Design and Implementation of a Digital Library

by James Richvalsky and David Watkins

Introduction

Have you ever wondered what a digital library is or how you would go about creating your own digital library? By describing the design and implementation process we followed in creating our own digital library, we intend to give the reader a simple definition of what a digital library is, and present some key design issues involved in creating a digital library. We focus on a UNIX based implementation, but we also include highlights of PC-based libraries.

Background

What is a Digital Library?

A **digital library** is a collection of information that is stored and accessed electronically. The information stored in the library should have a topic common to all the data. For example, a digital library can be designed for computer graphics, operating systems, or networks. These separate libraries can be combined under one common interface that deals with computers, but it is essential that the information contained within each library remain separate. The purpose of a digital library is to provide a central location for accessing information on a particular topic. The last thing a user wants to happen when he searches for information about computer graphics is to get information on operating systems. A digital library must keep topics separate, otherwise it would be totally useless. A digital library should also have a user interface that is easy to use.

A Digital Library for Computer Science Educators

The digital library we helped to develop is entitled "The Computing Laboratory Repository" [6]. It is endorsed by the ACM Special Interest Group on Computer Science Education (SIGCSE). This repository supports the collection of laboratory materials developed by computer science educators for use in curriculum development. This site (http://springfield.tcnj.edu:8000/~compsci/index2.html) currently supports the online submission of laboratory abstracts and permits retrieval via keyword searching. There are currently two separate sections of our digital library: the peer-reviewed and the editor-reviewed sections. The peer-reviewed section consists of on-line laboratories that are refereed by

a group of peer reviewers. The editor-reviewed section also contains on-line laboratories, but these labs are reviewed only by the editor of the site.

The SIGCSE Working Group on Closed Labs originally discussed the initial need for our laboratory repository in the summer of 1995 [1]. Work has continued in this area; specific issues addressed include the development of guidelines for submission [4] and increasing the accessibility of the library [5].

Design and Implementation

Requirements Specifications

One of the first design issues in the creation of a digital library is to prepare a list of high-level requirements. This list includes what information the library will contain, how that information will be generated, what audience the information is intended for, and how the data will be accessed. For our project, we wanted to create a repository for computer science laboratories. Instructors throughout the world should be able to visit a web site and submit information about the labs they use in their classrooms. From this description we see that: the information for the library should be gathered through on-line submissions; the intended audience is computer science educators around the world; and the data should be accessible through the web. All of these issues must be considered in the development of a digital library. A clear plan must be developed before one starts the detailed design and development of the library.

Hardware - PC or UNIX

Another important issue that needs to be assessed before site development begins is the storage location of the digital library. The web server must have access to the Internet, ample hard drive space, and the ability to handle the expected access load. Preferably, the computer will have a T1 or better connection to the Internet. This will allow faster access for users. If a PC is used, it should be at least a 100 MHz Pentium. Similar speeds are recommended for UNIX machines. Ample disk drive space for library materials is also recommended. We recommend at least 100MB of free disk space. This will give your library plenty of room to expand.

Once an architecture has been chosen, a web server must be chosen. It is common for users who have opted to use a PC to be running Windows NT. Two of the most popular web servers for this operating system are Microsoft's Internet Information Server and Netscape's Enterprise Server. If a UNIX machine was chosen, one of the numerous free web servers that can be downloaded from the web, such as NCSA HTTPd Web Server [7] (which we are currently using), can be used.

The choice of which hardware and supporting software should be used is an important decision. Many people consider UNIX machines to be more reliable than PCs. Also, the software for UNIX machines has been in use for many years (and thus is well tested), whereas PC software is relatively new. We

chose to use a UNIX machine attached to a T1 line because of the reliability and speed of this arrangement. We are currently using a SUN Ultra Workstation running SunOS 5.6. We have also experimented with a SUN Sparc Workstation running SunOS 5.5.1.

