Automatic Web Publishing

Òscar Sànchez i Vilar

A dissertation submitted to the University of Dublin,
in partial fulfillment of the requirements for the degree of
Master of Science in Networks and Distributed Systems

September 2000

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Òscar Sànchez i Vilar
September 2000

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed:

Òscar Sànchez i Vilar
September 2000

Acknowledgments

En anglès…

I would like to express my heart felt gratitude to those that gave me the opportunity to undertake this M.Sc. at Trinity College. In particular:

To my mother and my grandmother, because they deserve it.
To my supervisors, Dr. Pádraig Cunningham and Dr. Vinny Cahill, and the other lecturers at Trinity, for their support through the whole year.
To the Barcelona School of Informatics, for their financial support.
To Eircom, for the project.
To Catherine and Danny, for their help and for all we have shared this year in Dublin.

And in catalan…

M'agradaria expressar el meu més sentit agraïment a aquells que m'han brindat l'oportunitat de realitzar aquest màster al Trinity College. En particular:

A la meva mare i la meva àvia, perquè ho mereixen.
Als meus supervisors, Dr. Pádraig Cunningham and Dr. Vinny Cahill, *i la resta de proferssors al Trinity, pel seu suport durant tot l'any*
A la Facultat d'Informàtica de Barcelona, pel seu ajut econòmic
A Eircom, pel projecte
A Catherine i Danny, pel seu ajut i per tot el que hem compartir aquest any a Dublin.

Abstract

Research and Development departments of most important companies are currently producing technical documents and papers as a result of their own investigation projects. This type of documents normally develop concepts and ideas that often lead to a contribution to knowledge (or possibly to simple information) that may be interesting for other research projects in the department or for further investigation. In that kind of environments, a system for publishing and sharing information become a requirement for the success of the department.

Although many approaches to solve this problem have already been considered, such as shared system files and mailing lists, all of them offer to the user limited services and require extra input information to know which documents are of interest to each user. Besides, the growing demand for web based interactive solutions and for personalization has made them become obsolete.

This project will investigate the design of an automated system for the publication of technical documents on the World Wide Web. Using a simple interface, its distributed users will be able to publish the results of their research, and, automatically, the system will make the new document available for the rest of the users (once reviewed if necessary) by linking it from appropriate document collections, by building searching indices to allow to look for the document using keywords. The publication of a new document will be announced to potentially interested readers by emailing a notification to them; to decide who might be interested in a new document, the system will keep information about interests and past interactions with the system for each authorized user. Other features that the system will provide (if necessary) are the possibility of feedbacking information to report authors and publishing the same document in different formats.

Contents

List of figures

1. Introduction

1.1 The project

This project was proposed by Eircom's Research and Development Division. Its final aim is to investigate and implement a system to address some of the problems that often arise in environments in which scientific investigation is carried out.

Effective dissemination of information throughout a group of researchers is a key issue for the success of any research body. This is a significant issue since different projects can be related to each other, or results of an investigation might be relevant for the direction of other projects. Even old investigations may be of interest, so a repository of such documents becomes a real need for such groups. The web (Intranet for private corporation domains) has proved to be an appropriate media for this purpose due to its distinctive characteristics (available to most of the users within the organisation, capacity of storage, aptitude to manage different electronic formats, etc). Browsing facilities have proved to be essential in this context. Documents can be linked with appropriate collections (indexed by author, topic, date, etc), consequently making it easier to search a given document. Another helpful facility to help the collaboration between different investigations is the ability to search for words and or phrases that occur in any document included in the knowledge repository. This greatly reduces the effort spent locating relevant documents for a researcher.

Nevertheless, just providing researches with a means to publish and to search for information fails to take full advantage of this media possibilities': as the publication of documents grows, it becomes increasingly harder for the researcher to keep up to date with new relevant

published work. Thus, some guidance to potentially interesting documents is desirable to keep researchers from spending a great deal of time and effort looking for information.

This document presents a solution to these problems: a web-enabled networked digital library with support for multiple distributed users.

1.2 The structure of the document

Chapter 1: Introduction briefly introduces the project and the how this report is structured in different chapters.

Chapter 2: Digital libraries investigates what is a digital library, its benefits and disadvantages, the state of the art and examines the similarities and differences with the solution proposed in the document.

Chapter 3: Technologies explores the technologies available to develop a web application with the required attributes and analyses their pros and cons, explaining the choice made. As we will see, the main criteria have been the grade of scalability the technology permits, its portability, its market support and perspectives.

Chapter 4: System requirements establishes the requirements of the system, based on the project proposal and the meetings with Eircom that have been held during the course of the project. The existing system is outlined and the advantages of proposed solution are made evident.

Chapter 5: Analysis presents two of the diagrams that are created in the analysis phase of an Object Oriented development methodology. These

diagrams, elaborated in UML notation, are the use case view and the object model of the system.

Chapter 6: Design explains how the system is organized and the design decision that have been taken. It presents each of the subsystems that the solution makes use of, and describes in detail the implications of their use in the overall system. Section 6.1.3,Eircom R&D system is examined in depth since it implements most of the requirements stated in Chapter 4, System Requirements.

Chapter 7: Implementation describes how the designed system has finally been implemented, the decisions made at this stage of the development process, and gives a short overview of the system interface.

Chapter 8: Evaluation gives a brief discussion of the performance issues of the system, assessing its strengths and weaknesses, and finally suggests further work that could be carried out as an extension to this project.

Appendix A includes some examples and code referenced from the discussion

2. Digital libraries

2.1 Introduction

There are many different definitions of the digital library, and other terms can be found in the literature to describe the notion of a digital library, such as “electronic library” or “virtual library”. Research has been done to try to abstract common elements from different definitions of a digital library in order to create a universally accepted standard definition for digital library, without success. In defining the digital libraries, Drabenstott [Drabenstott] offers 14 definitions, published between 1987 and 1993. His summary definition is oriented to the point of view of library users, although he introduces some common aspects such as the distribution of the library or the fact that “digital libraries are not limited to document surrogates; they extend to digital artifacts that cannot be represented or distributed in printed formats.” Another report [Saffady] cites 30 definitions of the digital library, published between 1991 and 1994, to propose a new definition based in the common aspects of all definitions: "Broadly defined, a digital library is a collection of computer-processible information or a repository for such information… a digital library is a library that maintains all, or a substantial part, of its collection in computer-processible form as an alternative, supplement, or complement to the conventional printed and microfilm materials that currently dominate library collections". Other contributions can be found in [Borgman], [British Library] and [Duguid], each of them considering different aspects. Nevertheless, the definition that has gained most acceptance is the one proposed by the US Digital Library Federation [DLFed]: "Digital libraries are organizations that provide the resources, including the specialized staff, to select, structure, offer intellectual access to, interpret, distribute, preserve the integrity of, and ensure the persistence over time of collections of digital works so that they are readily and

economically available for use by a defined community or set of communities."

This definition gives an idea about the breath of the concept and partially explains why the terminology in the field is so confused, since the term “digital library” is commonly used to simply refer to the notion of “collection,” without reference to its organization, intellectual accessibility, or service attributes. A significant example of this extent is the simplistic definition by Peter J. Nürnberg [Nurnberg, et al]:” A physical library deals primarily with physical data, whereas a digital library deals primarily with digital data”.

Much of the confusion is no doubt attributable to differences of perspective. Digital libraries bring together facets of many disciplines, and therefore experts with different background are involved, including computer scientists, database vendors, commercial document suppliers, publishers, etc.

It’s noteworthy that, although “electronic library” is an accepted synonym for digital library, this is not similarly accepted for “virtual library”. Collier defines a “virtual library” as “an electronic library in which the users and the holdings are totally distributed, yet still managed as a coherent whole” [Collier]. Therefore, a virtual library can be considered as an extension to the digital library since it has no defined location (i.e. virtual).

From the early stages, the fundamental motivation for research and investment in digital libraries was the belief that they could provide a better way to deliver information to the user. Back in 1945, Bush introduced the idea of using technology to gather, store, find and retrieve information to contribute to science evolution [Bush V.]. At that time, the technology available hindered any plan to implement Bush’s ideas, but he established the bases for a new field: the digital library. This field was better defined in 1965, with the publication of the book “Libraries of the Future”, where

Licklider described the research and development needed to build a truly usable digital library. The book introduces the first notion of collections of documents using digital technologies. It is difficult to determine exactly when the first digital library came into being, mainly because of the lack of unanimous agreement about what can be considered a digital library. The development of automated libraries (libraries that manage their resources with computer technology) experienced real progress in the 1970's and matured in the 1980's, but it wasn't until the beginning of the 1990's when the first real digital libraries started to be functional. This is because a series of technical developments took place that removed the technological impediments to creating digital libraries.

Several technical report searching and indexing system have been developed in the past. Early in the 1990's, the physics *e-print* [Ginsparg] allows scientist to submit documents by email in TeX format, along with bibliographic information to index the documents. Developed in 1994, the HARVEST system in Computer Science extracts indexing information from documents and indexes them. Other examples of such systems can be found in [Ian].

One of the pioneer projects to create a digital library was the Mercury Electronic Library project, back in 1989, at Carnegie Mellon University (CMU). One of the project's aims was to build a retrieval system that could deliver full-text documents to the desktop, based on the existing high performance network; it went live with a dozen textual databases and a small number of page images of journal articles in computer science. From then on, uncountable projects on digital libraries have been carried out, and a lot of investment has been done in order to provide a better paradigm of information delivery. The most relevant ones are analysed in further sections.

2.2 Benefits

As introduced in the previous section, the motivation to build digital libraries was, from the beginning, to improve the paradigm of information delivered by physical libraries. This section outlines some of the benefits of digital libraries.

- **Finding of information**

One of the greatest advantages introduced by digital library is the ability to search the full content of the documents stored in it. While traditional libraries narrow this ability to search to a small number of metadata fields associated with the document (the fundamental tool for representing catalog records in computers is the MARC format [MARC]), readers of a digital library can be endowed with a means to search for word or phrases in the whole collection of electronic documents. Although older material is often available only in image format, since essentially all modern material is now printed via computers, it can generally be provided in text form and be searched. This solves the difficulties of searching in printed material.

- **Availability of information**

Information in digital libraries is always accessible to the users. As discussed by William Y. Arms [William], “a recent study at a British university found that about half the use of a digital library’s digital collections was at hours when the library buildings were closed”. Furthermore, since what is delivered to the user is a copy, which is not returned to the library, information is always available and never checked out to other readers, lost

or mis-shelved. This implies that documents can be used by simultaneous users (assuming copyright permission is available).

- **Access to information**

The use of digital libraries allows that a single electronic copy can be accessed from a great many locations; since libraries contain much information that is unique, researchers that want to consult a document have to physically move to the documents' location. With the use of digital libraries, these documents are delivered with electronic speed to the user's desk.

- **Preservation**

The question of preserving or archiving digital information is not a new one and has been explored at a variety of levels over the last five decades, but there is as yet no viable long-term strategy to ensure that digital information will be readable in the future. Preservation of digital documents is a completely different task from preservation of physical documents, since the later depends on having a permanent object and keeping it under guard. This is not an issue for digital material, since multiple copies of the document can be made to ensure that at least one will stay alive. Nevertheless, other problems arise, such as the longevity of the physical media on which the information is stored [Waters], a.k.a. technological obsolescence. Although different approaches to overcome this issue have been considered (such as "refreshing" digital information by copying it onto new media or migrating the material by transferring it from one hardware/software configuration to another, or from one generation of computer technology to a subsequent generation), Rothenberg suggests that the best solution is "to run the original software under emulation on

future computers. This is the only reliable way to recreate a digital documents original functionality, look, and feel" [Rothenberg],

- **Updates of information**

Updating a digital library is much easier than updating a physical library, since the only thing to do is to store the new document in the collection and, possibly, add some information to the library's indexes. Better yet, some libraries projects, such as the New Zealand Digital Library project, "gathers material from public repositories without any participation by authors or their institutions " and therefore it automates the task of updating the library [Ian].

- **Format of the information**

Printed documents used in traditional libraries have limitations when working with them, for instance, with statistical data. The digital library can provide information in any format (graphics, databases, video, etc) and they can even be locally modified when permitted by copyright. Furthermore, information delivered to users can be customised or adapted to user's needs, for instance, in larger type size for those with limited sight, in different formats according with the user's software, etc.

2.3 Disadvantages

Some authors have explored the disadvantages digital libraries can pose. Among them, Walt Crawford [Crawford] highlights that there are "enormous economic and ecological disadvantages to the all digital library". It has been estimated that people tends to print documents larger than 500

words, an therefore, a typical digital library would use at least fifty times as much paper as at present. Another issue for digital libraries is the copyright law, since the copyright holder can loose control of his published material: there is an obvious need for a unified schema to lend copyrighted material via a digital library (different countries have different copyright laws). The current system is not valid either for authors or for publishers: if libraries acquire a copy of the document and then each user can have its own copy, authors would only get paid for one copy, and publishers and editors would disappear. Another problem is the digitalisation of existing material. In addition to the resources that need to be utilised in order to digitalise the vast amount of existing printed material, libraries continue to acquire new print materials, and some of them, such as the Library Congress, do it faster than it digitalises old material.

