SloanSpace-DSpace File Transfer Component

by

Genevieve T. Cuevas

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

December 16, 2004

Copyright 2004 Genevieve T. Cuevas. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author

Department of Electrical Engineering and Computer Science
December 16, 2004

Certified by

Harold Abelson
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

BARKER
SloanSpace-DSpace File Transfer Component

by

Genevieve T. Cuevas

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

December 16, 2004

Thesis Supervisor: Harold Abelson

Abstract

This thesis demonstrates how to use Web services to integrate course management systems with digital repositories. We present a component that provides interoperability between SloanSpace, a course management system, and DSpace, a digital repository, both developed at MIT. In particular, a file transfer component was created that enables SloanSpace users to search and retrieve DSpace documents while in SloanSpace, and submit SloanSpace documents into DSpace. DSpace's web services provided the means for interaction between the systems. The architecture of the component was designed to handle not only the metadata mappings between SloanSpace and DSpace metadata, but mappings between file metadata of SloanSpace and other systems as well. Two scenarios were then created to test the effectiveness of the component. The test results demonstrate the ability of the component to decrease the amount of time spent in performing file transfers between the two systems. Most importantly, however, the component demonstrates more generally interoperability with digital repositories. It not only integrates SloanSpace with DSpace, but also allows for a more general integration with any other system.
Acknowledgements

I would like to thank Hal Abelson for teaching me how to write a thesis. I could not ask for a better advisor. I would like to thank Al Essa, for helping me construct my thesis. I would like to thank Andrew Grumet for guiding me in the development of my system. I would like to thank MacKenzie Smith and Richard Rodgers for providing the DSpace web services. Lastly, I would like to thank my family. None of this would have been possible without the love and support.
# Table of Contents

1. Introduction................................................................................................................. 7
2. Background ................................................................................................................. 8
   2.1 SloanSpace ........................................................................................................ 8
   2.2 DSpace ............................................................................................................. 10
   2.3 Scenario 1: Populating a SloanSpace File Storage Area via DSpace .......... 12
   2.4 Scenario 2: Submitting a SloanSpace file to DSpace ................................. 17
   2.5 What is the motivation behind this system? ..................................................... 18
   2.6 Challenges ....................................................................................................... 19
   2.7 Related Work .................................................................................................. 20
3. System Overview ...................................................................................................... 22
   3.1 System Functions .............................................................................................. 23
      3.1.1 Search/Retrieve ...................................................................................... 23
      3.1.2 Submit ................................................................................................... 29
   3.2 Data Model ....................................................................................................... 35
      3.2.1 Exploring the Data Model Requirements ................................................ 35
      3.2.2 Generalizing the Data Model ................................................................. 37
   3.3 System Implementation Details ....................................................................... 39
      3.3.1 Search Interface ..................................................................................... 39
      3.3.2 Retrieve Interface .................................................................................. 41
      3.3.3 Submit Interface .................................................................................... 43
4. Integrating the File Transfer Component with Other Systems ......................... 44
   4.1 Filling in the Tables Using the Add Schema Interface .................................... 45
   4.2 Providing the Code for the Submit and Retrieve Interfaces ......................... 51
   4.3 Implementing the Search Service Contract .................................................... 52
5. System Testing and Analysis .................................................................................... 52
   5.1 Testing the File Transfer Component ............................................................. 52
   5.2 Results of the Tests ......................................................................................... 53
   5.3 Discussion of the Test Results ........................................................................ 54
6 Future Work .............................................................................................................. 55
6.1 System Deployment .......................................................................................... 55
6.2 Integration with Other Systems ....................................................................... 56
References ............................................................................................................. 57
A Database Tables .................................................................................................. 58
B Critical Source Code .......................................................................................... 61
  B.1 dspace-get.tcl .............................................................................................. 61
  B.2 dspace-submit.tcl ........................................................................................ 70
  B.3 meta-view.adp .............................................................................................. 73
  B.4 meta-view.tcl ............................................................................................... 74
  B.5 schema-add.adp ............................................................................................. 77
  B.6 schema-add-2.tcl ........................................................................................ 77
  B.7 schema-add-fields.adp .................................................................................... 78
  B.8 schema-add-fields.tcl ................................................................................... 80
  B.9 schema-add-fields-2.tcl ................................................................................. 81
  B.10 schema-add-fields-cont.adp ......................................................................... 82
  B.11 schema-add-fields-cont.tcl .......................................................................... 83
  B.12 schema-add-fields-cont-2.tcl ....................................................................... 84
  B.13 search-url.adp ............................................................................................. 85
  B.14 search-url.tcl ............................................................................................. 86
  B.15 search-url-results.adp .................................................................................. 87
  B.16 search-url-results.tcl .................................................................................. 89
  B.17 dspace-search-procs.tcl .............................................................................. 90
  B.18 google-search-procs.tcl .............................................................................. 92
C Instructions for Integration With Other Systems ................................................. 95
  C.1 Filling in the database tables via the Add Schema Interface ......................... 95
  C.2 Adding the code for the submit interface ..................................................... 97
  C.3 Adding the search service contract implementation ................................... 99
  C.4 Adding the code for the retrieve interface ................................................... 101
D Installing the system into .LRN .......................................................................... 103
List of Figures

Figure 2-1: SloanSpace screenshot ............................................................................................. 9
Figure 2-2: DSpace home page ................................................................................................. 13
Figure 2-3: DSpace search results ............................................................................................ 14
Figure 2-4: DSpace search result ............................................................................................ 15
Figure 2-5: SloanSpace file storage area .................................................................................... 16
Figure 2-6: SloanSpace file upload .......................................................................................... 16
Figure 2-7: DSpace submission page ......................................................................................... 18
Figure 3-1: System overview ..................................................................................................... 22
Figure 3-2: Prof. Smith’s File Storage ....................................................................................... 24
Figure 3-3: Search query page ................................................................................................... 25
Figure 3-4: Search results page .................................................................................................. 26
1 Introduction

The number of systems developed to promote the use of technology in learning has risen dramatically as information technology resources have become more readily available. Many higher learning institutions and universities have directed much effort to the creation of course management systems, online courses, and other technologically enhanced learning tools. At the same time, the number of digital repositories being developed has also seen a similar growth rate. Many institutions and communities have created their own digital repositories. Journals, theses, books, software, and other published works now reside in the digital repositories provided by the institution, and members of the institution now have easy access to these digital resources.

It would be expected that the growth and abundance these systems would lead to efforts directed towards the interoperability between the systems. Education and learning tools equipped with direct access to digital repositories would result in more powerful and comprehensive systems. Digital repositories would also see an increase in usage if it can be accessed through other systems. However, a comparatively small amount of time and resources have been spent in making these integrations happen.

The system developed in this thesis provides one such integration. This work provides a component that enables interoperation between two systems developed at MIT – SloanSpace, a course management system, and DSpace, a digital repository. The component allows SloanSpace users to search and retrieve DSpace documents from SloanSpace and submit SloanSpace documents into DSpace. Moreover, because SloanSpace and DSpace follow different file metadata standards, the component contains a mapping interface that transforms file metadata from one system into the file metadata of the other system. Testing the component with two scenarios show that searching and retrieving DSpace documents using the file transfer component cuts the time (i.e. the time it takes using current system without the file transfer component) by 57%. Similarly,
submitting SloanSpace files into DSpace using the file transfer component cuts the time by 44%.

2 Background

2.1 SloanSpace

SloanSpace [1] is an online management system for courses and learning communities that enables information to be shared within each class or community. Each community or course in SloanSpace has a community area web page that stores and displays community content. Access to this community area is given only to community members. Furthermore, different types of access can be given to the members. These access types determine what types of actions members can perform in the respective community area.

Currently, all MIT Sloan School classes use SloanSpace to store and display class content. A typical class area in SloanSpace contains such content as class documents, a class calendar and syllabus, class news, and a class forum. Professors, teaching assistants, and administrators for that class are given a professor-type access to the class community area, which allows them to add and modify the displayed content. Students are typically given a student-type access, which restricts them from viewing or modifying certain content in the class area.

SloanSpace is also being used by various online communities at MIT. Examples of such communities are student groups and research groups. Through SloanSpace, members of the groups can communicate with each other online via the community forums. SloanSpace also enables them to share their files securely.

Each community area has an associated file storage area page, which displays files and related to that class or community as well as operations to the file. A link is also available for each file, which, when clicked, will take the user to the respective file area.
The file area contains links to perform operations on the file, such as editing the file or deleting the file. Operations on files and the file storage area can be restricted so as only to prohibit certain members from performing certain actions. For instance, only members of type professor or teaching assistant may modify or add files to the area. Either files or URLs can be added to the file storage area. Directories may also be added to organize the files.

SloanSpace is organized into packages. A package represents a single component or service. For example, the file storage package is the package associated with the file storage area in the community area where users can add and manage community files. The calendar package is the package associated with the calendar for the community. Each package comprises of the user interface files for that component, the library files containing processes, or operations, related to the component, and database files containing database table definitions and functions for the component. Below is a screenshot of main SloanSpace class page for the “Intro to CS” class.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>07:00 AM</td>
<td>08:00 AM</td>
</tr>
<tr>
<td>08:00 AM</td>
<td>09:00 AM</td>
</tr>
<tr>
<td>09:00 AM</td>
<td>10:00 AM</td>
</tr>
<tr>
<td>10:00 AM</td>
<td>11:00 AM</td>
</tr>
<tr>
<td>11:00 AM</td>
<td>12:00 PM</td>
</tr>
<tr>
<td>12:00 PM</td>
<td>01:00 PM</td>
</tr>
<tr>
<td>01:00 PM</td>
<td>02:00 PM</td>
</tr>
<tr>
<td>02:00 PM</td>
<td>03:00 PM</td>
</tr>
<tr>
<td>03:00 PM</td>
<td>04:00 PM</td>
</tr>
<tr>
<td>04:00 PM</td>
<td>05:00 PM</td>
</tr>
<tr>
<td>05:00 PM</td>
<td>06:00 PM</td>
</tr>
<tr>
<td>06:00 PM</td>
<td>07:00 PM</td>
</tr>
<tr>
<td>07:00 PM</td>
<td>08:00 PM</td>
</tr>
<tr>
<td>08:00 PM</td>
<td>09:00 PM</td>
</tr>
<tr>
<td>09:00 PM</td>
<td>10:00 PM</td>
</tr>
</tbody>
</table>

Figure 2-1: SloanSpace screenshot
The calendar package is responsible for the calendar component shown in the screenshot. Similarly, the forums package contains all the code files and user interface files that handle the forums component. The file storage area is reachable by clicking the “File Storage” tab in the top of the page. Again, this file storage area is handled by the file storage package.

SloanSpace is an implementation of .LRN [3], an open-source course application suite for online course management systems and learning communities. .LRN is based on the OpenACS framework [4], a toolkit used for building online community-oriented web applications. OpenACS, and SloanSpace, in turn, are implemented in Tcl.

2.2 DSpace

DSpace [2] is a digital repository that provides long-term storage for all types of digital content developed at MIT. Examples of content stored currently in DSpace are papers, theses, books, preprints, images, simulations, computer programs, and multimedia publications.

Content in DSpace is organized by communities and collections. All items belong to a specific collection, and all collections belong to a community. In addition, each item contains two types of data – the metadata, which describes the item, and the item content, stored as bitstreams. The item metadata is based on the Dublin Core metadata standard [5].

Access to the content stored in DSpace can be done via the DSpace web interface. Through this web user interface, users can browse or search for DSpace content. Users can also submit content into DSpace via this interface. The submission process consists of two tasks: the user must first enter the content description (or metadata) and then upload the file into DSpace. Users must also specify which collection to store the item in. Access to some of the collections and to the submission interface is restricted to authorized users.
DSpace also provides a web service to enable communication with other systems. Methods implemented in the web service include a search and browse function, an ingest function, and a deposit function. The search/browse function, which allows users to search and browse DSpace content, is based on SRW (Search/Retrieve Web Service). Through SRW, a user may enter a search or browse query via a URL, and will be returned an XML document containing results. For example, the URL query for a search for “math” returning the 1st result is:

http://dspace-demo.mit.edu:8080/SRW/search/DSpace?query=math&maximumRecords=1 &startRecord=1

When this URL is entered, the DSpace SRW service returns the XML document containing the search result. Below is part of that XML document:

```xml
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="/SRW/searchRetrieveResponse.xsl"?>
<searchRetrieveResponse xmlns="http://www.loc.gov/zing/srw/">
  <version>1.1</version>
  <numberOfRecords>1</numberOfRecords>
  <resultSetId>fov7co</resultSetId>
  <resultSetIdleTime>300</resultSetIdleTime>
  <records>
    <record>
      <recordSchema>default</recordSchema>
      <recordPacking/>
      <recordData><srwdc:dc xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:srwdc="info:srw/schema/1/dc-v1.1">
        <dc:contributor.author>Carroll, Lewis</dc:contributor.author>
        <dc:date.accessioned>2003-12-03T22:04:10Z</dc:date.accessioned>
        <dc:date.available>2003-12-03T22:04:10Z</dc:date.available>
        <dc:date.issued>2002-12-03T21:26:17Z</dc:date.issued> . . .
```

DSpace also provides a two SOAP based web service that allows users to submit and retrieve DSpace content. The ItemAccessService contains methods that allow users to retrieve DSpace files. Similarly, the ItemIngestService contains methods that allow users to deposit files into DSpace. For example, in order to retrieve a file from DSpace, the user calls the retrieveItem and retrieveBitstream SOAP methods of the ItemAccessService. The file id is given as an input to index the file. The retrieveItem request retrieves the file metadata associated with the file, while the retrieveBitstream request retrieves the file content encoded in a base64 string from DSpace.

2.3 Scenario 1: Populating a SloanSpace File Storage Area via DSpace

Say, for example, that Professor Smith, the professor for Physics 101, wants to populate the course’s SloanSpace file storage area. He feels that DSpace would be a good repository to search for such files. Prof. Smith can accomplish this task with the current system, but it would require him to interact explicitly with both SloanSpace and DSpace. In the following chapter, we'll see how this need to explicitly deal with both systems can be avoided.

Here is the current process Prof. Smith would go through in order to accomplish this task:

He first would first through DSpace via the DSpace web interface. The following screenshot shows the DSpace web user interface:
Once the search query "physics" is entered into the search textbox, as shown above, Prof. Smith clicks on the "Go" button to fetch the search results. The following figure is the screenshot of the page returned by DSpace, containing the search results for the query:
Prof. Smith then would browse through these results and choose whichever ones he feels is appropriate for the class. Once he has decided which files to add into the file storage area, he would then click on the link for that result. For example, suppose Prof. Smith decides that the first file listed above, “The conceptual structure of physics”, is a good candidate, he would then click on the file link to go to the page displaying the file information. The following is a page that displays the file information:
Suppose Prof. Smith simply wanted to add the URL of the file. He would then simply have to save the value for URI listed above, and then add this URL to the file storage area. Suppose however that he wanted to add the actual file into the file storage area. He would then need to save the file into his local computer, by either right clicking on the “View/Open” link above and choosing the “Save” option, or click on the link and then saving it into his computer from the menu bar. Once he saves it into his computer, he then logs on to SloanSpace and goes to the file storage area for the class. Here is the file storage area for Physics 101:
Finally, in order to upload the file into the file storage area, shown above, Prof. Smith would then click on the “Upload a file” link, and from there, proceed with uploading the file he saved in his computer from DSpace, into the file storage area. The “upload a file” screen is shown below:

Figure 2-6: SloanSpace file upload
Once Prof. Smith enters all the information and clicks on the Upload button, the file is then added into the file storage area.

