Multimedia Data Access and Storage

Mang Tak Rayson Chan

490.490DT Project In Information Technology

Supervisors:
Reinhard Klette, James Harper

CITR Tamaki
The University of Auckland
Tamaki Campus
Private Bag 92019
Glen Innes, Auckland
New Zealand

27th October 2000

Contents

Chapter 1

Acknowledgement

I would like to thank Reinhard Klette for inspiring the idea of building this system.

James Harper for helping to install the components and spending time on solving some tricky problems with the system, especially when getting some components compiled on the Digital Unix box this system have to implemented on.

Also I would like to thank Vincent Chung for giving advice on running a web server, security and web application.

Finally I would like to thank my good friend Yuk-Fai Cheng for his advice on Java applet and JDBC. Also countless hours on debugging the Java applet.

Rayson Chan
24/10/2000

Chapter 2

Introduction

2.1 Project Aim

As more multimedia items are available a systems must be implemented to keep track of, locate and extract these multimedia items efficiently. The main concern of this project is to produce a core system for building upon more advance storage structure and searching techniques.

One problem this system need to address is the portability issue, where different system architectures will be accessing this system, such as PC and Mac just to name a few. To make sure this system is available for as many platforms as possible the system is to be implemented using the web interface, and made accessible across an intranet.

Because of the multimedia nature of the objects this system will be dealing with, the system must recognise the differences between specific type of media or format (frequently known in this document as the *datatype*). Since the system will also need to let the user preview an item while searching before downloading the full size item, the system must also change accordingly for a particular datatype when embedding the preview in the HTML page which will be displayed to the end user.

Chapter 3

System Analysis and Design

To aid the design phase and help understand the overall system an Object-Oriented Analysis and evaluation was used throughout the entire duration of the project. The analysis was proved to be extremely useful for identifying flaws in the intermediate designs, producing an integral system and help understanding the problem domain more clearly. This is especially important on a system of this scale. The OOA design for this problem domain also make it possible for reusing of the results in future implementations [4, Pp 11–12].

3.1 Requirements

The requirement for this system is a multimedia search engine and navigation system. However, full-scale hypermedia system like Hyperwave[1] would be unnecessary and outside the scope of this project. In fact, time and resources available for the project made that impossible. Full-scale hypermedia frequently employ acyclic graph structure and automatic linking/unlinking. These features require sophisticated management system which will require a great deal of development effort.

Keyword search and hierarchical browse was navigation styles available in the core system. A media object management system was implemented for adding and updating the media objects in the system. A user database will also required for the authentication and logging of users on the system. The user database can be expanded to provide additional services such as setting the permission values on a particular object.

The system should be easy to use. As all transactions happen over the web a balance between amount text and graphics must be maintained, to leverage aesthetic and performance.

[1] `http://www.hyperwave.com`

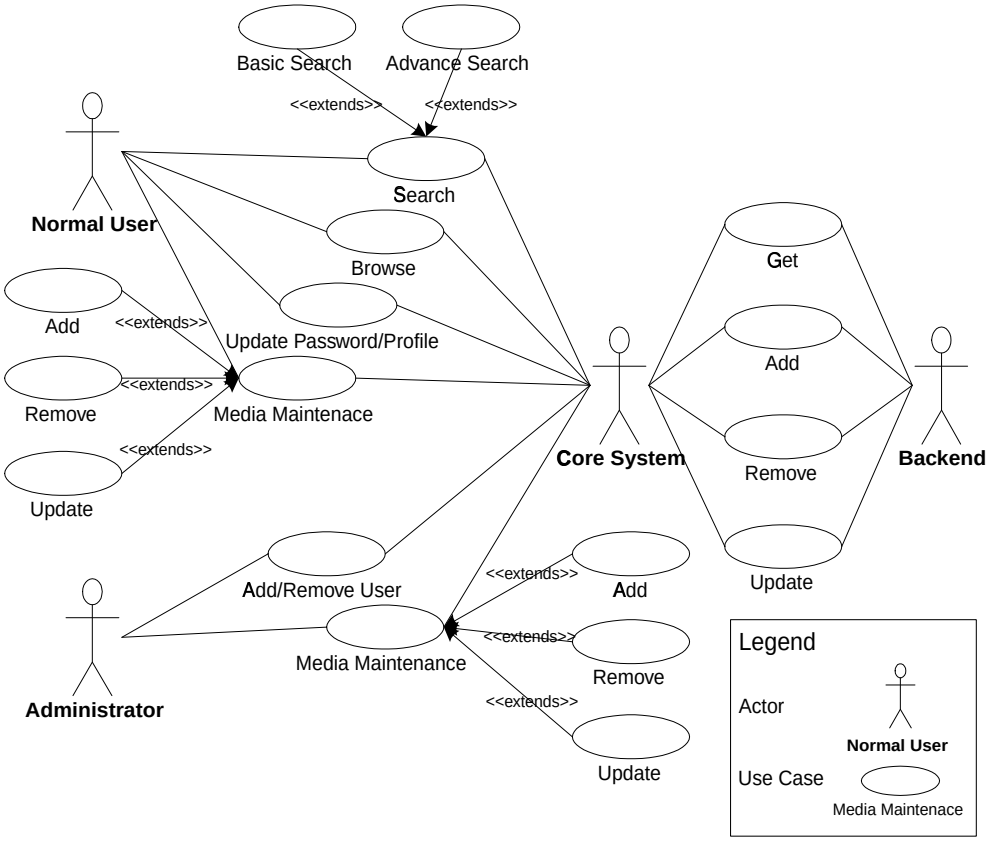

Figure 3.1: Use Cases of the system.

3.2 Use Cases

To best illustrate the interaction between different parts of the system a use cases diagram [15, Section 3 Part 6] is shown in **Figure 3.1**. A stick figure is an *actor*. Actors are different parties of the system. An ellipse represents *use case*. They are the interactions or events between actors.

Some of the interactions such as *search* are comprise of a number of sub-interactions, indicate by an arrow line labeled "<<extends>>". Also, *search* and *browse* are both part of the navigation styles available in the core system.

Important divisions of this multimedia search and management system are:

- User Interface
- Core System
- Backend

The diagram shown the core system is a key part of the entire system. It handles all the requests from user and it is also be the only interface

between users/administrator and the backend, since users are not allowed to deal directly with the database. The core system will also produce the graphical user interface needed for the end system. This is done to shield the user from the internals of the system, such as SQL queries to the database. In order to protect the system, the interface also defines specific tasks which can be accomplished by the user.

The backend of the system is for storing all information. There are only a limited number of basic functions on the backend, namely *get*, *add*, *delete* and *update*. The core system must have knowledge of the backend to used those functions.

3.3 Design

The system takes the client-server approach with thin client, which means the client-side processing is very limited. This is the path taken for most web applications, as it is simple yet robust, and could provide much more control over presentation and processing. With the use of Java applet, "smarter" thick client could take away some trivial processing off the server.

Figure 3.2 shows division of functions within a typical client-server application [17][Pp 654–657]. The three major components are:

Input/Output Component Formats and presents data in output devices. Uses *presentation logic* to manage the graphical user interface and data formatting.

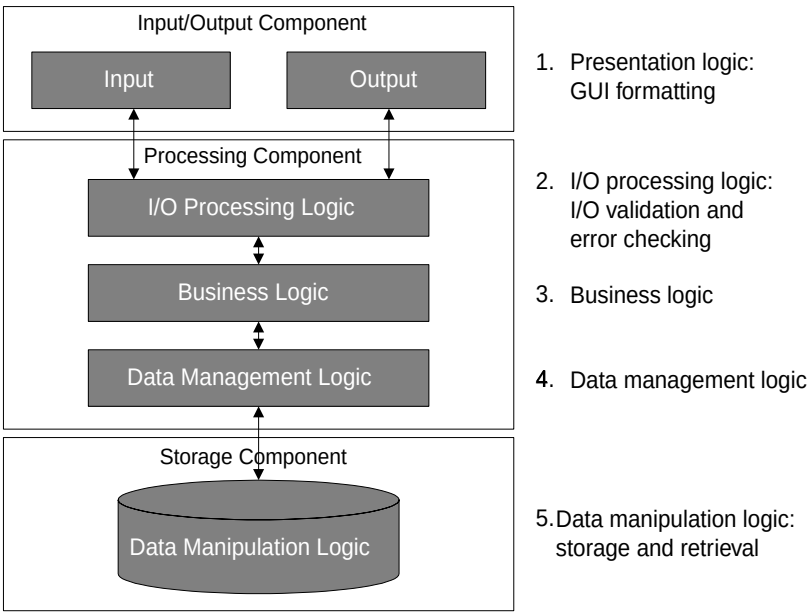

Figure 3.2: Client-Server application components. [17]

Processing Component Application code that performs data validation, error checking, and so on.