Storage - Database or File Structure

The next big decision to be made is how to store the files that will comprise the digital library. There are a number of options available. Two possible methods are using a database and creating a special directory and file structure. There are many databases to choose from. Examples include Oracle, MSQL, and Microsoft Access. If you use a PC, Access would probably be the database of choice, as it would be the most compatible with NT. Oracle is a good choice for a database on a UNIX machine. It is very reliable and fairly flexible. There are some problems that arise from using a database, however. The main problem involves the database's lack of flexibility. After creating a database, if a new field needs to be added, a new database needs to be created. On a number of systems, the transfer of data from the old database to the new one can be difficult and time consuming.

Instead of using a database, we chose to create our own special directory and file structure. For each laboratory, we created an abstract file. This file included information about the lab, such as the author's name, the title of the lab, the subject of the lab, and other relevant information. For each lab, we dynamically created a directory whose name is composed of the author's name and the lab title. This ensured a unique directory name for each lab. This can be accomplished using the following code fragment written in PERL:

```
     # retrieve information which was passed into the CGI script
     $authors = $in{'authors'};
     $title = $in{'title'};

     # convert spaces to underscores using string manipulation
function
     $authors =~ s/ /\_/g;
     $title =~ s/ /\_/g;

     # combine the author and title fields, separated by a '-' to
create a directory name
     $dir = $authors.``-''.$title;

     # create directory with correct file permissions
     mkdir(``$dir'', 777);
```

All files associated with each lab are stored in the appropriate directory. This design is more flexible than using a database. For example, new fields can be added by changing the CGI script only.

An example of the directory structure we created is shown below. Notice that the name of the directory

where the lab is contained consists of the author's name and the title of the laboratory. Included in this directory is a file which contains the lab abstract (lab) and other associated files. In this example, a Microsoft Word document and an image file are included.

```
     ./labs/official_labs/David_Watkins-
Implementation_of_a_Digital_Library/lab
     ./labs/official_labs/David_Watkins-
Implementation_of_a_Digital_Library/paper.doc
     ./labs/official_labs/David_Watkins-
Implementation_of_a_Digital_Library/image.jpg
```

Accessibility - Java or Standard HTML / Hi-Res or Lo-Res

The next major concern involves accessibility for users. This involves deciding on the target audience. It must be determined if the website should be accessible by everyone regardless of system specifications, or only by people with more advanced hardware and software. With the advances in computers, there is now a greater disparity between the connection speeds of users. The spectrum of computers ranges from those using 9600 baud modems to computers connected directly to a T3 line. This is a problem because it can affect the number of graphics that can reasonably be included on a website. For computers with very slow connections, graphics can become an unwanted nuisance and may cause users to turn away from a site.

To handle this problem, we chose to create separate low-resolution and high-resolution sections for our website. The low-resolution section contains HTML 2.0 webpages with no graphics. There is also no Java, JavaScript, or frames located on these pages. This page loads fast enough to be usable over all connections, and can even be viewed on text only browsers, such as Lynx. We also created a more advanced high-resolution site that contains frames, graphics, and JavaScript. This is more visually appealing (and arguably more functional); however, the content is the same as the low-resolution section.

Search Engines - Write Your Own or Commercial

An important part of every digital library is the search engine. There are a number of different options to choose from. You could take the time to create your own search engine or download a free search engine from the Internet. Some of the advantages of downloading a search engine include speed, price, and time. We tried writing our own search engine, but it could not compete with the one we downloaded from the web. The commercial search engine that we are using is WWWWais 2.5. You can find this engine available at no cost on the web. This search engine can handle Boolean variables and automatically generate links to search results.