2.4 Research areas

Nowadays, digital libraries are no longer experimental projects with short-term funding to get them started; the initiative that established digital libraries as a distinct field of research came in 1994, when the National Science Foundation (NSF), the Defense Advance Research Projects Agency (DARPA) and the National Aeronautic and Space Agency (NASA) created the Digital Libraries Initiative (DLI). This initiative provided funds worth $24 million for six four-year research projects to be carried out at six universities: The University of California at Berkeley, the University of California at Santa Barbara, Carnegie Mellon University, the University of Illinois, the University of Michigan and Stanford University. Currently (from 1998), this initiative is called Digital Libraries Initiative Phase 2, “a multiagency initiative which seeks to provide leadership in research fundamental to the development of the next generation of digital libraries, to advance the use and usability of globally distributed, networked information resources, and to encourage existing and new communities to focus on

innovative applications areas". New partners have joint the initiative, namely the National Library of Medicine (NLM), the Library of Congress (LOC), the National Endowment for the Humanities (NEH) and the Federal Bureau of Investigation (FBI) [DLI2]. There are countless other initiatives on digital libraries. The most significant ones are the Digital Library Federation (DLF), "a consortium of research libraries that are transforming themselves and their institutional roles by exploiting network and digital technologies", and the IEEE Digital Library Task Force, whose focus "is to promote research in the theory and practice of all aspects of digital libraries". The research carried out by these bodies, as well as by private corporations and universities encompass a range of interrelated technical, social and political issues. Currently, the main threads of research on digital libraries are: the object model (the structural relationship among components of the library and the user's view of these components); the user interfaces and human-computer interaction, since it is understood that major advances in usability will come from innovation in the interfaces and not the underlying databases or processing engines; information discovery (strategies and methods on finding information, as introduced in [Ian]); preservation of digital material to address technology obsolescence; format of the digital material and digitalisation of the existed printed material (a hot area of research is the support for searches in non-textual material, such as audio tracks and images); natural language processing, to provide better paradigms to search; and interoperability, exploring how to get different digital libraries to work together; a further investigation in these research areas has been carried out by Williams Y. Arms [Williams].

2.5 Eircom's R&D Digital library

The aim of this project is not to do research in any of the areas introduced in the previous chapter, since it would be too ambitious for the time and resources available; this project will try to investigate the best

solution to solve a problem that arises in many research bodies: the effective dissemination of information throughout the group. In addition, a fully operative application is expected to be implemented to address this issue in a concrete environment: the Eircom's Research and Development Division. Most of the requirements identified for the application are in close relation with the forces that motivate digital library research and that define digital libraries, such as full-text indexing of documents, search facilities, etc. In addition, other functionalites not covered by common digital libraries are needed. The system should provide the users with an automated tool to add documents to the library (in a "traditional" digital library this action is performed by a librarian or a team of librarians, but not by the library readers or users). In addition, a mechanism to recommend documents of potential to users is to be explored and implemented.

To sum up, Eircom's Research and Development Division digital library is an application that inherits most of the requirements of a digital library and adds on top some extra functionality. The proposed application is analysed in depth in the following chapters.

3. Technologies

This chapter analyses and assesses the technologies that are available to implement the proposed solution; since the application is to be used in an intranet environment, all the technologies examined here are web technologies.

3.1 Dynamically Generated Web Pages

In a web application with the characteristics defined in this document, the information provided to the user is not static, but changes with user interaction (after a user adds a new document to the library, subsequent pages that plot the library contents must show this new document). Two alternatives could be considered to keep the system up to date, although one of them turns out to be unpractical: either a web programmer manually changes the affected pages each time a modification to the system is done or pages are created dynamically using information stored in the server (database, macro files, XML documents, etc.). The second approach is the only valid one when dealing with non-trivial projects and will be discussed below. Dynamically created pages enable the project team to program the system behaviour and the application flow and to customise the system based on user preferences (internationalisation, presentation issues, etc). Depending on the technology used to generate these content-tailored pages, web programmers can access business logic and business processes, other web applications or legacy systems.

There are numerous ways to construct server-side dynamic applications. In this section, different technologies that can be used to construct this type of system are investigated and assessed.

3.1.1 CGI and Fast CGI

[CGI RFC Project] In 1993, the U.S. National Centre for Supercomputing Applications (NCSA) developed a standard for server side executable support called Common Gateway Interface (CGI). CGI permits interactivity between a client and a host operating system through the web via the Hyper Text Transfer Protocol (HTTP). It's a standard for external gateway programs to interface with information servers, such as a web server. A gateway is a program that handles information requests and returns the appropriate document or generates a document on the fly. Gateways conforming to this specification can be written in any language that produces an executable file. Some of the more popular languages to use include: VBScript, C, Perl, shells, and many others. Regardless the programming language it is written in, permission and resources to run it on the web server are required. When a URL that points to a CGI program is requested by a user, the web server uses the CGI protocol to invoke the program and to send/receive data from it. Nevertheless, the use of CGI poses several drawbacks: each time a CGI script is spawned, it creates an additional process on the server machine, slowing the server's response time (if the server is heavily accessed CGI is not applicable) and therefore compromising scalability. Also, if the CGI script is not set up correctly, security holes can occur on the server, making the application vulnerable. Another problem is that it is difficult to maintain state — that is, to preserve information about the client from one HTTP request to the next -- since HTTP is a stateless protocol.

FastCGI is a language independent, scalable, open extension to CGI that provides better performance without the limitations of server specific APIs [FastCGI]. Like CGI, FastCGI applications run in separate, isolated processes (so a buggy FastCGI application cannot crash or corrupt the core server or other applications). FastCGI is conceptually very similar to CGI, with a major difference: FastCGI processes are persistent; they need to be started prior to serving any requests, and after finishing one, they wait for a new request instead of exiting. If it dies, a FastCGI manager automatically restarts the process. To control all processes created by FastCGI, a process manager is needed. This process manager works with heuristics to estimate when new instances of applications need to be created. This makes FastCGI applications difficult to develop, because special directives need to be included in the web server configuration files.

3.1.2 Server Side Includes

Server Side Includes are directives which one can place into HTML documents to execute other programs or output data such as environment variables and file statistics.

While Server Side Includes technically are not really CGI (see 3.1.1 CGI and Fast CGI), they can be an important tool for incorporating CGI-like information, as well as output from CGI programs, into documents on the Web. An SSI is a command or directive placed in an HTML file through the use of a comment line. Basically with SSI one can execute shell or CGI scripts. When a client requests a document that includes SSI directives, the server parses the document looking for SSI directives to execute them. As a way of building dynamic pages, SSI isn't very efficient. It's particularly bad if you use more than one directive per page, because each directive is a separate process. Each SSI program you include on the page at least

doubles the server load for every viewing of that page. SSI creates a security risk because SSI directives can execute system programs.

3.1.3 Proprietary server extensions

Most web servers offer a native API (Application Programming interface) to allow the developer to access performance-enhancing features by embedding non-HTML commands within their HTML files. These commands are then processed when the file is requested and generate a combination of HTTP headers and HTML that is sent to the client, in a similar manner to the way SSI works.

The biggest advantage of API-based web server access is speed. Instead of a new process being required for each instance of an application called by the web server, a single multi-threaded application can handle all the traffic. This arrangement can drastically improve the performance of web applications. Not only performance is increased, but it also allows more sophisticated security since the application accessed through the server API runs as if it actually is the server. There are some drawbacks, nonetheless: first, each web server has a different API and any application must be customized for a given API, thus lacking of portability; since the applications run in the server's address space, buggy applications can corrupt the core server (or each other). A malicious or buggy application can compromise server security, and bugs in the core server can corrupt applications. The most used proprietary server extensions are Netscape Server API [NSAPI], Apache Modules [Apache API] and Microsoft® Internet Server API [ISAPI].

3.1.4 Server-side JavaScript

As with SSI, JavaScript can also be embedded in HTML files at the server side. These directives provide the developer with a means to connect to different relational databases, to share information at an application level and to access the file system on the server. Since HTML pages with server-side JavaScript eventually produce an HTML document to be sent to the client, they can include client-side JavaScript, as every HTML document does.

HTML pages that use server-side JavaScript are compiled into bytecode executable files. A web server that contains the JavaScript runtime engine runs these application executables. Development of JavaScript applications is comprised of two stages. In the first one, HTML pages (which can contain both client-side and server-side JavaScript statements) and JavaScript files are created and compiled into a single executable. In the second step, and once the page is requested by the user, the runtime engine executes any server-side statements to dynamically generate the HTML page to return to the client.

There are several drawbacks to server-side JavaScript, such as the intrinsic limitations of the language (object model, data types) and the need to have a web server that contains the JavaScript runtime engine.

3.1.5 Java™ Servlets

The Servlet API was developed to boost the advantages of the Java platform to solve the issues of CGI and proprietary APIs. Servlets are protocol and platform-independent server side components, written in Java

and compiled to an architecture independent bytecode, designed to enhance web servers' functionality.

A Servlet is a Java class and therefore needs to be executed in a Java VM by a service we call a Servlet engine. When a Servlet is requested, the class is dynamically loaded and stays in memory until it is unloaded or the Servlet engine is stopped, thus potentially handling more than one request without having to reload it. This request-response model is based on the behaviour of the HTTP protocol. Currently, almost all web servers are Servlet-enabled: some of them, like Sun's Java Web Server (JWS) and W3C's Jigsaw, which are completely written in Java, have a build-in engine; others, like Netscape's Enterprise Server, Microsoft's Internet Information Server (IIS) and Apache, require a Servlet engine add-on module, which has to be configured to intercept Servlet requests, serve them and send the response via the web server. The Servlet interface is the central abstraction of the Servlet API. All servlets implement this interface either directly, or more commonly, by extending a class that implements the interface. The two classes in the API that implement the Servlet interface are GenericServlet and HttpServlet. For most purposes, developers will typically extend HttpServlet to implement their servlets [ServletAPI]. These classes implement methods that express the lifecycle of the servlet: init, service and destroy. The servlet container is responsible for loading and instantiating a servlet (either on "start-up" or when the servlet is first accessed by a client); once the servlet is instantiated, the container must initialise it before it can handle the first request. This is done by calling the init method passing a unique object implementing the ServletConfig interface (through which the servlet gets initialisation parameters and a ServletContext describing the runtime environment where the servlet runs). Then, the servlet is ready to handle user requests. Each request, which is represented by an object of type Servlet request, generates a call to the service method, passing the request object. Further requests to the servlet generate new calls to the service method with new request objects representing the new requests, skipping so the loading, instantiation and initialisation process. It's not

loaded again until the servlet changes, and a modified servlet can be re-loaded without restarting the server. These requests may be concurrent, i.e. more than one can occur at the same time. By default, there must be only one instance of a servlet class per servlet definition in a container. Consequently, it could imply concurrency problem among requests if accessing shared data; this is solved by using the SingleThreadModel interface, that guarantees that one thread at a time will execute through a given servlet instance's service method by serializing requests or by maintaining a pool of servlet instances. The servlet instance is kept in memory at the container's discretion. When the container decides that a servlet should be unload (the policy differs among implementations), it must allow the servlet to release any resources it is using and save its state. It is done by calling the servlet's destroy method, but it's not allowed to call it until all threads in the service method finish or time out. The servlet's class may then become eligible for garbage collection.

Even more, the servlet API has interfaces that easy programmer's difficulties to deal with sessions (it is necessary to associate series of different requests from a particular user) over a stateless protocol such us HTTP. One of these interfaces is the HttpSession, which allows the servlet container to track user sessions, without requiring the developer to do it. Another interface, called ServletContext, defines a servlet's view of the web application, and allows a servlet to access resources available to it; there is one instance of this interface associated with each web application.

In functionality, servlets lie somewhere between Common Gateway Interface (CGI) programs and proprietary server extensions such as the Netscape Server API (NSAPI) or Apache Modules. Servlets have the following advantages over other server extension mechanisms:

- they are generally much faster than CGI scripts and FastCGI scripts because a different process model is used. Instead of creating a new

process for each request, the threads model is used, thus requiring only lightweight context switches among requests.

• they use a standard API that is supported by virtually all web servers.

• they have all the advantages of the Java programming language, including ease of development, strong type-check and security, and platform independence.

• they provide facilities to deal with session and security issues

• they can access the large set of APIs available for the Java platform.

3.1.6 JavaServer Pages™

JavaServer Pages™ technology is the Java™ platform technology for building web applications [JSPSpec]. JavaServer Pages™ are part of the Java 2 Enterprise Edition (J2EE). From an architectural point of view, JSP can be seen as a high-level abstraction of servlets that is implemented as an extension of the Servlet API (the first time a JSP is invoked, a servlet is generated by the container). They can be used in a cooperative and complementary manner, as we will see later on this section. The use of JSP implies a change in the way web applications are developed: instead of generating HTML code from a program, the actual program is embedded in the HTML document. Thus, a JSP document is a text-based file that intermixes template data (HTML tags) with dynamic actions to generate a response to a request from a client.

JSP add a basic feature to the advantages introduced by servlets: the separation of dynamic and static content. While in servlets each line of the dynamically generated document has to be created with an out.println(“dynamic code”) instruction (an onerous and time consuming task for long HTML pages), making it difficult to separate the role of a HTML

designer from a software developer. JSP allow page creators to make changes to the HTML document without serious risk of breaking the Java code. Nevertheless, this approach, called page-centric, in which the request invocations are made directly to the JSP page, is not ideal, since software developers and graphics designers have to share the same text file. Nevertheless it is suited for small projects with only one or two developers.

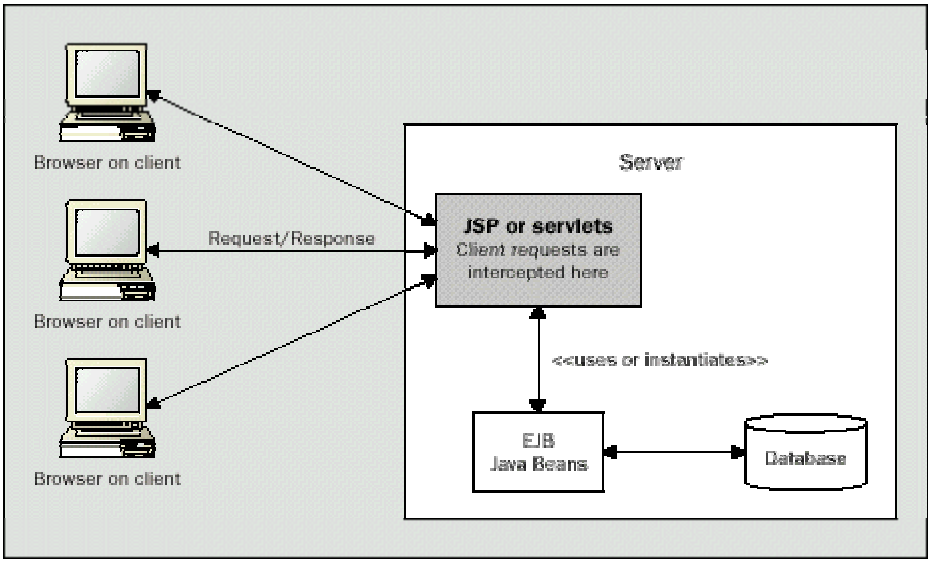


Figure 3—1: JSP Design Model 1

Indiscriminate use of this architecture usually leads to a significant amount of Java code embedded in the page; the use of JavaBeans (helper or worker beans) allows us to remove part of this Java code by encapsulating data access and business logic, as shown in Figure 3—1: JSP Design Model 1.