- *I/O processing logic* manages data entry validation and basic error checking.
- *Business logic* is applied through the code that represents the business rules. Business rules is a description of a policy, procedure, or principle within a specific business environment.
- *Data management logic* determines what data are needed for each transaction.

Storage Component uses *data manipulation logic* to deal with the actual data storage and retrieval from the physical storage devices.

The division within the system can roughly map to the outline in **Figure 3.2**. With the thin client approach much of the processing and logic are concentrated at the core system. Some of the Input/Output Component are also implemented in the core system.

The backend of the system roughly corresponds to the Storage Component in **Figure 3.2**.

Chapter 4

Specifications

4.1 General

4.1.1 Media

Because this application will require to display and process a large amount of multimedia data of various formats, and the user requirement leave the new media type implementation open for future expansion. The number of media categories this application can handle at the moment shall limited to only one:

- Image

It is possible to modify the system to support more datatypes. Core system must be modified to support new media types. Other media categories we had in mind and could expand support into for the future are:

- Text
- Video
- Sound
- Application
- Model

Although there is only one supported media category is at moment, more multimedia support can be added in the future. In fact, both the back-end design and the web interface front-end are capable to accommodating more variety of media formats. The system is also able to store unknown types with only basic object details.

To be consistent with the media type representation on the Internet the system uses MIME [7] (Check Appendix for current IANA registered MIME

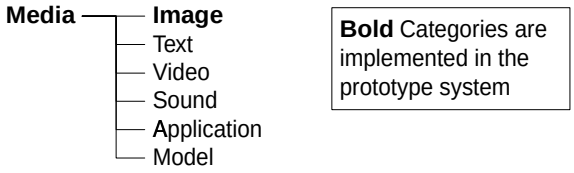

Figure 4.1: Media Hierarchy.

types). All object are associated with a MIME datatype. This scheme simplifies the uploading and displaying of media item on browser. Media categories in database also follow the Top-Level Media Types in MIME. Some media types in MIME are not supported, Composite types such as Multipart and Message are not supported.

For the purpose of keeping track of recognised datatypes in the system there is a datatype database to store all recognised types and formats and their MIME type. This database also have other purposes such as recalling the HTML tags for displaying a particular type of object on the user web browser. More specific uses of the information in this datatype database in the context of UI can be found in section 4.4.1, page 20 and for backend in section 4.2, page 12.

The system currently does not have common formats for storing full-size media object under a particular media category. It is up to the browser to present the media or save it. The amount of time and processing power needed to turn all full-size objects and previews into common formats far outweighs the advantages. It also takes time to evaluate which format is appropriate in each media category, taking into account all the factors such as storage requirement, cost, possibility of converting it from other formats. For example, in a case where all the images are stored in the system as Photo CD format. If a user attempt to upload a large image in non-Photo CD format, the system will have to convert it. This task is subject to the availability of conversion program, and a trade off between processing power and limited level of convenience. The value of the system can greatly reduced if only limited number of formats are supported.

4.1.2 Recommended Formats for Preview

Although the system is able to handle any datatypes, some datatypes are discuss here to ensure usability within most web browsers. This is mainly applies to *Preview Object* and not *Full-Size Object*. See 4.3.1 for more information about Preview Object and Full-Size Object.

There are many format and rendering techniques in digital images. Keeping that the capabilities of a browser in mind, we decided that only the most common formats should be used. For raster images, the system should support JPEG which is lossy but is suitable for pictures and is able to achieve

high compression rate without obvious defects that is detectable to human eye. The format is chosen based on the fact that it is flexible in terms of compression rate, file size and the trade off in image quality.

Lossy JPEG encoding alone may not be enough to satisfy all image requirements. Many images in the final application will likely to require lossless images. For this there are currently two formats that are by far the most common, GIF and PNG [2]. Both are able to handle alpha layer and are popular formats. However there is one shortcoming with PNG. Although it is a more advance format than GIF this format is not well supported, probably due to the complexity. Most image editors or viewers do not have all the functionality of PNG built-in. Hopefully this issue will be resolved in near future.

Even though there are plans to support panoramic images in the application (e.g. Quicktime VR) we encountered certain problems. First, these are often not open standards and therefore methods for producing such images may not be accessible cheaply and effectively (for example, special encoding software). Second, because this is likely to be a propriety format the possibility of extending or integrating the format may not be possible. Third, because of the failure to integrating or extending the software the end user may be bounded to install a special program or plug-in for viewing this format. For these reasons we decided to leave this option open for further investigation.

The current system is have limited support for text documents. The idea of searching is to look at only specific fields in the database and not inside the actual document. There are other systems that is available which mainly with structured text document, providing browsing and full text search on a text document. For example, the Greenstone system[1] is able to convert a variety of document format, such as plain text, postscript and pdf into it's own internal format, provide structure and searching facility on that document. Other capabilities including automatic relinking is also in that system. The current system currently do not allow searching within the Full-Size Object, this is also open to future investigation.

The intended application was to incorporate 3D objects and scenes. The 3D support under media category Model was not able to complete in time. The original intention was to convert the full-size object into a VRML[3] preview, because it is the best supported 3D format on browsers. However, other issues also appears as to how to convert other formats to VRML. Also, the searching process may also be a problem because the end user may have to invoke the VRML plugin for the browser to see each search result. User have a choice of providing an alternative preview when adding a new full-size object to the system, or have the system generate an preview if possible. This design feature of custom preview is not only limited to 3D objects but

[1] `http://www.nzdl.org`

is available to all media types.

4.1.3 Collection Hierarchy

The collection hierarchy is how objects in the system are grouped when presented to the user, the physical storage of objects in the system takes a different approach. We will discuss the backend in greater detail in section 4.2.3. Because of this reason collection hierarchy is only significant when the user is navigating or managing media objects.

The collection hierarchy in use right now is only two levels: *Collection* and *Grouping*. This is implemented for prototype and the limit should be eliminated in the future versions to allow more flexibility over the collection hierarchy. It should follow the tree structure collection but should avoid the acyclic graph structure. That is because acyclic graph structure can add a great deal of complexity to the final system and require the system to take over many more tasks, for example, the system will have to maintain referential integrity such as in the Hyperwave and Hyper-G systems [9]. Example would be when an intermediate collection is delete the system is required to automatic relinking of all referencing links, not only the children of the deleted collection. In addition to that, if acyclic graph is indeed used the search engine will have to modify to handle recursive search.

Because of the design of the backend, the use of tree structure must be reinforced by the core system when user is creating new collection structures. Otherwise the linking of collections could go out of control and becomes an acyclic graph, which is undesirable.

4.2 Backend

The backend of this system is responsible for persistent storage of all information. This includes the following sets of information:

User Information Information required to authenticate user, user personal information for making contact, and any user management attributes.

Permission Information This is the permission information for an object. On the final product this will be a matrix between all objects and all user domains.

Datatype & Presentation Information We recognise the difference between different media types. The method for presenting each type could be different, particularly between different media categories (e.g. sound and image). It is important to mark each multimedia object with a type and the tag for presenting such type.

