>
<tr>
<td>6.4</td>
<td>Image Content Model</td>
<td>67</td>
</tr>
<tr>
<td>6.5</td>
<td>Image Content Model (Detailed)</td>
<td>71</td>
</tr>
<tr>
<td>6.6</td>
<td>STORI Content Model</td>
<td>72</td>
</tr>
<tr>
<td>6.7</td>
<td>News Feed Content Model</td>
<td>73</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

1.1 Project CoolStateLA

In this current age of technology, more and more people are abandoning the traditional way of getting information and adopting much quicker and easier ways to get the information. This advanced technology allows simple newspapers to be more than just a newspaper. They can be a complete media company and they can use those different medias to get the word out.

Project CoolStateLA is the First Open Source 'Converged Newsroom' aimed to provide a Newspaper, Web site, Blog facility, RSS news feed filter, Text feeds to create Email alerts or SMS messages, Audio and Video broadcasting, Media sharing, News web casts and Internet radio. A complete content management solution is required to facilitate all this functionality. A system that provides an ability to create, manage, review, search, annotate, store and retrieve media. This system is being built by a team of Graduate students in California State University under the supervision of Dr. Russ Abbott (Computer Science Department, Calstate) and Prof. Jon Beaupre (Communication Studies, Calstate).

1.2 Asset Management System

The purpose of this thesis is to provide an 'Asset Management System' for
CoolStateLA. It is aimed to define a mechanism to ingest, annotate, organize, store and retrieve digital components or assets for CoolStateLA. Since CoolStateLA basically involves dealing with contents, this Asset Management System will serve as backbone or core of entire system. The proposed asset management system is built on top of Fedora which is a free open source repository software equipped with all the advanced functionality to full fill the unique requirement of this Asset Management System.

1.3 Scope of the Project

Although Project CoolStateLA is basically being developed for the California State University. It will be an Open Source project. There are other similar applications in market being used by some big newspapers like LA times but those applications usually cost over $1 million.

Universities and non-profit organizations' Magazines and Newspapers can't afford those commercial solutions. We are hoping and positive that this project will be very appealing to them. Although the Asset Management System is being built keeping in mind the unique requirements of CoolStateLA's architecture and requirements, It is extensible and flexible enough to be used by some other institution or organization. This asset management system is the first application that is built using the latest fedora version and it is using many advanced features of fedora that are not yet used by any other fedora based systems, as of this writing.
1.4 Intended Audience

This project will be of a great interest for small newspapers, media companies, libraries, educational institutes, fedora community and developers interested in java based frameworks or web services.
CHAPTER 2
ARCHITECTURE

2.1 Architecture Of CoolStateLA

CoolStateLA consists of following components

1. Content Management System
2. Workflow Management System
3. Asset Management System
4. STORI Manager
5. RSS Feed Filter
6. Wiki/Bulletin Board
2.1.1 Content Management System

The Content Management System will provide Newspaper facility and help publish the content to the website. A free OpenSource application called Joomla has been proposed for this purpose. Joomla provides a complete framework for publishing the contents on the website and the contents can be exported to the most common documents file format like pdf or doc, which is needed for the newspaper. Joomla has a strong community of
developers who have built several plug-ins and modules that can be added to basic joomla installation.

### 2.1.2 Workflow Management System

The purpose of this component is to define the Tasks, States, Activities, Responsibilities and the Information flow in the whole system. A few open source Workflow Managers were tested and found to have either insufficient functionality or had too many bugs. A Workflow Manager is currently being built to fulfill all the unique requirements of CoolStateLA.

### 2.1.3 Asset Management System

The Asset Management System is the core component of this project. This System is consist of tasks and decisions surrounding ingesting, defining, cataloging or organizing, storage and retrieval of digital contents. Asset management system is built on top of fedora repository Which is a free open source architecture built for the integration of data, interfaces, and mechanisms to achieve interoperability and extensibility.

### 2.1.4 STORI Manager

The STORI Manager will work with the workflow manager and handle all the STORI all the contents and meta data required for that. It will communicate with Asset
Management System and the RDBMS both and will decide what data needs to be stored in AMS and what data will go to the RDBS.

2.1.5 RSS News Feed Filters

RSS News Feed Filter will search and gather all the news that match certain criteria on large news agencies' websites, like CNN, New York Times, LA Times. A system based on ROME filters is currently being built by the CoolStateLA's development team.

2.1.6 Wiki/Bulletin Board

A wiki and forum will be available as a part of this project for general public to participate in the information sharing process. WikiMedia and PHPbb are proposed for this purpose.

2.2 Architecture of Asset Management System

The Proposed Asset Management System is comprised of following components.

1. Fedora Repository/Server
2. Fedora Client
3. Fedora Client API
4. CoolStateLA’s Fedora Client
5. Integration Interface
Figure 2.2 – AMS Architecture
2.2.1 Fedora Repository

Fedora is an acronym for Flexible Extensible Digital Object Repository Architecture. Fedora’s flexibility makes it capable of serving as a digital repository for a variety of use cases. Among these are digital asset management, institutional repositories, digital archives, content management systems, scholarly publishing enterprises, and digital libraries. It is an open-source software licensed under the Mozilla Public License.

Fedora Repository is where all the contents and their metadata will be stored. This repository provides a set of methods to ingest, search, organize, annotate and retrieve contents. Although fedora package comes with a pre-configured Servlet Container (Tomcat) and a DBMS (mckoi), they are not recommend for production system.

2.2.2 Fedora Client

Although any WebService /SOAP client can be used to call fedora's method, fedora provides its own client that is easy to use and has an Applet based interface to perform most of the required tasks.

2.2.3 Fedora Client API

Fedora Client provides a rich API to call fedora's method without much hassle. It basically lets your application communicate with the fedora client.
2.2.4 CoolStateLA's Fedora Client/Controller

A client application for Fedora has been built as a part of this thesis. This client provides methods to create FOXML Objects for ingesting into fedora, create relationships, build content model and Batch Import the contents.