Web developers soon realised that the best paradigm would be that in which presentation and content aspects could easily be split according with the responsibilities defined in the development team. Consequently, a new design pattern was considered, the so-called Model2. This methodology follows the Model-View-Controller (MVC) paradigm. In this

approach, JSPs are used to generate the presentation layer, and either JSPs or Servlets to implement programming tasks. The front-end component acts as the controller performing all the computation and setting all data and objects that the view (JSP), with only presentation responsibilities, will retrieve later on to generate the dynamic content.

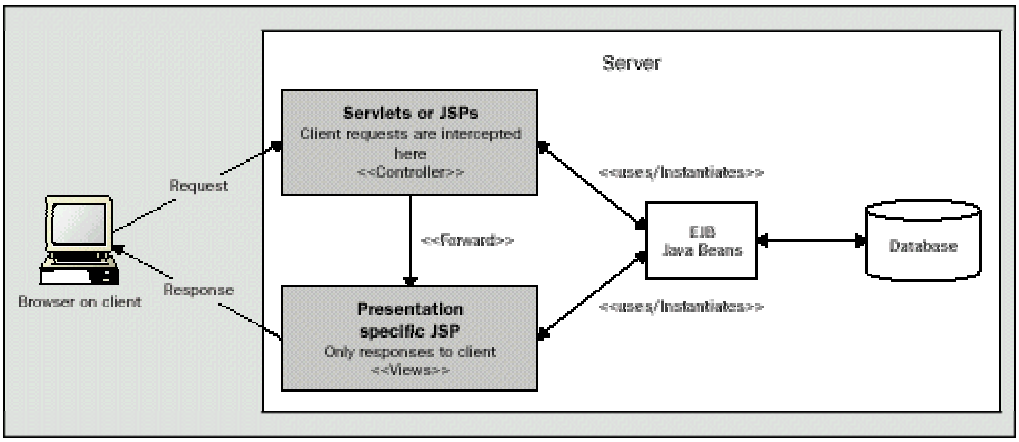


Figure 3—2: JSP Design Model 2

This is the approach that, when possible, has been followed in this project since it results in the cleanest separation of presentation from content, leading to clear separation of responsibilities of the developers and page designers in a team (in this case I was the only developer, but this is the pattern that should be followed regardless). In terms of presentation, JSP pages are the only choice, while for control proposes servlets can also be used. It is also possible use template engines for presentation but they haven't been chosen for this project because there is no standard specification of such engines and because of my lack of expertise in using them.

3.1.6.1 Java Beans

The JavaServer Pages (JSP) component model is centred on Java software components called Beans, which must adhere to the JavaBeans API. A Java Bean is nothing more than a java class that has been programmed following a set of intuitive conventions dictated by the JavaBeans specification. Figure 3—3: JavaBean example shows a simple JavaBean used in this project.

```
public class Category{

   private String strCat;
   private String strSubCatOf;

   public Category(String strCatParam, String strSubCatOfParam){
     this.strCat = strCatParam;
     this.strSubCatOf = strSubCatOfParam;
   }

   public String getCategory(){
     return this.strCat;
   }

   public String getSubCatOf(){
     return this.strSubCatOf;
   }

   public void setCategory(String strCatParam){
     this.strCat = strCatParam;
   }

   public void setSubCatOf(String strCatParam){
     this.strSubCatOf = strCatParam;
   }
}
```

Figure 3—3: JavaBean example

Instances of Bean classes are simply Java objects, so they can be referenced from Java code, but a Bean container can only manipulate it through its properties; this is because JSP containers interact with Bean objects through a process called introspection, using reflection (mechanism that allows the Bean container to examine any class at run time to determine its set of properties). As suggested earlier in this chapter, the use of JavaBeans enables the programmer to migrate the Java code representing the business logic and data storage from the JSP to the

JavaBean worker. This refactoring leaves a much cleaner JSP with less code, and then a better separation between presentation and content. Figure 3—4: UseBean example shows how to use a simple bean from a JSP. Another advantage that arises from the use of JavaBeans is that they are reusable software components, since they can be called from different JSPs.

```
1  <jsp:useBean id="myCategory" scope="session" class="Category" />
2
3  <HTML>
4  <BODY>
5    ... more code here ...
6  <jsp:setProperty name= "myCategory" property="Category" value="<%= strCategory %>" />
7  <jsp:setProperty name= "myCategory" property="SubCatOf" value="<%= strSubCatOf%>" />
8    ... more code here ...
9
10 </BODY>
11 </HTML>
```

Figure 3—4: UseBean example

3.1.6.2 HttpSessions

The Hypertext Transfer Protocol (HTTP) is by design a stateless protocol. Nevertheless, in most web applications, as the one described in this document, there is a need to associate series of different requests from a particular user with each other. Although it can be programmed by using different tracking strategies (URL rewriting or cookies), the Servlet API v2.2 offers an easy to use interface called `HttpSession`. This interface enables the programmer to store and retrieve objects from the `HttpSession`. The JSP Specification defines a default object, called `session` that can be accessed at any time and that is transparently mapped on the `HttpSession` defined in Servlets.

3.1.7 Active Server Pages

Active Server Pages is an open, compile-free application environment in which you can combine HTML, scripts, and reusable ActiveX server components to create dynamic and powerful Web-based business solutions. Active Server Pages enables server side scripting for IIS with native support for both VBScript and Jscript. [ASP]

Active Server Pages (ASP) technology is similar in concept to Netscape's server-side JavaScript, but is implemented as an ISAPI application integrated into Microsoft's Internet Information Server, which implies that ASP technology is basically restricted to Microsoft Windows-based platforms (although ASP technology is available on other platforms through third-party porting products).

Java Server Pages are the counterpart from Sun to Microsoft's ASP. These technologies, similar in concept, have many differences that led me to choose JSP as the server-side technology used in the application discussed here. Some of these differences include:

- JSP are platform and server Independent
- JSP technology uses the Java language for scripting, while ASP pages use Microsoft VBScript or Jscript, thus allowing the user to access a wide collection of Java API's and all the J2EE services (besides other advantages, such strong-type check mechanism, security, protection against system crashes, etc)
- Maintenance is easier with JSP, due to the lack of scalability of ASP's scripting languages.

3.1.8 PHP

"PHP: Hypertext Pre-processor" (PHP) is an open-source, server-side scripting language for creating dynamic Web pages. Its syntax is C-like, but is a distinct scripting language that works with many web servers. PHP code appears in the HTML pages embedded within simple delimiters, in a similar way to SSI, JSP and ASP do. Although it has support for many databases, and for interfacing to other services using protocols such as IMAP, SNMP, NNTP, POP3, HTTP, scripts to control the application logic have to be written within the HTML document, requiring a developer to create the dynamic pages. As considered in section 3.1.6 JavaServer Pages ™, this prevents a proper definition of roles within the web development team, compromising the application maintenance.

3.2 Web server and JSP Container

A web server is a program that, using the client/server model and the World Wide Web's Hypertext Transfer Protocol (Hypertext Transfer Protocol), serves web components to users via a web browser. It can be used for serving pages over an intranet or over the Internet. Apache Web Sever version 1.3.12 for Win32 (created by Apache Software Foundation) has been used in the development of this project, basically because it can be used free of charge, it has been shown to be substantially faster than many other free servers, and because of my expertise on the server. The application resulting from this project is to be deployed in a web site using Microsoft's Internet Information Server running in a Microsoft's Windows NT 4.0 machine. Therefore, no proprietary API can be used in the solution, otherwise the system would not be portable to the deployment environment.

A servlet container is a runtime shell that manages and invokes servlets on behalf of users. Jakarta-Tomcat, developed by Apache Software Foundation, is a servlet container with a JSP environment, compliant with Java Servlet 2.2 API and Java Server Pages 1.1 specification. As a servlet container, it manages and contains servlets through their lifecycle and provides the network services (along with a web server) to set requests and responses. As a JSP environment, it delivers requests from clients to the requested JSP page and requests from the JSP page to the client. There are basically two types of servlet containers: stand-alone servlet containers (integral part of a Java-based web server) and servlet containers that need to be installed as and add-on component to a web server via that server's native extension API. Within this second option, containers can be classified into in-process or out-of-process, depending on whether the process executing the Java Virtual Machine that runs the container is opened inside the web server's address space or not. While in-process containers provide good performance but are limited in scalability, out-of-process containers have worse response time but better scalability. Jakarta-tomcat can operate in either of the three modes; for this project, bearing in mind that Apache Web Server is not a Java-based web server, the decision to be made was between the defined add-on modes; since the difference in performance is not significant for the project requirements, the out-of-process mode has been used.

3.3 Web application and Web Application Archive (WAR)

A web application is a collection of servlets, Java Server Pages, HTML documents, and other web resources that might include image files, compressed archives, and other data. It also includes a web application deployment descriptor. A web application may be packaged into an archive or exist in an open directory structure. All compatible servlet containers must accept a web application and perform a deployment of its contents into their

runtime. [ServletAPI]. This archive file is created by using standard JAR tools. The standard extension used for web application archives is .war.

The web application deployment descriptor (DD) conveys the elements and configuration information of a web application between Developers, Assemblers, and Deployers. These descriptors are XML files specified by a Document Type Definition (DTD). This DDT can be found at [ServletAPI]. The web application deployment descriptor created for this project can be found in Appendix A.1 Web application descriptor (web.xml)

3.4 Presentation

One of the requirements of the system is to integrate its interface with the Eircom's Research and Development Intranet; one additional requirement posed by the use of the subsystems described in chapter 6 Design is the interface integration between them; therefore, great effort has to be put in the presentation of the application. Two technologies have been used to achieve this goal, Cascading Style Sheets and Dynamic HTML.

3.4.1 Cascading Style Sheets

CSS is a W3C recommendation to separate presentation from content in web documents. Currently there are two recommendations (CCS1, CSS2) and a third one is being developed (CSS3). Style sheets provide a means for authors to specify how they wish documents written in a mark-up language such as XML or HTML to be formatted. These languages are designed to structure documents, not to format how the documents are to be displayed (it depends on the client). By using CSS, one the developer can separate content from formatting (it leaves a much cleaner HTML

document, in a similar way JSP Model2 does), he has more control over formatting than using HTML, and can ensure a uniform appearance across an application. By integrating existing CSS in the site, the seamless integration of the application has been achieved. The only problems are that not all browsers support CSS properly; one on every 20 users have browsers that don't support style sheets.

A style sheet is simply a text file, with the .css extension, which is written according to the grammar defined in the various recommendations. This grammar defines rules that allow the developer to assign presentation attributes to HTML tags or to define classes of attributes that can be assigned to HTML objects [CSS].

```
.mylinklight {
   color:#ffeeee;
   font-weight:500;
   font-family:arial;
   word-spacing:0;
   text-decoration:none;
   letter-spacing:0;
   margin-left:20;
   margin-right:30;
   font-size:12pt;
   background-color:#3366cc
}
```

Figure 3—5: CSS Example

3.4.2 Dynamic HTML

DHTML has been used to create some dynamic effects in the pages of the application, as well as to generate the windows that allow the user to navigate across different hierarchies (see chapter 7 Implementation).

DHTML is the combination of several built-in browser features in fourth generation browsers that enable a web page to be more dynamic. It uses Cascading Style Sheets (CSS), HTML and JavaScript as underlying

technologies and therefore is an entirely client-side technology that doesn't require any plugin in the web browser to create these effects. The basic drawback to this technology is that the main browsers (Microsoft Internet Explorer and Netscape Navigator) haven't implemented DHTML the same way, making it difficult to develop effects that work in the same manner in both browsers. An example of DHTML, used in the login page, can be found in A.2 DHTML code for login page.

3.5 Information storage

As mentioned in chapter 4 System Requirements, information about users must be stored in order to implement a recommendation system. Two approaches to keep such information can be followed: use client-side technologies such as cookies or maintain a database in the server. The next subsections examine these approaches.

3.5.1 Cookies (client-side storage)

Cookies are a general mechanism which server side connections can use, to both store and retrieve information on the client side of the connection. When a server sends and HTTP object to a client, it can also attach a piece of information in text format, containing, among others, a description of the range of URLs for which that information is valid. By setting that range of URL's to be a superset of all the URL's used by the application, this information will be sent back to the server each time the client requests an HTTP object that is part of the application. If this information were defined to be all the information the application needs to know about the client, no database would be needed on the server-side.

A cookie is introduced to the client by including a Set-Cookie header as part of the HTTP response; typically this will be generated by server-side technologies such as the ones discussed in the previous sections (3.1 Dynamically Generated Web Pages). The syntax for this header is:

Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure Where: • **NAME** is the name of the cookie, • **VALUE** is the value associated to the cookie, • **DATE** defines the valid life-time of the cookie, • **DOMAIN_NAME** and **PATH** define to which URL's the cookie must be sent back, • **SECURE** marks the cookie to only be transmitted if the communications channel with the host is a secure one. (For more information on these fields, see [Cookies])

Table 3-1: Cookie syntax header

But, if information about other users is needed when a given user interacts with the system (as in this application, when a new document is added by a user to the library all potentially interested users must be notified by the system) it implies that all users need to store information about the rest of the users, which may compromise privacy and make it difficult to update such replicated information. Another possible downside to be considered is that the number and size of cookies is limited: a client can store up to a total of 300 cookies, narrowing this number to a maximum of 20 cookies per server or domain, with a maximum size of 4 kilobytes per cookie. Depending on how the application is designed, these limitations could be an issue. But the principal factor why it is unwise to rely on cookies to store important information just in the client-side is that the user has the ability to reject cookies. To sum up, cookies are an inadequate mechanism due to the application requirements. Therefore, a server-side mechanism is needed; in order to ease the design, a database will be used.