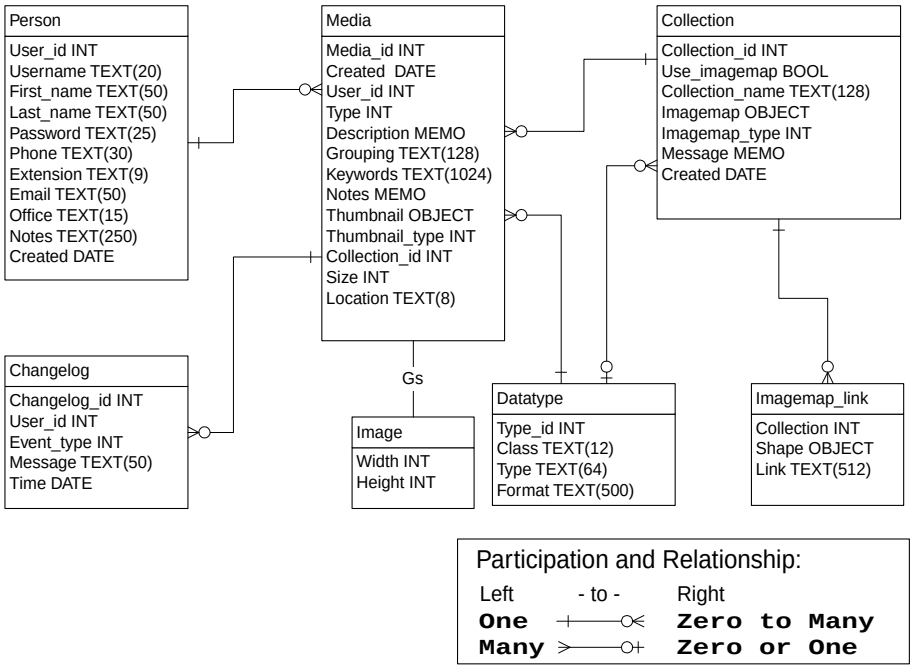

Figure 4.2: Entity-Relationship Diagram using Crow's Foot.

Structural Information This is for keeping the logical structure of the multimedia content in the system. The navigation and display of proper information will depend on the Structural Information.

Object Details The attributes on each object, the creation time, keywords, caption, dimension (in case of image) or run length (in case of video). Some of these fields are also searchable.

Object Data The full-size data and the preview of an object. Since the full-size data may be big some considerations must be made on performance issues.

The backend would be implemented using a combination of file-system and database. That harness both the capabilities of a database and the efficiency of the file-system. Although the data maybe separated consistency of data is still remains a priority.

4.2.1 Database Design

The Entity-Relationship diagram **Figure 4.2** is the layout of the database and is drawn using the Crow's Foot methodology [17, Pp 258–263]. Here is a detailed explanation of the diagram:

Media Represents a single basic Media Object (also refer to as an *object*), the basic object has all the attributes common to any media type or

format. Other Specific types of media, such as video, sound or image, should inherit this basic class. The `Thumbnail` and `Thumbnail` fields are for storing the preview and it's corresponding datatype. Note that all type information are referring to the `Datatype` table. The field `Location` is the file name which the full-size object is stored under.

Image Is the only class extending the base Media class. It has two fields in addition to all the fields appeared in Media.

Person Contains all the personal and system information required to identify a user. Personal information is for contacting the user.

Changelog When an object is updated records are new entries are added in this class. When presenting details for an object this modification records should also shown.

Collection A object has the `Collection_id` field for referring to a collection. This class also has an option to use an Imagemap. The Imagemap is directly saved in the database as binary data in the `Imagemap` field, and it's type information in the `Imagemap_type` field.

Imagemap_link This is useful when the imagemap is active for a collection. When user enabled the imagemap and begin editing the imagemap all link area information are stored here.

Datatype Datatype is the class for describing the MIME type of a piece of data, whether it is a media object, preview or imagemap. It also has the formatting and tag information for properly displaying it in the web browser. User is required to create a new type and provide these information if the system encounter a format foreign to it.

Please note that this database structure is only designed for the prototype system. In the final product the database structure should probably be different.

4.2.2 Search Query

At the moment searching is not handled by the core system. The role of core system is to take user request, then process it into requests the backend can understand, and expect the backend to response with data the core system can understand. Therefore, the backend also taken the task of searching for the right object.

A number of searching techniques were proposed. Some of them are:

Basic Boolean Search Search using `AND`, `OR` and `NOT` with nesting brackets. Searchable fields are restricted to only `Description`, `Grouping` and `Keywords` in the class `Media`.

Advance Search Closely resemble Basic Boolean Search but provide more user controls in HTML forms and instruction on how to use these forms. User will also have control over the output and can restrict the search domain to certain fields or collections.

Similarity Search This is only a proposed searching technique and should be used on only images. It's purpose is to find other pictures with similar features or colour to a particular image.

4.2.3 Special Datatypes and Storage

The system is unique in a way it has to deal with a number of interesting datatypes, which are normally binary and require special attention from the backend system. That is also the reason that motivated us into believing that database alone for storing and retrieving data is not enough. We used both file-system and database for storing data.

Performance and efficiency are both equally important factors when the decision was made. Database is very fast at finding a target, and is a good choice for querying and saving structured information. File-system on another hand is good for retrieving large files and unlike a database, it adds very little overhead, in terms of processing and space efficiency. But file-systems are generally unsearchable unless a specific target is known. The conclusion lead us into using the file-system for keeping reasonably large file. Therefore file-system becomes the best choice for saving *full-size object*, without the added overhead in a database. A *preview* however, has different requirements. A preview is regularly access in a search or when the user is navigating the structure. This calls for speed and searchability over space efficiency, and the preview is best kept in the database as binary data. Also, a typical preview is smaller than the full-size object it represents. Therefore the overhand in space should be insignificant.

When a user need to access a full-size object extra steps must be made to locate and extract the data. For information on locating the full-size object (`Location` field in the `Media` class) the core system must do a search on the database. This is a trade off between using less space and spending more processing time finding the object. This will justify especially when the object may reach upwards of a few megabytes, perhaps tens or hundreds of megabytes, saving the binary data in database will introduce significant amount of overhead.

The *Collection Hierarchy* navigation structure appeared on page 12 is also store alongside with the object details and the preview.

4.3 Core System

4.3.1 Storing Object

When storing a media object the system keep an object in 2 parts: a *full-size object* and a *preview object*. That is done to speed up the display of previews during search. It is possible to keep only one object in the system and create all previews on the fly but the additional processing time and performance penalty would not justify.

As mentioned in the last section, the system is able to create new or save custom preview. When the user choose to create a preview in the system the system should generate a preview through some utilities on the server side from the full-size image given, which user must submit when adding an new object. If the system is not able to generate a preview it will leave the preview blank. In which case the user will be able to submit a custom preview after the object is properly added to the system. The user can also completely bypass the automatic preview generation and submit an alternative preview.

In addition to normal object a collection can also accept an image. Each collection has an imagemap and an imagemap_preview field for storing the background image for the collection imagemap. Note that this 2 fields only accept image and not other datatypes.

There is also the datatype associated with any object (Preview, Full-Size or Imagemap) in the system. When saving the full-size object and during the creation of a preview object, the system finds out about the object's datatype and save that property. This property is used when displaying the object.

There is a way to create new media format (or datatype) in the system. The user must provide the proper MIME type and category for a new format. Also the proper HTML tags for presenting the object in an HTML page is required from the user to properly enable the display of this datatype.

4.3.2 User Management

There are many tasks in user management. Although in the core system the administrator can define many normal users the prototype system do not make many distinctions between normal users except to present their personal information. Since the environments where this system deployed are multiuser environments the user management scheme should expand to include permission and access on objects and collections. There should also be more than 2 classes of users (currently normal user and administrator), where individual user has adjustable privilege assigned by administrator.
The current User Management scheme only make distinction between a normal user and administrator. The prototype system allows administrator to:

	Object 1	Object 2	Object 3
Domain 1	Read	Read	Read/Write
Domain 2	-	Read	-
Domain 3	Read/Write	Read/Write	Read

Table 4.1: Example of a Control Matrix.

- Add User Account
- Remove User Account
- Update User Details/Password
- Update Administrator Details/Password

And functions for normal user are:

- Update User Details
- Update User Password

Proposed permissions, attributes and functions are:

Object Attributes Media Objects and collections in the system should employ an access control mechanism similar to a *control matrix* [18]. Since the Collection/Object and Owner is one-to-one relationship, where as Collection/Object and Domain is many-to-many these two different sets of information should implemented in separate data structure (e.g. two database tables).