2.2.5 Integration Interface

This Interface will consists of a set of methods that will allow communication between Asset Manager and other Components of CoolStateLA.
CHAPTER 3
DIGITAL REPOSITORIES FOR AMS

This chapter will concentrate on the selection of right repository for this project based on certain criteria that are discussed in Section 3.2.1

3.1 Digital Repositories

A digital repository is simply a digital library in which collections are stored in digital formats and accessible by computers [1]. It is an environment, which supports the full life cycle of creation, storage, preservation, dissemination and use of data, information and knowledge[2]. They provide the resources, including the specialized staff, to select, structure, offer intellectual access to, interpret, distribute, preserve the integrity of, and ensure the persistence over time for the collections of digital works so that they are readily and economically available for use by a defined community or set of communities[2].

Some of the leading institutional repository software platforms are

Archimede

· bepress
· CDSware
· CONTENTdm
· DSpace
3.2 Selection of Right Repository for CoolStateLA’s AMS

This Section will talk about the unique requirements and driving factors in this project that led to the selection of fedora repository.

3.2.1 Factors to Consider

There are several factors or criteria in general for the selection of a repository based on the budget, functionality requirements and the size of a project. The following factors were considered for the selection of a repository for the Asset Manager.

3.2.1.1 Cost /Budget

CoolStateLA is currently being funded only by California State University, Los Angeles. Since CoolStateLA is a low budget project, one of the key element to consider while choosing a repository is cost. This project is intended to be completely open source project, therefore, preferably an open source free repository is most suitable for this project.
3.2.1.2 Functionality

Although, the more functionality a repository can provide, the better it is, but at the least, this project requires versioning, search ability, Extensibility, Customized MetaData, Batch Processing of contents, content model, platform independence and a rich API.

3.2.1.3 Reliability

This project is expected to have a large number of contents that will be kept in the repository. Only a repository that can handle that kind of large data will be suitable for this project. A repository should also provide an easy way to schedule and create backups.

3.2.1.4 Security

Although, pretty much all the contents in this project will be publicly available to read, only limited number of users will have write access to the repository. The repository should provide SSL and encryption features.

3.2.1.5 Documentation and Support

A nice and efficient application with proper documentation and support is a waste. Good documentation can help quick learn the repository and use the its full functionality.
3.2.1.6 Activities

The repository should have a strong and active developer community who could build the patches and fix any bugs in the application as they occur or reported.

3.2.2 Shortlisting

Initially four free open source repositories were considered for this project based on their functionality, reliability, size and nature of the organizations using them and their developer and user communities. Those four repositories are

1. Fedora.
2. Dspace
3. E-Print
4. Remository

Remository is an open source repository originally developed for MAMBO (A Content Management System) but now it is also available as a plugin for Joomla which makes it really easy to install and use with Joomla. However, its limited functionality (versioning, custom metadata, API, search capabilities) and less active community makes it unsuitable for big systems such as CoolStateLA.

E-Prints was dropped in the early stages of this project because of its inability to work on windows platform. However “E-Prints 3.0 for windows” was released later on
back in Oct 2007 but it requires tons of extra applications to install first.

Both Fedora and DSpace were installed, experimented with and evaluated in this project.

The following table provides a good comparison between the two repositories.

<table>
<thead>
<tr>
<th>Feature</th>
<th>DSpace</th>
<th>Fedora</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary Bitstream Retrieval</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Versioning</td>
<td>No</td>
<td>YES</td>
</tr>
<tr>
<td>Batch Import</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Metadata search</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Full text search</td>
<td>YES</td>
<td>YES (with Gsearch)</td>
</tr>
<tr>
<td>User Roles</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Web Interface</td>
<td>YES</td>
<td>YES (3rd party)</td>
</tr>
<tr>
<td>Work Flow</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Integration with other app.</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Open API</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>OAI/DC Support</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Programming Language</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>Content Model Support</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Metadata Browsing and sorting</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Platform Supported</td>
<td>Win/Linux/Unix</td>
<td>Win/Linux/Unix</td>
</tr>
<tr>
<td>Database Support</td>
<td>PostGre/MySQL/Oracle</td>
<td>PostGre/MySQL/Oracle/Mckoi</td>
</tr>
<tr>
<td>3rd Party tools &amp; app</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>
TABLE 3.1-DSpace Vs Fedora Cont.

<table>
<thead>
<tr>
<th>Feature</th>
<th>DSpace</th>
<th>Fedora</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good API Documentation</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Good User Documentation</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Community Help</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Handle Large No. of Contents</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Good Security Mechanism</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Although DSpace and Fedora both have most of the features required for this project, Fedora's versioning feature, powerful API and easy integration makes it ideal for this project.
CHAPTER 4
FEDORA REPOSITORY

This chapter will discuss the fedora data structures, mechanism for ingesting and retrieving objects, the available APIs in fedora, security mechanisms and its interface.

4.1 Configurations and Environment

It is always an important decision to choose the right operating environment and configuration for an application and can have significant effects on the performance, reliability, security and efficiency of the system.

Fedora supports both Windows and Unix/Linux platforms. It comes with a pre-configured java based database called Mekoi and the Tomcat Servlet Container or Server. These two can be used for a development server but they are not recommended for a production server due to the performance and security issues.

A small configuration can cause huge delays in the completion of any relevant task. The following graphs show some interesting data taken by a team of Fedora community.
Figure 4.1- API-M Testing-Roundtrip Time Per Operation[6]
Some of the terms used in these graphs are explained later on in this chapter but these graphs clearly show that the right configuration and DBMS selection can significantly improve the performance of our system.

4.2 Fedora Objects

Fedora Supports 4 different kind of objects.

A Data Object in a Fedora repository describes content (data and metadata) and a
set of associated behaviors or services that can be applied to that content. Data objects comprise the bulk of a repository [7].

A *Behavior Definition* (BDef Object) is a template for client-side services, defining a set of abstract operations (methods) and their client-side arguments. Association of a BDef with a digital object augments the basic behavior of the object with the operations defined in the BDef template [8].