3.5.2 Database (server-side storage)

Microsoft Access has been chosen as the relational database management system to store application information since it was the only one accessible from the web site where the application was to be installed. Details about database design are discussed in section 6.2 Database Design. To act on the database, the Java Database Connectivity (JDBC™) 2.0 API has been used, since it provides access to "virtually any tabular data source" from programs written in Java, facilitating portability. To use the JDBC API with a particular database management system, a JDBC technology-based driver is needed. Nevertheless, a JDBC-ODBC Bridge driver can also be used, making most Open Database Connectivity (ODBC) drivers available to programmers using the JDBC API and thus enabling the programmer to access any ODBC data source from Java. The Bridge is implemented as the sun.jdbc.odbc Java package and contains a native library used to access ODBC. Whenever possible, Sun recommends the use of a pure Java JDBC driver instead of the Bridge and an ODBC driver, since this configuration completely eliminates the client configuration required by ODBC, as well as eliminating the potential that the JVM could be crashed by errors in the native code.

To be able to use the database, the first thing is to load the mentioned driver and register it with the `DriverManager`, a class in the `java.sql` package. Then you need to open a connection to the database, resulting in a `java.sql.Connection` object. A Connection object represents a native database connection and provides methods for executing SQL statements. To do so, the getConnection method has to be invoked in the `DriverManager`, asking it to locate a suitable driver from amongst those loaded. Once a `Connection` is established, a `Statement` is used for executing a static SQL statement and obtaining the results produced by it, if any. This `Statement` object provides methods to execute both Data Definition and Data Manipulation operations defined in the ANSI SQL-2 standard, via its methods `execute()`, `executeUpdate()` and

`executeBatch()`. For those `Statements` that return data from the data source (“select” commands), a `ResultSet` interface is provided to implement typical methods to move along the `ResultSet`, to retrieve columns from it, etc. [Java 2 API]

To improve database access time, a connection pool mechanism has been used. A connection pool is a cache of open connections that can be used and reused, thus cutting down on the overhead of creating and destroying database connections. This overhead is considerable, since an application can easily spend several seconds every time it needs to establish a connection. For short database transactions, more system resources may be consumed connecting to the database than executing the actual transaction. Figure 3—6: Database Connection Pool shows a scheme of this mechanism.

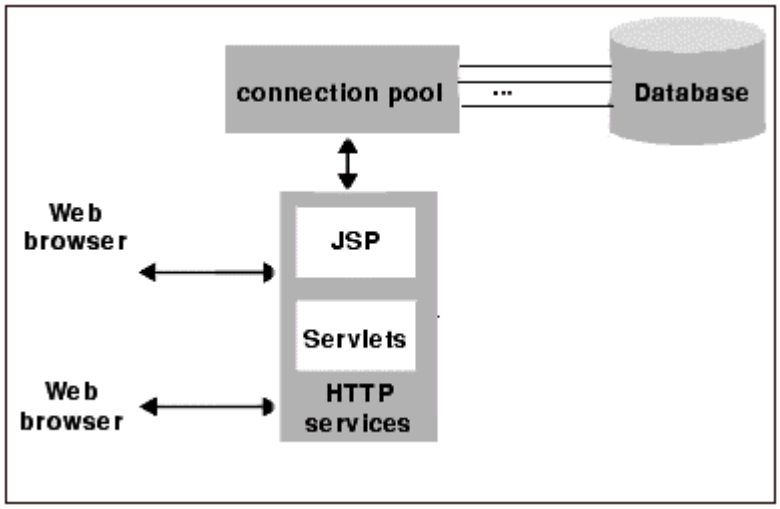


Figure 3—6: Database Connection Pool

The database connection pool [ConnectionPool] consists of manager class that provides an interface to multiple connection pool objects. Each pool manages a set of JDBC Connection objects that are shared throughout the application. The `DBConnectionManager` class is implemented according to the Singleton design pattern, ensuring this way that only one

instance of this class will exist at a time. Other objects can obtain a reference to this instance through a static method (class method). The `DBConnectionPool` class represents a pool of connections to one database, identified with a JDBC URL; in this case: jdbc:odbc:EirDL, where jdbc is the protocol identifier, odbc is the driver identifier (JDBC-ODBC Bridge) and EirDL is an ODBC System Data Source.

4. System Requirements

According to Merlin Dorfman and Richard H. Thayer, a software requirement can be defined as: " A software capability needed by the user to solve a problem or achieve an objective or a software capability that must be met or possessed by a system or system component to satisfy a contract, specification, standard, or other formally imposed documentation."

This section analyses the different requirements this project has to meet, by examining different sources of information to determine them.

4.1 The problem

One of key objectives in Eircom´s Research and Development Division is to disseminate its investigation results effectively throughout the group. This project was proposed to improve the extent to which this is currently done. In a research group, where the outcome of a research project may be significant for other ongoing investigations or to establish the scope of future projects, a mechanism for the diffusion of such information throughout the group is a key issue for the success of the research body. Another concern to improve the collaboration among the group is to ensure that its members are up to date with published work produced by other researchers on relevant topics for their investigation. Finally, another problem to be solved is the difficulties for the researchers to publish the reports generated since this task is often done by hand, which is time-consuming and makes maintenance difficult.

4.2 Project proposal

The main functionality expected from this project was established from the following project proposal:

This project will investigate the design of an automated system for the publication of technical documents on the World Wide Web. The system will allow its distributed users to commit documents for publication on the web and will then take care of all aspects of making the document available via the web including automatic forwarding of the documents for review prior to publication (if required), linking the document from appropriate document collections, building search indices to allow documents to be retrieved based on content or keywords, possibly publishing the document in different formats, etc. The web interface to the system should be designed to guide readers to documents of potential interest based on their past use of the system and stated interests and might provide features such as automatic generation of reports of new documents that become available, email notifications of new documents to potential readers, etc.

This proposal has been refined during the course of the project. Some aspects have been discarded after analysing the problem in depth, such as the publication of the document in different formats: there is no real need to do that, bearing in mind that the researcher can download the document and convert it to the format he is interested in using available converter tools. Having different versions of the same document in the server complicate the maintenance of the system without any advantage on the other hand. New functions have been required as well. The most significant is that the user must have a means to search for words or phrases within the whole collection of documents in order to ease his search for relevant documents to his research.

4.3 Existing system

The existing system currently being used at the Research and Development Division to address the problem defined in the previous

section is far from optimal and doesn't address all the aspects mentioned above. The only means a researcher has to obtain information about other's research is to browse through a set of individually managed pages and follow the existing links. Each time a new document is to be published, first of all the researcher has to launch an ftp client to store the file/s in an appropriate directory in the web server. Then, he has to edit by hand the HTML page where the document is to be placed; no further action is taken. Therefore, potentially interested readers are not notified about the presence of the new document, nor is the document linked to a proper collection to ease the search for the document. The rest of the researchers have no means to search for words or phrases within the document. Another drawback of this is the lack of a uniform interface, since each user is responsible for the maintenance of their own page.

4.4 Additional requirements

Some non-functional requirements were deduced in the early meetings held at Eircom's premises.

- The system was to be deployed in the Research and Development Division's Intranet site. Therefore, in addition to having to be a web-enabled application, its interface must integrate with the existing interface of the site.
- The application must operate in a Microsoft's Internet Information System 4.0 running on Microsoft's Window's NT 4.0.
- The mail system for the notifications function is cc:Mail.

Other requirements imposed on the system which have been especially considered are:

- **Portability**: a commonly accepted definition for a portable application is: "*An application is portable across a class of environments to the degree that the effort required to transport and adapt it to a new environment in the class is less than the effort of redevelopment*".

All design decisions have been taken bearing in mind that the initial environment for which the application was developed might change in the future, and trying to minimise the effort needed to migrate to a new environment. Thus, whenever possible, standards (a commonly accepted specification for a procedure or for required characteristics of an object) have been followed.

- **Scalability:** An application is considered to be scalable if, unchanged, it can handle increasingly complex problems. Often scaling yields problems such as reduced efficiency. Since the number of documents in the system is expected to eventually grow, it has to be prepared to deal with a high load of documents without compromising response times.

5. Analysis

Once the requirements of the system have been examined, the system analysis phase may be undertaken. There are two basic steps to be covered when following an Object Oriented approach: to create a set of use case diagrams and to decide the data model of the system. These steps are covered in the following sections.

5.1 Use case diagram

During the first stage in the OO analysis the requirements are refined into a set of use case diagrams. The following figure (Figure 5—1: Use case diagram) shows the use-cases established for Eircom's R&D Digital Library:

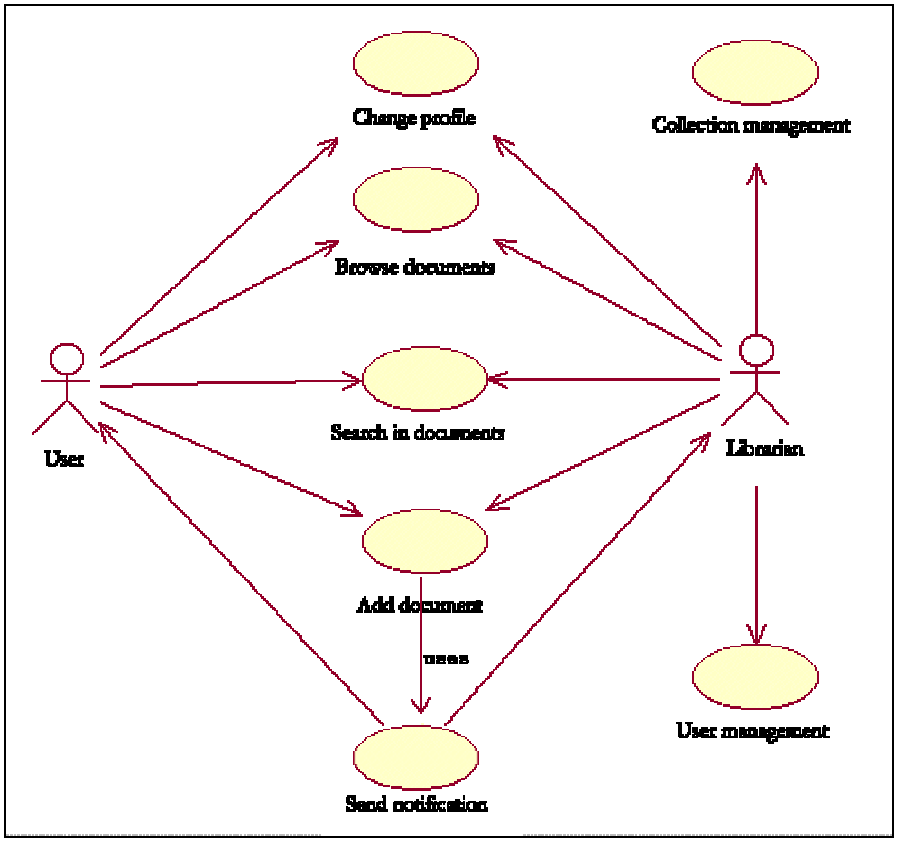


Figure 5—1: Use case diagram

A use case diagram describes who are the external components in the system (mostly live persons or other applications) and what activities they perform using it. Two external components (a.k.a. actors) have been identified: the librarian and users. These actors define roles that may be mapped to users (for instance, a single user can be granted both roles). As shown in Figure 5—1: Use case diagram, librarians can perform all the operations plain users can, plus management of users (addition, deletion or update) as well as collection management (creating new collections, building collections or deleting collections).

5.2 Object model

Another task to be performed in the analysis stage of the development process is to model the domain of the application, i.e. to represent a static structure of the system. The object model, a.k.a. class diagram, shows the existence of classes and their relationships in the logical view of the system. A class in the diagram is a collection of objects with common structure, common behaviour, common relationships and common semantics.

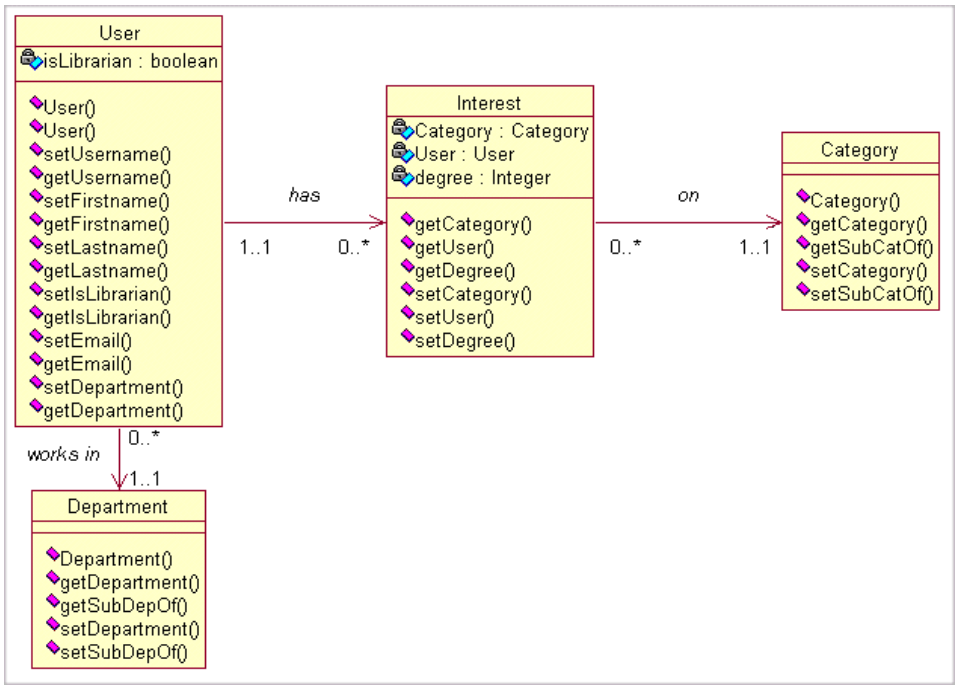


Figure 5—2: Object model diagram

6. Design

As stated in section 4 System Requirements it is expected that the system will provide the user with a means to perform searches for words that appear in any part of the document within the whole collection of documents. Two approaches to implement this functionality must be considered. One is to operate directly on the source files with a pattern matcher like grep or it is possible to create data structures with information extracted from the source documents (indexes, trees, etc) to ease the retrieval task. Both have pros and cons: grep is simple to use and does not require any programming effort, but it does not scale well with the size of the collection; on the other hand, processing the collection to create the mentioned data structures is a scalable solution as long as the data structures are scalable, but this implies developing a complex system. One example of a retrieval mechanism that makes use of a fast exhaustive searching is the GLIMPSE (GLobal IMPlicit Search) system [GLIMPSE], which scans the entire text for each query within a reasonable search time. Nevertheless, this system is intended to search for files in a UNIX file system. Hence, it cannot be considered for the proposes of this project.