- Owner (a Domain) of a Collection/Object

Domain-Collection/Object Attributes This is the *control matrix*, a relationship between a specific object and domain.

- Permission to Read Collection Information & Imagemap Preview
- Permission to Read Object Information, Preview & full-size Object
- Permission to Write Collection Information & Imagemap Preview
- Permission to Write Object Information, Preview & full-size Object

Domain Attributes This is a group of users. Individual user can belong to many domains.

- Permission to Create New Collection

- Permission to Create New Object
- Permission to Update Domain-Collection/Object Attributes
- View Database Only (Designed for public access, override any Permission set).
- Domain members (or users).

Domain Functions Any members of a domain can update attributes in the control matrix if provided they had the right to do so as indicated in Domain Attributes. The domain attributes are however off-bounds to user, only the administrator can modify any Domain Attributes.

- Add New Collection/Object
- Update Domain-Collection/Object Attributes

User Functions These are functions or routines a user can carry out.

- Update User Details
- Update User Password

User Attributes These are attributes associated with a user.

- Permission to Update User Details
- Permission to Update User Password

Administrator Functions The administrator will need the following routines for user management tasks. There are no attributes associated with administrator.

- Add new User
- Update User Details
- Update User Password
- Update Administrator Details/Password
- Update any Domain-Collection/Object Attributes
- Update any Domain Attributes
- Update any User Attributes

4.3.3 Data Consistency

Data consistency is a cooperation between the core system and the backend. It's main purpose is to keep the data in consistent state. Data usually become inconsistent if a transaction, consist of many queries, failed before it is fully completed. This is especially important when the system is access by many users and inconsistent state can occur. An example of such situation can illustrate the idea:

> Two users are trying to update the details for a particular media object at almost the same time (hence two concurrent transactions). The normal approach to take at this point is to lock all the rows in any tables that will involve updating. After the first transaction has ended the second transaction would begin.

The system should be able to lock table so that data is only written to by one transaction at any instance. There is however one problem in this example. The changes made by the first transaction will be overwritten by the second transaction. This is a problem we are aware of but there is little we can do to help. HTTP is inheritly stateless and in order to keep the system simple little effort is made to keep track of individual user and active connections. It is still possible to notify the second user in the example of the changes made by the first user using methods such as putting session ID in the URL or cookies, and notify the server of the session in the updating request. However we will leave this problem at the moment and implement that feature only when there is a demand for it.

4.3.4 Robustness

Core system is the only interface between the user and the system. It is also responsible for many tasks and the system is only as strong as this point, therefore robustness is an issue on any multiuser system. Web applications are particular sensitive to robustness issues, especially when almost all users are on remote machines, and a single mistake can potentially cripple the hosting computer. Both security and stability and part of the problem.
The following events test a system tolerance to mistakes [8]:

1. User unknowingly entered an incorrect or harmful input into the system.
2. Unwelcomed users may gained access to the system and manipulating the data in the system, or vandalise the system making it unusable for legitimate users.
3. Bugs in the coding cause the system to fail.

For 1 and 2 there are several known problems with web-based system. One is the input to the system must be check or filter when necessary. This was implemented on the prototype system. Another is denial-of-service attack the remote user can initiate on the system. The system uses the HTML form-based upload [12], and a continuous stream of data can consume all the available memory on the host system. This should be avoid using a limit on the size of data the user can upload.

We also did exhaustive tests on existing scripts to reduce the number of possible bugs, and tried to eliminate as many as possible in the prototype

system. The number of bugs in the current system should reduced. However most tests must be made to identify any hidden flaws in the system.

4.4 User Interface

4.4.1 Presenting Media Object

Presenting the preview and full-size object to the user also posed some challenges, mainly due to ways different media types are shown on an HTML page (for example, `<IMG>` tag for Image, and `<OBJECT>` for embedded object). Because all objects has an associated MIME-type it is possible to present the media object in appropriate construct inside a HTML page. The tag information is also store in the datatype database.

The user can also download the full-size object from the browser interface. The action taken to whether save, present the object internally in browser or through plug-in will be an decision made by the browser and the user. Normally a web browser would attempt to display a multimedia object through a plug-in where possible, only offer user a choice to save the file when it is unable to display the object internally or with any installed plugin.

4.4.2 Unsupported Media Types

Certain media types may be unsupported on the user web browser. For example, 3D and panoramic images. It is up to the User Interface to decide whether to download a plug-in or save the object. Modern browsers normally have automatic support for downloading plugins.

4.4.3 Timely Response and Interactivity

There is also a need for the system to give quicker response, preferably immediately. There is bounded to be a pause when loading a new page. This is also involved the backend operation.

One possibility of having almost immediately response is to embed `JavaScript` in the HTML page. We experimented with JavaScript, and it is very useful in tasks such as highlighting of fields the actual field input checking still have to be done on the server side because it requires processing not possible in JavaScript. Other HTML page elements can also be manipulated using JavaScript[14], such as layers.

Another solution besides JavaScript to provide interactivity is to use `Java` applets. We also did an actual implementation of Java applet on the system and it is indeed capable of a lot more than JavaScript. It can accomplish tasks such as accessing and manipulating the database, directly

bypassing the core system. However there are certain concerns such as security holes it may open and behavior across different browsers.

These problems with Java and JavaScript will discuss in more depth in Chapter 5.

4.4.4 Consistency

Consistency is also an important part of a hypermedia system. The problem of getting lost in the hypermedia space comes in when the site consist of more than a few pages. Even though many parts of the certain system follows the tree structure with a common parent consistent links and visual clues makes it easier to navigate and understand the structure sooner.

Chapter 5

Implementation

The prototype system implements some of the design features of the system described in Chapter 3. Not of the features were implemented as a fully implemented system would require much more time and resource.

5.1 Components

We chosen these software components when implementing the system:

Backend The backend consists of a database and the file-system. File-system is not named here because there are no specific requirements on file-system.

- `PostgreSQL`[1] 7.0.2, Object-Relational Database

Core System These components are responsible for capturing HTTP requests from the user browser and return HTML output.

- `Apache`[2] 1.3.13, web server
- `ImageMagick`[3] 5.2.3 and 5.2.4, for processing images
- `Perl`[4] 5.004 and 5.005, produces output on user browser through CGI and web server
- Perl Module: `CGI.pm`[5], Simple Common Gateway Interface Class, for formatting HTML and CGI
- Perl Module: `Pg.pm`[6], Perl5 extension for PostgreSQL

[1] `http://www.postgresql.org`
[2] `http://www.apache.org/httpd`
[3] `http://www.wizards.dupont.com/cristy/ImageMagick.html`
[4] `http://www.perl.com`
[5] Included in the `Perl` distribution.
[6] Included in the `PostgreSQL` distribution.

- Perl Module: `HTTPD::UserAdmin.pm`[7], Management of HTTP server user databases
- Perl Module: `Image::Magick.pm`[8], Perl extension for calling ImageMagick's libmagick routines, for interfacing with Image Magick Graphics Manipulation package

User Interface Any Web Browser capable of displaying graphics and understands `CSS Level 1` [11] (Cascading Style Sheets) and `Java 1.1`[9], `JavaScript` is optional. We tested the system with:

- `Microsoft Internet Explorer` 5.0 and 5.5 on Windows NT/2000
- `Netscape Communicator` 4.7 and 4.75 on Windows, Digital Unix and Linux
- `Netscape Navigator` 2.02 on MacOS 8.1

The system was implemented on `Digital Unix` 4.0 and `Redhat Linux` 6.2 platforms. Some problems were encountered when setting up the components. When we started this project we were aware of the fact that compatibility would arise. These problems and other difficulties will be discuss later.

5.2 Backend

This and next 2 sections describes the components and discusses the reasons behind choosing a particular components. Major evaluation factors were cost, compatibility with the design and performance.

5.2.1 PostgreSQL DBMS

PostgreSQL is an object-relational database management system (OR-DBMS) [16]. Concepts such as *class*, *inheritance*, *type* and *functions* where added to the traditional Relational Database Model to make easier to implement in more variety of applications. The database was built on relational database model (therefore *object-relational*) and therefore adapting the database model from other relational database designs would be much easier then converting a relational design to object-oriented database model.