*Behavior Mechanism* (or BMech) registers within Fedora the capability of web service(s) to perform the operations defined by a specific BDef. This registration includes defining service binding metadata encoded in the Web Service Description Language (WSDL) and also a data profile of the BMech. The data profile defines the types of inputs that are considered compatible with the service. In particular it declares the MIME types that are needed by the respective web service to perform its task.

*Content Model* (or CModel) is a digital object that in addition to being used to store information that will allow you to validate whether a data object constitutes a valid object corresponding to that content model. The Content Model is also an important piece for doing disseminations in the Content Model Architecture. A Data Object will indicate which Content model they represent via a special RELS-EXT DataStream [8].

### 4.3 Datastreams and Disseminators

A Datastream is a component of a Fedora digital object. It represents some
MIME-typed stream of content. The datastream is a description of this content and a pointer to the content’s location. The datastream is not, however, equivalent to a file on a file system because the datastream may encapsulate bytestream content internally in the case of XML content stored with the object, but all other content is a reference to content that exists external to the repository[7]. A Datastream can be a Managed content, Inline XML, Redirect or a reference to some External content or asset.

A Disseminator is the component in a digital object that is used to associate behaviors (i.e., services) with the object. So a Disseminator is a set of service subscriptions between the data object and a pair of behavior objects. The data object defines requirements for presentation of the data referenced within it by explicitly referencing the behavior objects [8].
Figure 4.3 – Fedora Object
4.4 Storage and Retrieval Mechanism

This section will cover all the necessary details for creating, ingesting, organizing and retrieving the objects or contents. However, we will not discuss method calls/API in this section. That will be covered in subsequent sections.

4.4.1 Ingesting Objects

Ingesting objects simply means calling the fedora ingest method and providing a bytestream of array containing the object details like Properties, Datastreams, Disseminators and Relationship information in XML form.

4.4.1.1 Meta-data

Meta-data defines the data or contents stored in an object so that they could be organized in a way that is not only logical but also allows easy retrieval. Fedora supports Open Archive Initiative (OAI) and hence the Dublin core metadata.

The Open Archives Initiative (OAI) is an attempt to build a "low-barrier interoperability framework" for archives (institutional repositories) containing digital content (digital libraries). It allows people (Service Providers) to harvest metadata (from Data Providers). This metadata is used to provide "value-added services", often by combining different data sets. [5]

The Dublin Core metadata is stored in a datastream called DC and has the
following fields in fedora

Title

Creator

Subject

Description

Publisher

Contributor

Date

Type

Format

Identifier

Source

Language

Relation

Coverage

Rights

In addition to this basic Dublin Core metadata a customized metadata can be defined using a separate datastream.
4.4.1.2 FOX·L

FOX·L is a simple XML format that directly expresses the Fedora digital object model. As of Fedora 2.0, digital objects are stored internally in a Fedora repository in the FOX·L format. In addition, FOX·L can be used for ingesting and exporting objects to/from Fedora repositories [9]

At the highest level, the FOX·L XML schema defines elements that correspond directly to the fundamental Fedora digital object components. Below is a brief sketch of these elements.

4.4.1.3 Properties of Object

Here is a sample fedora XML that defines the properties.

```xml
<digitalObject PID="uniqueID">
  <!-- there are a set of core object properties -->
  <objectProperties>
    <property/>
    <property/>
    <property/>
    ...
  </objectProperties>
  <!-- there can be zero or more datastreams -->
  <datastream>
```
<datastreamVersion/>
<datastreamVersion/>
...
</datastream>

!-- there can be zero or more disseminators -->

<disseminator>

<disseminatorVersion/>
<disseminatorVersion/>
<disseminatorVersion/>
...
</disseminator>
</digitalObject>
### 4.4.1.4 Defining Datastreams

The following attributes have to be defined while defining a datastream.

#### Table 4.1 - Datastream Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Unique Identifier for a datastream</td>
<td>Anything</td>
</tr>
<tr>
<td>State</td>
<td>Whether a datastream is available or not</td>
<td>A(Active) or D(Disabled)</td>
</tr>
<tr>
<td>Control-Group</td>
<td>Provides the type of datastream, ie, XML Managed</td>
<td>M,E,R,X</td>
</tr>
<tr>
<td></td>
<td>Contents, External or Redirect</td>
<td></td>
</tr>
<tr>
<td>Versionable</td>
<td>Whether to maintain the versions of datastream</td>
<td>true, false</td>
</tr>
<tr>
<td>Datastream Version</td>
<td>The version of datastream being ingested</td>
<td>Anything in format DS.0 DS.1..</td>
</tr>
<tr>
<td>Mimetype</td>
<td>Mime type of contents in the Datstream</td>
<td>text/xml, html, image, video etc</td>
</tr>
<tr>
<td>Label</td>
<td>Label of datastream</td>
<td>Anything</td>
</tr>
<tr>
<td>Size</td>
<td>Size of datastream in bytes</td>
<td>Any numbers</td>
</tr>
<tr>
<td>Creation Date</td>
<td>When the datastream was created. If not set explicitly</td>
<td>A data + time in format 2004-12-10T00:21:58.000 Z</td>
</tr>
<tr>
<td></td>
<td>fedora automatically fills in this value</td>
<td></td>
</tr>
</tbody>
</table>
The following XML is an example of datastream XML

<foxml:datastream ID="DC" STATE="A" CONTROL_GROUP="X"
 VERSIONABLE="true">
  <foxml:datastreamVersion ID="DC.0" MIMETYPE="text/xml"
 LABEL="Default Dublin Core Record" SIZE="488"
 CREATED="2004-12-10T00:21:58.000Z">
    <foxml:xmlContent>
      <oai_dc:dc
 xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
        <dc:title>FOX·L Reference Object</dc:title>
        <dc:creator>Sandy Payette</dc:creator>
        <dc:subject>Fedora documentation</dc:subject>
        <dc:description>FOX·L showing how a digital object is encoded for persistent storage in a Fedora repository</dc:description>
        <dc:publisher>Cornell CIS</dc:publisher>
        <dc:identifier>demo:999</dc:identifier>
      </oai_dc:dc>
    </foxml:xmlContent>
  </foxml:datastreamVersion>
</foxml:datastream>
4.4.1.5 Defining Disseminators

As of Fedora 2.2, the Disseminators were part of the data objects. However, in the latest release of Fedora, i.e., Fedora 3.0, a new model object has been introduced called CModel (explained in Chapter 6). All the dissemination information is in CModel and every data object that conforms to that CModel object inherits all the disseminators available in CModel object.