As we will see in this chapter, the proposed solution uses a public domain system, mg, to perform this task. The provision of this functionality to the system and the use of the mg software have turned out to be determinant to the rest of the system design.

6.1 System architecture

Figure 6—1: System architecture shows the modules that make up the proposed solution, their auxiliary components and the existing relationships

among them. Each of these modules is thoroughly examined in the following subsections.

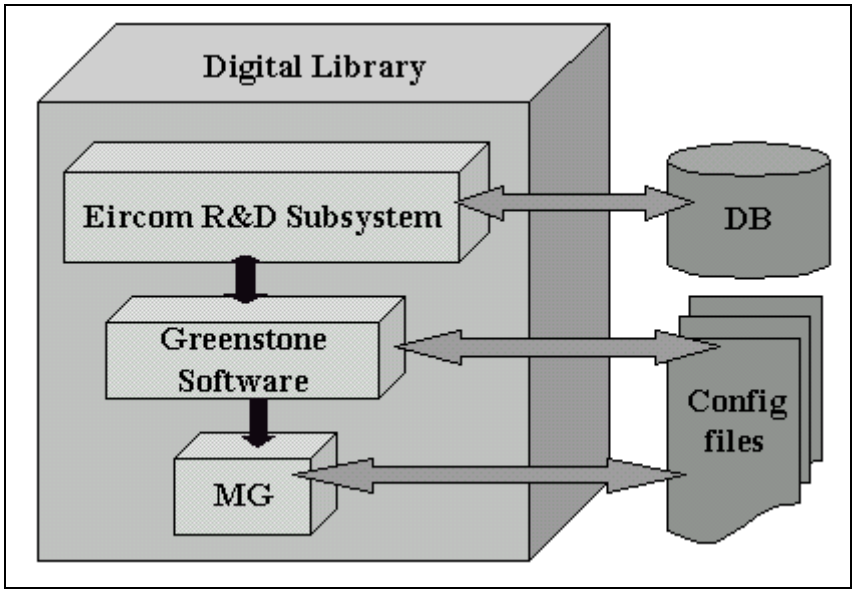


Figure 6—1: System architecture

The system requirement to allow the user to perform full-text searches of all the documents determined the overall architecture of the system; it demanded to use an indexing tool (mg, section 6.1.1), and this tool in turn demanded to use the greenstone software to provide a web interface to it. Finally, the use of this software determined how the top-level module was to be developed.

6.1.1 The mg system

The MG (Managing Gigabytes) system is a public domain collection of programs that comprise a full-text retrieval system [MG] that gives access to a collection of documents, every single word of which is indexed in the system. Mg is written in C, initially for Unix machines (although a Windows version is currently available) and is made up of two subsystems: the first

one is used to compress and index a collection of documents (creating a database out of them) and the second is an interactive query tool (this query tool operates only on the indexes to do the searching, so the original collection of documents is no longer needed, thus requiring far less storage space).

The text compression and index construction process is carried out by the command `mg-build` *`collection-name`*. This script uses some others, basically, `mg_get, mg_passes` and `mg-query`.

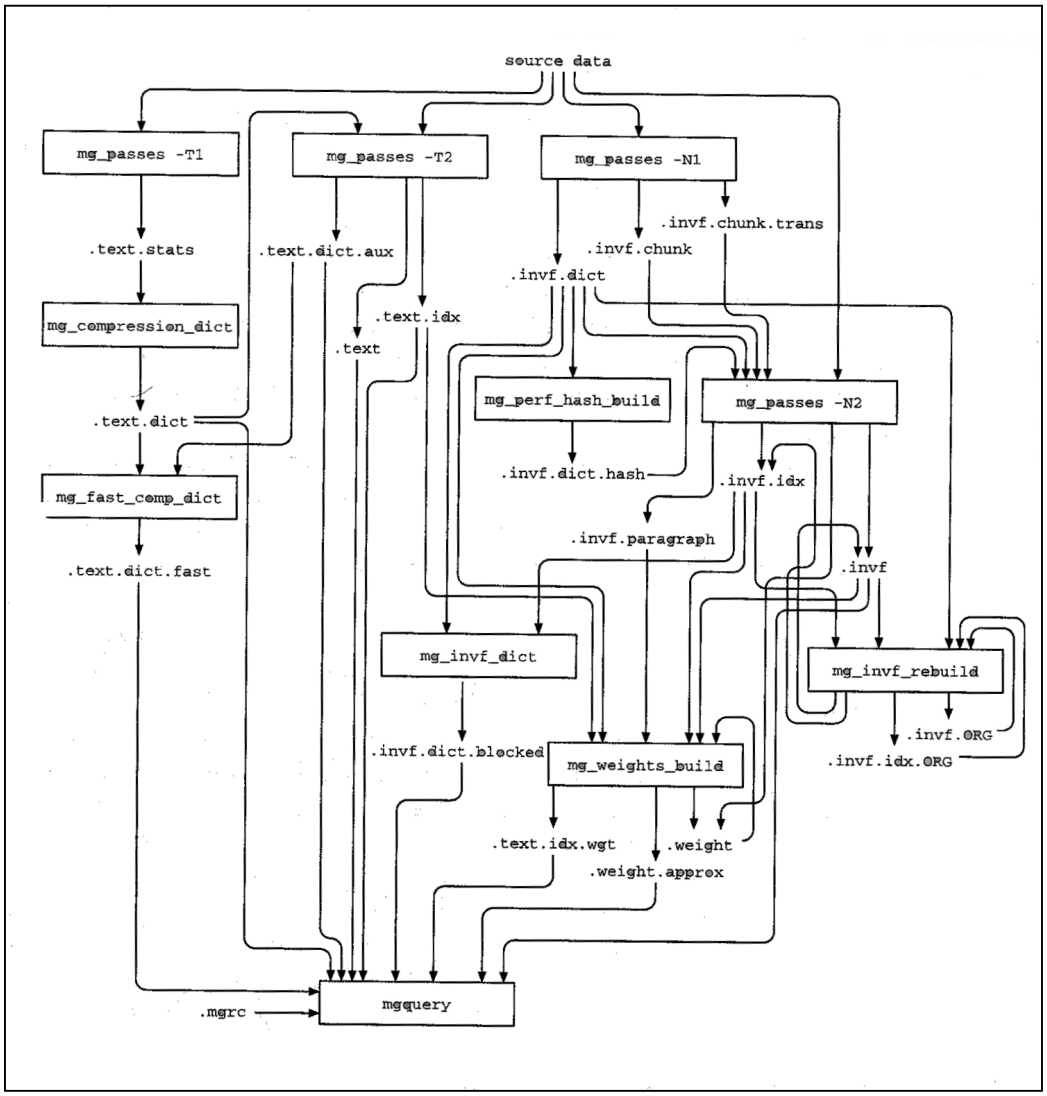


Figure 6—2: Internal mg structure

At the text compression stage, the `mg_get` script provides raw documents to the `mg_build` command. It accepts as a parameter the name of a collection and outputs all the documents to be stored, adding a ^B at the end of each document. The user has to implement a new version of this command to deal with different types of documents. After it, `mg_passes` is invoked several times with different parameters to create different structures. This script compresses the text using a “Huffman coder for text compression” (Huffman coding calculates the output representation of a symbol, based on a probability distribution, assigning short codewords for likely symbols and long ones for rare ones) and a “zero-order word-based semi-static model for ASCII text” (which first creates a lexicon with all the words and then compresses the text using the parameters accumulated in the first phase).

Once the text and images have been compressed, they have to be indexed; the mg system uses the inversion method, which creates different inverted files during the process and merges them. Then, the index is compressed, even if this is not a very important step for the final result since the size of the index is typically around 0.1 times the size of the original collection.

Finally, the command `mgquery` allows the user to query all documents in the system, through its command-line interface. The system supports four type of queries:

- Boolean queries: The user can type terms to search for linked with Boolean connectors.
- Ranked queries: The system searches for several terms without connectors, and then returns an ordered list of documents by closest matches with a similarity score.
- Approximated ranked queries: The same as before, but less accurate in order to decrease response time.

- Number of document: The user can ask the system for a certain document, giving its number, since all documents in the collection are classified with one.

Nevertheless, and despite being a very useful tool, the mg system has several drawbacks. Firstly, it has a simple command-line interface, as it's intended to be a retrieval engine. Secondly, the system can only handle a single interactive session at a time so it doesn't allow concurrent users. Finally, it only works with a static document collection: the whole collection of documents has to be indexed before any search on the documents can be done, and once a collection is built, no updates to the collection can be done; to add a new document, the entire collection has to be rebuild, which can be time-consuming for large collections. To solve some of these limitations, another system, the greenstone software, has been developed on top of it. The next section explores this software, analysing its functionality and limitations.

6.1.2 Greenstone system

Greenstone is a GPL Digital Library Software package for building multimedia collections and making them available via a browser and uses mg as its kernel. It is the result of an ongoing research project in the Computer Science Department at Waikako University in New Zealand, started in 1995, and it is publicly available through the Internet [Greenstone]. The aim of the project is "not to run a new library but to develop the underlying technology and make it available so that others can use it to create their own collections".

Information in a library using this software is stored in collections of documents (group of related documents). Metadata (descriptive information

about the documents) can be associated to documents. Source documents can be in any format from which ASCII text can be extracted in order to index them.

6.1.2.1 Search and browsing facilities

The system provides browsing and powerful search facilities to find information in a collection. It automatically organises documents according to metadata associated with them; thus, no links have to be inserted by hand. Searching is full-text (making the entire text of all documents available for searching), and depending on the metadata fields associated to documents in a collection, the user can choose between indexes to be search (title, author, etc). Figure 6—3: Greenstone search interface shows these facilities for a demonstration collection. The metadata defined for this collection is shown in the navigation bar: subject, title and organisation. Users can type several search words in the search box and search for some or all of them, depending on what they select from the “list box” above the search box. In case the user chooses to search for some of the words that are likely to appear in the document he is looking for, the list of documents that the system return is ordered according to how many search words it contains, considering rare words more relevant than common ones. Quotation marks must be used when looking for phrases within documents.

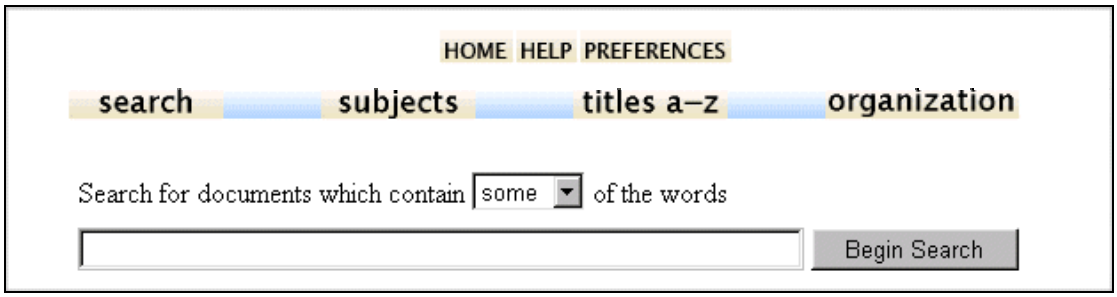


Figure 6—3: Greenstone search interface

The greenstone software also allows advanced users to configure the system, create new collections, update them, add metadata to documents, etc. This is the trickiest part of the system, since most of this operations must be carried out dealing with configuration files and command-line instructions. Section 6.1.3 Eircom R&D system, describes a system that provides a web interface to this mechanism, and automates most of the processes that would have to be carried out manually if working with this software.

6.1.2.2 Building collections

To be able to build a collection, the user needs to know how the greenstone software is structured in the file system and where he can locate the configuration files used by the system. Figure 6—4: Greenstone collection structure shows this.

Figure 6—4: Greenstone collection structure

- The *archives* directory stores documents in GML format
- The *building* directory stores the indexes during the building process
- The *etc* directory stores configuration files about the collection
- The *images* directory stores images used in the collection
- The *import* directory stores the source files for the collection
- The *index* directory stores the indexes that are publicly available to search the collection.

This is a two stages process. First, source documents have to be converted to a format that can be indexed, i.e. text has to be extract from the document. This step, called *import*, takes every file stored in the import directory and converts it to a GML file (Greenstone Markup Language), which is basically a text file with metadata associated, and stores it in the *archives* directory. Once the new material has been imported, the collection should be rebuilt. This step, carried out by the command *buildcol*, creates the indexes for both searching and browsing. The MG software is used to do the searching. The *buildcol* process invokes the MG module *mgbuild* (see 6.1.1 The mg system) to create the set of indexes on associated metadata and to compress the information contained in the GML files.

Every collection in the library must have a configuration file (*etc/collect.cfg*) that defines which metadata is defined for the documents in the collection and therefore which indexes must be created for the collection. It has also to be specified the types of the source documents that the collection will comprise, so that an appropriated “plugin” (Perl module) to convert the raw material to GML documents.