SQL in PostgreSQL

PostgreSQL uses the most widely adopted query language standard in the database world, SQL. The SQL was standardised by ISO and ANSI com-

[7] `http://search.cpan.org`, search for `HTTPD`
[8] Included in the `ImageMagick` distribution.
[9] `http://java.sun.com`

mittees and formally known as SQL/92 in the ratified standard ISO/IEC 9075:1992. A detailed description of SQL/92 was given in [5].

Large Object

Large Object is a binary object in PostgreSQL. Other databases usually call it a BLOB (Binary Large Object). 8192 bytes is the maximum size a normal field can be. A large object is an independent object in the database that can accept value beyond the length of 8192 bytes. The user can create a large object in the database and reference it in a table. When the user no longer needs a large object the object unlinked (or removed) from the database and all references to object cleared. A large object does not necessary correspond to a table and sometimes this can be a problem. Especially when there are large number of large objects the system have to keep track of. There are SQL functions the user can execute to remove orphaned large object, which eases the lost objects problem somewhat.

There are specific functions when working with large objects. It mimics the file system using `import` (like `copy`) an `unlink` (like `delete`).

Evaluation

PostgreSQL was chosen because of it's support for the object-oriented design of the system, especially the class Media and it's subclasses appeared in **figure 4.2** on page 13. Another reason is because the database is open-source and is available at no cost. Another important factor is it uses standard SQL (with some extensions), the impact involve when switching database will be will lower.

PostgreSQL does have some problems. One of those would be some non-standard SQL constructs it contains. Because of it extra functionality a number of non-SQL/92 standard query constructs were added. These constructs may cause problems should the system switch to a new DBMS.

Alternatives

Other databases were also considered. `Interbase`[10] and `mySQL`[11] were other available choices, we did not consider commercial databases because of the potential cost. The two other candidates was either open-source or cost very little and can provide higher performance than PostgreSQL, but both lacking the object-orientedness in PostgreSQL, so we favor PostgreSQL.

[10] `http://www.interbase.org`
[11] `http://www.mysql.com`

5.3 Core System

5.3.1 Apache Web Server

Apache is one of the most popular web servers being used on the Internet[12]. In the widely deployed user base the server is well tested, security holes and bugs are minimised.

Apache also have support for module loading, called *Dynamic Shared Object (DSO) Support*. This function allows apache to "learn" new abilities without recompiling the entire server. New modules are compiled and on server startup, and can be loaded at the administrator discretion. The base distribution of Apache contains some default modules, and more modules are available on the Internet[13].

Mod_include

A number of modules was particularly important in the prototype system, one of those being `mod_include`. It is for parsing *server-side includes*, which is a dynamic HTML page generates on the fly. This is very useful for creating standard format page with common title bar and links, providing consistency within a website.

Mod_perl

Another module that gained some attention was `mod_perl`. Although we did not used the modules in the prototype system previous tests done with the module was impressive. Perl scripts that was used to generate CGI output was greatly improve in terms of speed. It precompiles a Perl script and keep it in memory. When there is a request for that script the compiled script is immediately run, unlike in CGI, which have to load the Perl interpreter to compile and run the script.

Mod_auth

The `mod_auth` was also used in the prototype system. It was used to provide Basic HTTP authentication[6] for the system. It can also used to provide a more secure Digest authentication but support on typical web browser was not available at the time.

Mod_auth_pgsql

Last module we consider using was `mod_auth_pgsql`. It can used add a useful feature to the user authentication scheme by looking up the username and password inside the database. With mod_auth username and password are

[12] `http://www.netcraft.com`
[13] `http://modules.apache.org`

saved in a text file on the file-system, and may cause some consistency problem between user information stored in the database and those in the text file.

Implementation Issues

With the four modules mentioned only mod_include and mod_auth was used in the prototype system. The other modules were not used because certain problems they caused which require more time invest into solving the problems.

One problem with mod_perl is that the programs are cached and sometimes the caching problem produces inconsistent output between different HTTP requests. With mod_perl the server have a tendency to repeat the output of the previous request, regardless of request user or any user submitted parameters. This problem caused some privacy concern on a multiuser system, since there more than one user using the system. The outputs to a number of web browsers can get mixed up.

Another problem is the mod_auth_pgsql module. It is a good idea to keep user authentication information and user person information together in the database but it requires extra Perl packages installed on the system and a number of Perl scripts require modification to write user authentication information directly to the database. This was not done within the limited amount of time.

The basic server is able to generate dynamic HTML page using the CGI interface. This is very simple interface for generating HTML pages. Detailed description of how it works and alternatives please refer to section 6.3 on page 36.

5.3.2 ImageMagick Image Manipulation Package

ImageMagick is one of it's kind. At the time of writing, we were not aware of any other image manipulation packages that works on command line or has Perl interface like ImageMagick does. It is a sophisticated program capable of complicated complex image processing and has many graphics filters. It has support for a large (50+) number of image formats, including some video formats.

Purpose

The main reason this is used in the system is to provide a consistent Preview Image format everyone can understand. Although the full-size object stored in the system do not have a unify format, the preview, on the other hand, must be stored in the limited number of formats a web browser can display. A conversion program is necessary to convert these the full-size objects into *JPEG* preview image, typically seen on the Internet. The program also does

retouching on the image to make an image more aesthetically pleasing. For example, framing the image and add text to the image. There is however some time penalty on the process, but it is a small price to pay for an otherwise impossible task of producing preview images.

Problems

The package does have it's drawback. One of those is it's reliant on external image support. The operating system must contain the shared library during compile, for example, the JPEG library, before it can read or write JPEG images. Another problem is the lacking of plug-in structure. The program requires recompile whenever a new image format is needed.

Implementation

We implemented this only on the Linux platform to produce previews for image objects. The compilation failed on Digital Unix and requires more investigation into the problem.

5.3.3 Perl Interpreter

Perl was a major part of the system, almost all of the dynamic content was generated using Perl scripts. The output of a Perl script is very simple. The input from the user side sent through the CGI interface, and script would pick it up from the CGI interface and begin processing. The content of a HTML page, prefix by the HTTP header, is printed to the standard output of the script. The standard output from the script is captured by the CGI interface on the web server. CGI will be describe in more details on page 36.

Features

There are many reasons why we choose to use Perl. One of which is the ability to manipulating text strings with ease. Perl has built in regular expression matching and string manipulation much like the Unix utilities `awk` and `sed`. This is very important because web forms processing mostly deals with text string and pattern matching. Much of the functions were simplified by the pattern matching facilities.

The powerful yet simple to use pattern matching facility can be demonstrate with one example. In the system, script regularly need to check users input for mistakes, dangerous characters or input length. To check a user input string `userInput` contains only whitespace and alphabetical characters, length fewer than 20 characters we could do this in Perl:

```
$userInput =~ /^[\w\s]{0,20}$/
```

Perl also have a different sense of types. In which a numeric value can convert to a string, or vice versa, without the extra step of explicitly converting it. The concept of true and false is also quite different in Perl. Perl does not officially have a boolean 'true' and 'false' yet. When an expression or a variable is empty or null string, it is false. If it has content then it's true. Sometimes this simplifies the process of evaluating user input.

Portability of Perl is also a factor in the final decision. The Perl interpreter is available across multiple platforms, including Windows, Mac and many variant of Unix.

Because Perl is already widely use, there are numerous packages for extending the capabilities of Perl. Some of which was used in building the prototype system and we shall discuss each modules (also refer as 'packages') in details.

CGI.pm

The CGI.pm module is for managing HTTP or HTML tasks. A non-exhaustive number of tasks are listed below:

- Capturing CGI input
- URL Encoding and Decoding
- Formatting most (if not all) available HTML elements up to version 4.01, with parameters for each element.
- Manage HTTP Upload

These are features of CGI.pm that was used in the prototype system. The CGI.pm also automatically accomplish the mundane task of decoding and returning `query string` (See *Common Gateway Interface (CGI)* on page 36).

There are also a number of built-in safety features in CGI.pm. One of those is the ability to restrict size of a POST request in HTTP, to stop overflow in host from unterminated POST requests. The programmer can also choose to turn off the ability to receive POST request entirely.