2.4 Scenario 2: Submitting a SloanSpace file to DSpace

Suppose that Jane, a member of the Dolphins Research Group, wanted to submit a group research paper into DSpace. In order for all the members to be able to contribute to the paper, the file was uploaded in the Dolphins Research Group SloanSpace file storage area. Once the file was ready for submission, Jane and her group were ready to submit the paper into DSpace. Here is the submission process Jane would currently go through in order to accomplish this task:

First, Jane would download the file from the file storage area into her computer. She would then go to the DSpace web submission interface, and enter all the metadata information for the file. A screenshot of one of several pages for the DSpace web submission interface is shown in the following figure:
There are 8 screens Jane would need to go through in order to complete the submission. The “Describe” tab, highlighted in red in the screen above, shows where she is in the submission process. The screens query Jane for the file metadata. Examples of metadata are the title, the author, the type, and the language, as shown above.

2.5 What is the motivation behind this system?

The two scenarios described above shows content transfer between SloanSpace and DSpace. However, imagine if there was some component that would simply enable users to perform these content transfers without having to switch between both environments. Even more so, the component would make use of the file information already stored in the system, and use that information when performing the content transfer, instead of
having the user supply this information once again. This component is the system
developed in this thesis.

SloanSpace and DSpace are merely two of several systems at MIT developed to
incorporate technology into the learning environment. OpenCourseWare, which places
MIT course materials on the web for free, is another one of these systems. Efforts to
integrate OpenCourseWare with DSpace are also being made. The vision for the future is
that all of these different learning environments can interoperate with each other, thus
building a very comprehensive environment for the users. This system in this thesis is
the first of such integrations.

Most importantly, however, the system developed in this system demonstrates more
generally interoperability with digital repositories. Although more and more digital
repositories are being developed, a relatively small effort has been made to integrate
other systems with these repositories. The system developed in this thesis not only
integrates SloanSpace with DSpace, but also allows for a more a general integration with
any other system.

2.6 Challenges

The main problem to be solved in this thesis deals with the metadata handling. The files
from DSpace and SloanSpace have metadata associated with them, but the specific
metadata stored in the SloanSpace files and DSpace files is different. The metadata for
DSpace files is based on the Dublin Core metadata standard. The metadata standard
specifies elements for the metadata, such as the title, the author, the publication date, the
type, and the publisher, among others. The SloanSpace files also have data associated
with it, such as the title, the user who uploaded the file into SloanSpace, the date the file
was uploaded, the file size, and the file type. Since the metadata specifications for both
systems are different, the file metadata for files transferred from one system to the other
must be adjusted to map to the file metadata specifications for the other system. For
instance, a file coming in from DSpace contains Dublin Core metadata. In order to add
the file to SloanSpace, the file must contain SloanSpace specific metadata. Thus the
system must contain a mapping module that maps the DSpace metadata to the SloanSpace data.

Another challenge in this thesis is developing a user interface that will make it easier for users to transfer files between the two systems. Files being transferred already contain metadata from the system they are coming from. Thus the file submission process to the new system must be simpler than the current submission process for that system. For instance, as shown in Scenario 2, in order to submit files into DSpace (via the DSpace web interface), users enter file metadata through a series of screens. Since the Dublin Core metadata standard contains a significant number of elements, the process can be lengthy. In the file transfer interface developed in this thesis, submitting a file into DSpace from SloanSpace should be a faster and simpler process since the file being submitted already contains metadata from SloanSpace. In other words, a user should not have to enter the DSpace metadata that maps directly to the SloanSpace metadata. Thus the user interface must pre-populate DSpace metadata from the SloanSpace metadata using information from the metadata mapping module.

Finally, the design of the system should be more general to include integration with other repositories, and not specific solely for integration with DSpace. For example, the metadata mapping module must also be able to map SloanSpace metadata with metadata of any other system or repository.

2.7 Related Work

Awareness of the need for integration between systems and digital repositories has been growing over the past few years. For example, an effort has been made by IMS and OKI to develop standards of integration among education systems and repositories. The work done by these two organizations focus mainly on developing the specifications for interoperability between systems and repositories, whereas the system developed in this thesis is an actual implementation of a system that provides this interoperability.
The IMS Digital Repositories Interoperability (DRI) Specification provides specifications for digital repository interoperation of common repository functions. It specifies five core interactions between systems and repositories. These five interactions include search/expose, gather/expose, request/deliver, submit/store, and alert/expose [9]. The search/expose interaction defines the process in which systems search metadata exposed by content repositories. The gather/expose interaction defines the process in which systems request metadata that is exposed by the repository. The request/deliver interaction involves the process in which a system requests access to the learning object exposed through the search operation. The submit/store interaction defines the process in which a system submits content to the repository. This interaction refers to the IMS Content Packaging Specification as a standard on how to package and export the content. Lastly, the alert/expose specification defines the process in which repositories alert systems on new or updated metadata or resources.

The Open Service Interface Definitions (OSIDs) developed by the Open Knowledge Initiative (OKI) provides specifications for integration in an education technology environment. These specifications describe how components of education technology systems interact with one another. OSIDs provide a layer of abstraction between the client application and the service application [10]. Implementation details of the service application need not be known by the client application in order for the client to interoperate with the service application. Similarly, details of the client application are hidden from the service application. OSIDs simply specify what is needed from the service and what is expected out of the client. Repository OSIDs are OSIDs developed for interoperability between digital repositories and other components. With repository OSIDs, clients don’t need to know the implementation details of each particular repository, and instead simply provide data the OSID specifies is expected out of the client [11]. The repositories on the other hand would provide data that the OSIDs have specified for them to provide to the client.
3 System Overview

The file transfer component described in this thesis integrates SloanSpace with DSpace, as illustrated in the figure below.

The component sits in SloanSpace, and interacts with DSpace, directly through two of DSpace’s web services – the SOAP web service and SRW (Search/Retrieval Web Service). Three types of operations can be performed by this component. The search operation searches DSpace content through SRW. The retrieve operation gets files from DSpace and places them in the proper SloanSpace area. The submit operation submits files from SloanSpace to DSpace. Both the retrieve and submit operations are performed via the DSpace SOAP web service.

The file transfer component contains a storage element, which consists of database tables used to store both the metadata mappings and the specific file metadata. These database
tables are created in the SloanSpace database, and can therefore reference other tables in
the database.

This chapter first shows how the system functions. In particular, it shows the how the
system behaves in the two scenarios described in the previous chapter. Then it describes
the data model of the system, on which the database tables are based. Finally, it
describes the implementation details of the system.

3.1 System Functions

3.1.1 Search/Retrieve

The search and retrieve interface allows SloanSpace users to search and retrieve files
from DSpace, without leaving the SloanSpace environment.

Recall the first scenario described in the previous chapter, where Prof. Smith wants to
populate his Physics 101 SloanSpace file storage area. Although he was able to
accomplish his task, he had to leave the SloanSpace environment and go to DSpace to
search the files. Then he had to save the file in his own computer, after which he could
then finally upload the file to the SloanSpace file storage area.

The search and retrieve interface developed in the system in this thesis makes Prof.
Smith’s job much easier and speeds up the process, by enabling Prof. Smith to search and
retrieve DSpace files, while never leaving the SloanSpace environment. Here now is the
process Prof. Smith would go through in the same scenario, but using the system in this
thesis:

He would first go to the file storage area. Here is the screenshot of the file storage area in
this system:
In this system, the file storage area now contains a “Search for Files” link, shown above. To proceed with the DSpace file search, Prof. Smith would now click on this link. When he clicks on the link, he is directed to the first page of the search interface, which is the page in which Prof. Smith can enter the search query. As described in the previous chapter, Prof. Smith then enters “physics” as the search query. Below is the search interface page, with the query “physics” typed in the text box for the query:
As shown in the page above, Prof. Smith also has an option of searching Google and a restricted version of Google where it would only search through DSpace URLs, indicated above by the “Google” and “DSpace-restricted Google” radio buttons respectively. An example of search using these different domains will be shown later in this section.

Once Prof. Smith has entered the query into the text box, as shown above, he would now click on the “Search” button to get the search results. Below is the search results page returned for the query “physics”:

![Search Query Page](image-url)
Each result in the search results page for a DSpace search displays the result title, the result description, and the URL of the document, as shown above. In addition, the result contains two links displayed to the right of the URL – the “Add URL” and the “Add File” link. If Prof. Smith wanted to add the URL of the file to the file storage area, he would click on the “Add URL” link. If he instead wanted to add the file itself in the file storage area, then he would click on the “Add File” link. Suppose Prof. Smith wanted to add the first document, “The conceptual structure of physics” to the file storage area, he then clicks on the “Add File” link for this result. After he clicks on this, he is then redirected back to the file storage area, which now contains the added file. Here is a screenshot of the newly updated file storage area:
A design issue arose of whether or not to throw away the extra metadata. That is, the file coming from DSpace contained other metadata values that SloanSpace does not need. Although throwing away the extra metadata allows for simplicity and does not require the addition of extra storage space, the metadata would be useful when the system is extended to allow for integrations with other systems, since these other systems may use the extra metadata. Thus for this system, extensibility was chosen over simplicity.

Using the system in this thesis, Prof. Smith then did not have to leave SloanSpace to search DSpace. Furthermore, he did not have to first save the file into his local computer. Most importantly, however, Prof. Smith did not need to enter all the file information, as he did when using the current system. Recall that when using the current system, Prof. Smith had to upload the file manually to SloanSpace, which required him to fill out the SloanSpace file information. In particular, he had to fill out the title, description, and file location. Using the file transfer component in this thesis, Prof. Smith did not need to fill this out. Instead, the retrieve interface mapped the DSpace metadata values of the file to
the SloanSpace metadata values, and automatically filled out this information, thus speeding up the file transfer process.

In addition to searching through the DSpace domain, users can also search through Google. For instance, suppose Prof. Smith was not satisfied with the search results returned by DSpace. He can then search through Google by the following process: He first goes to the file storage area, as he did before, and clicks on the “Search for Files” link. Now, instead of selecting the “DSpace” button in the search query page as he did in the previous scenario, he now selects the “Google” button. This is shown below:

![Google search](image)

Figure 3-5: Google search

He then clicks on the “Search” button as he did before to get the search results. Here in the following figure is the search results page for the “Google” search for “physics”:
Note that the “Add File” link is not available for the Google page, since the results returned are web sites instead of web documents returned in the DSpace search. Thus, only the URL’s of the results can be added to the file storage area.

3.1.2 Submit

Through the file transfer component, SloanSpace users would be able to submit files in their file storage area to DSpace, while never leaving the SloanSpace environment.
Recall the second scenario described in the previous chapter, where Jane wanted to submit a group research paper into DSpace. In order to do this using the current system, Jane had to go to the DSpace submission web user interface, which, through a series of screens, queried her for the file metadata.

The submit interface in the file transfer component developed in this thesis makes the process in this scenario easier and faster by accomplishing two things. First, using the file transfer component, Jane no longer has to leave the SloanSpace environment to submit files into DSpace. And second, the submit interface pre-populates the entries for the DSpace file metadata values by mapping the SloanSpace metadata values to the corresponding DSpace metadata values. Thus Jane will no longer have to fill out values for file metadata entries that SloanSpace already maintains. Here now is the process Jane would go through for the second scenario described in the previous chapter, but this time using the submit interface developed in this thesis:

First, Jane would go to the file area for the file that she wants to submit to DSpace. Here is the file area for the file:

![File area screenshot](image)

Figure 3-7: File area
To proceed with the DSpace submission, Jane would click on the “Upload to DSpace” link, shown above as one of the actions for the file. Clicking on this directs Jane to the page that contains the form that queries for the file DSpace metadata values. Some of the metadata values are filled out, depending if there exists a mapping from a SloanSpace file metadata field to the respective DSpace field. Here are two screenshots of this page. The first screenshot shows the top of the page. The second screenshot shows the page when scrolled to the bottom.

Figure 3-8: Metadata entry
In the page above, all the fields and all their display information, including the field label, the display text above the field input element, the input element (i.e. the text box, select list, or text area associated with the field), and the “Add More” button (for fields that can have multiple values) are all dynamically generated. Moreover, the author, title, publication date, and description values were pre-populated. After Jane finishes filling out the rest of the values, she then clicks on the “Upload to DSpace” button at the bottom of page, which would finally submit the file to DSpace. Note that Jane also has an option.
of saving the current metadata entries, and come back to the submission process later, on canceling the process, by clicking the “Save” or “Cancel” buttons respectively.

After the submission is made, the submit interface returns a page the status of the submission. That is, it shows whether or not the submission was successful. The success of the submission depends on the values Jane submitted. For example, if the value for the field is required for submission, but Jane has failed to fill it out, then she will be directed back to the pre-populated entry page with a message for field that was unsuccessfully fill out. Here is an example of that returned page, when Jane did not fill out the required title field. Note the red error text, “Please enter a title”, next to the title field:

![Metadata entry error text](image)

**Figure 3-10: Metadata entry error text**
Finally, once all the field values are successfully entered, the interface submits the file to DSpace. Here is the page returned to Jane indicating that the submission was successful:

![Successful submission](image1)

**Figure 3-11: Successful submission**

In order to indicate that the file has already been submitted to DSpace, a new icon is associated with the file. The new icon is similar to the old file icon, except that it shows a “D” beside it. Here is a screenshot of the updated file storage area:

![Updated file storage area](image2)

**Figure 3-12: Updated file storage area**

The icon for the “Dolphins Research Paper” shows a “D” next to the file icon. In addition, the submit interface won’t allow the file to be submitted to DSpace. The
interface simply returns a page telling the user that the file has already been submitted to DSpace.

Using the file transfer component, Jane no longer has to leave the SloanSpace environment. More importantly, Jane doesn’t need to fill in some of the metadata fields, whose values can be mapped to SloanSpace file values, making the submission process faster and easier.

3.2 Data Model

In order to explain the design choices for the data model, this section first takes a closer look at the two scenarios and explores the types of data needed to be managed and stored in order to perform the specific functionalities. Then, the section explains how this data was modified to allow for generality. That is, the section describes how the data was structured to not only contain information specific to DSpace, but also to contain information for other remote systems that wish to integrate with SloanSpace as well.

3.2.1 Exploring the Data Model Requirements

Recall the second scenario where Jane wants to submit a paper into DSpace. In particular, recall what happens once Jane clicks on the “Upload to DSpace” link in the file area. Illustrated below is a summary of this process:
As seen in the illustration, the storage element contains the information needed to generate the data in the pre-populated page. In particular, recall the two types of data dynamically generated in the pre-populated page. The first is the metadata fields, and the information associated with those fields. This information includes the display label and text for the field, the input type, and flags indicating whether the field value is required upon submission and whether or not the field can have multiple values. The second type of data dynamically generated is the pre-populated values for the field, where the values are the SloanSpace field values that map to the respective DSpace value. Thus the storage element needed to store data containing the DSpace metadata fields and their information, including the display information, the multiple and required flags, and the SloanSpace – DSpace mapping information. In addition, once the file was submitted to DSpace, the submit interface tagged the file so as to indicate that the file was already submitted. In order to do this, the storage element then needed to store a record of the files that were submitted to DSpace.

Now, recall the first scenario where Prof. Smith wants to search and retrieve files from DSpace into the file storage area for his class. In particular, recall what happens once Prof. Smith has clicked on the “Add File” link associated with a particular search result. Illustrated below is a summary of this process:

Figure 3-14: Retrieve process summary
The illustration above shows the need for the file transfer component’s storage element to additionally store two types of information. The first is the DSpace-SloanSpace mapping, shown in the second step of the process. The second is the DSpace metadata for the file being retrieved from DSpace, shown in the fourth step of the process.