4.4.1.7 Establishing Relationships

As of Fedora 2.2, the only way to create a relationship between two objects was to create XML for datastream and add it to an object as a datastream but Fedora 3.0 provides addRelationship method that takes simple object pids and relationship name, and creates the relationship.

These relationships define the Atomistic Content Model, which is explained in Chapter 6.

4.4.2 Retrieving Objects

Fedora provides tons of methods and ways to retrieve objects based on their meta
data or relationships. The use of the right method and ways is important for performance of your system and depends on your content model.

Fedora Objects can be retrieved using a search on the dublin core meta-data through APIA methods. The results are usually returned as a FieldResults Array. Each item represents an object in fedora and provides all information through getters.

4.5 Application Programming Interfaces (APIs)

Fedora provides a large number of APIs that allows the programmers and developers from different background and domain knowledge to build applications for Fedora. Although fedora is built using Java language and uses all the Java based frameworks, the use of Web Services made it possible for a lot of PHP, Perl and other languages' programmers to build tools and interfaces for Fedora.

4.5.1 Java API

Since fedora is developed using Java language. It provides a rich Java API. The complete API specification is available on Fedora's website.

4.5.2 API-Access

The Fedora Access service defines an open interface for accessing digital objects. The access operations include methods to do reflection on a digital object (i.e., to discover
the kinds of disseminations that are available on the object), and to request disseminations. The major function of the Fedora Access service is to fulfill a client's request for dissemination. To support disseminations, the underlying repository system must evaluate the behavior associations specified in a digital object, and figure out how to dispatch a service request to a supporting service with which the digital object associates. The supporting service may be internal to the repository system, or it may be an external web service that the repository must call upon. The underlying repository system facilitates all external service bindings on behalf of the client, simply returning a dissemination result via the Access service layer [10]
Table 4.2 APIA Methods[11]

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>describeRepository</code></td>
<td>provides information about the Fedora repository server, including name, version, base URL</td>
</tr>
<tr>
<td><code>getDatastreamDissemination</code></td>
<td>gets the specified datastream's contents</td>
</tr>
<tr>
<td><code>getDissemination</code></td>
<td>gets the specified dissemination request</td>
</tr>
<tr>
<td><code>getObjectProfile</code></td>
<td>gets the Object Profile of the specified object</td>
</tr>
<tr>
<td><code>getObjectHistory</code></td>
<td>gets the change history of the specified object</td>
</tr>
<tr>
<td><code>findObjects (search)</code></td>
<td>performs a search on the repository given specified search criteria and returns a result set for matching objects</td>
</tr>
<tr>
<td><code>listDatastreams</code></td>
<td>gets a list of datastreams for the specified object</td>
</tr>
<tr>
<td><code>listMethods -</code></td>
<td>gets a list of disseminator methods for the specified object</td>
</tr>
<tr>
<td><code>resumeFindObjects (search)</code></td>
<td>gets the next set of items (&quot;hits&quot;) in a search result set. Used when a prior findObjects request specified that the result set should be returned with a maximum number of hits at a time.</td>
</tr>
</tbody>
</table>
4.5.3 API-Management

The Fedora Management service defines an open interface for administering the repository, including creating, modifying, and deleting digital objects, or components within digital objects. The Management service interacts with the underlying repository system to read content from and write content to the digital object and datastream storage areas. The Management service exposes a set of operations that enable a client to view and manipulate digital objects from an abstract perspective, meaning that a client does not need to know anything about underlying storage formats, storage media, or storage management schemes for objects. Also, the underlying repository system handles the details of storing datastream content within the repository, as well as mediating connectivity for datastreams that reference external content. [12]

4.5.4 REST-API

The Fedora REST API is a new, experimental feature that exposes a subset of the Fedora Access and Management APIs as a RESTful (Representational State Transfer) Web Service. The REST API may replace the API-A-Lite and API-M-Lite interfaces in future versions of Fedora.[13]

As the Fedora REST API is an experimental interface, it should not be considered stable. In particular, future releases may implement a different subset of the Fedora Access and Management APIs and may employ different URL endpoints or HTTP
methods. Finally, the REST API has not undergone security testing, so production repositories are cautioned against enabling the REST API without thorough evaluation.

Table 4.3 -API-A Methods (REST)[13]