Security - File Privileges

A lot of time and effort is needed to successfully secure your website; we will just mention a few key

security issues when using UNIX. First, the biggest problem we encountered was dealing with file permissions. Our site had to be able to create files on-the-fly in order to store the information that was gathered from a webpage. A standard web server runs under the default user ID of ``nobody.'' Any file created with a CGI script will have ``nobody'' as the owner of the file. This turns out to be a security risk because the file permissions have to be set to world readable, world writeable, and world executable. There are a couple solutions to this problem. One solution is to use the ``chown'' command in your CGI scripts. Unfortunately administrator privileges are needed to use this command, and most system administrators will not grant this access. A more viable solution is to install and run a web server and edit the configuration files. This way, the user ID under which the web server runs can be set. We decided to set our server to run as the user ID of the account under which we are developing our site. This is shown below by a code segment from the httpd.conf file of the NCSA HTTPd Web Server [7].

```
     # User/Group: The name (or #number) of the user/group to run
HTTPd as.
     User compsci
     Group dept
```

Another option is to secure certain areas of the website. Both the Apache and NCSA web servers for UNIX feature server-side security access setup. In a configuration file called ``htaccess'', users who are allowed to access certain areas of your website can be defined. When a user browsing the web tries to access a webpage in a secure area, a user login and password box appears. Only a correct login and password will allow the user to access that section of the website. On the NCSA web server, this can be accomplished by adding a file (.htaccess) in the directory you want to secure. The .htaccess file should contain the following lines.

```
     AuthUserFile /home/login_name/.password/.htpasswd
     AuthGroupFile /home/login_name/.password/.htgroup
     AuthName SIGCSE Administration Authorization
     AuthType Basic

     <Limit GET>
     require group name_of_group
     </Limit>
```

Languages

CGI Scripts

A **CGI Script** is used to take information from the web, manipulate it, and return a result. These scripts can be written in a number of languages. The two most popular languages for CGI scripts are C and PERL. We chose to use PERL for a number of reasons. PERL has a syntax which is very similar to C and has a number of string manipulation functions which are very convenient. In addition, it does not

require compilation (since it is interpreted) and it is a ``pure" scripting language. There are free libraries available to handle a lot of the overhead of passing data into the script from a web browser.

The library we used (cgi-lib.pl) to capture the information passed to our CGI script is available at no cost over the web. An example of how we used the cgi-lib.pl library is shown below.

```
     # tell PERL you will be using functions from cgi-lib.pl
     require 'cgi-lib.pl';

     # call the function from the above library that parses and
returns the data that was
     # passed into the script, the function returns the data in an
array with the name of 'in'
     &ReadParse(@in);

     # retrieve the data from the author field and save it in a
variable
     $author = $in{'author'};
```

In order for a script to be executed, the web server has to know that the script is a CGI script. This can be done in two ways. First, you can tell the browser that all files ending with a particular extension are CGI scripts. For example, a web server could be set so that all CGI scripts have to end with the ``.cgi" extension. When the server tries to load a file with that extension, it will think the file is a script, and will therefore try to execute it. If a script was saved with any other extension, the web server would just return the contents of the script as plain text and would not execute the script. The advantage of using this method is that scripts can be saved in any location.

The second method does not give you this freedom. With this method the web server must be told which directories contain scripts. When the server tries to access any file within that directory, it knows that the file is a CGI script, so it tries to execute it. Any script not saved in this directory will be processed as a plain text file. The typical name given to these CGI directories is ``cgi-bin." For the NCSA web server, a cgi-bin can be setup by adding the following line to the srm.conf file:

```
ScriptAlias /~username/cgi-bin/     /home/.../username/www/cgi-bin/
```

This line tells the web server that CGI scripts exist in the ``www/cgi-bin" directory of the ``username" account. The advantage of using this method is that it provides a central storage location for CGI scripts. This method also gives the administrator more flexibility by allowing him to designate which users are allowed to use CGI scripts. If the administrator does not want someone to have this privilege, he will not give them a cgi-bin directory.

JavaScript

JavaScript was created by Netscape to be used in its browser to support more advanced websites than HTML allows. One feature of JavaScript is the ability to determine what browser the user is running and load a webpage that would better suit this browser. JavaScript can also make websites more dynamic. We used JavaScript to add visual effects to our website, such as the highlighting of textual information when the mouse cursor is placed over it.