In summary, the scenarios described above require the storage element to store the following information. First, it needs to store the DSpace metadata fields and the information associated with them. This information includes the display information, the multiple and required flags, and the mapping information. Mapping information in both directions, that is, SloanSpace field to DSpace field and DSpace field to SloanSpace field needs to be stored since the submit interface uses the first type of mapping mentioned, and the retrieve interface uses the second. Second, it needs to store a record of the files being submitted to DSpace. And third, it needs to store the DSpace metadata information for files retrieved from DSpace.

3.2.2 Generalizing the Data Model

The most important design decision for the data model was to structure the data in a way that would easily allow other remote systems to integrate with SloanSpace. In order to accomplish this task, the data stored needed to be generalized for any remote system, not just DSpace, but still meet the data model requirements outlined in the previous section. Thus, the data model requirements were modified as follows. First, instead of simply storing the DSpace metadata fields and the information associated with them, the data model was modified to now store any type of metadata field, from any system. This data type is called metadata_fields. However, since each field in metadata_fields can now belong to any system, the fields then needed to contain an extra property indicating which metadata schema (or system) it belongs to. Second, instead of simply storing a record of the files submitted to DSpace, the data model now stores records of files submitted to any remote system. This data type is called metadata_submissions. Like the metadata_fields data type, each record needed to contain an extra property indicating which metadata schema (or system) the file was submitted to. Finally, instead of simply
storing the DSpace metadata information for files retrieved from DSpace, the modified
data model stores any remote system’s metadata information for files retrieved from that
system. This data is called **metadata_field_values**.

In addition, three other data types needed to be created to complete the generalized data
model. First, as described above, both the **metadata_fields** and **metadata_submissions**
contain an extra property that indicates which remote system the data belongs to. In
order to do this, an extra data type was created that stores all the remote systems
integrating with SloanSpace. This of course includes DSpace as one of its records. Let
us call this new data type, the **metadata_schemas**. Second, the SloanSpace metadata
exists in several tables. For instance, the file title exists in one SloanSpace table, while
the file creator exists in another. Thus, the field mapping in the **metadata_fields** can’t
simply list the SloanSpace metadata field name. Instead, a new data type was created to
solve this problem, where each record contains the SloanSpace metadata field name and
the SloanSpace table and column that contains the value for that field. Let us call this
new data type, the **ss_metadata_fields**. Lastly, when other systems, along with DSpace,
are integrated with SloanSpace in the future, the metadata values of the retrieved files
coming from a remote system can be used in the pre-population step of the submission
process into another system. In order to be able to do this, a new data type was created
that stored mapping information between the fields of remote systems. This data type is
called **metadata_mappings**.

Thus, in summary, six data types (database tables) were created:

1. **metadata_schemas** – stores information about the different metadata systems
   integrated with SloanSpace.
2. **metadata_fields** – stores the information about the metadata fields.
3. **ss_metadata_fields** – stores information about the SloanSpace file metadata. More
   specifically, it gives the table and column locations of the metadata values.
4. **metadata_field_values** – stores the remote system metadata information for the
   retrieved files
5. **metadata_submissions** – stores the records of the file submissions to remote systems.
6. **metadata_mappings** – stores mapping information between the metadata of remote systems.

SloanSpace, and in turn, the file transfer component, uses Oracle for its relational database.

### 3.3 System Implementation Details

#### 3.3.1 Search Interface

The main design decision made in developing the search user interface was to allow for generality. That is, not only should the search interface enable users to search through DSpace, but the design of the interface should also allow the search interfaces of other remote systems to be easily built and integrated with the current search interface.

In the first scenario described in section 3.1, Prof. Smith has an option of searching through both the DSpace and the Google domain. The search query page in which Prof. Smith entered the query contained radio buttons indicating which domain to search through. When Prof. Smith clicked on the search button, the interface then searched through the proper domain, and returned the respective results. The only difference between processes of searching DSpace and Google was in the step that fetches the search results from the given domain, and the parsing of those results. The use of ACS service contracts allowed for this task of developing a more generalized search interface.

ACS service contracts is a package available in OpenACS, and, in turn, is available in SloanSpace. Service contracts provide a way to develop interfaces or contracts, which can then be implemented by other packages. The contracts specify operations that implementers are required to fill.

The search service contract contained a paged search operation, paged_search. This operation takes as input a query string, a page number, and the number of results per page. The output of this operation is the search results, indexed on the page number. The
number of search results returned is the number of results per page indicated in the input. For instance, if the page number is 2 and the number of results per page is 5, then the operation will return the 6th through the 10th search results. Each search result is an array of three strings. The first value in the array is the string value for the title of the search result document. The second value is the URL of the document. The third value is the string containing the search result document description.

The DSpace search interface implements the search service contract, and therefore contains a method that fulfills the contract requirements of the paged_search operation. As the paged_search operation specifies, the DSpace paged_search operation takes as input a query string, a page number, and the number of results per page. This method then searches DSpace content via the DSpace SRW web service. The SRW web service allows remote systems to search through DSpace, through its SRU (Search and Retrieve URL service) service. Through SRU, remote systems can formulate search requests to DSpace via a URL. The query URL consists of two parts, separated by a “?” symbol. The first part specifies the SRW server location, and the second part specifies the query string and other query options or elements, where each query option is separated by a “&” symbol. Each search option contains the option tag followed by an “=” sign which is followed by the option value. For example, using the first scenario in section 3.1, Prof. Smith’s query to search for content in DSpace containing the word “physics”, starting with the 3rd search result, and returning a maximum of 5 results would be:

http://dspace-demo.mit.edu:8080/SRW/search/DSpace?query=physics&maximumRecords=5&startRecord=3

The DSpace paged_search method makes the search request to DSpace by calling this URL. The response returned by the SRW service is an XML document which contains the search results and the Dublin Core metadata for the result. The method then parses the XML document, using the XML parsing processes of the TCL Tdom package, in order to obtain the Dublin Core metadata values for the result document’s title, description, and URL.
In order to further demonstrate the usability of the search service contract, a Google search interface was also developed. The Google paged_search method implements the paged_search operation of the service contract. This method searches Google content through Google’s SOAP-based web service. In particular, it calls the doGoogleSearch SOAP request, and is returned a SOAP response, which is then parsed using the SOAP methods of the TclSOAP package. The SOAP response returned by the Google web service contains the title, the description, and the URL of the search results.

Thus, once Prof. Smith has clicked the “Search” button in the search query page, the search interface calls the search service contract paged_search operation for specific implementer, depending on which domain radio button was selected in the search query page. Finally, the search results page displays the results of the paged_search operation.

### 3.3.2 Retrieve Interface

The “Create URL” method in SloanSpace simply takes in a title, description, and URL, and adds that to the file storage area. Thus, once the “Add URL” is clicked, the retrieve interface simply needs to call the “Create URL” method using the title, description, and URL values returned in the search result.

The “Add File” interface on the other hand, can’t simply take the values returned in the search result, since it actually needs to fetch the contents of the file and the extra file metadata. In order to do this, the interface makes the SOAP requests to the DSpace ItemAccessService SOAP-based web service. The ItemAccessService contains the SOAP requests “retrieveltem” and “retrieveBitstream”. The “retrieveltem” request asks the service to return the Dublin Core metadata for the file, encoded in XML. The “retrieveBitstream” request, on the other hand, asks the service to return the bitstream content of the file. In order to know which file contents to return, both methods require the SOAP service client to supply the file id, which is the file URL returned by the SRW service.
Using the "retrieveItem" and "retrieveBitstream" SOAP requests, the "Add File" component then works as follows:

```
process add_file (file_id) {
    1. titlefields = lookup metadata fields in metadata fields table that SloanSpace title field maps to
    2. descfields = lookup metadata fields in metadata fields table that SloanSpace description field maps to
    3. Initialize titlefieldsvalue, descfieldsvalue
    4. xml_doc = retrieveItem(file_id)
    5. namevalue-array = xml_parse(xml_doc)
        a. xml_parse also sets titlefieldsvalue and descfieldsvalue
    6. bitstream = retrieveBitstream(file_id)
    7. temp = create_file(bitstream)
    8. upload_file(titlefieldsvalue, descfieldsvalue, temp.loc)
    9. add values in namevalue-array to metadatafieldvalues table
}
```

The first step of the process looks in the "metadata fields" table for the Dublin Core metadata elements to which the SloanSpace title field maps to, and stores this list of elements in an array. The second step does the same for the SloanSpace description field. The third step initializes the variables titlefieldsvalue and descfieldsvalue, which will contain the values that map to the SloanSpace title field and the SloanSpace description field, respectively. Once these arrays are set and the variables are initialized, step 4 then calls the "retrieveItem" SOAP request, which returns an XML file. Step 5 parses this XML file to get all of the file’s Dublin Core metadata element name and value pairs. During the XML parsing, if the metadata element being read is in the titlefields or descfields arrays, then the value of this element is concatenated to the current titlefieldsvalue or descfieldsvalue, respectively. Thus, after step 5, the namevalue-array contains all the file’s Dublin Core metadata name-value pairs, while titlefieldsvalue contains the value for the file’s SloanSpace title and descfieldsvalue contains the value for the file’s SloanSpace description. Step 6 then calls the "retrieveBitstream" request, which fetches the bitstream content of the file. Step 7 saves
this bitstream content into a temporary file. Step 8 then calls the same file upload process used when a SloanSpace user manually uploads a file into the file storage area, which takes in the SloanSpace file title, the SloanSpace file description, and the file location. Step 8 calls this process using title_fields_value, desc_fields_value, and the temporary file location as the input. Finally, step 9 adds the file metadata values from the name_value_array into the "metadata field values" table.

### 3.3.3 Submit Interface

The most important process of the submit interface is dynamic generation of the pre-population page, reached when a user clicks on the "Upload to DSpace" link the file area, as shown in the second scenario in section 3.1. Recall from the second scenario, the generated pre-populated page when Jane clicked on the "Upload to DSpace" link. A portion of the screenshot for this page can be seen below:

![Screenshot of the Submit Interface](image)

The dynamic generation for this page works as follows: First, the submit interface looks up the **metadata_fields** table for all the fields and their corresponding field information. For each field, the submit interface then generates an entry for the field, containing the
field label, the display text, an input element, and an optional “Add More” button, depending if the field’s multiple flag is true. Finally, the submit interface looks up the SloanSpace – DSpace mapping of the field. If a mapping exists, then the interface looks up the ss_metadata_fields table for the SloanSpace table and column name containing the SloanSpace mapped value for this field. It then calls the appropriate database query to fetch this value, and pre-populates the field input element with this value. As shown above, the author, title, and publication date are already pre-populated by the submit interface.

Once the metadata values have been filled and the user has clicked on the “Upload to DSpace” button, the submit interface generates the SOAP request to the DSpace ItemIngestService SOAP-based web service. Three ItemIngestService SOAP requests must be called in a particular order to submit the file into DSpace. The first SOAP request is “depositItem”, which deposits the file metadata, encoded in XML. Thus before this SOAP request is made, the interface first encodes the metadata into XML. After “depositItem” is called, the interface must then make the “depositBitstream” SOAP request. This submits the file contents to DSpace, encoded as a base64 string. Finally, the interface calls “depositComplete” which finalizes the file submission process.

If the “depositComplete” request returns true, indicating the success of the DSpace file submission, the interface adds the file to the “metadata submissions” table, and then redirects the user to a page displaying a message indicating the success of the file submission.

4 Integrating the File Transfer Component with Other Systems

The process for integrating another system with the file transfer component consists of several steps. First, the file transfer component’s database tables need to be filled with the metadata information of the system being integrated. Then, the code that communicates with the web service, both to submit and retrieve files from the system
remotely, must be provided. Lastly, a search service contract implementation that
searches through the system’s domain must be implemented.

To demonstrate the process of integrating a system with the file transfer component, this
chapter will show the steps taken to integrate a sample repository called “JJ Digital
Repository”.

4.1 Filling in the Tables Using the Add Schema Interface

The first step in making the integrating involves filling in the file transfer component’s
database tables with the metadata information and metadata mappings of the system
being integrated. In order to simplify this process, an “Add Schema” user interface was
developed. This interface queries the user for the metadata information of the system
being integrated then adds the information to the database.

For instance, suppose Jane Jacobs now wanted integrate the “JJ Digital Repository” with
SloanSpace. Say that the metadata for the files in her system contain three fields each –
author, description, and language, where the language can only be either English or
Spanish. Furthermore, say that author was a required field, and that it could contain
multiple values. In other words, the file can have multiple authors. The process would
then proceed as follows:

First, Jane goes to the main “Add Schema” page, and enters the schema name and the
number of metadata fields of the schema. This page is shown in the following
screenshot:
Then, Jane clicks on the “Add Schema” button to proceed to the field addition page. Here, Jane fills out the appropriate field information. The field information consists of the field name, the SloanSpace mapping, the mapping type, the required field flag, the multiple values flag, the display name, and the display type, as shown in the following screenshot:

![Figure 4-1: Adding a schema](image)

---

46
Once Jane has filled out the field information, she then clicks on the “Submit and Continue” button. Clicking on this button will then direct Jane to the next page, which queries her for the field display information which is used in the submit interface’s pre-populated metadata field query page. Here in the following screenshot is the field display information query page, with the appropriate information filled out:
Finally, Jane clicks on the “Submit” button to finish the process. The following screenshot shows the page that indicates the success of the submission:
This process creates the data necessary for the integration. Through this process, the submit interface can now dynamically generate the metadata entry page, using the fields in this schema. To demonstrate this, here is a demo page that will generate the metadata entry page for a file.

Figure 4-5: View new schema metadata

This page, created simply for the purpose of demonstrating the "Add Schema" functionality, allows a user to choose which system to upload the file to. To show that
the “Add Schema” worked for the JJ Digital Repository Jane created, the JJ Digital Repository option is selected. Once the “Upload to Schema” button is now clicked, the dynamically generated metadata entry page for a JJ Digital Repository file submission is displayed. Here below is a screenshot of that page:

![Screenshot of metadata entry page](image)

**Figure 4-6: New schema metadata page**

Note that the author field was pre-populated with the SloanSpace author field, which Jane indicated during the “Add Schema” process. In addition, note that the author entry input is a text box, while the description input is a text area, and the language entry input is a select list. Furthermore, the author entry contains an “Add More” button since this field was specified to allow multiple values.
4.2 Providing the Code for the Submit and Retrieve Interfaces

Once Jane has added the metadata schema and fields to the database, she must now provide the code that communicates with the “JJ Digital Repository” web service. This code makes the necessary calls to the web service in order to submit the file into the repository. For example, say that the web service contains the SOAP method “submitToJJ” that takes in the base64 encoded content and the values for the metadata field. Jane would then need to create a file with the code that does the following:

- create SOAP request for “submitToJJ”
- encode the file content into base64
- call the SOAP request with the base 64 content and the metadata values

Then, Jane would need to specify, in meta-submit.tcl (which can be found in the appendix), to redirect to this file when the “Upload” button is clicked and the schema id equals the schema id for the “JJ Digital Repository” schema.