<table>
<thead>
<tr>
<th>API Method</th>
<th>HTTP Method</th>
<th>Example URL Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>describeRepository</td>
<td>not implemented</td>
<td></td>
</tr>
<tr>
<td>getObjectProfile</td>
<td>GET</td>
<td>/objects/demo:29</td>
</tr>
<tr>
<td>listMethods</td>
<td>GET</td>
<td>/objects/demo:29/methods</td>
</tr>
<tr>
<td>listDatastreams</td>
<td>GET</td>
<td>/objects/demo:29/datastreams</td>
</tr>
<tr>
<td>getDissemination</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getObjectHistory</td>
<td>GET</td>
<td>/objects/demo:29/versions</td>
</tr>
<tr>
<td>findObjects</td>
<td>GET</td>
<td>/objects?pid=true&amp;terms=demo:29</td>
</tr>
<tr>
<td>resumeFindObject</td>
<td>GET</td>
<td>/objects?sessionToken=xyz</td>
</tr>
<tr>
<td>getObjectHistory</td>
<td>GET</td>
<td>/objects/demo:29/versions</td>
</tr>
<tr>
<td>API Method</td>
<td>HTTP Method</td>
<td>Example URL Endpoint</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>ingest</td>
<td>POST</td>
<td>/objects/new</td>
</tr>
<tr>
<td>modifyObject</td>
<td>PUT</td>
<td>/objects/demo:29?label=foo</td>
</tr>
<tr>
<td>getObjectXML</td>
<td>GET</td>
<td>/objects/demo:29/objectXML</td>
</tr>
<tr>
<td>purgeObject</td>
<td>DELETE</td>
<td>/objects/demo:29</td>
</tr>
<tr>
<td>addDatastream</td>
<td>POST</td>
<td>/objects/demo:29/datastreams/DS99?</td>
</tr>
<tr>
<td>modifyDatastream</td>
<td>PUT</td>
<td>/objects/demo:29/datastreams/DS99</td>
</tr>
<tr>
<td>ByReference</td>
<td></td>
<td></td>
</tr>
<tr>
<td>modifyDatastream</td>
<td>PUT</td>
<td>/objects/demo:29/datastreams/DS99</td>
</tr>
<tr>
<td>ByValue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setDataStreamState</td>
<td></td>
<td>not implemented</td>
</tr>
<tr>
<td>setDataStreamVers</td>
<td></td>
<td>not implemented</td>
</tr>
<tr>
<td>ionable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>compareDataStream</td>
<td></td>
<td>not implemented</td>
</tr>
<tr>
<td>mChecksumRequest</td>
<td></td>
<td>not implemented</td>
</tr>
<tr>
<td>getDataStream</td>
<td></td>
<td>not implemented</td>
</tr>
<tr>
<td>getDatasstreams</td>
<td></td>
<td>not implemented</td>
</tr>
<tr>
<td>getDataStreamHist</td>
<td></td>
<td>not implemented</td>
</tr>
<tr>
<td>ory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>purgeDataStream</td>
<td>DELETE</td>
<td>/objects/demo:29/datastreams/RELS-EXT</td>
</tr>
<tr>
<td>getNextPID</td>
<td>GET</td>
<td>/objects/nextPID</td>
</tr>
</tbody>
</table>
4.5.5 Resource Index

The Resource Index is the Fedora module that provides the infrastructure for indexing relationships among objects and their components. Examples of relationships between digital objects include well-known management relationships such as the part-whole links between individual chapters and a book, and semantic relationships useful in digital library organization such as those expressed within the Functional Requirements for Bibliographic Records (FRBR).

Fedora expresses relationships by defining a base relationship ontology [RELS-EXT] using RDFS and provides a slot in the digital object abstraction for RDF expression of relationships based on this ontology. Assertions from other ontologies may also be included along with the base Fedora relationships. All relationships are represented as a graph that can be queried using an RDF query language. The query interface to the Resource Index is exposed as a web service [risearch], [Fedora Relationships].

The Resource Index Search Service (RISearch) is a web service that exposes the contents of a repository's Resource Index for outside use [14].

Example Syntax for Requesting Tuples

http://localhost:8080/fedora/risearch?type=tuples
    &flush=[true (default is false)]
    &lang=iTQL
    &format=CSV|Simple|Sparql|TSV
    &limit=[1 (default is no limit)]
4.5.6 Basic OAI

The OAI Protocol for Metadata Harvesting (OAI-PMH) is a standard for sharing metadata across repositories. Every Fedora digital object has a primary Dublin Core record that conforms to the schema at: http://www.openarchives.org/OAI/2.0/oai_dc.xsd. This metadata is accessible using Fedora's OAI-PMH2.0 Provider Interface. Currently, only the Dublin Core metadata for each object may be disseminated via this interface [15].

The Fedora OAI Provider Interface uses the standard OAI-PMH syntax which can be found on the OAI web site (http://www.openarchives.org). Fedora will accept any valid OAI-PMH2.0 request.

The Fedora OAI-PMH Provider Interface is implemented as a java servlet and can be accessed using the following syntax:


4.6 Security Mechanisms

Fedora uses Sun's XACML (eXtensible Access Control Markup Language) and an XACML-based policy enforcement module. Developed by the OASIS Consortium,
XACML is an XML-based markup language to encode access control policies.

4.7 Available Clients

Fedora server can be accessed remotely in several different ways.

4.7.1 Fedora-Client Applet

It is a small applet based fedora client that comes with fedora and can be used to perform most of the tasks such as ingesting, searching and deleting objects and establish relationships.

4.7.2 Commandline Utilities

The Fedora client comes with several useful command-line utilities under server/bin folder.

fedora-dsinfo

fedora-export

fedora-find

fedora-ingest

fedora-ingest-demos

fedora-convert-demos
fedora-purge
fedora-modify
fedora-batch-build
fedora-batch-ingest
fedora-batch-buildingest

4.7.3 Internet Browser

As explained earlier, fedora provides a REST style API. So a lot of fedora's functions can be accessed through simple url (post or get style) from an internet browser.

4.7.3 WebServices Browser

Any WebServices browser such as Eclipse Web Service Browser can be used to call any method on the server through SOAP calls.

4.8 FEZ

Fedora software doesn't come with any web interface, however, there are tons of community built interfaces are available. FEZ is one of the most popular interface for fedora built by University of Queensland.

FEZ was installed as a part of this project. It provides a complete framework for
building a new repository, ingesting and retrieving objects and a nice workflow system.

However, the place where it lacks is allowing a customized display and content model.

Although FEZ's documentation indicate that its possible but takes a lot of efforts and there is no documentation on how to do that.
CHAPTER 5
CONTROLLER FOR ASSET MANAGEMENT SYSTEM

Fedora is a framework that provides a Backend logic and It is up to the user or programmer how they want to use that. It is comparable to a database where a mechanism to create database, tables, fields and procedures is provided but an application creates them based on how the application wants to store and retrieve information.

This chapter talks about the Controller or Business Logic Module that was written as a part of this thesis to provide an easy way for ingesting, annotating, storing, deleting contents/objects and implementing a content model.

5.1 Functionality Overview

This controller is needed between this backend logic and the front-end interfaces. The Controller takes a Data and Metadata and translates it into Fedora understandable language, ie, FOXML using the DOM parser and sends it to the fedora client. The Presentation Layer doesn't need to understand or create the FOXML structure. This controller is also responsible for creating Content Models.