Pg.pm

`Pg.pm` is a database driver for PostgreSQL. It is able to establish connection with the database server, execute query and extract results of a query. It is also able to execute large object functions and Pg.pm treat large object much like files streams in Perl. The user can append or overwrite a large object.

The Pg.pm is a propriety driver than contains functions not compatible with other database drivers. There is an initiative to create a common

interface for database access, called *DBI* (Database independent interface for Perl, using the DBI.pm module). Many major databases already have drivers using DBI, including PostgreSQL. This was not used in the prototype system was bad timing. The DBI driver for PostgreSQL was discovered after the system went well under way, and switching driver would require extensive changes to code written. And DBI.pm was not installed in the prototype system.

HTTPD::UserAdmin.pm

This is a module for managing the username:password database for the Apache user authentication. It understands basic and digest authentication scheme stored under text file or database. It has many regularly used username:password management functions such as add user, remove user and recall user password. It is also able to encode a password when adding a new user, both basic or digest scheme. However for database access it requires the DBI.pm modules appeared in the discussion on Pg.pm, and no reason to store password in the PostgreSQL database when Apache cannot use PostgreSQL to authenticate (requires Apache module `mod_auth_pgsql`). The absent of DBI.pm one factor as to why username:password database was not stored in PostgreSQL.

Image::Magick.pm

This Perl module make functions of ImageMagick available for Perl. All of ImageMagick functions can be used and some of which was used in the system:

- Resize
- Border
- Annotate – Add text to image.
- Get – For extracting image attributes, dimension of image was obtained through this function.

Implementation Issues

Perl itself generally works fine. Implementation problems we have to deal with related to Perl are often from Perl modules, and they usually have very few problems. We were able to use all the Perl modules named except for the Image::Magick.pm module. Problem with this module was related to the compilation problem of it's sister program ImageMagick, and it was unusable on one the Digital Unix platform. The problem can be bypass by taking out the automatic preview generation feature from the system.

Alternatives

Other alternatives of Perl we are aware of are *PHP*[14], *Java Server Pages (JSP)*[15] and *Python*[16]. PHP and JSP are easy to use languages, the problem is they are too dependent on the web server. Both of these only works when the Apache web server has the proper modules installed. But the fact that many web server today are Apache and that should not be an major issue. Database drivers are also available for either PHP and JSP (JSP uses the same JDBC driver in Java). JSP is also supported on other web servers. PHP mainly is available on Apache servers.

Python is a scripting language that is also gaining some attention. It is easier to use than Perl, more object-oriented and getting more support drivers. A Python driver is also available in the basic PostgreSQL distribution. It is also independent of the web server because it is an externally interpreted language like Perl.

All of the three are viable alternatives to Perl. We believe the path to choose depends upon the performance of the each system and the programming style preferred. We did not have the resources to test every options so there are no solid information on performance.

Disadvantages

The Perl language itself does have a steep learning curve. It has a weak sense of variable typing and the syntax of different data structures can cause occasional problems. Unlike PHP and JSP, which was designed from ground up specifically for web application, Perl lacks important features and flexibility that makes programmer job of maintaining the application easier.

One feature in JSP not present in Perl is the separation between code and HTML. In Perl both code and HTML are mixed together in the same file, or is very hard to split. The combining of HTML and code causes frequent problem with debugging (both HTML and program code), made the file excessively long, and made the job of designing a page much harder. In JSP the program code can be kept on a different file from HTML, and there is no code and HTML mix up.

We expect this and other features which present in other web application languages could gradually overtake the place of Perl.

5.4 User Interface

Portable output plays a very important part in the user interface design. HTML was used to improve portability. Other decisions made was also

[14] `http://www.php.net`
[15] `http://www.apache.org/tomcat/`
[16] `http://www.python.org`

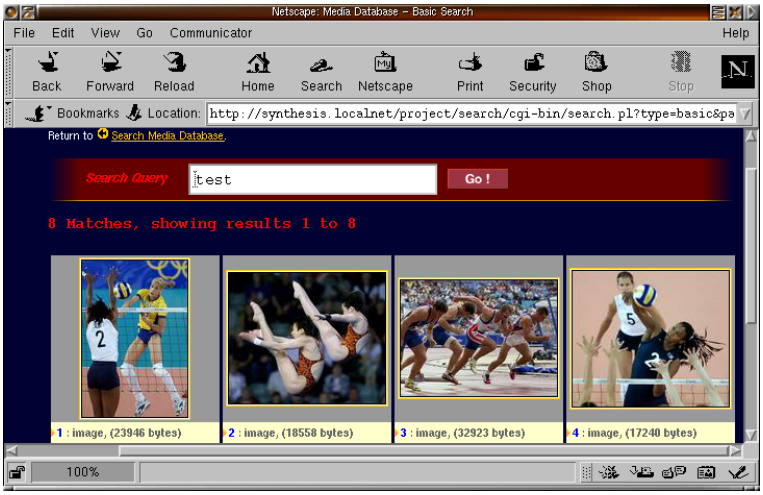

Figure 5.1: Prototype system: Screen shot of the Basic Search Screen.

based on this important factor, such as the use of Cascading Style Sheets (CSS) and Java.

5.4.1 HTML and CSS

To make sure all standard browsers can read the pages they are constructed using only `HTML` 4.01, and no propriety tags (i.e. "Netscape" or "Microsoft" tags) were used.

`CSS` is another area most older browser is likely to fail. We tested Netscape 2.02 and Internet Explorer 3 which has little or no CSS support and the result was no surprise. CSS did worked on newer browsers such as Netscape 4.7+ and IE 4+, with some minor differences in interpretation and formatting. It is worth noting that most browsers today still not fully compliant with CSS level 1. Generally older browsers are likely to have some problems reading CSS, and we urge those users to upgrade.

5.4.2 JavaScript

The concept of `JavaScript` was good in which to provide interactivity at web browser and remains simple to use. JavaScript are code embedded in the HTML page and can be interpreted by the user web browser if it understands JavaScript. Netscape was first to include JavaScript in their browsers. Microsoft later also supported JavaScript but started making modifications to the original JavaScript. At this stage a HTML page wishing to use JavaScripts often requires two sets of scripts to make sure browsers from both manufacturers understands.

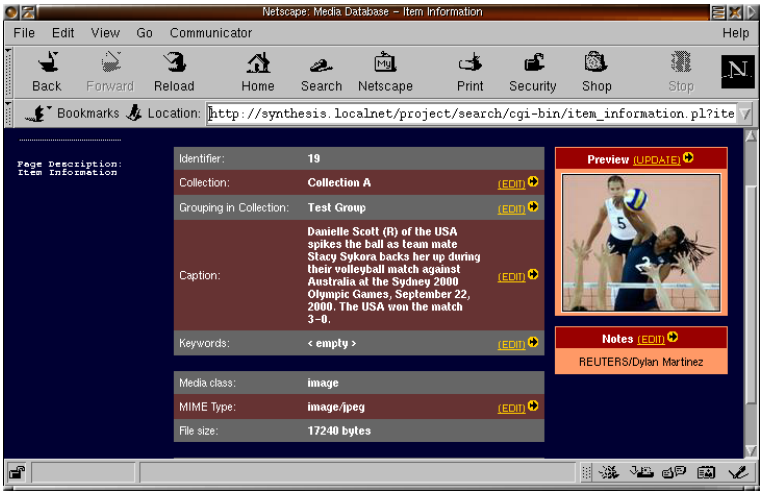

Figure 5.2: Prototype system: Screen shot showing Object Information.

5.4.3 Java

`Java` is a choice we hope could provide interactivity while maintaining compatibility. Although there are some minor differences between different Java Virtual Machines on different browsers the problems were solvable in time before this report was produced.