Similarly, Jane would need to add the code that retrieves a file from the repository, via the web service. For instance, say that the web service contains two SOAP methods “retrieveMetadataFromJJ” and “retrieveContentFromJJ”, that both take in a file id. The “retrieveMetadataFromJJ” method returns the file metadata and the “retrieveContentFromJJ” returns the base 64 encoded file content. Jane would then need to create a file with the code that does the following:

- create SOAP request for “retrieveContentFromJJ”
- call the SOAP request for “retrieveContentFromJJ” with the file id
- save the content into a temporary file
- create SOAP request for “retrieveMetadataFromJJ”
- call the SOAP request for “retrieveMetadataFromJJ” with the file id
- add the temporary file into the file storage area, and use the title and description fields returned by the “retrieveMetadataFromJJ” method

51
• add the metadata values returned by the “retrieveMetadataFromJJ” method into the
  metadata_field_values table

4.3 Implementing the Search Service Contract

The last component needed to complete the integration is the implementation of the
search service contract. Like both the DSpace and Google implementations, Jane needs
to create a “JJ Digital Repository” implementation that contains a paged_search operation
that takes in a query string, a page number, and the number of results per page. The
operation then searches “JJ Digital Repository” content via the repository’s web service.
Finally, it returns the parses then returns the search results, returned by the web service.

5 System Testing and Analysis

This section describes the tests run to measure the effectiveness of the file transfer
component, and discusses the results of the tests.

5.1 Testing the File Transfer Component

To show the effectiveness of the search and retrieve interface, a test was conducted using
the first scenario. The test comprised of running the task in the first scenario, first
without using the file transfer component, and then using the file transfer component.
Recall that the task consisted of searching for the word “physics” in DSpace, and then
adding the search result entitled “The Conceptual Structure of Physics” into the file
storage area. The times it took to accomplish the task both without the file transfer
component with the file transfer component were recorded and compared.

To show the effectiveness of the submit interface, a similar test was conducted, this time
using the second scenario. Recall that the task of the second scenario consisted of
submitting a file that was already in SloanSpace into DSpace. Like the first test for the
search and retrieve interface, this test comprised of running the task in the second
scenario first without using the file transfer component, and then using the file transfer
component. The times it took to accomplish the task without the file transfer component and with the file transfer component were recorded and compared.

Note that the file transfer component tests were done on a development version of SloanSpace, and not the actual deployed version of SloanSpace. Furthermore, the tests were run against a development DSpace web service.

5.2 Results of the Tests

Below are summary of results obtained from three users: (Note that the time is recorded in minutes).

Here first are the results from the search and retrieve test:

<table>
<thead>
<tr>
<th>Search and Retrieve Test</th>
<th>Time w/o component</th>
<th>Time w/ component</th>
<th>(Time w/ component) / (Time w/o component)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3:16</td>
<td>1:25</td>
<td>.434</td>
</tr>
<tr>
<td></td>
<td>4:07</td>
<td>1:34</td>
<td>.381</td>
</tr>
<tr>
<td></td>
<td>2:15</td>
<td>1:03</td>
<td>.467</td>
</tr>
</tbody>
</table>

Table 5-1: Results for the search and retrieve test

Here now are the results from the submit test:

<table>
<thead>
<tr>
<th>Submit Test</th>
<th>Time w/o component</th>
<th>Time w/ component</th>
<th>(Time w/ component) / (Time w/o component)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4:50</td>
<td>2:43</td>
<td>.562</td>
</tr>
<tr>
<td></td>
<td>3:58</td>
<td>2:25</td>
<td>.609</td>
</tr>
<tr>
<td></td>
<td>4:40</td>
<td>2:26</td>
<td>.521</td>
</tr>
</tbody>
</table>

Table 5-2: Results of the submit test

53
The results for the search and retrieve test, shown in Table 1, show that using the file transfer component speeds up the task in the first scenario significantly. On average, ratio of the time to accomplish the task with the component to the time to accomplish the task without the component is .427:1. Thus, using the file transfer component cuts the time to accomplish the task without the component by 57%, which is a little more than half.

The results for the submit test, shown in Table 2, also show a significant decrease in time spent performing the task in the second scenario when using the current system without the file transfer component vs. using the file transfer component. On average, the ratio of the time to accomplish the task with the component to the time to accomplish the task without the component is .564:1. Thus, using the file transfer component cuts the time to accomplish the task without the component by 44%, which is a little less than half.

5.3 Discussion of the Test Results

The test results show that the search and retrieve interface significantly cuts down the time to search DSpace, and to place the search results into the respective file storage area. Several factors contribute to this improvement. First, the user performing the search need not leave the SloanSpace environment. So time is no longer spent switching between the two environments. Second, with the file transfer component, the user no longer has to save the file into the local computer. And third, the user no longer has to fill in the information for the uploaded file. Recall that when uploading a file into SloanSpace, the user has to fill out the title, description, and file location (in the local computer). When using the file transfer component, this information is filled in automatically. Thus a significant amount of time is saved.

The results also show that the submit interface significantly cuts down the time to submit a file into DSpace, if the file was already uploaded into SloanSpace. This is again due to several factors. First, as in the search and retrieve interface, the user performing the
submission need not leave the SloanSpace environment. And second, the file transfer component pre-populates the metadata so that users no longer have to fill in data that SloanSpace already keeps track of.

More importantly however, the results show the usefulness of interoperability between SloanSpace and DSpace, and more generally, the interoperability between systems and repositories. This project shows integrating SloanSpace with DSpace allows for faster file transfers between the two systems by using the data already stored by the systems, cutting down time to accomplish the tasks approximately by half.

6 Future Work

6.1 System Deployment

The file transfer component currently runs on a development version of SloanSpace, and communicates with a development version of the DSpace web services. The hope for the future is for this component to actually be deployed and used in the deployed version of SloanSpace.

However, in order for the file transfer component to be deployed, two major issues need to first be addressed. The first issue deals with authentication. Currently, users who wish to submit files into DSpace must be registered DSpace users. With the file transfer component however, any SloanSpace user can submit files into DSpace. DSpace has no way to authenticate the users who are submitting files to their system if the users submit files through the SloanSpace file transfer component. In order then for the system to be deployed and used, a component must be developed that allows DSpace to authenticate and authorize the user before performing any file transfer operations.

The second issue deals with which DSpace collection the files go to when submitted via the file transfer component. Currently, in DSpace, when a user wishes to submit a file, he first specifies which DSpace collection he wants to add the file in. The file transfer
component currently submits all files into a demo collection. An example solution to this problem would be for DSpace to create a collection specifically for files coming in from SloanSpace. Using this, the file transfer component would then only need the identification for this collection. Although this simplifies the process, it is not very flexible and not very organized, since papers from SloanSpace can be very varied, as they can come from different SloanSpace communities. Another potential solution to this problem would be for the file transfer component to provide the users with a list of DSpace collections, and have the user choose the collection he wishes to add the file to. The problem with this is that certain collections can be restricted, and so there must also be a way to know which collections can be accessed.

6.2 Integration with Other Systems

Future work can also be directed towards integrating more systems with SloanSpace. The architecture of the file transfer component allows this to be done easily, as shown in chapter 4. For example, a useful integration would be to integrate OpenCourseWare with DSpace. OpenCourseWare is a system that places MIT course materials on the web for free. The course materials are not the materials of the current semester, but the material of a past semester. Thus, in order to construct an OpenCourseWare page, it would simplify the process if material from SloanSpace could be transferred easily into OpenCourseWare. Integration could also be made with other digital repositories. This would enable users to search through more domains.
References


   http://www.imsglobal.org/digitalrepositories/driv1p0/imsdri_infov1p0.html#1263439

   http://www.okiproject.org/documents/About%20OKI.pdf

   http://www.okiproject.org/documents/OKIManagingComplexity_rel_1_0.pdf


A Database Tables

-- The table contains the metadata schemas used by the system.
-- Each row in the table contains the schema name and the table
-- name of the schema table. (The schema table is the table containing
-- information for that specific schema. For every schema added, a
-- new schema table is created.)

cREATE TABLE METADATASCHEMAS
    (SCHEMA_ID INT,
     SCHEMA_NAME VARCHAR(100)
    CONSTRAINT METADATASCHEMAS_TABLE_NAME_PK PRIMARY KEY,
     CONSTRAINT METADATASCHEMAS_NAME_NOT_NULL NOT NULL)

CREATE SEQUENCE SEQ_SCHEMA_ID START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER TRG_SCHEMA_INSERT
BEFORE INSERT ON METADATASCHEMAS
FOR EACH ROW
BEGIN
    IF :NEW.SCHEMA_ID IS NULL THEN
        SELECT SEQ_SCHEMA_ID.NEXTVAL INTO :NEW.SCHEMA_ID FROM DUAL;
    END IF;
END;
/

-- This table will contain the SloanSpace metadata. Each row contains
-- the SS metadata name, the original SloanSpace table or view,
-- and the column that the field is mapped to.

CREATE TABLE SLOANSPACE_FILEMETADATA
    (FIELD_NAME VARCHAR(50)
    CONSTRAINT SSFILENAME_PK PRIMARY KEY,
     MAPPING_TABLE_OR_VIEW VARCHAR(50),
     MAPPING_COL_NAME VARCHAR(50)
    CONSTRAINT SLOANSPACE_FILEMETADATA_FIELDNAME_PK PRIMARY KEY,
     CONSTRAINT SLOANSPACE_FILEMETADATA_MAPPIGNCOLNAME NULL
    )

-- This table contains the fields and the field information for the Dublin
-- Core metadata schema. It is specific to the Dublin Core schema. Each
-- row contains the field id, the field name, the SloanSpace metadata field
-- that the Dublin Core field is mapped to, and the mapping certainty value,
-- which specifies how certain the mapping is between the two fields.

CREATE TABLE METADATAFIELDS
    (FIELD_ID INT,
     FIELD_NAME VARCHAR(50)
    CONSTRAINT METADATAFIELDS_TABLE_NAME_PK PRIMARY KEY,
     PRETTY_NAME VARCHAR(100)
    CONSTRAINT METADATAFIELDSPRETTYNAME_NOT_NULL NOT NULL,
     SCHEMA_ID INT
    CONSTRAINT METADATAFIELDS_SCHEMA_FK REFERENCES METADATASCHEMAS
    NOT NULL,
     MAPPING_SS_FIELD VARCHAR(50)
    CONSTRAINT METADATAFIELDS_MAPPING_FK REFERENCES SLOANSPACE_FILEMETADATA
    --table or view name of column to which this field is
map_type
integer

constraint metadata_field_map_type_ck

check (map_type in (1,0,-1)),
--map type = 1 if from field to ss,
0
--if both to and from, and
-1 if from ss to field

display_text
varchar(200),
display_type
varchar(30),
-- type of input display, e.g. text, textarea, select,
-- radio, etc.
-- display_type will be inside <display_type> and
-- </display_type>
display_attributes
varchar(200),
-- attributes of the display, ex. for textarea rows=3
-- cols=50, etc.
-- ex. <display_type display_attributes></display_type>
display_elements
varchar2(4000),
-- elements of display (for select lists)
-- ex. <display_type display_attributes>display_elements</display_type>
error_text
varchar(500),
-- text displayed field is required, but left empty
required
char(1)
default 'f'

constraint metadata_fields_required.nn
not null

constraint metadata_fields_required_ck
check (required in ('t', 'f')),
-- indicates whether or not the field is required upon
-- submission
multiple
char(1)
default 'f'

constraint metadata_fields_multiple.nn
not null

constraint metadata_fields_multiple_ck
check (multiple in ('t', 'f'))

create sequence seq_field_id start with 1 increment by 1;

create or replace trigger trg_field_insert
before insert on metadata_fields
for each row
begin
if :new.fieldid is null then
select seq_field_id.nextval into :new.fieldid from dual;
end if;
end;
/

create table metadata_field_values
(file_id integer
constraint metadata_values_file_id_nn
not null,
field_id integer
constraint metadata_values_field_id_nn
not null,
field_value varchar2(4000),
schema_id integer
constraint metadata_values_schema_nn
not null);

--map type = 1 if from field_1 to field_2,
0
--if both to and from, and
-1 if from field 2
to field 1

create table metadata_schema_mappings
(field_id_1 integer
constraint metadata_mappings_fl_nn
not null,
schema_id_1 integer
constraint metadata_mappings_sl_nn
not null)
create table metadata_submissions
(
    file_id integer
        constraint metadata_subm_file_id_fk
            references cr_items
        constraint metadata_subm_file_id_nn
            not null,
    schema_id integer
        constraint metadata_subm_schema_fk
            references metadata_schemas
        constraint metadata_subm_schema_nn
            not null
);

-- VIEWS --
create or replace view metadata_file_view
as
select i.item_id as file_id,
     i.name,
     r.title as file_name,
     r.publish_date,
     r.description,
     r.content,
     r.content_length,
     r.mime_type
from cr_items i, cr_revisions r
where i.live_revision = r.revision_id;

create or replace view metadata_user_view
as
select o.object_id as filejid,
     p.first_names II ' ' II p.last_name as fullname
from acs_objects o, persons p
where o.creationuser = p.person_id;

-- DATA --

--insert the sloanspace file metadata
insert into sloanspace_file_metadata values ('author', 'metadata_user_view', 'full_name');
insert into sloanspace_file_metadata values ('title', 'metadata_file_view', 'name');
insert into sloanspace_file_metadata values ('publish_date', 'metadata_file_view', 'publish_date');
insert into sloanspace_file_metadata values ('description', 'metadata_file_view', 'description');
### B Critical Source Code

#### B.1 dspace-get.tcl

```tcl
ad_page_contract {
    Add File From DSpace

    @author Genevieve Cuevas (gtcuevas@mit.edu)
    @creation-date 1 Apr 2004
}

folder_id:integer,notnull
schema_id:integer,notnull
{itemID:trim none}
{title:trim ""
{description:trim ""

-validate {
    valid_folder -requires {folder_id:integer} {
        if ![fs_folder_p $folderid] {
            ad_complain "[ file-storage.lt_The specified parent_]"
        }
    }
}

set old_title $title
set old_desc $description
set title ""
set description ""

set context [fs_context_bar_list -final "Add File From DSpace" $folderid]

# get the dspace-export file
# base this on file title or filename (however dspace web service is formatted)
# set content [util_httpget http://web.mit.edu/gtcuevas/Public/export.txt]

### DSPACE WEB SERVICE CALL ###
::SOAP::create GetFile \
    -uri "http://dspace-14.mit.edu:8080/axis/services/ItemAccessService" \
    -proxy "http://dspace-14.mit.edu:8080/axis/services/ItemAccessService" \
    -name "retrieveItem" \
    -action "" \
    -params { epersonID string itemID string }

set personid rrodgers@mit.edu
set itemid http://hdl.handle.net/123456789/23
if [catch {set contentencoded [GetFile $personid $itemid]} errmsg] {
    ad_return_complaint 1 "Error getting the file from DSpace"
    ad_script_abort
}

set content [::base64::decode $content_encoded]

# get the field name of the fields that map to ss title and description
set titles \
    [db_list get_titles \
        "select field_name from metadata_fields where mapping_ss_field='title' and schema_id=$schema_id and (map_type=1 or map_type=0)"
    ]
set descriptions \
    [db_list get_desc \n        "select field_name from metadata_fields where mapping_ss_field='description' and schema_id=$schema_id and (map_type=1 or map_type=0)"
    ]

# parse export file
if { [catch {dom parse $content doc} errmsg] } {
    return
}
```