5.2 Object Types

The object types here are basically fedora's data objects. The meta data attached to
them and the relationships between them make them logically a collection or item object.

5.2.1 Collection Object

A Collection object is simply a data object with rel:isMemberOf relationship with other data objects, where the relationship is to this object. Such an object can be used for creating Gallery, Stori, Feed collection or any other set of object that should logically be together.

5.2.2 Item Object

Item object is a data object with rel:isMemberOf relationship with Collection object. So logically we can say that all the item objects are part of or member of collection object or collection. Item objects can be image objects, audio or video object or even other collection objects.
5.3 Ingesting Contents

The Controller provides two different ways to ingest Contents into fedora
5.3.1 Single Ingest

Single ingest or single object ingest is simply creating a standalone object inside the repository without any relationships that is defined or has any meanings in content model.

5.3.2 Batch Ingest

Batch ingest allows to ingest multiple contents /objects through a batch job. Currently, the controller just supports the batch import of images. It can take a directory of images, create one object per image which involves building all the necessary meta data and Datastreams and adding relationships and once its done the object will be ingested into Fedora. The created object will be a part of collection and will conform to image content model.
Figure 5.2 – Batch Ingest Model
5.4 Searching Contents

This controller takes the following three approaches to search the contents.

5.4.1 Searching Dublin Core

This one simply searches the contents using the Dublin core meta data attached with that content. It is usually faster then Custom Metadata search but slower then RISearch.

```java
Condition []conditions = {new Condition("pid",ComparisonOperator.has,"cools*")};

String []rfields = {"pid","label","relation"};

FieldSearchResult fsr = apia.findObjects(rfields,new NonNegativeInteger("50"), new FieldSearchQuery(conditions,null));
```

This above code searches all the objects that contain cools+[string]

5.4.2 Searching Custom Metadata

Since a lot of meta data required to clearly define and identify the contents is not available in dublin core. The controller implements its own meta data. To search through
that meta data the controller uses DOM parser to read the metadata into a HashMap with
meta data name as the key. This HashMap can be searched using basic java methods.

```java
public Map<String, Element> elemList = new HashMap<String, Element>();
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    //Using factory get an instance of document builder
    DocumentBuilder db = dbf.newDocumentBuilder();
    //parse using builder to get DOM representation of the XML file
    Document dom = db.parse("http://localhost:8080/fedora/get/" + objPid + "/CoolsD");
    Element docEle = dom.getDocumentElement();
    NodeList nl = docEle.getChildNodes();
    removeAll();
    if(nl != null && nl.getLength() > 0) {
        for(int i = 0 ; i < nl.getLength();i++) {
            Node el = nl.item(i);
            element elem = new element();
```
5.4.3 Resource Index Search

The controller sends a request to mulgara database with an ITQL query and the results are returned from mulgara in the form of subject, relationship, object. A method is written to separate objects from rest of the results and put it in a list.

RISearchHelper ris = new RISearchHelper();

String query = "select $s $p $o from <#ri> where $s $p $o and $p <mulgara:is> <info:fedora/fedora-system:def/relations-external#isMemberOf> and $o <mulgara:is> <info:fedora/"+object+">";

try {
    String str = ris.search(query);

    for (String ss:processStr(str))
        rlist.put(ss);
5.5 POJO to FOXML

As explained in the previous chapter, Fedora allows to ingest object as Fedora Object XML (FOXML). Building FOXML is a pretty tedious process. Our Controller provides a much simpler way to create POJOs and convert them to FOXML.

5.5.1 XML Parser

Our Asset Manager uses the Xerces which allows XML parsing and generation. Although any XML stream can be parsed using DOM (Document Object Model) or SAX (Simple API for XML) parser. We have used the DOM parser, since most of the XML documents are small in size and each document gets traversed multiple times.

5.5.2 FOXML Tags

FOXML specification includes a set of tags which is not large but requires a lot of attributes to be set with those tags.

5.5.2.1 Attributes

These are much like the properties of an XML tag and just provide more information related to that tag.

<foxml:digitalObject VERSION="1.1" PID="demo:999"
Xerces provides a simple approach for setting attributes.

```java
root.setAttribute("VERSION", "1.1");
root.setAttribute("PID", namespace+:""+identifier);
root.setAttribute("xmlns:xsi","http://www.w3.org/2001/XMLSchema-instance");
root.setAttribute("xsi:schemaLocation",
"info:fedora/fedora-system:def/foxml#
http://www.fedora.info/definitions/1/0/foxml1-1.xsd");
```

Where root is the root Node(Element or Tag) of the XML tree.
5.5.2.2 Namespaces

Namespaces disallow any ambiguity in the names of XML elements. It can be anything of our choice but usually it is a real URI.

5.5.2.3 Tag Values

The tag values are simply whatever is in between starting and closing tag, i.e.,

<TagName> Value </TagName>

The values can be set using

TagNode.setTextContent(value);

5.5.2.4 Building Tree

The XML tree is built by appending tags to each other, so as an example

<FatherTag>

<ChildTag> Value </ChildTag>

</FatherTag>
To create a tree, we will create elements of both FatherTag and ChildTag and then simply do

FatherTag.append(ChildTag)

5.5.3 Generating Datastreams

This is simply a matter of generating tags, defining the necessary attributes, setting the values and appending them to each other to form a tree. Here is an example of building dublin core.

```java
Element elemds = xml.doc.createElement("foxml:datastream");
elemds.setAttribute("ID", "DC");
elemds.setAttribute("STATE", "A");
elemds.setAttribute("CONTROL_GROUP", "X");
elemds.setAttribute("VERSIONABLE", "true");
Element elemdsv = xml.doc.createElement("foxml:datastreamVersion");
elemdsv.setAttribute("ID", "DC.0");
elemdsv.setAttribute("MIMETYPE", "text/xml");
elemdsv.setAttribute("LABEL", "Default Dublin Core Record");
elemdsv.setAttribute("SIZE", "488");
// elemdsv.setAttribute("CREATED", "2008-12-10T00:21:58.000Z");
```
Element xmlcon = xmldoc.createElement("foxml:xmlContent");