The following code segment shows how the browser version can be checked using JavaScript:

```
    <script language=``JavaScript''>
    bname = navigator.appName;
    bvers = parseInt(navigator.appVersion);
    if (bname == ``Netscape'' && bvers >=3)
    {
    # insert function which you want to call when the client browser
is Netscape
    # version 3 or greater
    }
    </script>
```

Conclusion

There are a number of issues that need to be accounted for when creating a digital library. The most important issues are security and flexibility. If a site is not secure, important data can easily be lost or corrupted. If a site it not flexible, hours will be spent making small site changes.

Another important issue is maintaining the quality of the data in the digital library. If a digital library accepts all submissions, it will probably contain a lot of useless information. The way we tackled this issue was by having either the editor of the site or a group of peer reviewers critique the newly submitted information.

Finally, the most important issue is the ever changing technology. The World Wide Web is still very young and advances continue to make development much easier. It is important to keep up with all of the standards that are being made for the web (which are being created by agencies like the World Wide Web Consortium [8]). An example of this is the Dublin Core [2]. The Dublin Core is a 15-element metadata element set intended to facilitate discovery of electronic resources. Metadata is simply ``data about data" [3]. It is a methodology and language for describing online learning resources that will facilitate effective searching.

The 15 elements of the Dublin Core include: Title, Subject, Description, Source, Language, Relation, Coverage, Creator, Publisher, Contributor, Rights, Date, Type, Format, and Identifier. The meta tags are HTML code that are placed within the head section of an HTML document. An example of two of the

tags is shown below.

```
<META name=``DC.creator'' content=``John Doe''>
<META name=``DC.subject'' content=``Digital Libraries''>
```

The ``DC'' indicates that the meta tags are part of the Dublin Core element set. The first tag indicates the creator of the document, in this case John Doe. The second tag is the subject of the document, which in our example is ``Digital Libraries.'' All of the other elements in the Dublin Core follow the same pattern. For example, if a tag indicating the title of the document was to be created, the name of the tag would be ``DC.title.'' The title of the document would be placed in the content field of the tag created.

By keeping up to date with the latest technologies and standards, a digital library that will be useful for thousands of visitors can be created.

References

1

L. Cassel, H. Taylor, chairs. SIGCSE Working Group on Closed Labs *NECC*, Baltimore, Maryland, June 1995. Unpublished manuscript.

2

Dublin Core, http://purl.oclc.org/metadata/dublin_core/, visited April 15, 1998.

3

IMS Meta-data, http://www.imsproject.org/metadata/, visited April 15, 1998.

4

D. Joyce, D. Knox (joint chairs), J. Gerhardt-Powals, E. Koffman, W. Kreuzer, C. Laxer, K. Loose, E. Sutinen, A. Whitehurst ``Developing Laboratories for the SIGCSE Computing Laboratory Repository: Guidelines, Recommendations, and Sample Labs Report of the Working Group on Designing Laboratory Materials for Computing Courses,'' ACM ITiCSE 97, Uppsala, Sweden, June 1997, SIGCSE Special Issue 1997, pp. 1-12.

5

D. Knox, ``Enhancing Accessibility of Lab Materials,'' SIGCSE Bulletin, Vol. 29, No. 4, December 1997, pp. 20-21.

6

D. Knox, ``SIGCSE Computing Laboratory Repository,'' http://springfield.tcnj.edu:8000/~compsci/index2.html, visited April 15, 1998.

7

NCSA HTTPd Web Server, http://hoohoo.ncsa.uiuc.edu/docs/Overview.html, visited April 15, 1998.

8

World Wide Web Consortium, http://www.w3.org/, visited April 15, 1998.

James Richvalsky and David Watkins are students at the College of New Jersey and can be reached at richvals@barney.tcnj.edu and watkins@barney.tcnj.edu, respectively.