The most important commands that can be used in the configuration file are shown in the following table:

`Creator` *the creator of the collection* **`indexes document:Title document:text`** *the indexes defined on the collection. This collection defines an index on the Title of the document and another one on the text of the documents. Therefore, users will be able to search for words or phrases that appear either on the title or in the text.* **`plugin IndexPlug`** **`plugin HTMLPlug`** *List of plugins to control the import process. IndexPlugin mission is to associate metadata to documents, while HTMLplugin converts documents form HTML to GML format.* **`classify Hierarchy hfile=sub.txt metadata=Subject sort=Title`** *describes which files define metadata about the documents. This command specifies that the file named sub.txt contains the subject associated with the documents. The structure of this file is analysed at the end of this chapter.*

Table 6-1: Collection configuration commands

As introduced in Table 6-1: Collection configuration commands, IndexPlug associates metadata to files. This is done by manually writing a file defining which metadata values are to be associated to each file in the collection; every single file to be added to the collection must appear in this file, called *index.txt*, with its metadata, otherwise it won't be included in the collection (see Figure 6—6: Greenstone index.txt configuration file) Other configuration files that have to be updated manually are sub.txt and org.txt, stored in the *etc* directory. The first one is used to maintain a hierarchy of categories upon which a document can be classified and therefore linked into appropriated collections. The second one, with identical structure, is used to maintain the organisational structure of the division.

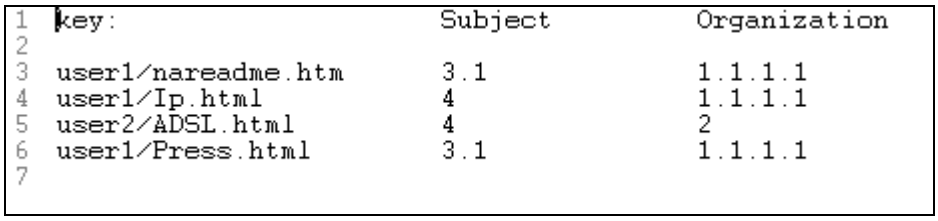
The following figure (Figure 6—5: R&D Collection configuration file) shows the configuration file used for the default collection in Eircom R&D Division.

```
# Configuration file for the default collection in Eircom
# R&D Division

creator       sanchezo@tcd.ie
maintainer    sanchezo@tcd.ie
public        true

indexes        document:text
defaultindex   document:text

plugin         GMLPlug
plugin         HTMLPlug
plugin         ArcPlug
plugin         IndexPlug
plugin         RecPlug

classify      Hierarchy hfile=sub.txt metadata=Subject sort=Title
classify      AZList metadata=Title
classify      Hierarchy hfile=org.txt metadata=Organization sort=Title
classify      List metadata=Howto


collectionmeta collectionname    "R&D Collection"
collectionmeta iconcollection    "_httpprefix_/collect/RDCollection/images/demosm.gif"
collectionmeta iconcollectionsmall   "_httpprefix_/collect/RDCollection/images/demosm.gif
collectionmeta .document:text    "documents"
```

Figure 6—5: R&D Collection configuration file

Figure 6—6: Greenstone index.txt configuration file shows the structure of the *index.txt* file. It is a sequential file with an entry for each document in the library. Only documents included in this configuration file are recognised by the greenstone software. A document in the library is described with three fields: *key, subject* and *organisation*. The first one tells the greenstone software where in the file system it can find the document. The second one,

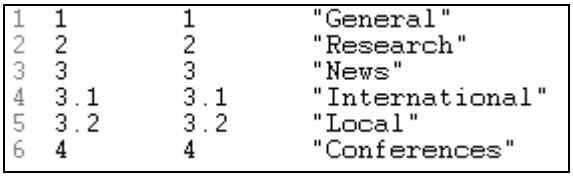
subject, is and identifier for the category of document that has been associated with the document. This field must match with an entry in the *sub.txt* configuration file. Finally, the same pattern applies for the third field in the file, *organization*, which is used to classify documents upon the company's organisation chart based on the author of the document.



```
1  key:                        Subject         Organization
2
3  user1/nareadme.htm          3.1             1.1.1.1
4  user1/Ip.html               4               1.1.1.1
5  user2/ADSL.html             4               2
6  user1/Press.html            3.1             1.1.1.1
7
```

Figure 6—6: Greenstone index.txt configuration file

Figure 6—7: Greenstone sub.txt configuration file plots the structure of this file. The identifier used in the index.txt file to classify documents depending on their category (or subject) is duplicated in this file, as each entry in the file must specify this value in its two firsts fields (this is required by the design of the greenstone software). Finally, a description for that code must be provided. Note that the structure of the *org.txt* is identical to this one.



```
1  1       1       "General"
2  2       2       "Research"
3  3       3       "News"
4  3.1     3.1     "International"
5  3.2     3.2     "Local"
6  4       4       "Conferences"
```

Figure 6—7: Greenstone sub.txt configuration file

The greenstone software is widely used as a technology to run digital libraries. The fact that it is available at no charge under the terms of the GNU (General Public License) and that it offers a full range of facilities, has encouraged universities (Middlesex University, Rutgers University), United Nation agencies (such as the United Nations University) and government organisations to adopted it. Nevertheless, the requirements of this project were beyond the capabilities of the system; the greenstone software is

conceived to be mono-user: a “librarian” has to store the source files in the directories afore mentioned, then he has to create the configuration files manually and then build the collection; there are no mechanisms available to add files to a collection through a web interface; since it is mono-user, there is no concept of users in the system and therefore no recommendation schema can be defined. The next section (6.1.3 Eircom R&D system) describes a system designed to address these drawbacks.

6.1.3 Eircom R&D system

This module, built on the greenstone software, has been conceived in order to address the problems and limitations discussed above, so that the application fulfils the system requirements. The functionality this system is responsible of is:

- **To introduce the concept of users**. The system keeps track of authorised users by storing a record in the database for each of them. Users must log onto the system to be authenticated and to have access to the library. Once the user has logged in, the system, using HttpSessions, maintains this information during the course of the session. Users can be managed by a librarian.

- **To automate the publication of documents**: users are provided with a means to upload a file to the collection. A pop-up window allows the user to choose which file from their hard disk they want to upload. Once the document has been chosen, the user must declare the topic of the document (category). To do so, another pop-up window will show the hierarchy of categories available. Once a category has been selected, the file will be transparently stored in an adequate directory in the web server and added to the appropriated

configuration files in the greenstone software (i.e. *index.txt*), as described in Section 6.1.2.2.

- **To allow users to change their profile**: As mentioned before, once the user has logged in, information about him or her is maintained by the system. This information (first name, last name, email, department, if they have librarian rights or not, and the stated user interests in categories of documents) is available for updating at any moment. If the user selects to update its profile, his or her details are shown on the screen, along with a list of categories to which he or she is subscribed and with a list of all available categories to which the user might like to subscribe. Through this screen, the user can update his or her interests, which are automatically updated for further interactions with the system.

- **To add and delete categories of documents in a distributed way.** Users can update at any moment the categories of documents defined in the system, by adding new categories at any level of the hierarchy or by deleting them. These changes transparently update the configuration files of the greenstone software, which turns out to be an onerous task when it has to be maintained manually.

- **To send notifications** about new documents to potentially interested users. When a user adds a document to the collection, an email is sent to all the uses that might be interested in that documents. A user is supposed to be interested in a category of documents if that category is registered in his or her profile, as well as if he or she has uploaded documents classified upon that category. The notification informs the user about the file that has been uploaded, its author, and the category it has been classified in.

6.1.3.1 User Profiling

Users of the digital library are in need of a recommendation service that permits them to be up to date with relevant documents available in the library. If the system implements such mechanism, it has to keep track of its users interests by means of a method called user profiling. The recommendations are then done based on this profile. Two approaches to this method must be considered: content-based, in which the system tries to recommend items similar to those a given user has liked in the past, and collaborative, in which the system identifies users whose tastes are similar to those of the given user and then recommends items they have liked. This second approach has been discarded for this solution, because of its shortcomings: firstly, when a new document appears in the collection, no information about this document is available (no other users have read it, so the system doesn't know whether other users are interested in the document or not, and if they are, to what extend); secondly, it is difficult to deal with users who have unusual interests compared to the rest of the users.

A third approach to this problem can be taken: a combination of both methods, trying to incorporate the advantages of content-based and collaborative recommendation whilst inheriting the disadvantages of neither. This approach has been successfully applied in [Fab], but unfortunately is not suitable for the system described here. The main reason to discard any collaborative approach is that the expected number of researches is not big enough for the results of this method to be accurate. Since the researchers are organised in groups and subgroups in a hierarchical manner, and each group is specialised in a certain area, collaboration would be restricted to research groups, where generally, all researchers are interested in all documents generated by the group.

Therefore, a content-based recommendation schema has been followed, as introduced in this section. Users are provided with a mechanism whereby they can manipulate their profile stating the categories of documents they are interested in. In addition, users are automatically considered to be interested in a category of documents when they add a document on that category to the library.

6.2 Database Design

The database, stored in the server-side, has been created to automatically generate the configuration files used by the greenstone software and to implement the functionally assigned to the Eircom R&D system. A brief description of its tables is noteworthy.

Categories: describes the categories of the documents that have been defined in the system. Since categories are structured hierarchically, two fields are needed: the name of the category (*Category*) and which category the category subcategory of (*SubCatOf*). Top-level categories (those without parent) leave this second field as null. A third field (*IndexCod)* is used to generate the configuration file about categories.

Departments: same structure as Categories, but about departments.

Users: information about users is needed in order to: let users log onto the system (username/password); inform other users via email that a new document has been added to the collection by a specific user (first name and last name); determine which operations a given user is allowed to perform (IsLibrarian); send others notifications about new documents that might be of interest of them (email); link a specific user's documents to appropriated collections (department).

Interests: information about users interests in certain categories of documents must be stored in order to send them notifications when a new document of a category they are interested in is added to the library. Since the cardinality of the relationship is N:N (a user can be interested in multiple categories and a category may be of interest of multiple users), this information cannot be stored in either of the tables in a normalised form and therefore the use of this table is justified.

Figure 6—8: Database Schema shows the database design introduced in the previous paragraphs

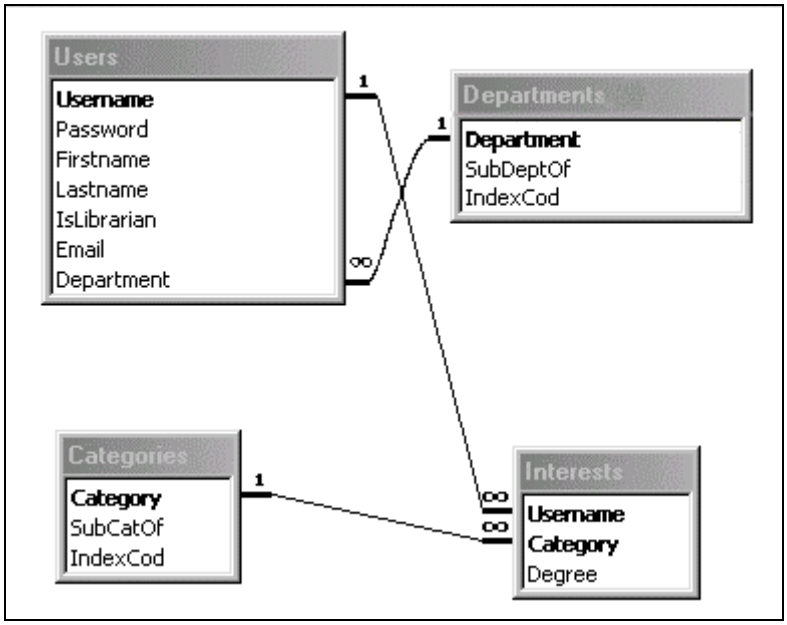


Figure 6—8: Database Schema

7. Implementation

This chapter describes how the system presented over the previous chapters has been implemented, focusing in the top-level module of the system. The description of this module covers how the functionality assigned to this module in the design chapter has been transformed into a working solution.

7.1 mg and greenstone modules

While the mg system has been used as is, without any modification, the greenstone software has been adapted in different ways in order to allow seamless integration with the top-level module, the R&D system. Although it offers a set of files to customise its interface, these files only deal with the interface for the functionality offered by the software, which is to be expected. Therefore, these files have had to be restructured to plug the functionality added by the new module. These configuration files are stored in the *gsdl/macros* directory in the application tree; their extension is .dm and are organised as a set of macros used by the system to generate the html files. A macro associates a value to a label with the following pattern: *_namemacro_{content}*. Macros can appear as well as content for other macros. An example of these files is included in appendix A.3 Greenstone home page configuration file. This file defines the content for the home page of the library.

This part of the implementation process turned out to be one of the trickiest parts of the entire project. The fact that no documentation is provided with the software about which macros are defined in which file, and

that these files are structured in packages with macros that overwrite other macros defined in other packages, made this task last longer than planned.

7.2 R&D module

A detailed description about this system and how it interacts with the user has been covered in 6.1.3 Eircom R&D system. The following pages cover the programming decisions taken to implement each of the functions previously analysed.

7.2.1 Distributed users

The system stores a row in the table *Users* in the database for each of the users registered in the system. Before having access to any of the functions of the system, the user must log on, using a common username/password form. When the user fills these fields, the information provided is checked against the database via a servlet called CheckP.java. If the data introduced allows the authentication of the user, access is granted to the system. The servlet retrieves all the user information from the table (first name, last name, email, etc), creates a bean of type `User` and stores it in the `HttpSession`. The first thing the rest of the pages do is to retrieve the bean from the `HttpSession`. If the reference is valid, it implies that the user has successfully logged into the system, and therefore is allowed to visit that page. If not, the user is redirected to the login page to log into the system (the same approach is taken when the user introduces a wrong combination of username and password). As an aside, an extra task performed by this servlet is worthy to mention. In its `init()` method, which is called when the servlet is loaded in the servlet container, it creates an

instance of the `DBConnectionManager` class, which is used to create and manage the pool of `Connections` introduced in section 3.5.2 Database (server-side storage). This servlet is configured to be loaded on Tomcat's start-up; this way, the servlet (and therefore the `DBConnectionManager`) is loaded before any interaction with the system is carried out.