Element oai_dc = xmldoc.createElement("oai_dc:dc");
oai_dc.setAttribute("xmlns:oai_dc","http://www.openarchives.org/OAI/2.0/oai_dc/");
oai_dc.setAttribute("xmlns:dc","http://purl.org/dc/elements/1.1/");

Element dctitle = xmldoc.createElement("dc:title");
Element dccreator = xmldoc.createElement("dc:creator");
Element dcsubject = xmldoc.createElement("dc:subject");
Element dcdescription = xmldoc.createElement("dc:description");
Element dc.publisher = xmldoc.createElement("dc:publisher");
Element dcdate = xmldoc.createElement("dc:date");
Element dc.contributor = xmldoc.createElement("dc:contributor");
Element dctype = xmldoc.createElement("dc:type");
Element dcrights = xmldoc.createElement("dc:rights");
dctitle.setTextContent(title);
dccreator.setTextContent(creator);
dcs.propertyContent(subject);
dcdescription.setTextContent(description);
dcpublisher.setTextContent(publisher);
dcpublisher.setTextContent(contributor);
dcpublisher.setTextContent(date);
5.5.4 Building Objects

A Fedora object is nothing but a combination of datastreams with some extra tags and some properties of the object. With the above mentioned techniques of building XML a fedora object can easily be created.
The only new thing is setting the properties of object. But no matters what the fedora term is for a part of FOXML, it is still simply a Tag or tree of Tags. Here is an example of setting the properties.

```
// setting header for xmlobject

String[] NAME =
{
"http://www.w3.org/1999/02/22-rdf-syntax-ns#type",

String[] VALUE = {"FedoraObject","A",objLabel};

Element objproperties = xmldoc.createElement("foxml:objectProperties");

for (int i=0;i<NAME.length;i++)
{

    // Child i.

    Element e = xmldoc.createElement("foxml:property");
    e.setAttributeNS(null, "NAME", NAME[i]);
    e.setAttributeNS(null, "VALUE", VALUE[i]);
    objproperties.appendChild(e);
```

return objproperties;
Whenever we work with digital content, it is with an established set of expectations for how an intellectual work may be expressed and we also want to communicate our intellectual works effectively to others. To do this, we have an expectation that the important aspects of our digital works are delivered accurately to users accessing them [16].

Traditionally, the term "content model" used (and originated) with the publishing community to describe the physical structure of the intellectual work. Users and collectors of intellectual works often add value by organizing works, annotating them, and producing ways to find them or information within them. However digital technology enables ways to create and use content in ways that cannot be achieved with physical media [16].

Digital content may also incorporate functions as a part of its nature. A spreadsheet and an image are hardly any different if there is no software to make it function. Those features which must be present to provide an authentic experience of the digital content are called the "essential characteristics" and they can be captured in a content model to ensure durability of the experience as format and technology changes over time. Content models can be used...
1.) for content classification to facilitate discovery.

2.) For validation usually at ingest or modification.

3.) as a template when content is created to generate user interfaces, drive workflows, describe content components, or to manage policy enforcement.

We use the term "content model" to mean both:

1. Content structure as used by publishers and other traditional content-related professions

2. A computer model describing an information representation and processing architecture

By combining these very different views, C·A has the potential to provide a way to build an interoperable repository for integrated information access in our organizations and to provide durable access to our intellectual works [16].

An understanding of design decisions behind this "first-generation" C·A is a key element for community participation in future generations of C·A development. Most important is an understanding of three significant and interrelated developments in software engineering:

1. object-oriented programming

2. design patterns

3. model-driven architectures
The Content Model Architecture (CMA) describes an integrated structure for persisting and delivering the essential characteristics of digital objects in Fedora. Two primary ways of thinking about content models have emerged and both are supported by the CMA. The first approach is focused on using complex single-object models and is commonly called "compound." The second approach is to use multi-object models and is commonly called "atomistic" or "linked" [16].

In the CMA, the "content model" is defined as a formal model that describes the characteristics of one or more digital objects.

In the CMA, the concept of the content model is comprehensive, including all possible characteristics which are needed to enable persistence and delivery of the content. This can include structural, behavioral and semantic information. It can also include a description of the permitted, excluded, and required relationships to other digital objects or identifiable entities [16].

Since we anticipate that a large number of digital objects will conform to the same content model, they may be treated as a class.

CModel both represents the notion of the class and can contain the content model. The CMA does not require that digital objects explicitly conform to its architecture or explicitly declare any of its metadata elements beyond providing well-formed Fedora digital objects - unless you want to use the advanced features provided by the Fedora repository [16].
The minimum requirement to participate in the CMA is for a digital object to assert a relation to record its class' identity.

The remaining functionality is enabled by creating the content model and storing it within Fedora digital objects like any other content. [16]

6.1 Important Factors to Consider

Now that we have established a basic understanding of content model. Let's try to understand the requirements for a content model. As defined earlier in this chapter, there are two kind of content models atomistic and compound. The key element in atomistic model is Relationships and in the compound model is Datastreams.

Let's discuss the compound model first.

In order to choose the right Datastreams for our objects, first we need to know what kind of contents we want to persist, how we want to search them and present them. The following figure shows a simple data object that represents an image.
DC means Dublin core Datastream and CMD means customized meta data and it is another datastream.

So why did we choose to use CMD Datastream. The answer is simply because we needed to keep the information about the Date that picture was taken. There is no room for this information in Dublin Core so only way to have this information or any other information that is not a part of Dublin core is to create a new Datastream and put that information in there. Although such Datastream can be simple text with values separated by comma or tabs but the preferred and recommended way is to use XML. Content is another Datastream that simply redirects to Image datastream. This will help accessing any object's main content regardless of whether its an Image, video, audio or some other content. Image is a Managed datastream and is stored inside fedora. A call to this Datastream will return the actual image.
The point to be noted here is how we selected the right datastreams, type of Datastreams and their contents based on what information or data we want to retrieve and how we want to retrieve. A simple ER diagram can usually answer What and a combination of a detailed Use case and data flow diagram can answer How.