61
```plaintext
set root [get documentElement]
set mods_field ""
set mods_value ""
set file_loc ""
set file_md [list]

foreach child [root childNodes] {
    set childName [child nodeName]
    lappend file_content $childName
    if [string equal $childName fileSec] {
        foreach fileSecChild [child childNodes] {
            if [string equal [fileSecChild nodeName] file] {
                foreach fileGrpChild [fileSecChild childNodes] {
                    if [string equal [fileGrpChild nodeName] fileGrp] {
                        foreach fileGrpChild [fileGrpChild childNodes] {
                            if [string equal [fileGrpChild nodeName] file] {
                                set parsed_file_loc [fileGetAttribute OWNERID noval]
                                if { ![string equal $parsed_file_loc noval] } {
                                    set file_loc "$parsed_file_loc"
                                }
                            }
                        }
                    }
                }
            }
        }
    } elseif [string equal $childName dmdSec] {
        foreach dmdChild [child childNodes] {
            if [string equal $dmdChildnodeName mdWrap] {
                foreach mdChild [dmdChild childNodes] {
                    if [string equal [mdChild nodeName] xmlData] {
                        foreach field [mdChild childNodes] {
                            set mods-field mods:abstract
                            set mods-value [field text]
                        }
                    } elseif [string equal $childName mods:accessCondition] {
                        if [string equal [field nodeType] TEXT_NODE] {
                            set mods-field rights_uri
                            set mods-value [field text]
                        }
                    } elseif [string equal $childName mods:classification] {
                        set attr_val [field getAttribute authority noval]
                        if [string equal [field nodeType] TEXT_NODE] {
                            set mods-field subject_1cc
                            set mods-value [field text]
                        } elseif [string equal $attr_val lcsh] {
                            set mods-field subject_lcc
                            set mods-value [field text]
                        } elseif [string equal $attr_val ddc] {
                            set mods-field subject_ddc
                            set mods-value [field text]
                        }
                    }
                }
            }
        }
    }
}
```

62
set mods_field subject lcsh
set mods_value [$field text]
} elseif [string equal $attr_val mesh]
set mods_field subject mesh
set mods_value [$field text]
} elseif [string equal $attr_val local]
set mods_field subject other
set mods_value [$field text]
} elseif [string equal $attr_val noval]
set mods_field subject classification
set mods_value [$field text]
}
}
} elseif [string equal $fieldName mods:extension]
if [string equal [$field_child nodeType] ELEMENT_NODE]
if [string equal [$field_child getAttribute encoding naval] iso8601]
if { [llength [$field_child childNodes]] == 11 }
set element_child [$field_child firstChild]
if [string equal [$element_child nodeType]
TEXT_NODE]
set element_name [$field_child nodeName]
if [string equal $element_name
mods:dateAccessioned]
set mods_field date_accessioned
set mods_value [$field_child text]
} elseif [string equal $element_name
mods:dateAvailable]
set mods_field date_available
set mods_value [$field_child text]
} elseif [string equal $element_name
mods:dateSubmitted]
set mods_field date_submitted
set mods_value [$field_child text]
}
} elseif [string equal $fieldName mods:genre]
if [string equal [$field_child nodeType] TEXT_NODE]
set mods_field type
set mods_value [$field text]
} elseif [string equal $fieldName mods:identifier]
set attr_val [$field getAttribute type noval]
if [string equal [$field_child nodeType] TEXT_NODE]
if [string equal $attr_val govdoc]
set mods_field identifier-govdoc
set mods_value [$field text]
} elseif [string equal $attr_val isbn]
set mods_field identifier_isbn
set mods_value [$field text]
} elseif [string equal $attr_val ismn]
set mods_field identifier_ismn
set mods_value [$field text]
} elseif [string equal $attr_val issn]
set mods_field identifier_issn
set mods_value [$field text]
} elseif [string equal $attr_val local]
set mods_field identifier_local
set mods_value [$field text]
} elseif [string equal $attr_val sici]
set mods_field identifier_sici
set mods_value [$field text]
} elseif [string equal $attr_val uri]
set mods_field identifier_uri
set mods_value [$field text]
} elseif [string equal $attr_val noval]
set mods_field identifier
set mods_value [$field text]

63
set mods-value [$field text]
}

} elseif [string equal $fieldName mods:language]
if [string equal [$field_child nodeType] ELEMENT_NODE]
mods:languageTerm} {
    if [string equal [$field_child nodeName] authority noval]
nodeType] TEXT_NODE} {

    set element-attr [$field_child getAttribute
authority noval]

    if { [llength [$field_child childNodes]] == 11 }
        if [string equal [$field_child firstChild] element-attr rfc3066]
            set mods-field language_iso
            set mods-value [$field_child text]
        elseif [string equal $element-attr noval]
            set mods-field language
            set mods-value [$field_child text]
        }
    }
}

} elseif [string equal $fieldName mods:name]
if [string equal [$field_child nodeType] ELEMENT_NODE]
mods:namePart} {
    if [string equal [$field_child nodeName] authority noval]
nodeType] TEXT_NODE} {

    if { [llength [$field_child childNodes]] == 11 }
        if [string equal [$field_child firstChild] element-attr rfc3066]
            set mods-field language_iso
            set mods-value [$field_child text]
        elseif [string equal $element-attr noval]
            set mods-field language
            set mods-value [$field_child text]
        }
    }
}

} elseif [string equal $fieldName mods:note]
if [string equal [$field_child nodeType] TEXT_NODE] {
    if { [$field hasAttribute xlink:simpleLink] == 1}
        set mods-field description_uri
        set mods-value [$field text]
    else {
        set attr-val [$field getAttribute type noval]
        if [string equal $attr-val provenance] {
            set mods-field description_provenance
            set mods-value [$field text]
        } elseif [string equal $attr-val sponsorship] {
            set mods-field description_sponsorship
            set mods-value [$field text]
        } elseif [string equal $attr-val "statement of responsibility"] {
            set mods-field
description_statement_of_responsibility
            set mods-value [$field text]
        } elseif [string equal $attr-val noval] {
            set mods-field
description
            set mods-value [$field text]
        }
    }
}

} elseif [string equal $fieldName mods:originInfo]
if [string equal [$field_child nodeType] ELEMENT_NODE]
nodeType] TEXT_NODE} {
encoding noval]

mods:copyrightDate] {
  set mods_field date_copyright
  set mods_value [$field_child text]
} elseif [string equal $element_name
  set modsfield datecopyright
  set modsvalue [$fieldchild text]
elseif [string equal $element-name
  set modsfield date-created
  set modsvalue [$field-child text]
} elseif [string equal $element-name
  set modsfield date-issued
  set mods_value [$field-child text]
} elseif [string equal $attr-val noval] {if [string equal $element_name
  set modsfield publisher
  set mods_value [$field_child text]
}

} } elseif [string equal $fieldName mods:physicalDescription] {if [string equal $fieldName mods:physicalDescription]
  set element_name [$field_child nodeName]
  if [string equal $element_name mods:extent] 
  set mods_field format_extent
  set mods_value [$field_child text]
} elseif [string equal $fieldName mods:relatedItem] {if [string equal $fieldName mods:relatedItem]
  set text_node [$field_child firstChild]
  if [string equal $text_node nodeType] ELEMENT_NODE] {
    if [string equal $text_node nodeType] ELEMENT_NODE] {
      if ( [llength [$field-child childNodes]] == 1 ) {
        if [string equal [$field-child firstChild]
          set element_name [$field_child nodeName]
          if [string equal $element_name mods:extent] 
            set mods_field format_extent
            set mods_value [$field_child text]
        } elseif [string equal $element-name
          set mods_field format
          set mods_value [$field_child text]
        } elseif [string equal $element-name
          set mods_field format_mimetype
          set mods_value [$field_child text]
      }
    }
}
} elseif [string equal $field getAttribute type noval] {
    if [string equal $field_child nodeType] ELEMENT_NODE] {

        if ( [llength [$field-child childNodes]] == 1 ) {
          if [string equal [$field-child firstChild]
            set mods_field $text_node
            set mods_field
set mods_value [$text_node text]

} elseif [string equal [string equal $attr_val $attr_val constituent] {
    set mods_field relation_haspart
    set mods_value [$field text]
} elseif [string equal $attr_val otherVersion] {
    set mods_field relation_version
    set mods_value [$field text]
} elseif [string equal $attr_val original] {
    set mods_field relation_isbasedon
    set mods_value [$field text]
} elseif [string equal $attr_val otherFormat] {
    set mods_field relation_isformatof
    set mods_value [$field text]
} elseif [string equal $attr_val host] {
    set mods_field relation_ispartof
    set mods_value [$field text]
} elseif [string equal $attr_val series] {
    set mods_field relation_ispartofseries
    set mods_value [$field text]
} elseif [string equal $attr_val isReferencedBy] {
    set mods_field relation_isreferencedby
    set mods_value [$field text]
} elseif [string equal $attr_val succeeding] {
    set mods_field relation_isreplacedby
    set mods_value [$field text]
} elseif [string equal $attr_val replaces] {
    set mods_field relation_replaces
    set mods_value [$field text]
} elseif [string equal $attr_val requires] {
    set mods_field relation_requires
    set mods_value [$field text]
} elseif [string equal $attr_val original] {
    if [string equal xlink:simpleLink] {
        set mods_field source_uri
        set mods_value [$field text]
    } else {
        set mods_field source
        set mods_value [$field text]
    }
} elseif [string equal $fieldName mods:subject] {
    if [string equal $field_child nodeType] ELEMENT_NODE}
if { [llength [$field-child childNodes]] == 1 } {
    if [string equal [[]$field-child firstChild

    mods:geographic] {
        set mods-field coverage-spatial
        set mods-value [$field-child text]
    } elseif [string equal $element_name

    mods:temporal] {
        set mods-field coverage-temporal
        set mods-value [$field-child text]
    } elseif [string equal $element_name

    mods:topic] {
        set mods-field subject
        set mods-value [$field-child text]
    }
    }
} elseif [string equal $fieldName mods:tableOfContents] {
    if [string equal [[]$field-child nodeType] TEXT_NODE] {
        set mods-field type
        set mods-value [$field text]
    }
    } elseif [string equal $fieldName mods:titleInfo] {
    if [string equal [[]$field-child nodeType] TEXT_NODE] {
        set attr-val [$field getAttribute type noval]
        if [string equal $attr-val alternative] {
            set mods-field title_alternative
            set mods-value [$field text]
        } elseif [string equal $attr-val noval] {
            set mods-field title
            set mods-value [$field text]
        } elseif $fieldName mods:tableOfContents] == 2] {
    if [string equal $fieldName mods:name] {
        set role-ele [$field firstChild]
        set namePart-ele [$field lastChild]
        if [string equal $role-ele nodeName] mods:role] {
            if [string equal $namePart-ele nodeName]

    mods:roleTerm] {
        if { [llength [$roleTerm-ele childNodes]] == 1 ] {
            set roleTerm-ele [$role-ele firstChild]
            if [string equal $roleTerm-ele

    text] {
        set role_term [$roleTerm-ele text]
        if [string equal $role_term advisor]

    set mods-field

    set mods_value [$NamePart-ele

    author] {
        set mods-field contributor_author

    set mods-value [$NamePart-ele

    editor] {
        set mods-field contributor_editor

    set mods-value [$NamePart-ele

    illustrator] {
        set mods-field

    contributor_illustrator
```tcl
set mods_value [${namePart}_ele

text] elseif [string equal $role_term other] {
set mods_field contributor_other
set mods_value [${namePart}_ele

text] } } } } } if ![string equal $mods_value"] { lappend file_md [list $mods_field $mods_value]
if { [search -exact $titles $mods_field] != -1} {
append title "$mods_value"
}
if { [search -exact $descriptions $mods_field] != -1} {
append description "$mods_value"
}
}
}
}

### DSPACE WEB SERVICE CALL ###
::SOAP::create GetContent \\
-name "retrieveBitstream" \\
-action "" \\
-params { epersonID string bitstreamID string }

set bitstream_content [GetContent $personid $file_loc]
set tempfile_loc [acs_root_dir]/packages/file-storage/www/dspace-temp/tempfile
set out [open tempfile_loc w]
fconfigure $out -translation binary
.puts -nonewline $out [::base64::decode $bitstream_content]
close $out

### add the file into sloanspace ###
# Check for write permission on this folder
ad_require_permission $folderid write

# Get the user
set user_id [ad_conn user_id]
# Get the ip
set creation_ip [ad_conn peeraddr]
# Get file mime_type
set mime_type [cr_filename_to_mime_type -create $file_loc]

# Get the file name part of the upload file
if ![regexp [^/\] ]+ $file_loc filename] } { # no match
set filename $file_loc
}
# Get the title
if ![empty_string_p $title] } {
set title $filename
```
calls
db_transaction {
    set file_id [db_exec_plsql new_lob_file {}]
    set version_id [db_exec_plsql new_version {}]
    db_dml lob_content -blob_files [list $tempfile_loc]
    # Unfortunately, we can only calculate the file size after the lob is uploaded
    db_dml lob_size {}
B.2 dspace-submit.tcl

ad_page_contract {
    Try meta
} {
    schema_id:integer
    file_id:integer
}

set url_query [ad_conn query]

###
### check if submit type = addmore
###

set addmore_start [string first addmore $url_query]
if { $addmore_start > -1 } {
    set query_length [string length $url_query]
    set substring [string range $url_query $addmore_start [expr $query_length - 1]]
    set addmore_end [string first $substring =]
    set addmore_string [string range $substring 8 [expr $addmore_end - 1]]
    append url_query "&multiple-"$addmore_string"
    ad_returnredirect "meta-view?$url_query"
    ad_script_abort
}

###
### check if submit type = cancel
###

set submit_type [ns_queryget submit_type]
if [string equal $submit_type Cancel] {
    ad_returnredirect "file?file_id=$file_id"
    ad_script_abort
}

###
### check to see if file was already submitted
###

if { [db_Orilrow get-val "select * from metadatasubmissions where fileid=$fileid and schema_id=$schema_id"] == 1 } {
    ad_returnredirect "dspace-submitted?schema_id=$schema_id&fileid=$fileid"
    ad_script_abort
}

set has_required 0

###
### check for empty fields
###
db_foreach get_required "select * from metadata_fields where schema_id=$schema_id" {
    #get empty fields
    if [string equal $required t] {
        if [empty_string_p [ns_queryget $field_id]] {
            append url_query "&empty_req_field=$field_id"
            set has_required 1
        }
    }
}

###
### upload to dspace
###

set mylist [list]
set xml_file "<dublin_core/>
"

#process fields
if { $has_required == 1 } { 
    ad_returnredirect "meta-view?url_query"
    ad_script_abort
} else {
    db_dml delete_old_values;
    delete from metadata_field_values where file_id = :file_id and schema_id = :schema_id
}

db_foreach get_fields "select * from metadata_fields where schema_id=$schema_id"
{
    #set field name tag for xml file
    regsub -all "." $field_name "split fname
    set elt [lindex $split fname 0]
    set qual "none"
    if { [llength $split_filename] > 1 } { 
        set qual [lindex $split_filename 1]
    }

    #get value
    if [ns_queryexists $field_id] { 
        if ! [empty_string $ns_queryget $fieldid] { 
            set value [ns_queryget $fieldid]
            lappend mylist "$fieldid = $value"
            #add value to database
            db_dml insert_value { 
                insert into metadata_field_values values (:file_id, :fieldid, :value, :schema_id)
            }
        }
    }

    #insert value into xml file
    append xml_file "<dcvalue element="$elt" qualifier="$qual">$value</dcvalue>
"
}

# get multiple values
if [ns_queryexists multiple-$field_id] { 
    set mult_list [ns_querygetall multiple-$field_id]
    foreach mult $mult_list { 
        if ! [string equal $mult ""] { 
            db_dml insert_mult_val { 
                insert into metadata_field_values values (:file_id, :fieldid, :mult, :schema_id)
            }
        }
    }
}