The `ImapEditor` is an applet written using Java. The user is able to edit

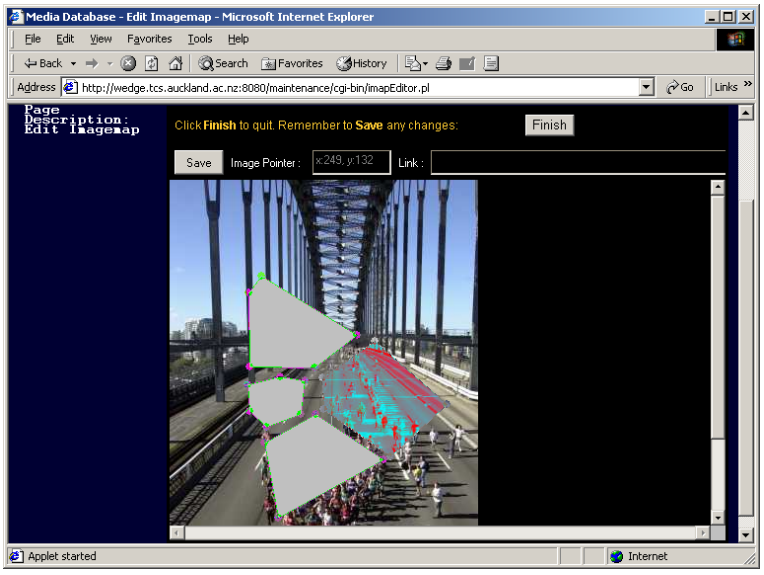

Figure 5.3: Prototype system: Screen shot of the Imagemap Editor `ImapEditor`.

imagemap area polygon and link using this applet. The user can save the link and polygon. Any previouly saved links and polygons are recall when this applet is loading.

Chapter 6

Technology

This chapter will briefly discuss a number of technology used and implementation notes of this system in regard to the technology.

6.1 Hypertext Markup Language (HTML)

6.1.1 Cascading Style Sheets (CSS)

`Cascading Style Sheets (CSS)` [11] is tied together with HTML and CSS Level 1 was used extensively throughout the entire site. The main purpose of CSS is to divide the formatting in an HTML document from it's content. Before the appearance of CSS, HTML content are mixed with the formatting. The task of updating formatting or content is often a daunting task because the programmer have to navigate through a sea of <`FONT`> and other HTML tags, and often have to sort through the formatting parameters in a tag to find the part of the page requiring update. This is no longer the case with CSS. The formatting for a particular tag (e.g. <`A href`> for hyperlink) can be update at a single point in an external file or the Style Sheet section of a HTML file. Format applies globally to the entire HTML page and the programmer no longer need to manually update the formatting at every single part of the page.

CSS Level 2 also became an official W3C Recommendation in May 1998[1], notable changes were support for downloadable fonts, element positioning and tables. However, for many web browsers the support for CSS Level 1 is still incomplete.

In the system CSS files are stored in the `/resource` directory and named with the `.css` extension. We used Server Side Include to attach it's content to the HTML page.

[1] `http://www.w3.org/Style/CSS/`

6.2 Hypertext Transfer Protocol (HTTP)

HTTP was used when the World-Wide Web began in 1990. In the HTTP/1.1 specification [1] it is describe as *"an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems... A feature of HTTP is the typing of data representation, allowing systems to be built independently of the data being transferred."*

6.2.1 Dynamic Query

It is possible to request dynamic content or return variables back to server from browser (or user agent). HTTP defines dynamic retrieval of data from the server through several methods [1, Pp 30–31]. Important methods are (summarised from [10]):

GET Returns the data asked for by the user agent. Variables (called `query string`) submitted by user are appended to the URL (See RFC 1738 for more details). The server-side script or program can lookup the submitted variables sent from the browser in an environment variable named `QUERY_STRING` through the server CGI. Content obtained through GET also have a potential for being cached. This request method is also restrictive. For safety reason, the web server is designed to accept only URL up to a certain length, making the GET method much more restrictive when submitting large amount of data.

POST Tells the server to accept the data and do something with it. The server would expect the variables submitted by the user web browser in the HTTP request body and unlike the GET method, which uses the URL for transmitting variable. On the web server the CGI interface will echo the submitted data to the script or program's standard input, where the program/script can capture data from. Results which was transmitted by this method cannot be cached.

6.2.2 Requesting Multimedia Data

The `Content-Type` attribute in the HTTP header [1] and design of HTTP itself provides a lot of flexibility over how the actual object is served over the network. The actual object, regardless of text or binary, is send from the server to the browser as an individual stream. Upon receiving the stream the recipient can interpret the data as indicated by the server in the Content-Type header, with a MIME value of the data type and format. One example is a HTML page and an image embedded inside that page using the <`IMG`> tag. The browser opens the HTML page, the server sends the page. The browser sees the image inside the HTML page and request the image through the URL indicated by the <`IMG`> tag. The URL could be an actual image

or a CGI script. The server sends the image and label the Content-Type as `image/jpeg`, the browser would interpret it as an JPEG image inside the HTML page.

6.3 Common Gateway Interface (CGI)

Initially implemented on the NCSA's httpd web server[2], and was later carried into the Apache server together with the original httpd codebase[3]. CGI is a simple interface between a web server and external program for generating dynamic HTML pages. The operation of CGI is simple, when the server receives a request it either:

GET Places user submitted data in the environment variable `QUERY_STRING`, which a script or program can read.

POST Echoes the user submitted data to the standard input of a script or program.

The script/program can also find out the HTTP method used through the environment variable `REQUEST_METHOD` in CGI, to eliminate guesswork on how to capture the submitted data.

The script/program generate HTML output on it's standard output. CGI echoes this to the web server and the web server passes the entire standard output to the user browser. Because the web server or CGI do not add anything to the output, the script/program must generate a HTML output with properly formed HTTP header.

6.3.1 Alternatives

The CGI is good design for legacy scripts or programs that use standard input and output for generating dynamic content. Performance may still suffer from `folk`-ing or spawning of the script/program responsible for generating the content. Other alternatives are available to replace this model of dynamic content generation. `PHP`[4] is one that do not require spawning of external interpreter or program when generating content. It is a module inside the Apache web server and it works parallel with the web server, without the use of CGI.

6.4 Structure Query Language (SQL)

This description of SQL was extracted from [17]:

[2] `http://hoohoo.ncsa.uiuc.edu/`
[3] `http://httpd.apache.org/ABOUT_APACHE.html`
[4] `http://www.php.net`

> Structure Query Language(SQL) is one of relational model's standard language. This language is composed of about thirty command and is designed to work with any application that requires the manipulation of data stored in a relational database. Almost all relational software supports SQL, and many software vendors have developed extensions to the basic SQL command set.
>
> Because SQL's vocabulary is rather limited, SQL is relatively easy to learn.
>
> SQL is a nonprocedural language; that is, the user specifies *what* must be done, but not *how* it is to be done. To issue SQL commands, end users and programmers do not need to know the data storage format nor the complex activities that take place when any SQL command is executed.
>
> Yet,as useful and as powerful as SQL is, it is not meant to stand alone in the application arena. Data entry is possible but awkward, as are data corrections and additions... interfaces are created with the help of GUI-based software.

SQL was used in the Perl scripts in the system. Every database query, whether it is updating, searching or adding was executed using SQL.

6.5 Java and JDBC

6.5.1 Java

Java was used to some extend when implementing the system. It was clear at the beginning that some interactivity was not possible through the use of HTML forms or JavaScript. The `ImapEditor` for easy updating and editing of imagemap is a prime example. Java is the only solution for providing the interactivity we required and still well supported on many web browsers from different manufacturers.

A number of requirements in the imagemap editor is difficult to implement in HTML, if not impossible, particularly flexible drawing abilities and database queries. The level of interactivity is also a major drawback with HTML. User normally expect immediate response in the changes and update. That is not possible in HTML because it will requires time to submit to the server and for the browser to receive the results, and all this happens over a network connection. Java solves this by putting the code on the user side, making response faster and also relieving the server from a lot of extra work.

When dealing with untrusted applets security is paramount in Java. The restriction on untrusted applets can be tight, especially when the security

manager in a web browser allows an untrusted applet downloaded from a web site to do only limited number of tasks. This also made the testing procedure more difficult that it would with normal Java application.

We used JDK 1.1 to ensure compatibility with older browsers.

6.5.2 JDBC

JDBC[5] is the Java Data Access API and it used for standardising database access. Drivers for many databases are available. Standard functions for connecting and maintaining database connection are available, as with query and database interpretation functions. The JDBC driver we used also have additional functions available for working with polygons and large object type, making the downloading and updating of picture and imagemap area possible.
There are 4 types of JDBC drivers[6]:

1. JDBC-ODBC bridge provides JDBC API access via one or more ODBC drivers. Contains native ODBC binary code that could introduce an security exception in applet.