Another key factor in choosing the Datastream is an understanding of Fedora system and how is it being used in your application. For example, We know photographer is an important information and will be searched frequently. The Dublin core is indexed in fedora and methods are provided to retrieve any Dublin core element. However Customized Meta data is although indexed but our application is not using that indexing So retrieval of meta data from customized meta data datastream will be way slower then the dublin core. Thats why we used the dublin core's contributor field to represent photographer.

Now that we have some understanding of how a compound model is built ,lets discuss how these compound models are linked together to build an atomistic or linked model.

Fedora allows to define your own relationships. So We can define relationships such as isA ,has, isnotA, isPartOf, isChildOf etc between two objects as long as it makes sense to have such a relationship between those objects. For example, an object that represent your Bank Account can not have isA relationship with an object that represents an Image. That would be illogical. But isNotA relationship between these two does make sense. If you have a basic understanding of common data structures like Array or Set then you can understand
that the relationship between an Array and its element can be defined as has or contains if
type array to element, and if its element to array relationship then it can be isPartOf,
isContainedBy or isElementOf. So if we have to define a collection of same kind of objects
inside Fedora we can simply create a collection object and create isPartOf relationship from
any object that wants to be part of this collection to the collection object.

See the figure below to understand it a little better.
Figure 6.2 – Atomistic Content Model
In this figure, we have built a gallery of images and every image object has isMemberOf relationship with the Gallery object.

6.2 Building Content Model in Fedora

There are two basic approaches for using content models. First, the content and content models can be disseminated to applications able to interpret them to deliver the essential characteristics of the content. Disseminating the content model to external services can be also be used when creating new digital objects for ingest, validation, transformation and replication.

Table 6.1 – Fedora Content Model Objects [16].

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data (Data)</td>
<td>A container for content</td>
</tr>
<tr>
<td>Service Definition (BDef)</td>
<td>A container for service definitions, an element of a content model</td>
</tr>
<tr>
<td>Service Deployment Mechanism(BMech)</td>
<td>A container for service deployment mechanisms</td>
</tr>
<tr>
<td>Content Model (CModel)</td>
<td>A container for content models</td>
</tr>
</tbody>
</table>
Figure 6.3 – Fedora Content Model [16]
We have created the following model for our system.

Figure 6.4 – Image Content Model

This model is explained in the next section but the thing to notice here is the relationships between object and the basic idea of these objects as explained in the above table.
6.3 Image CModel Sample

We are trying to make our Image object a part of Image CModel so that we could use the disseminators defined in that CModel. Another idea of having CModel is to validate (disabled by default) any model that conforms with it. DS-COMPOSITE-MODEL contains a list of datastream that are expected to be in any object that conforms with a CModel and the Mime Type of that datastream. This information allow a CModel object to be used as a template for the new objects.

Below is the contents of DS-COMPOSITE-MODEL datastream in this model.

```xml
<dsCompositeModel xmlns="info:fedora/fedora-system:def/dsCompositeModel#">
  <dsTypeModel ID="DC">
    <form MIME="text/xml"/>
  </dsTypeModel>
  <dsTypeModel ID="RELS-EXT">
    <form MIME="text/xml"/>
  </dsTypeModel>
  <dsTypeModel ID="Image">
    <form MIME="image/jpeg"/>
  </dsTypeModel>
</dsCompositeModel>
```
It simply says that any object that conforms with this CModel has to have two XML datastreams and one Image datastream.

The BDef object does nothing more then defining the name and parameters or inputs of the disseminators. It is more or less like an abstract class in Java. The disseminators are defined in a datastream inside bmech called MethodMap. Below is the MethodMap of the BDef object shown in the above diagram

```xml
<fmm:MethodMap name="MethodMap - BDef Object for Image Transform"
xmlns:fmm="http://fedora.comm.nsdlib.org/service/methodmap">
  <fmm:Method label="Converts an image to grayscale" operationName="grayscale"/>
  <fmm:Method label="adjusts the brightness" operationName="brightness">
    <fmm:UserInputParm defaultValue="" label="" parmName="brightAmt" passBy="VALUE" required="true"/>
  </fmm:Method>
  <fmm:Method label="Watermarks an image" operationName="watermark">
    <fmm:UserInputParm defaultValue="" label="" parmName="wmText" passBy="VALUE" required="true"/>
  </fmm:Method>
  <fmm:Method label="zooms an image" operationName="zoom">
    <fmm:UserInputParm defaultValue="" label="" parmName="zoomAmt"
```
The bmech object is the one that defines all the bindings to a webservice. Each parameter is appended to the base url of a webservice through a mechanism.

6.4 CoolStateLA’s Content Model

Below are the content models for image gallery, stori and new feeds.
Figure 6.5 – Image Content Model (Detailed)
Figure 6.6 – STORI Content Model
Please note that the models for STORIs and Feeds do not show the CModel, BMech and BDef objects, rather they just show the disseminators. This is done for simplicity but in practice these disseminators will be implemented using fedora's CModel.
CHAPTER 7

FUTURE WORK

The Controller built as a part of this thesis implements only one content model, i.e., the Image Gallery Model. The implemented Gallery Model can be used as a reference for building STORI, Feed or any other models.

The customized meta-data class (CoolsMD) currently contains a fixed set of meta-data. Ideally, it should be flexible enough to add any meta-data. At the least, this class needs to be extended to support the meta data used in the content models shown in previous chapter.

The integration will be done once the other components in CoolStateLA define and agree on a communication protocol and interfaces.
A system consists of fedora and a custom built controller is proposed and appears to fulfill not only the current requirements of the CoolStateLA but also that system will be flexible and extensible enough to fulfill any future requirements of CoolStateLA. The proposed Asset Management System is way more powerful and provide more functionality then we originally expected. The content model presented in this thesis not only captures most of the contents and meta-data related requirements but also suggests a mechanism for effectively delivering those contents.
REFERENCES


http://www.fedora.info/definitions/1/0/api/#apia


http://www.fedora.info/definitions/1/0/api/#apia


http://www.fedora-commons.org/documentation/3.0b1/userdocs/digitalobjects/cmda.html