#append end tag to xml file
append xml_file "</dublincore>
"

#either save or submit
if [string equal $submit_type Save] { 
    ad_returnredirect "dspace-submitsaved?file_id=$file_id"
    ad_script_abort
} else {
    ### ADD OTHER SUBMIT CALLS HERE ###

    if { $schema_id != 1 } { 
        ad_returnredirect "file?file_id=$file_id"
    }
}
# Turn XML file into base64 binary string
set xml_file_base64 [::base64::encode $xml_file]

set file_name [db_string get fn "select file_name from metadata_file_view where file_id = $file_id"]

# Get the file content and save as a base64 encoded string
set tempfile /web/gen/www/temp
set blob_file [db_blob_get_file "get_content" "select content from metadata_file_view where file_id = $file_id" -file $tempfile]
set open_file [open $tempfile r]
fconfigure $open_file -encoding binary
set pure_file [read $open_file]
close $open_file
set encoded_string [::base64::encode $pure_file]

### DSPACE WEB SERVICE CALL ###

## Create SOAP Requests
::SOAP::create DepositItem \ 
  -uri "http://18.42.6.79:8080/axis/services/ItemIngestService" \ 
  -proxy "http://18.42.6.79:8080/axis/services/ItemIngestService" \ 
  -name "depositItem" \ 
  -action "" \ 
  -params { epersonID string collectionID string docBytes base64Binary }

::SOAP::create DepositBitstream \ 
  -uri "http://18.42.6.79:8080/axis/services/ItemIngestService" \ 
  -proxy "http://18.42.6.79:8080/axis/services/ItemIngestService" \ 
  -name "depositBitstream" \ 
  -action "" \ 
  -params { ticket string fileName string bitstream base64Binary }

::SOAP::create DepositComplete \ 
  -uri "http://18.42.6.79:8080/axis/services/ItemIngestService" \ 
  -proxy "http://18.42.6.79:8080/axis/services/ItemIngestService" \ 
  -name "depositComplete" \ 
  -action "" \ 
  -params { ticket string }

## Call SOAP Methods
set personid rrogers@mit.edu
set collectionid http://hdl.handle.net/123456789/2
if [catch {set ticket [DepositItem $personid $collectionid $xml_file_base64]} errmsg]
  {ad_return_complaint 1 "Error depositing metadata into DSpace" ad_script_abort}
if [catch {set depositBitstream [DepositBitstream $ticket $file_name $encoded_string]} errmsg]
  {ad_return_complaint 1 "Error depositing file bitstream into DSpace" ad_script_abort}
if [catch {set depositComplete [DepositComplete $ticket]} errmsg]
  {ad_return_complaint 1 "Error depositing file bitstream into DSpace" ad_script_abort}

db_dml insert_submission {
  insert into metadata_submissions values (:file_id, :schema_id)
}
ad_returnredirect "dspace-submitsuccess?file_id=$file_id" ad_script_abort}
B.3 meta-view.adp

<master>
  <property name="title">Upload to DSpace</property>
  <property name="context">@context;noquote</property>
  <form method=get action="@submit_file_name@">
    <input type=hidden name=schema_id value="@schema_id@">
    <input type=hidden name=file_id value="@file_id@">
    <table>
      <multiple name=ds>
        <tr height=40 valign=bottom>
          <td>&lt;td align="right"&gt;&lt;b&gt;@ds.pretty_name@&lt;/b&gt;&lt;/td&gt;
          <td>
            &lt;if @ds.display_text@ not nil&gt;
              &lt;small&gt;@ds.display_text@&lt;/small&gt;
            &lt;/if&gt;
          &lt;/td&gt;
        &lt;/tr&gt;
        &lt;multiple name=multiplefields&gt;
          &lt;if @multiplefields.fid@ eq @ds.fieldid@&gt;
            &lt;tr&gt;
              &lt;td&gt;
                &lt;if @ds.display_type@ eq "text"&gt;
                  &lt;textarea name=multiple-@ds.fieldid@ @ds.display_attributes@ value="@multiplefields.fval@"&gt;&lt;/textarea&gt;
                &lt;/if&gt;
                &lt;if @ds.display_type@ eq "textarea"&gt;
                  &lt;textarea name=multiple-@ds.fieldid@ @ds.display_attributes@ value="@multiplefields.fval@"&gt;&lt;/textarea&gt;
                &lt;/if&gt;
                &lt;if @ds.display_type@ eq "select"&gt;
                  &lt;select name=multiple-@ds.fieldid@ @ds.display_elements;noquote@&gt;&lt;/select&gt;
                &lt;/if&gt;
                &lt;if @multiplefields.fid@ eq @ds.fieldid@&gt;
                  &lt;input type=submit name=addmore-@ds.fieldid@ value="Add More"&gt;
                &lt;/if&gt;
              &lt;/td&gt;
            &lt;/tr&gt;
          &lt;/if&gt;
        &lt;/multiplefields&gt;
      &lt;/multiple&gt;
    &lt;/table&gt;
    &lt;tr&gt;
      &lt;td colspan=2 height=20&gt;&lt;/tr&gt;
      &lt;input type=submit name=submit_type value="Save"&gt;
      &lt;input type=submit name=submit_type value="Cancel"&gt;
      &lt;input type=submit name=submit_type value="Upload to @schema_name@"&gt;
    &lt;/tr&gt;
  &lt;/table&gt;
&lt;/form&gt;
B.4 meta-view.tcl

ad_page_contract {
  Try meta
} {
  schema_id:integer
  file_id:integer
}

set schema_name [db_string get_sn "select schema_name from metadata_schemas where schema_id=$schema_id"]

set context [fs_context_bar_list -final "Upload to $schema_name" $file_id]

## set the submit file
if [string equal $schema_name DSpace] {
  set submit_file_name dspace-submit
}

set empty_fields [list]

# check for empty required fields
if [ns_queryexists empty_req_field] {
  set empty_fields [ns_querygetall empty_req_field]
}

set startrow 1
set numrows 0
multirow create multiple_fields fid fval

db_multirow –extend [value empty] ds get_dsmetadata {
  select * from metadata_fields where schema_id = :schema_id
} {
  # check for empty field
  set empty [nssearch –exact $empty_fields $field_id]
  # check for and set existing field values
  if [ns_queryexists $field_id] {
    set value [ns_queryget $field_id]
  } else {
    set db_values [db_list get_vl "select field_value from metadata_field_values where file_id=$file_id and field_id=$field_id"]
    if { [llength $db_values] == 1 } {
      set value [lindex $db_values 0]
    } elseif { [llength $db_values] > 1 } {
      set i 0
      foreach db_value $db_values {
        if { ![string equal $db_value **] } {
          if { $i == 0 } {
            set value $db_value
          } else {
            if [string equal $multiple t] {
              multirow append multiple_fields $field_id $db_value
            } elseif [string equal $multiple f] {
              append value "$db_value"
            }
          }
        }
        set i [expr $i + 1]
      }
    } else {
      # get from other schemas
      set value ""
      set db_mapped_fields_1 [db_list get_mfl "select field_id from metadata_schema_mappings where field_id_1=$field_id and (map_type=0 or map_type=1)"]
      set db_mapped_fields_2 [db_list get_mfl "select field_id_1 from metadata_schema_mappings where field_id_2=$field_id and (map_type=0 or map_type=1)"]
    }
  }
}

# get from other schemas
set value ""
set db_mapped_fields_1 [db_list get_mfl "select field_id from metadata_schema_mappings where field_id_1=$field_id and (map_type=0 or map_type=1)"]
set db_mapped_fields_2 [db_list get_mfl "select field_id_1 from metadata_schema_mappings where field_id_2=$field_id and (map_type=0 or map_type=1)"]

set db_values [db_list get_vl "select field_value from metadata_field_values where file_id=$file_id and field_id=$field_id"]

if { [llength $db_values] == 1 } {
  set value [lindex $db_values 0]
} elseif { [llength $db_values] > 1 } {
  set i 0
  foreach db_value $db_values {
    if { ![string equal $db_value **] } {
      if { $i == 0 } {
        set value $db_value
      } else {
        if [string equal $multiple t] {
          multirow append multiple_fields $field_id $db_value
        } elseif [string equal $multiple f] {
          append value "$db_value"
        }
      }
    }
    set i [expr $i + 1]
  }
} else {
  # get from other schemas
  set value ""
  set db_mapped_fields_1 [db_list get_mfl "select field_id from metadata_schema_mappings where field_id_1=$field_id and (map_type=0 or map_type=1)"]
  set db_mapped_fields_2 [db_list get_mfl "select field_id_1 from metadata_schema_mappings where field_id_2=$field_id and (map_type=0 or map_type=1)"]
}

74
get-mf2

"select field_id_2 from metadata_schema_mappings where
field_id_1=$field_id and (map_type=0 or map_type=-1)"

set db_mapped_fields [concat $db_mapped_fields_1 $db_mapped_fields_2]
if { [llength $db_mapped_fields] >= 1 }
    foreach mfield $db_mapped_fields {
        set db_vals [db_list get_vals "select fieldvalue from
metadata_fieldvalues where file_id=$fileid and field_id=$mfield"
        if { [llength $db_vals] == 1 }
            set value [lindex $db_vals 0]
        elseif { [llength $db_vals] > 1 }
            set j 0
            foreach db_val $db_vals {
                if { ![string equal $db_val ""] }
                    if {$j == 0 }
                        set value $db_val
                    else
                        if string equal $multiple t
                            multirow append multiple_fields $fieldid $db_val
                        elseif [string equal $multiple f]
                            append value "$db_val"
                        
                set j [expr $j + 1]
            
        
        #get from ss table
        if { [string equal $value ""] }
            if { ![exists_and_not_null mapping_ss_field] && ($map_type == 0 ||
            $map_type == 1) }
                set value **
                set has_mapping \n                [db_orlrow \n                get_mapping \n                *select mapping_table_or_view,mapping_col_name from
sloanspace_file_metadata where field_name='$mapping_ss_field'"
            if {$has_mapping == 1 }
                query get_value value onevalue \n                "select $mapping_col_name from $mapping_table_or_view where
file_id=$file_id"
            } else {
                set value **
            
        
        #check for multiple values in querystring and append to multiple_fields multirow
        set mult_list [ns_querygetall multiple-$field_id]
        set list_length [llength $mult_list]
        foreach mult $mult_list {
            multirow append multiple_fields $fieldid $mult
        }

        #replace selected in select lists with new value
        if { [string equal $display_type select] }
            if { ![empty_string_p value] }
                if { ![string equal $value ""] }
                    set string_length [string length $display_elements]
                    set val_index [string first value="$value" $display_elements]
                    if { $val_index > -1 }
                        set start_index [expr [string length value-$value] + $val_index + 2]
                        set start_string [string range $display_elements 0 [expr $start_index
- 1]]
set end_string [string range $display_elements $start_index [expr $string_length - 1]]

set display_elements $start_string
append display_elements " selected"
append display_elements $end_string
B.5 schema-add.adp

<master>
<br>
<table width=100% cellspacing=0 cellpadding=0>
<tr><td><font size="+1" color=#003366><u>Add Schema</u></font></td></tr></table>
<br>
<form action="schema-add-2" method=get>
<table>
<tr><td align="right">Schema Name:</td>
<td colspan=2><input type=text name="name" value="" size=50></td>
</tr>
<tr><td align="right">Number of Fields:</td>
<td align="left" colspan=2><input type=text name="num_fields" value="" size=7></td>
</tr>
</table>
<input type="submit" value="Add Schema">
<a href="schema-add-help">Add Schema Help</a></form>

B.6 schema-add-2.tcl

ad_page_contract {
    page to add a new nonversioned object to the system

    @author Genevieve Cuevas (gtcuevas@mit.edu)
    @creation-date 01 April 2004
} {
    name:trim
    num_fields:integer
}

set schema_id [db_string get_id "select seq_schema_id.nextval from dual"]

db_dml add_schema {
    insert into metadata_schemas values (:schema_id,:name)
}

ad_returnredirect "schema-add-fields?schema_id=$schema_id&num_fields=$num_fields"
B.7 schema-add-fields.adp

```xml
<master>
<table width="100%" cellspacing=0 cellpadding=0>
<tr><td><font size="+1" color="#003366"><u>Add Fields</u></font></td><td><u><br></u></td></tr>
</table>
<form method=get action="schema-add-fields-2">
<input type=hidden name="schema id" value=@schema-id@>
<input type=hidden name="num_fields" value=@num_fields@>
<table>
<multiple name=fields>
<tr><td align="right">Field @fields.rownum@ Name:</td><td colspan=2><input type=text name="field_@fields.rownum@" value=""></td></tr>
<tr><td align="right">SloanSpace Mapping:</td><td colspan=2><select name="ssm_@fields.rownum@">
<option value="none">None</option>
<option value="author">Author</option>
<option value="title">Title</option>
<option value="publishdate">Publish Date</option>
<option value="description">Description</option>
</select>
Mapping Type:
<select name="ssmt_@fields.rownum@">
<option value="0">Both Ways</option>
<option value="1">Remote Schema to SloanSpace</option>
<option value="-1">SloanSpace to Remote Schema</option>
</select>
</td></tr>
<tr><td align="right">Required:</td><td colspan=2><select name="req_@fields.rownum@">
<option value="t">Yes</option>
<option value="f">No</option>
</select>&nbsp;&nbsp;&nbsp;&nbsp;Has Multiple Values?
<select name="mult_@fields.rownum@">
<option value="t">Yes</option>
<option value="f">No</option>
</select>
</td></tr>
<tr><td align="right">Display Name:</td><td colspan=2><input type=text name="pn_@fields.rownum@" value=""></td></tr>
<tr><td align="right">Display Type:</td><td colspan=2>
<input type=radio name="dt_@fields.rownum@" value="text">Text Field
<input type=radio name="dt_@fields.rownum@" value="textarea">Text Area
<input type=radio name="dt_@fields.rownum@" value="select">Select List:
</td></tr>
<tr><td colspan=2 height=20></td></tr>
</multiple></form>
78
B.8 schema-add-fields.tcl

ad_page_contract {
    page to add a new nonversioned object to the system

    @author Genevieve Cuevas (gctcuevas@mit.edu)
    @creation-date 01 April 2004
}
{
    schema_id:integer
    num_fields:integer
}

multirow create fields fname

for {set i 1} {i <= $num_fields} {incr i} {
    multirow append fields "field_${i}"
}
B.9 schema-add-fields-2.tcl

ad_page_contract {
  page to add a new nonversioned object to the system

  @author Genevieve Cuevas (gtcuevas@mit.edu)
  @creation-date 01 April 2004
}

  schema_id:integer
  num_fields:integer
}

set mylist [list 1]
set options_query ""

# process fields from schema-add-fields
for {set i 1} {$i <= $num_fields} {incr i} {
  set field_name [ns-queryget field_$i]
  set ss_mapping [ns-queryget ssom_$i]
  set map_type [ns-queryget ssmt_$i]
  set required [ns-queryget req_$i]
  set is_mult [ns-queryget mult_$i]
  set pretty_name [ns-queryget pn_$i]
  set display_type [ns-queryget dt_$i]

  lappend mylist $fieldname
  lappend mylist $pretty_name

  if [ ![string equal $field_name ""] && ![string equal $pretty_name ""] ] {
    if [string equal $ss_mapping "none"] {
      dbdml addfield "insert into metadata_fields
                   values($field-id, '$fieldname', '$pretty_name',$schema-id,null,$map-type,
                      null, '$display-type',null,null,null,null, '$required', '$is-mult')"
    } else {
      dbdml add_field "insert into metadata_fields
                      values($field_id,'$field_name','$pretty_name',$schema_id,'$ss-mapping',
                      $map_type,null,'$display_type',null,null,null,
                      '$required','$is_mult')"
    }
  } else {
    dbdml add_field "insert into metadata_fields
                      values($field_id,'$field_name','$pretty_name',$schema_id,'$ss-mapping',
                      $map_type,null,'$display_type',null,null,null,
                      '$required','$is_mult')"
  }

  if [string equal $display_type "select"] {
    set num_options [ns-queryget num_ops_$i]
    append options_query "$num_ops_$field-id=$num_options"
  }
}

# redirect to continue page
ad_returnredirect "schema-add-fields-cont?schema_id=$schema_id$options_query"
### B.10 schema-add-fields-cont.adp