7.2.2 Publication of documents

When a user chooses the option to upload a new file to the collection, and after checking that they have successfully logged in, a form is presented to the user. In this form he or she has to select the file containing the document from his hard disk; this can be done by pressing the *Browse* button in the form, which pops up a window to navigate the file system. This behaviour is achieved by defining the "input type" HTML tag of this field as FILE. Figure 7—1: Uploading a document shows a screenshot for this action. The form data is sent as `multipart/form-data` as described in RFC 1867 [RFC1867]. Since multiple types of information can be included in the same HTTP request after the submission of the form, this data is sent with headers that specify the type of information that is being sent and the boundaries of the stream. These delimiters are the only means the web server has to split the sent data into the different fields. The web server needs to parse the stream to obtain these fields. There are several java classes available on the Internet to perform this task and to free the programmer from having to deal with the mentioned delimiters. The com.oreilly.servlet package is one of the most complete solutions. One of the classes in the package, called `MulipartRequest`, is the utility class to handle `multipart/form-data`. While it cannot handle nested data (multipart content within multipart content) or internationalised content (such as non Latin-1 filenames), it can receive arbitrarily large files, which is the only feature required by this application. All the constructors of this class have as a parameter the `HttpRequest` object. Once the

`MulipartRequest` object has been created, methods are provided to `getFile(), getParameter(), readAndSaveFile(),…`

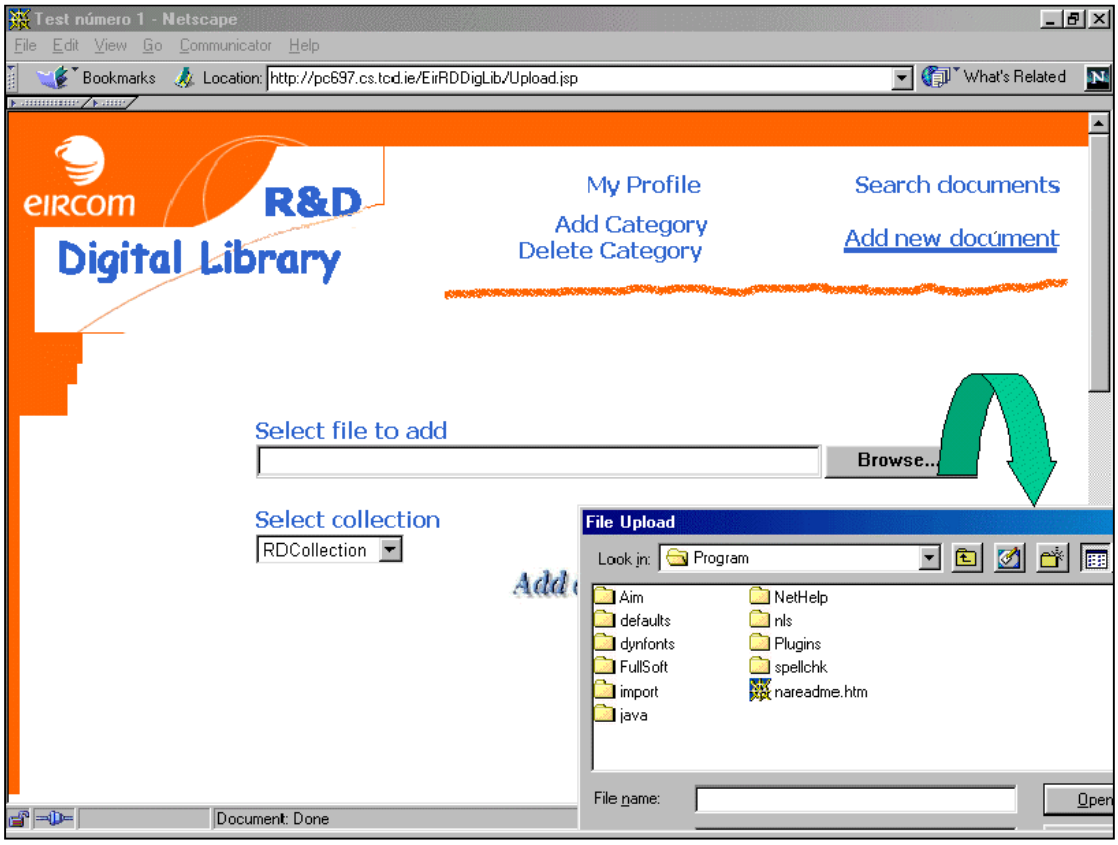


Figure 7—1: Uploading a document

After having indicated the name of the file and the collection to which the file is to be uploaded, the user must specify the category of the document. To do so, a new window pops-up to show the available categories in the system. This is shown in the next section, 7.2.3 Addition and deletion of categories. Once all the information regarding the new file has been gathered, the UploadLast JSP writes the file to the appropriate directory in the file system, updates the file *index.txt* (it appends a new line at the end of the file) to let the greenstone software know about the new file, and sends email notification to researchers that might be interested in the newly updated document. In order to actually send the emails, it uses a java

class as a bean, *MailBean.java*. It loops over the users that have been selected from the database and for each of them, it sets the bean's properties (recipient, body, subject, etc) to customise the email sent to each user.

7.2.3 Addition and deletion of categories

Several functions of the system need to present the hierarchy of categories currently available in the system to the user. Some of them, such as "Add Category" and "Delete Category", also allow the user to modify these categories of documents. In order not to replicate the code in different pages, a JSP (Categories.jsp) and a servlet (CatsRec.java) have been implemented in such a way that they can customise their output depending on the operation the user wants to perform. Figure 7—2: Categories shows this customisation depending on the action being executed.

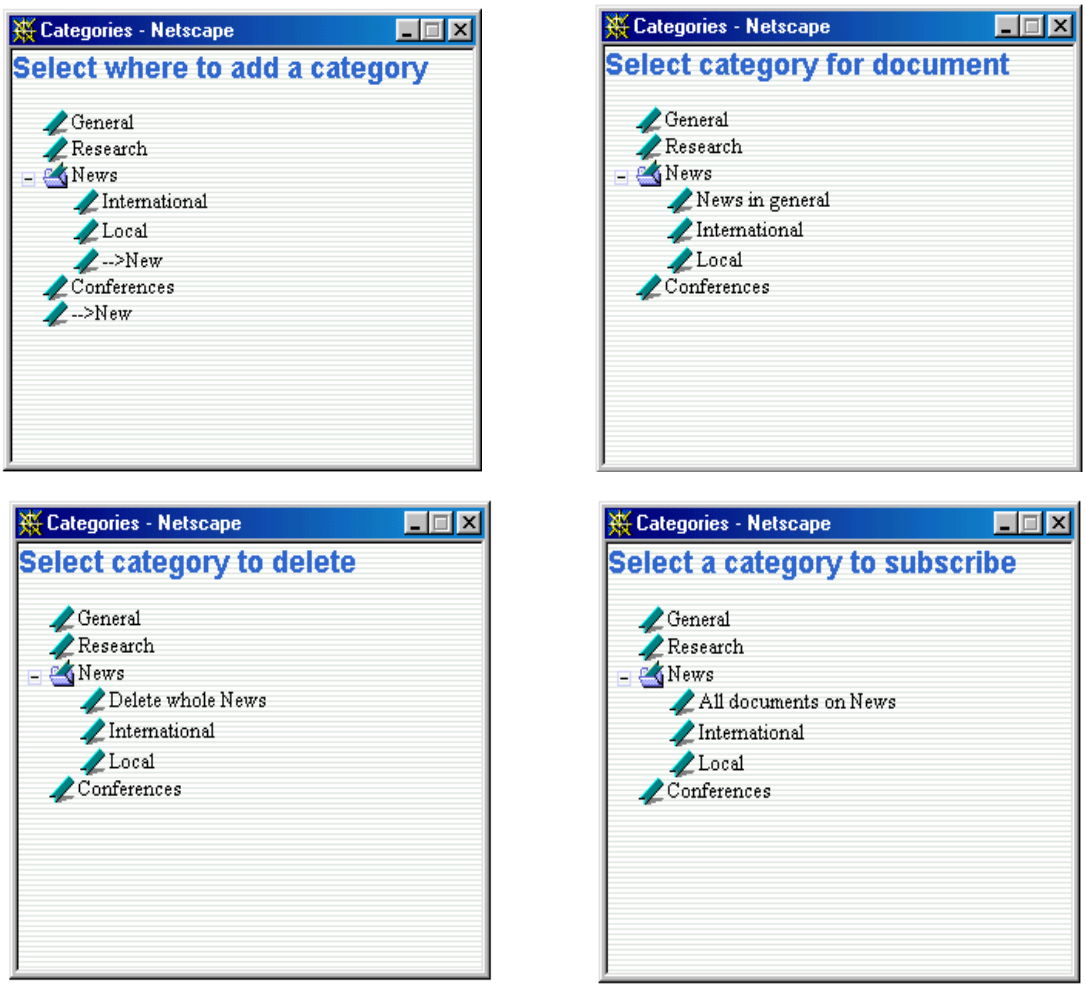


Figure 7—2: Categories

The use of two components (the JSP and the servlet) is justified since HTML (in fact, JavaScript code) output has to be generated (by the jsp) and also some business logic has to be programmed (by the servlet), as discussed in 3.1.6 JavaServer Pages ™. The JSP obtains the information from the servlet and then it displays it. The servlet has to create a hierarchical structure from a relational table from the database. The interface that it exposes to the JSP returns a String with the JavaScript that the JSP generates in the response object and that is returned to the user. This JavaScript code is responsible for popping a window with the hierarchy that allows the user to expand or contract nodes dynamically without having to regenerate the structure. Figure 7—3: JavaScript code example introduces the pattern to create dynamic trees showing how to create a tree with two children, one of them being a tree itself with one child, and Figure 7—4: JavaScript code to generate the hierarchy of categories shows the code to generate the tree shown in Figure 7—2: Categories. This solution proves to be very efficient, since no unnecessary reloads or queries to the database are carried out.

```
<SCRIPT LANGUAGE="JavaScript1.2">
                <!--
                  var mySubTree = new Tree();
                  mySubTree.addTreeItem("subTree item a");

                  var myTree = new Tree();
                  myTree.addTreeItem(mySubTree);
                  myTree.addTreeItem("my tree item A");

                //-->
</SCRIPT>
```

Figure 7—3: JavaScript code example

The following is an explanation on how the servlet creates the String representing the whole structure: is worthy to understand how it works: it is done by recursively calling a method in the servlet `getSubCatsRec`. This method gets an array of Category (see Figure 3—3: JavaBean example) created with all the rows in the Categories table in the database, and parses it, detecting which nodes have to be added to the String. For instance, while

dealing with the top-level categories it searches the array looking those that have no parent (its SubCatOf property is null). Then, for each top-level category, it calls the method again trying to find its children, i.e. those categories that have its SubCatOf property set to that top-level category, and so on. The leaves are identified when no category references them as parent category.

```
<SCRIPT LANGUAGE="JavaScript1.2">
 window.onload = loadTrees;
 // load Tree component
if (document.layers) {
  document.writeln('<SCRIPT SRC="tree_n4.js"><\/SCRIPT>');
} else if (document.all) {
  document.writeln('<SCRIPT SRC="tree.js"><\/SCRIPT>');
}

// load my trees
function loadTrees() {
            window.cat0 = new Tree( "cat0" );
           cat0.addTreeItem("General","opener.location.href='/EirRDDigLib/AddCat2.jsp?Actio
n=Add&Cat=General';close();");
           cat0.addTreeItem("Research","opener.location.href='/EirRDDigLib/AddCat2.jsp?Act
ion=Add&Cat=Research';close();");
            window.cat3=new Tree("News");
            null
            cat3.addTreeItem("International","opener.location.href='/EirRDDigLib/AddCat2.jsp?
Action=Add&Cat=International';close();");
           cat3.addTreeItem("Local","opener.location.href='/EirRDDigLib/AddCat2.jsp?Action=
Add&Cat=Local';close();");
            window.cat3.addTreeItem("--
>New","opener.location.href='/EirRDDigLib/AddCat2.jsp?Action=Add&Cat=News';close();");
            cat0.addTreeItem(cat3);
           cat0.addTreeItem("Conferences","opener.location.href='/EirRDDigLib/AddCat2.jsp?
Action=Add&Cat=Conferences';close();");
            cat0.addTreeItem("--
>New","opener.location.href='/EirRDDigLib/AddCat2.jsp?Cat=&Action=Add';close();");
            showTree(window.cat0,5,35);
}
</SCRIPT>
```

Figure 7—4: JavaScript code to generate the hierarchy of categories

Comparing these two figures, the reader can observe that in the second one an extra parameter is used when adding an element to a tree. This parameter specifies the JavaScript action to be taken if the user clicks on that entry and is generated dynamically by the servlet. So, for instance, the action

```
"opener.location.href='/EirRDDigLib/AddCat2.jsp?Action=Add&Cat=General';close();"
```

calls the JSP called *AddCat2.jsp* with the query string *Action=Add&Cat=General* (a presentation view without programming code responsibilities that asks for the name of the category to be added). This JSP is displayed in the application's main window, as instructed by *opener.location.href*. Finally, it closes the window where the categories were displayed. This action is also customised for each of the system's functions that use the categories tree. For example, when subscribing to a category as described in 7.2.4 Change profile, this action would be

```
opener.location.href='/EirRDDigLib/AddCatUser.jsp?Action=Subscribe&Cat=Local';close();"
```

which calls another JSP with another query string to perform another action. Whatever is the action to be performed, if it changes the existing hierarchy of categories, these changes are made both to the table of *Categories* and the file *sub.txt*, since this is regenerated using the new data in the table.