2. A native-API partly Java technology-enabled driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. This driver also requires loading of some binary code.

3. A net-protocol fully Java technology-enabled driver translates JDBC API calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server. In general, this is the most flexible JDBC API alternative.

4. A native-protocol fully Java technology-enabled driver converts JDBC technology calls into the network protocol used by DBMSs directly.

The PostgreSQL driver is a type 4 JDBC driver [13]. That made the applet implementation possible because there is no loading of native code which could cause a security exception.

We used JDBC 1.0, which is compatible with JDK 1.1. JDBC 2.0 is also available but not used. JDBC 2.0 is a new standard implemented in Java 2.0.

[5] `http://java.sun.com/products/jdbc/index.html`
[6] `http://java.sun.com/products/jdbc/driverdesc.html`

Chapter 7

Conclusion

7.1 Current Status

The current prototype implemented the following functions:

Backend Mostly completed. It will accept new datatype when necessary. New structure must be added if new User Management scheme and Multi-level Collection Hierarchy is to be used.

- Database Design
- Database Driver

Core System Partially completed.

- Web Server
- User Authentication
- Basic User Management
  - Add User
  - Remove User
  - Update Profile
  - Update Password
- Basic Media Management
  - Add Object
  - Add Collection
  - Object Information
  - Collection Information
    - Update Collection Information
    - Set Collection Imagemap
    - Update Collection Imagemap
- Two levels Collection Hierarchy

- Basic Boolean Search
- Generate User Interface

User Interface Completed for prototype

It is very important to point out the system is not complete. The prototype system is mostly ready for basic search, with only two levels Collection Hierarchy.

7.2 Urgent Improvements

The Collection Hierarchy must be changed to accept multi-levels tree structure. The current 2 levels structures in the prototype system is not sufficient for most applications. The navigation scheme in the current User Interface must also change accordingly to handle large amount of data and multi-levels Collection Hierarchy. Otherwise the value of the system would be limited. More user testing is also necessary to make the system more accessible.

7.3 Lower Priority Improvements

These are items which do not require immediate attention:

- The complete user management scheme using *control matrix* mechanism should be implemented to allow large number of users with different privilege level.
- Introduce other media categories: Video, Model (3D), Text, etc.
- Introduce new search mechanisms: advance search, similarity search for images, etc.

Bibliography

[1] Tim Berners-Lee, Jim Gettys, HJeffrey C. Mogul, Henrik Frystyk Nielsen, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* Internet Engineering Task Force, May 1996.

[2] Thomas Boutell and Tom Lane, editors. *PNG (Portable Network Graphics) Specification 1.0.* W3C, 1.0 edition, October 1996.

[3] Rikk Carey, Gavin Bell, and Chris Marrin. *ISO/IEC 14772-1:1997 Virtual Reality Modeling Language (VRML97).* VRML Consortium Incorporated, 1997.

[4] Peter Coad and Edward Yourdon. *Object-Oriented Analysis.* Prentice-Hall, second edition, 1991.

[5] C.J. Date and Hugh Darwen. *A Guide to the SQL Standard: A user's guide to the standard database language SQL.* Addison-Wesley, 1997.

[6] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication.* Internet Engineering Task Force, June 1999.

[7] Ned Freed and Nathaniel S. Borenstein. *Multipurpose Internet Mail Extension.* Internet Engineering Task Force, November 1996.

[8] Simson Garfinkel and Gene Spafford. *Web Security & Commerce.* O'Reilly & Associates, 1997.

[9] Frank Kappe. *HyperWave: The Next Generation Web Solution*, chapter 10: The Hyper-G Server, pages 115–133. Addison-Wesley, 1995.

[10] Ben Laurie and Peter Laurie. *Apache: The Definitive Guide*, chapter 4, pages 75–77. O'Reilly & Associates, second edition, Feb 1999.

[11] Håkon Wium Lie and Bert Bos. *Cascading Style Sheets, level 1.* W3C, December 1996.

[12] Larry Masinter and Ernesto Nebel. *RFC1867: Form-based File Upload in HTML.* Internet Engineering Task Force, November 1995.

[13] Peter T. Mount. *ProgreSQL Programmer's Guide*, chapter 21: JDBC Interface, pages 184–189. Progres Global Development Group, 1999.

[14] Netscape. *Client-Side JavaScript Guide, version 1.3.*

[15] OMG. *OMG Unified Modeling Language Specification.* OMG, 1.3 edition, June 1999.

[16] PostgreSQL Development Team. *PostgreSQL Tutorial.*

[17] Peter Rob and Carlos Coronel. *Database System: Design, Implementation, and Management.* Course Technology, third edition, 1997.

[18] Abraham Silberschatz and Peter Baer Galvin. *Operating System Concepts.* Addison-Wesley, fifth edition, 1998.

Appendix A

Database Schema

```
-- Create object type "lo".
create function lo_in(opaque)
        returns opaque
        as '/users/studs/grad/mcha140/project/pgsql/lib/modules/lo.so'
        language 'c';

create function lo_out(opaque)
        returns opaque
        as '/users/studs/grad/mcha140/project/pgsql/lib/modules/lo.so'
        language 'c';

create type lo (
        internallength = 4,
        externallength = variable,
        input = lo_in,
        output = lo_out
);

create function lo(oid)
        returns lo
        as '/users/studs/grad/mcha140/project/pgsql/lib/modules/lo.so'
        language 'c';

create function lo_manage()
        returns opaque
        as '/users/studs/grad/mcha140/project/pgsql/lib/modules/lo.so'
        language 'c';


-- Table "collection"
```

```
create table collection (
       collection varchar(128) not null primary key,
       use_imagemap bool default 'false',
       imagemap lo,
       imagemap_type integer,
       message varchar(4096),
       created timestamp default 'now'
);

--create trigger t_imagemap before update or delete on collection for
--each row execute procedure lo_manage(imagemap);

-- Table "media"
create table media (
       id serial not null primary key,
       type integer not null,
       created timestamp default 'now',
       creator oid,  -- Updated
       description text not null,
       grouping varchar(128) not null,
       keywords varchar(1024),
       notes text,
       thumbnail lo,
       thumbnail_type integer,
       collection oid,  -- Updated
       size integer,
       class varchar(12),
       filename char(8)
);

--create trigger t_thumbnail before update or delete on media for
--each row execute procedure lo_manage(thumbnail);

-- Table "image"
create table image (
       width integer,
       height integer
) inherits (media);

-- Table "person"
create table person (
       username varchar(20) not null primary key,
       first_name varchar(50) not null,
       last_name varchar(50) not null,
```

```
        password varchar(25),
        phone varchar(30),
        extension varchar(9),
        email varchar(50),
        office varchar(15),
        notes varchar(250),
        created timestamp default 'now'
);

-- Table "imagemap_link"
create table imagemap_link (
        collection oid not null,  -- Updated
        shape polygon not null,
        link varchar(512)
);

-- Table "changelog"
create table changelog (
        id integer not null,
        username oid,  -- Updated
        event_type smallint,
        message varchar(50),
        time timestamp default 'now'
);

-- Table "datatype"
create table datatype (
        type_id serial not null primary key,
        class varchar(12) not null default 'media',
        type varchar(64) not null default 'application/octet-stream'
);

-- These to get started.
insert into person (username, first_name, last_name)
values ('admin', 'Nobody', 'Nobody');

insert into datatype (class, type) values ('image', 'image/jpeg');
insert into datatype (class, type) values ('image', 'image/gif');
insert into datatype (class, type) values ('image', 'image/png');
```

Appendix B

IANA Assigned MIME Types

Latest version of the latest registered MIME types on the Internet can be found at:
`http://www.isi.edu/in-notes/iana/assignments/media-types/media-types`

[Due to the size of this document, it is not included with this report.]

Appendix C

Source Code

The source code of all Java and Perl scripts can be found on the Internet at:
`http://www.tcs.auckland.ac.nz/~btech/btech2000/mcha140/htdocs.tar.gz`