```xml
<master>
<table width="100%" cellspacing=0 cellpadding=0>
<tr><td><font size="+1" color="#003366"><u>Add Fields (cont.)</u></font></td></tr>
<br><br>
<form action="schema-add-fields-cont-2" method=get>
<table>
<input type=hidden name=schema id value="@schema-id@">
<multiple name=fields>
<tr><td colspan=3><b>Field Name: @fields.field_name@</b></td></tr>
<tr>
<td align="right">Display Text:</td>
<td colspan=2><input type=text name=dt_@fields.fieldid@ size=100></td>
</tr>
<tr>
<td align="right">Error Text:</td>
<td colspan=2><input type=text name=et_@fields.field-id@ size=100></td>
</tr>
<if @fields.display-type@ eq "text">
<tr>
<td align="right">Text Field Size:</td>
<td colspan=2><input type=text name=tfs_@fields.fieldid@ size=7></td>
</tr>
</if>
<if @fields.display-type@ eq "textarea">
<tr>
<td align="right">Text Area Rows:</td>
<td colspan=2><input type=text name=tar_@fields.field_id@ size=7>
	.Columns:<input type=text name=tac_@fields.field_id@ size=7></td>
</tr>
</if>
<if @fields.display-type@ eq "select">
<multiple name="options">
<if @options.fieldjid@ eq @fields.fieldid@>
<tr>
<td align="right">Option Text:</td>
<td colspan=2><input type=text name=opt_@fields.field-id@_@options.index@ size=20>
	.Value:<input type=text name=opv_@fields.fieldid@_@options.index@ size=20></td>
</tr>
</if>
</multiple>
</if>
</multiple>
<tr><td colspan=3 height=20></td></tr>
<multiple name="num-ops_list">
<input type=hidden name=@num-ops_list.numops-string@ value=@num-ops_list.num_ops_value@>
</multiple>
<tr><td colspan=3 height=20></td></tr>
<tr><td align="left">
<input type=submit value=" Submit ">
</td></tr>
<tr><td align="right"><a href="schema-add-help">Add Schema Help</a></td></tr>
</table>
</form>
```
B.11 schema-add-fields-cont.tcl

ad_page_contract {
    page to add a new nonversioned object to the system

    @author Genevieve Cuevas (gtcuevas@mit.edu)
    @creation-date 01 April 2004
}

    schema_id:integer
}

multirow create options field_id index

#this is the multirow for the num_ops key in the query
multirow create num_ops_list num_ops_string num_ops_value

db_multirow fields get_fields {
    select * from metadata_fields where schema_id = :schema_id
}

    if [string equal $display_type "select"] {
        if [ns_queryexists num_ops_$field_id] {
            set num_ops [ns_queryget num_ops_$field_id]
            for {set i 1} {$i <= $num_ops} {incr i} {
                multirow append options $field_id $i
            }
            multirow append num_ops_list num_ops_$field_id $num_ops
        }
    }
}

}
B.12 schema-add-fields-cont-2.tcl

ad_page_contract {
  page to add a new nonversioned object to the system

  @author Genevieve Cuevas (gctuevas@mit.edu)
  @creation-date 01 April 2004
}

  schema_id:integer

#process fields for schema-add-fields-cont

dbforeach set_fields "select * from metadata_fields where schema_id=$schema_id" {
  set display_text null
  set error_text null
  set display_attributes null
  set display_elements null

  ### set field values ###
  if [ns-queryexists dt_$fieldjid] {
    set display_text [ns-queryget dt_$fieldid]
  }
  if [ns-queryexists et_$field-id] {
    set error_text [ns-queryget et_$fieldjid]
  }
  if [string equal $display_type "text"] {
    if [ns-queryexists tfs_$fieldjid] {
      set size [ns-queryget tfs_$fieldid]
      set display_attributes "size=$size"
    } elseif [string equal $display-type "textarea"] {
      set rows ""
      set cols ""
      if [ns-queryexists tar_$fieldjid]
        set rows "rows=[ns-queryget tar_$fieldid]"
      } if [ns-queryexists tac_$fieldjid] {
        set cols "cols=[ns-queryget tac_$fieldid]"
      } set display_attributes "$rows $cols"
  } elseif [string equal $display_type "select"] {
    set display_attributes ""
    set display_elements ""
    if [ns-queryexists num_ops_$fieldjid] {
      set num_ops [ns-queryget num_ops_$fieldid]
      for {set i 1} {i <= $num_ops} {incr i} {
        set option_text ""
        set option_value ""
        set opt_str opt_$fieldid
        append opt_str "_$_i"
        set opv_str opv_$fieldid
        append opv_str "_$_i"
        if {[ns-queryexists $opt_str] && [ns-queryexists $opv_str]} {
          set option_text [ns-queryget $opt_str]
          set option_value [ns-queryget $opv_str]
          append display_elements "<option value="$option_value">$option_text</option>"
        }
      }
    } else {
      #add values to database
      db_dml set_values "update metadata_fields set display_text='$display_text', error_text='$error_text', display_attributes='$display_attributes', display_elements='$display_elements' where field_id=$field_id"
    }
  ad_returnredirect "schema-add-fields-done"
}
B.13 search-url.adp

<form method=GET action="search-url-results">
<input type=hidden name="folder_id" value="$folder_id$">
<table border=0>
<tr>
<td align=right>Search:</td>
<td colspan=2><input size=70 name="searchstring" value=""></td>
</tr>
<tr>
<td></td>
<td><input type=submit value="Search" &nbsp;&nbsp;&nbsp;&nbsp; in &nbsp;&nbsp;&nbsp;&nbsp;
<input type=radio name="searchtype" value="dspace" checked>DSpace
<input type=radio name="searchtype" value="google" >Google
<input type=radio name="searchtype" value="googledspace">DSpace-restricted Google</td>
</tr>
</table>
</form>
B.14 search-url.tcl

ad_page_contract {
    page to add a new nonversioned object to the system
}

@author Genevieve Cuevas (gtcuevas@mit.edu)
@creation-date 01 April 2004

} {
    folder_id:integer,nonnull
    {type "fs_url"}
    {title ""
    {lock_title_p 0}
} -validate {
    valid_folder -requires {folder_id:integer} {
        if ![fs_folderp $folderid] {
            ad_complain "[_ file-storage.lt_The_specified_parent_]"
        }
    } -properties {
        folder_id:onevalue
        context:onevalue
    }
}

# check for write permission on the folder
ad_require_permission $folder_id write

# set templating datasources
set pretty_name "Search DSpace"
if ![empty_string_p $pretty_name] {
    return -code error "[_ file-storage.No_such_type]"
}

# set context [fs_context_bar_list -final [_ file-storage.Search [list pretty_name
$pretty_name]] $folder_id]
set context [fs_context_bar_list -final "Search DSpace" $folder_id]

# Should probably generate the item_id and version_id now for
# double-click protection

# if title isn't passed in ignore lock_title_p
if ![empty_string_p $title] {
    set lock_title_p 0
}

# Message lookup uses variable pretty_name
set page_title [_ file-storage.simple_add_page_title]
### B.15 search-url-results.adp

```xml
<master>
<property name="title">Search Results</property>
<property name="context">@context;noquote@</property>

<form method=GET action="search-url-results">
<table>
<tr align="right">Search:</tr>
<td><input size=50 name="searchstring" value="@searchstring@"></td>
</tr>
<tr>
<td><input type=submit value="Search"></td>
</tr>
</table></form>

<hr>
<table cellspacing=0 cellpadding=0 width=100%>
<tr>
<td align="right">Results <b>@pageminval@ - @pagemaxval@</b> of <b>@numrecords@</b> for <b>@searchstring@</b>:</td>
<td align="center"><a href="search-url-help">Search Help</a></td>
</tr>
</table>

<table>
<multiple name="urls">
<tr><td><a href="@urls.url@">@urls.title;noquote@</a></td></tr>
<tr><td>@urls.description;noquote@</td></tr>
<tr height=30><td>&nbsp</td></tr>
</multiple>
</table>
```

87
B.16 search-url-results.tcl

ad_page_contract {
    Search results in DSpace

    @author Genevieve Cuevas (gtcuevas@mit.edu)
    @creation-date 1 Apr 2004
}

{ folder_id:integer,nonnull
  searchstring:trim
  { searchType dspace
    { pagenum:integer 1
      } -validate
        valid_folder -requires { folder_id:integer } {
          if ![fs_folder_p $folder_id] {
            ad_complain "file-storage.ltThe-specified_parent_"
          }
        }
    }
}

# check for write permission on the folder
ad_require_permission $folder_id write

set context [fs_context_bar_list -final "Search Results" $folder_id]

# set pagination variables
set recordsperpage 5
set pagemaxval [expr $pagenum * $recordsperpage]
set pageminval [expr $pagemaxval - $recordsperpage + 1]
set nextpage [expr $pagenum + 1]
set prevpage [expr $pagenum - 1]

multirow create urls title url description

if [string equal $searchtype google]
   set results [acs-sc::invoke -operation paged_search -contract URLSearcher -impl GoogleSearcher -call-args [list $searchstring $recordsperpage $pagenum]]
elseif [string equal $searchtype googlespace]
   set results [acs-sc::invoke -operation restricted_paged_search -contract URLSearcher -impl GoogleSearcher -call-args [list $searchstring $recordsperpage $pagenum]]
else
   set results [acs-sc::invoke -operation paged_search -contract URLSearcher -impl DSpaceSearcher -call-args [list $searchstring $recordsperpage $pagenum]]

set resultslist [lindex $results 1]
foreach result $resultslist {
    multirow append urls [lindex $result 0] [lindex $result 1] [lindex $result 2]
}

set numrecords [lindex $results 0]
The "dspace searcher" searches and retrieves dspace urls.

@author gtcuervas@mit.edu
@version $Id: dspace-search-procs.tcl,v 1.0 04/14/04 09:51:04 peterm Exp $

namespace eval dspace_search {

ad_proc -private search_url {
    query
    }
    
    Implements the search operation for URLSearcher.
    }
    
    set res1 [list "Google" "www.google.com" "google website"]
    set res2 [list "Yahoo" "www.yahoo.com" "yahoo website"]
    set res3 [list "MIT" "web.mit.edu" "mit website"]
    return [list $res1 $res2 $res3 $query]

ad_proc -private paged_search_url {
    query
    results_per_page
    page_num
    }
    
    Implements the paged search operation for URLSearcher.
    }
    
    #initialize results list
    set results [list]
    
    #set pagination vars
    set pagemaxval [expr $page_num * $results_per_page]
    set pageminval [expr $pagemaxval - $results_per_page + 1]
    
    regsub -all "*" $query "" url_query
    if { [catch {set content [ns-httpget http://dspace-
domo.mit.edu:8080/SRW/search/DSpace?q=%22$url_query%22&maximumRecords=$results_per_page&startRecord=$pageminval]} errMsg] }
        return
    
    #set doc [dom parse $content]
    if { [catch {dom parse $content doc} errMsg] }
        return
    
    set root [$doc documentElement]
    
    set recordTitle ""
    set recordUrl ""
    set recordDesc ""
    set numrecords 0

    foreach child [$root childNodes] {
        set childName [$child nodeName]
        if [string equal $childName numberOfRecords] {
            set numrecords [$child text]
        } elseif [string equal $childName records] {
            foreach recordsChild [$child childNodes] {
                set recordsChildName [$recordsChild nodeName]
                if [string equal $recordsChildName record] {
                    set recordDesc ""
                    foreach recordChild [$recordsChild childNodes] {
                        set recordChildName [$recordChild nodeName]
                        if [string equal $recordChildName recordData] {
                            foreach dataChild [$recordChild childNodes] {
                                set dataChildName [$dataChild nodeName]
                                if [string equal $dataChildName srw_dc:dc] {

```
```

B.17 dspace-search-procs.tcl

ad_library {

    The "dspace searcher" searches and retrieves dspace urls.

    @author gtcuervas@mit.edu
    @version $Id: dspace-search-procs.tcl,v 1.0 04/14/04 09:51:04 peterm Exp $
}

namespace eval dspace_search {

ad_proc -private search_url {
    query
    }
    
    Implements the search operation for URLSearcher.
    }
    
    set res1 [list "Google" "www.google.com" "google website"]
    set res2 [list "Yahoo" "www.yahoo.com" "yahoo website"]
    set res3 [list "MIT" "web.mit.edu" "mit website"]
    return [list $res1 $res2 $res3 $query]

ad_proc -private paged_search_url {
    query
    results_per_page
    page_num
    }
    
    Implements the paged search operation for URLSearcher.
    }
    
    #initialize results list
    set results [list]
    
    #set pagination vars
    set pagemaxval [expr $page_num * $results_per_page]
    set pageminval [expr $pagemaxval - $results_per_page + 1]
    
    regsub -all "*" $query "" url_query
    if { [catch {set content [ns-httpget http://dspace-
domo.mit.edu:8080/SRW/search/DSpace?q=%22$url_query%22&maximumRecords=$results_per_page&startRecord=$pageminval]} errMsg] }
        return
    
    #set doc [dom parse $content]
    if { [catch {dom parse $content doc} errMsg] }
        return
    
    set root [$doc documentElement]
    
    set recordTitle ""
    set recordUrl ""
    set recordDesc ""
    set numrecords 0

    foreach child [$root childNodes] {
        set childName [$child nodeName]
        if [string equal $childName numberOfRecords] {
            set numrecords [$child text]
        } elseif [string equal $childName records] {
            foreach recordsChild [$child childNodes] {
                set recordsChildName [$recordsChild nodeName]
                if [string equal $recordsChildName record] {
                    set recordDesc ""
                    foreach recordChild [$recordsChild childNodes] {
                        set recordChildName [$recordChild nodeName]
                        if [string equal $recordChildName recordData] {
                            foreach dataChild [$recordChild childNodes] {
                                set dataChildName [$dataChild nodeName]
                                if [string equal $dataChildName srw_dc:dc] {

```
foreach dcChild {$dataChild childNodes} {
    set dcChildName [lindex $dcChild nodeName 0]
    if [string equal $dcChildName dc:identifier.uri] {
        set recordUrl [lindex $dcChild text 0]
    } elseif [string equal $dcChildName dc:title] {
        set recordTitle [lindex $dcChild text 0]
    } elseif [string equal $dcChildName dc:description] {
        append recordDesc " "
        append recordDesc [lindex $dcChild text 0]
    } else {
        set result [list $recordTitle $recordUrl $recordDesc]
        lappend results $result
    }
}
return [list $numrecords $results]
B.18 google-search-procs.tcl

ad_library {

    The "google searcher" searches and retrieves google urls.

    @author gtcuevas@mit.edu
    @version $Id: dspace-search-procs.tcl,v 1.0 04/14/04 09:51:04 peterm Exp$
}

namespace eval google_search {

    ad-proc -private search-url query
    implements the search operation for URLSearcher.
    set resl [list "Google" "www.google.com" "google website"]
    set res2 [list "Yahoo" "www.yahoo.com" "yahoo website"]
    set res3 [list "MIT" "web.mit.edu" "mit website"]
    return [list $res1 $res2 $res3 $query]

    ad-proc -private paged_search-url query
    implements the paged search operation for URLSearcher.
    #initialize results list
    set results [list]
    #set pagination vars
    set google_page_num [expr $page_num - 1]
    set start_index [expr $google_page_num * $results_per_page]
    #set google soap variables
    set endpoint http://api.google.com/search/beta2
    set schema http://www.w3.org/2001/XMLSchema
    set Key {orDfgkBQFHKCjAlmJ3TqqHksuu+SUmZm}
    #google soap method call
    ::SOAP::create doGoogleSearch
    -proxy $endpoint
    -params {key string q string start int maxResults int
    filter boolean restrict string safeSearch boolean
    lr string ie string oe string}
    -action urn:GoogleSearchAction
    -encoding http://schemas.xmlsoap.org/soap/encoding/
    -schema [list xsd $schema]
    -uri urn:GoogleSearch
    set unparsedresult [doGoogleSearch \$Key \$query \$start_index \$results_per_page \false \"
    false 
    "utf-8 \utf-8]

    #-- parse the results
    set resultTagIndex [lsearch -exact $unparsedresult resultElements]
    set resultsIndex [expr $resultTagIndex + 1]
    set resultList [lindex $unparsedresult $resultsIndex]

}
set totalNumResultsTagIndex [lsearch -exact $unparsedresult estimatedTotalResultsCount]
set totalNumResultsIndex [expr $totalNumResultsTagIndex + 1]
set totalNumResults [lindex $unparsedresult $totalNumResultsIndex]

foreach record $resultList {
    set urlTagIndex [lsearch -exact $record URL]
    set urlIndex [expr $urlTagIndex + 1]
    set titleTagIndex [lsearch -exact $record title]
    set titleIndex [expr $titleTagIndex + 1]
    set snippetTagIndex [lsearch -exact $record snippet]
    set snippetIndex [expr $snippetTagIndex + 1]
    set title [ns-striphtml [lindex $record $titleIndex]]
    set url [ns-striphtml [lindex $record $urlIndex]]
    set snippet [ns-striphtml [lindex $record $snippetIndex]]
    set result [list $title $url $snippet]
    lappend results $result
}
return [list $totalNumResults $results]

ad-proc -private restricted_paged_search_url {
    query
    results_per_page
    page_num
    restriction
} {
    #initialize results list
    set results [list]
    #set pagination vars
    set google_page_num [expr $page_num - 1]
    set start_index [expr $google_page_num * $results_per_page]
    #set google soap variables
    set endpoint http://api.google.com/search/beta2
    set schema http://www.w3.org/2001/XMLSchema
    set Key {orDgkBqF9KHCAInJ3FqqHksuw+SUmZm}
    #google soap method call
    ::SOAP::create doGoogleSearch
    -proxy $endpoint
    -params (key string q string start int maxResults int \
        filter boolean restrict string safeSearch boolean \
        lr string ie string oe string)
    -action urn:GoogleSearchAction
    -encoding http://schemas.xmlsoap.org/soap/encoding/ \
    -schema [list xsd $schema]
    -uri urn:GoogleSearch
    set unparsedresult [doGoogleSearch
        $Key \
        $query \
        $start_index \
        $results_per_page \
        false \
        $restriction \
        false \
        "" \
        utf-8 \
        utf-8]
    #-----parse the results
    set resultTagIndex [lsearch -exact $unparsedresult resultElements]
set resultsIndex [expr $resultTagIndex + 1]
set resultList [lindex $unparsedresult $resultsIndex]

set totalNumResultsTagIndex [lsearch -exact $unparsedresult estimatedTotalResultsCount]
set totalNumResultsIndex [expr $totalNumResultsTagIndex + 1]
set totalNumResults [lindex $unparsedresult $totalNumResultsIndex]

foreach record $resultList
    set urlTagIndex [lsearch -exact $record URL]
    set urlIndex [expr $urlTagIndex + 1]
    set titleTagIndex [lsearch -exact $record title]
    set titleIndex [expr $titleTagIndex + 1]
    set snippetTagIndex [lsearch -exact $record snippet]
    set snippetIndex [expr $snippetTagIndex + 1]
    set title [ns_striphtml [lindex $record $titleIndex]]
    set url [ns_striphtml [lindex $record $urlIndex]]
    set snippet [ns_striphtml [lindex $record $snippetIndex]]
    set result [list $title $url $snippet]
    lappend results $result

return [list $totalNumResults $results]
C Instructions for Integration With Other Systems

Here are instructions for how to integrate other systems with the file transfer component. Three major steps need to be completed in order to make the integration:

1. Fill in the database tables via the Add Schema interface.
2. Add the code files for the submit interface.
3. Add the implementation for the search service contract.
4. Add the code files for the retrieve interface.

C.1 Filling in the database tables via the Add Schema Interface

1. Go to the add schema interface at: http://your_url/dotlrn/file-storage/schema-add
2. Enter the schema name and the number of fields. For example, enter “JJ Digital Repository” for schema name and “3” for number of fields as shown below:

   ![Add Schema Interface Screenshot]

   Click on the “Add Schema” button when finished.
3. Fill in the field information with the metadata field information of your metadata schema. The information queried is as follows:

   [Field Information]

   [Metadata Field Information]

   [Information Queried]

   [Field Information]
a. **Name**: specifies the name of the field.

b. **SloanSpace Mapping**: specifies which SloanSpace field it maps to.

c. **Mapping Type**: which direction the mapping goes.

d. **Required**: specifies whether or not a value for this field must be supplied when submitting into your system being integrated.

e. **Has Multiple Values**: specifies whether or not the field can contain multiple values.

f. **Display Name**: specifies the name of the field displayed in the submit user interface.

g. **Display Type**: specifies the input type of the field value.

h. **# of Options**: this is only relevant if the display type selected is “Select List”.

This specifies how many options the select list will have.

Below are sample values:

4. Click on the “Submit and Continue” button when finished.

5. Fill out the rest of the field information. The information queried is as follows:

   a. **Display Text**: specifies the text appearing on top of the input form, containing instructions for filling out that field.

   b. **Error Text**: specifies the text that appears when this field is filled out incorrectly.
c. **Text Field Size**: specifies the size of the text field, if the display type is “text field”.

d. **Text Area Rows**: specifies the number of rows of the text area, if the display type is “text area”.

e. **Columns**: specifies the number of columns of the text area, if the display type is “text area”.

f. **Option Text and Option Value**: specifies the option text and option values of the select list, if the display type is “select list”.

Below is an example of the field information for jj_author, created above:

![Add Fields (cont.)](image)

6. Click “Submit” when done. This concludes filling out the database tables.

**C.2 Adding the code for the submit interface**

In order to complete the submit portion of the integration, you would first need to provide the code that communicates with your web service method that submits files into your system. For example, say the “JJ Digital Repository” created above has a web service with a method called “SubmitIntoJJ(content, jj_author, jj_title, jj_description)”.

“SubmitIntoJJ” has as input the content, encoded in base64, and the values for the
metadata fields jj_author, jj_title, and jj_description. Thus, the code must contain a call
to this method. Here is a sample of what the tcl code file for the “JJ Digital Repository”
submit component will look like. Let’s name this file “jj-submit.tcl”.

**jj-submit.tcl**

```tcl
ad_page_contract {
  Try meta
} {
  schema_id: integer
  file_id: integer
}

## get the jj_author, title, and description fields
set author [ns_queryget $author_field_id]
set title [ns_queryget $title_field_id]
set description [ns_queryget $description_field_id]

## get the file contents and encode it to a base 64 string
set content [::base64::encode $file]

## call the web service “SubmitIntoJJ” web service method
::SOAP::create SubmitIntoJJ
-uri “http://www.jjdigitalrepository.com/webservice
-name “SubmitIntoJJ”
-params (content string, jj_author string, jj_title string, jj_description string)
SubmitIntoJJ $content, $author, $title, $description

## redirect to the file area
ad_returnredirect “file?file_id=$file_id”
```

Once this file is created, you would now need to call this code when the “Upload” button
is clicked in the submit user interface, if the schema_id specified is the schema_id of your
schema. To do this, you would need to modify the meta-view.tcl file as follows. Look
for the line in meta-view.tcl that says “### set the submit file . . . “. This looks like:

```tcl
### set the submit file
if [string equal $schema_name DSpace] {
  set submit_file_name dspace-submit
}
```

Add to this the following:

```tcl
if [string equal $schema_name <your_schema_name>] {
  set submit_file_name <your_code_filename>
}
```
For example, for the jj_submit.tcl file above, the new piece of code will look like:

```tcl
### set the submit file
if [string equal $schema-name DSpace] {
    set submit_file_name dspace-submit
} else {
    set submit_file_name jj_submit
}
```

C.3 Adding the search service contract implementation

In order to complete the search component of the integration, you must add an implementation of the search service contract that searches your system, through your web service.

To do this, first create the service contract operations to the database. Do this by creating a file called `<system>-search-create.sql`. For example, for “JJ Digital Repository”, create a file called “jj-repository-search-create.sql”. The contents of the file are as follows:

```sql
DECLARE foo INTEGER;
BEGIN
    -- create implementation
    foo := acs_sc_impl.new {
        impl_contract_name => 'URLSearcher',
        impl_name => 'JJRepositorySearcher',
        impl.pretty_name => 'JJ Digital Repository URL Search',
        impl.owner_name => 'jjrepository_search'
    };

    -- create paged search operation
    foo := acs_sc_impl.new_alias {
        impl_contract_name => 'URLSearcher',
        impl_name => 'JJRepositorySearcher',
        impl.operation_name => 'paged_search',
        impl.alias => 'jjrepository_search::paged_search_url',
        impl.pl => 'TCL'
    };

    -- add binding
    acs_sc_binding.new {
        contract_name => 'URLSearcher',
        impl_name => 'JJRepositorySearcher'
    };

END;
/
SHOW ERRORS
```

Copy the file contents above and replace all instances of “JJRepository” with your system name.
Also create the drop file. For example, here are the contents of “jj-repository-search-drop.sql” file:

```
declare
foo integer;
begi
  acs_sc_binding.del(
    contract_name => 'URLSearcher',
    impl_name => 'JJRepositorySearcher'
  );
  foo := acs_sc_impl.delete_alias{
    impl_contract_name => 'URLSearcher',
    impl_name => 'JJRepositorySearcher',
    impl_operation_name => 'search'
  );
  foo := acs_sc_impl.delete_alias{
    impl_contract_name => 'URLSearcher',
    impl_name => 'JJRepositorySearcher',
    impl_operation_name => 'paged_search'
  );
  acs_sc_impl.del{
    impl_contract_name => 'URLSearcher',
    impl_name => 'JJRepositorySearcher'
  };
end;
/*
show errors
```

Copy the file contents above, and replace all instances of “JJRepository” with your system name. Add both these files to your /packages/file-storage/sql/oracle directory.

Now, you are ready to supply the code of the implemented operation. First, create the file <system>-search-procs.tcl, and add this file to your /packages/file-storage/tcl directory. Now copy the contents below:

```
ad_library {
  The "<system> searcher" searches and retrieves <system> urls.
@author gtcuevas@mit.edu
@version $Id: dspace-search-procs.tcl,v 1.0 04/14/04 09:51:04 peterm Exp 5$
}
namespace eval dspace_search {
  ad_proc -private search_url {
    query
  } { 
    Implements the search operation for URLSearcher.
  } { 
    //fill in search code here
  }
}
```
Fill in the code starting at the line “//fill in search code here”, with the code that searches your system.

After this is done, you must now add the radio button for this search implementation. To do this, open “search-url.adp”, and add the following after the line “<input type=radio name="searchtype" value="googledspace"...“:

```html
<input type=radio name="searchtype" value="your_system_name">
```

Now, add the following to “search-url-results.tcl”, after the line “-call_args [list $searchstring $recordsperpage $pagenum “dspace”]”, with the following:

```tcl
} elseif [string equal $searchtype jjrepository] {
    set results [acs_sc:invoke 
        -operation restricted_paged_search 
        -contract URLSearcher 
        -impl JJRepositorySearcher 
        -call-args [list $searchstring $recordsperpage $pagenum "dspace"]]
```

Change all instances of “JJRepository” above with your system name.

Once all these pieces have been implemented, you are now ready to integrate the retrieve component.

### C.4 Adding the code for the retrieve interface

The steps for adding the retrieve interface are as follows. First, create the code file that communicates with your web service method that fetches files from your system’s web service. For instance, say “JJ Digital Repository” has 2 web service methods: GetJJFileContent and GetJJFileMetadata. Both these methods have as input, file_id, which is the id of file you want to fetch. The GetJJFileContent method returns a base64 encoded string containing the file contents, and the GetJJFileMetadata method returns the metadata in XML format. The code file, “jj-get.tcl”, will be as follows:
jj-get.tcl

```tcl
ad_page_contract {Add File} {
    folder_id:integer
    schema_id:integer
}

#call the web service methods
::SOAP::create GetContent
    -uri "http://www.jjdigitalrepository.com/webservice
    -name "GetJJFileContent"
    -params {file_id string}
::SOAP::create GetMetadata
    -uri "http://www.jjdigitalrepository.com/webservice
    -name "GetJJFileMetadata"
    -params {file_id string}
set content GetContent $file_id
set metadata GetMetadata $file_id

#parse the metadata file to get the parameters
#add the file into the folder with folder_id = $folder_id
```

Once this file has been created, modify the “search-url-results.adp” file, adding the following line after the line “<b>[Add File]</b>”:

```tcl
<if @searchtype@ eq "your_system_name">
    &nbsp;&nbsp;<a
    href="your_get_file?folder_id=@folder_id@&schema_id=2&itemID=@urls.url@&title=@urls.title &description=@urls.description@"
    >
    <b>[Add File]</b></a></if>
```

Replace “your_system_name” with your system name, and “your_get_file” with the get code file you created in the previous step.

Once this is all done, restart your .LRN server. Enjoy!!!
D Installing the system into .LRN

2. Unxip Thesis.tar
3. Go to the main directory, “Thesis”
4. “Thesis” contains 3 directories:
   a. file-storage
   b. fs-portlet
   c. sql
5. Go to file-storage, and do the following:
   a. Copy all the files in file-storage/sql/oracle/ and place them in your .LRN packages/file-storage/sql/oracle/ directory
   b. Copy all the files in file-storage/sql/oracle/ and place them in your .LRN packages/file-storage/sql/oracle/ directory
   c. Copy all the files in file-storage/sql/oracle/ and place them in your .LRN packages/file-storage/sql/oracle/ directory
   d. For all the files in file-storage/www/Modified, copy them and paste them into your .LRN packages/file-storage/www/ directory, replacing all the original files in .LRN with these modified files.
   e. Make a directory called dspace-temp into the .LRN packages/file-storage/www/ directory, and set permissions so that the directory is writable by all users.
   f. Copy all the files in file-storage/www/Modified/resources/ and place them in your .LRN packages/file-storage/www/resources/ directory.
6. Now, go back up to the fs-portlet directory in “Thesis”, and do the following:
   a. Copy the file in fs-portlet/www/Modified/ into your .LRN packages/fs-portlet/www/ directory (replacing the original .LRN file with this modified file).
7. Restart the server.