7.2.4 Change profile

User's can change their profile at any time by clicking on this option at the top of the page. This selection calls a JSP called *ChangeProfile.jsp* which displays the data about the user stored in the User bean associated with the current session, the categories to which the user is supposed to have an interest in, and a pop-up window with the whole hierarchy of categories to allow him to subscribe to any of them (see Figure 7—6: Change Profile). The action in this case is to call a JSP controller (AddCatUser.jsp) that interacts with the database to store a new row in the i*nterests* table. Finally, if everything work successfully the user is redirected to the *ChangeProfile.jsp*, where the newly updated information is shown to the user, who can do further changes in the profile. Figure 7—5: AddCatUser.jsp shows the code of this JSP.

```
<%-- AddCatUser.jsp (Controller), changes DB and includes other jsp --%>
<%-- Page directives: use sessions and import the --%>
<%@ page session = "true" %>
<%@ page import ="java.sql.*" %>
<%@ page import ="java.io.*" %>

<%-- Use the single instance of the DBConnectionManager with application scope --
%>
<jsp:useBean id="connMgr" scope="application" class="DBConnectionManager"/>
<%-- Use the single instance of the DBConnectionManager with application scope --
%>
<jsp:useBean id="userSession" scope="session" class="User" />
<% String strError = "/errorPage.jsp"; %>
//Check if the user has successfully logged in

<% if((session.isNew() == true ) ||(((session.isNew() == false ) &&
(session.getAttribute("userSession") == null))||(session.getAttribute("userSession") !=
null)&& (((User)session.getAttribute("userSession")).getUsername().equals("")))){
   request.setAttribute("error",(String)"NotLogged");
%>
<%-- if not logged,forward the user to the error page --%>
    <jsp:forward page="<%=strError%>" />
<% } else {
    <%--Retrieve parameters from the query string --%>
    String strAction = request.getParameter("Action");
    String strCat = request.getParameter("Cat");

    User theUser = (User) session.getAttribute("userSession");
    Connection con = connMgr.getConnection("EirDLConns");

   Statement stmt = null;

   String strSQL = null;
   if (strAction.equals("Subscribe"))
        strSQL="INSERT INTO Interests(Username,Category) VALUES('"+
        theUser.getUsername() +"'," +strCat +")";
   else
     strSQL="DELETE FROM Interests WHERE Username='"+
theUser.getUsername()
                  + "' AND Category='"+strCat +"'";
     try{
        stmt = con.createStatement();
        int dummy = stmt.executeUpdate(strSQL);
        stmt.close();
       // Free the connection. It goes to the pool to be taken later
       // if needed
       connMgr.freeConnection("EirDLConns",con);
     }
     catch (SQLException e){
        response.sendRedirect("../error3.html");
     }
  } %>
<jsp:include page="ChangeProfile.jsp" flush="true" />
```

Figure 7—5: AddCatUser.jsp

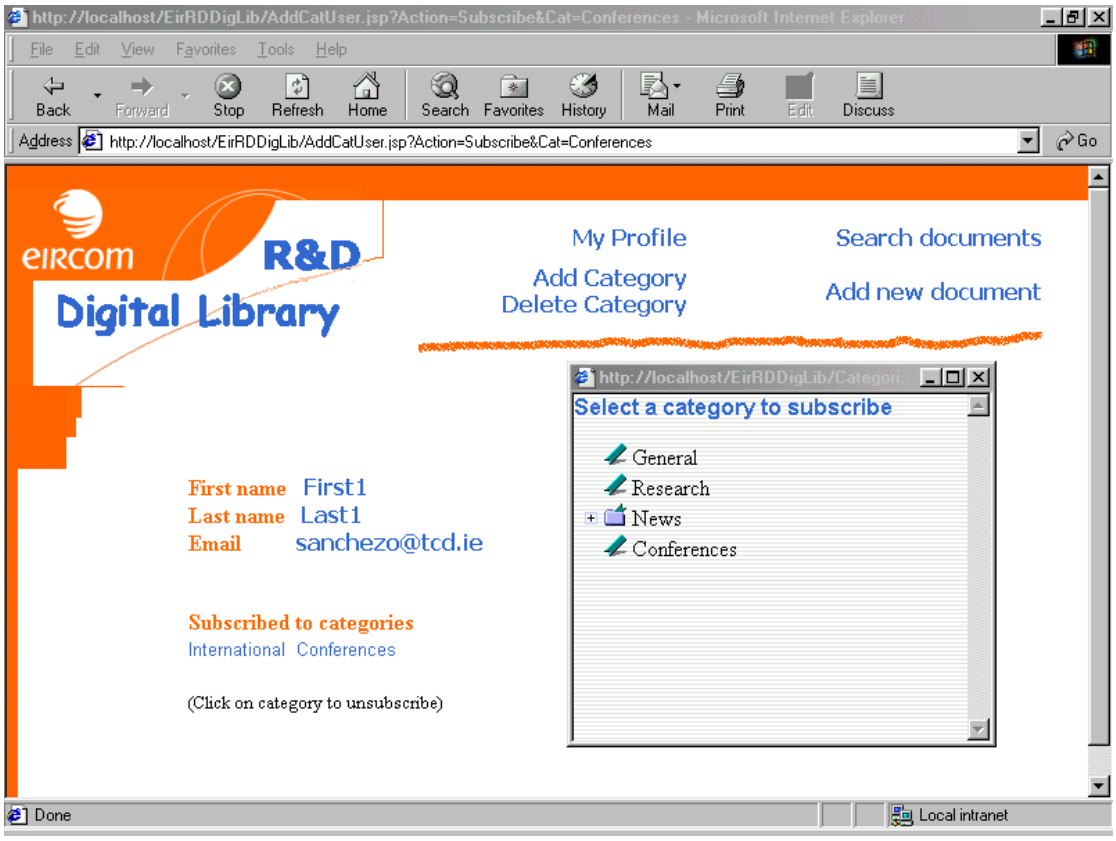


Figure 7—6: Change Profile

7.2.5 Librarian's extra functions

Finally, a short description about the librarian extra functionality introduced in previous chapters is worthy. The librarian can, in addition to all the functions discussed so far, manage collections of documents, users of the library and the organisational structure of the division.

The options to manage collections (build or create them) are mainly provided by the greenstone software, and therefore, out of the scope of this chapter. Nevertheless, an extra function has been added by this module relating to the management of collections. This is the publication of newly built collections, an action that has to be performed by hand in the

greenstone software (moving the new indexes to the *index* directory under the greenstone root directory).

Regarding the management of users, new functions to add, update, delete and list users have been added. All of them use the table of *Users* as a source of information.

Finally, the maintenance of the organisation of the departments is done in the same way as for the categories, but using the table of *Departments* and the file *org.txt* instead of the table *Categories* and the file *sub.txt* (see Addition and deletion of categories for details).

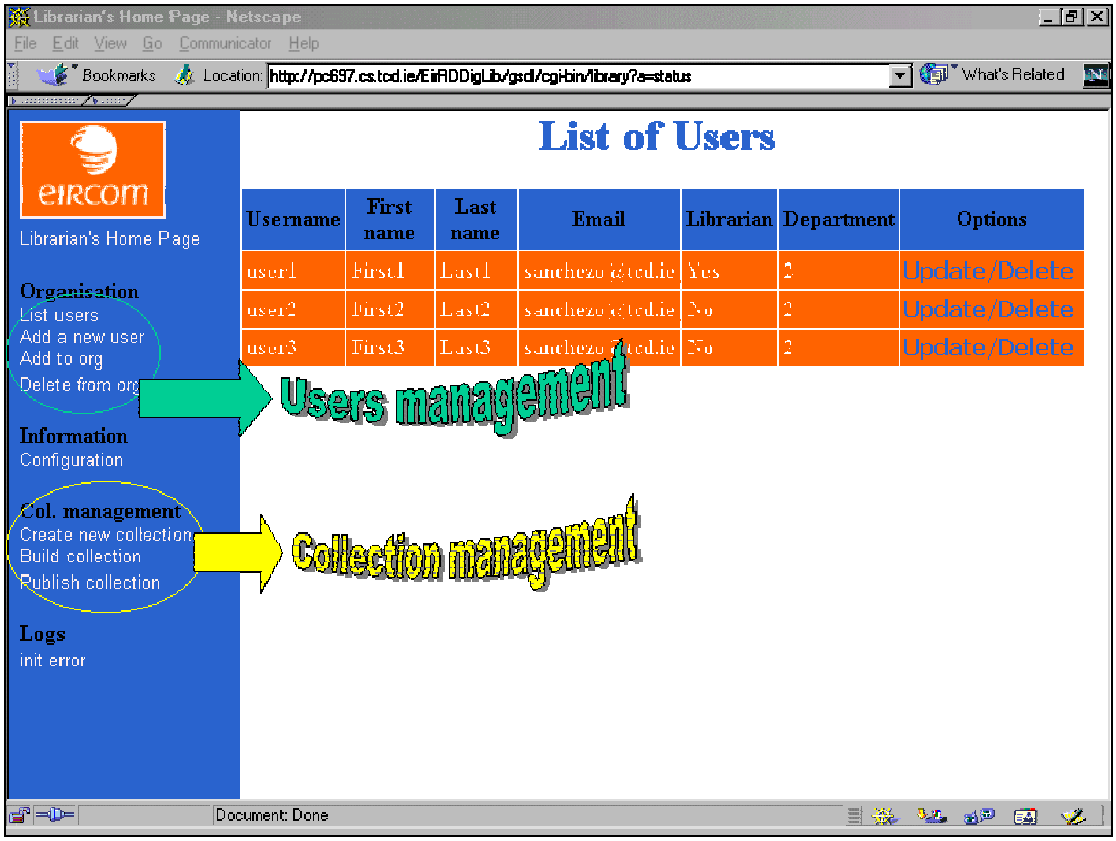


Figure 7—7: Librarian's Home Page

8. Evaluation

This project had two distinct objectives: the investigation of the best solution for the problem proposed and the implementation of a finished product to be deployed in a working environment.

The analysis of the functionality expected from the system turn out to have many similarities with the characteristics of a digital library, and therefore research was focused on this topic and the technologies that are available to implement a digital library. As discussed in this document, digital libraries are complex systems with many different areas of expertise involved and many social and economic implications. From this research, a better understanding of the scope and limitations of digital libraries was achieved, and new features were considered for the application to be built, such as the full-text indexing of the documents. All the literature on digital libraries agrees that the ability to search in all the documents stored in the library is one of the most important benefits introduced by digital libraries. The software to implement the tools to index documents and perform the searches on them is highly specialised and requires deep knowledge of its fundaments, and to try to implement would be an ambitious project itself. As introduced in this document, this is one of the active research areas in digital library. Thus, an investigation on the available tools to implement point out the mg system to be the most adequate for the proposes of this project. The use of this system had significant implications in the overall design of the solution due to its implementation as explained in the design chapter. To provide a web interface to the system, the greenstone software was used; finally, the extra functionality required was built on top of this software.

Regarding to the application delivered, it uses the best distributed paradigm available for this type of projects and a set of standard technologies backed by important companies. To ease maintenance, design

patterns such the Model-View-Controller pattern have been followed, as well as coding conventions. Scalability has been regarded as an important aspect for the system. The connection pool to the database and the indexing of documents are to solutions in this direction as well as two improvements to the performance of the system.

In conclusion, the objectives for this project have been achieved, and the outcome is a fully operative application that meets the requirements defined in Chapter 3 Requirements, a user manual and this report. Nevertheless, and due to the scope of the field, some extensions or modifications to provide new functionality could be considered, as introduced in the next section.

8.1 Further work

The solution described in this document has been designed and implemented following patterns to ease possible further work in this project. From the early stages in the development process, portability has been considered as a fundamental requirement in order not to tie the system to its initially intended environment and therefore to encourage its use. This increases the possibilities for the system to be extended or modified to provide extra functionality. As discussed in this document, digital libraries are a very dynamic field, with a lot of research being carried out. Not all the areas being explored are meaningful for this project; for instance, interoperability between digital libraries is not an issue, since this library has been created to facilitate the dissemination of information within the division. Others, in the other hand, are relevant for this work, such as preservation of digital material, organisation of the library, natural language processing and the format of the information.

The system is not equipped with any facility to deal with preservation of the material kept by the library and therefore backups need to be done by hand, as well as any refreshing or any migration of the information has. Since formats in which information is stored are frequently replace by new versions, future readers will need to be able to recognise the formats to display the information successfully.

Regarding to organisation of the library, it is noteworthy that in this system it is imposed by the greenstone software. Information has to be stored in collections or documents, and an important issue is how to organise it for storage and retrieval. This software considers all documents in the library as single objects; if a more complex model is needed, a best solution needs to be explored.

Although powerful search facilities are implemented in the system that allow the user to search for words or phrases in any part of the document, as well as coping with ranked searches, searching of text can be greatly enhanced if the search tool understands some of the structure of language, and therefore queries can be submitted using natural language; a further project in artificial intelligence could investigate how to add this feature to this system.

Another topic that can suggest further work is the format of the information stored in the library. This implementation only deals with documents in HTML and text, since the whole document is indexed and hence textual information is needed. As a consequence of this, if a researcher wants to add a document in another format, this document must be converted to an appropriated format before being added to the collection. If the system is required to deal with another format of information, a plugin has to be written to extract textual information from it, as discussed earlier in the document. More advanced features can be explored, such as speech recognition, to enable the user to search within audio tracks, or even image

recognition (automatic extraction of features from pictures), although this would be a very ambitious project.

Finally, if the system is to be used in other environments that the one it was intended for, the user profiling and recommendation mechanism designed in this application might not be the optimal one, depending on different factors, such as the number of users, the organisation of them, the number of documents that the library will store, among others. In this case, other solutions should be explored in order to improve the system.

9. References

[Apache API] Apache API
http://www.apache.org/docs/misc/API.html

[ASP] Microsoft Active Server Pages
http://msdn.microsoft.com/workshop/server/asp/aspover.asp

[Borgman] Borgman, Christine L. (1999). What are digital libraries, who is building them, and why? In T. Aparac (Ed.), Digital libraries: Interdisciplinary concepts, challenges and opportunities (pp. 29). Zagreb: Benja.

[British Library] British Library, Digital Library Programme Team. (1999, June 7). The British Library Digital Library Programme: Towards the digital library.
http://www.bl.uk/services/ric/diglib/digilib.html

[Bush V.] Vannevar Bush. “As we may think”, The Atlantic Monthly, 176 pp. 101-108 (July 1945).
http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm

[Collier], Collier Mel. (1997) Towards a General Theory of the Digital Library. In *Proceedings of the International Symposium on Research, Development and Practice in Digital Librarie*s.
http://www.DL.ulis.ac.jp/ISDL97/proceedings/collier.html.

[ConnectionPool] Database Connection Pooling
http://webdevelopersjournal.com/columns/connection_pool.html

[Crawford] W. Crawford, PaperPersists: Why physical library collections still matter, Online, Jan. 1998.
http://www.onlineinc.com/onlinemag/OL1998/crawford1.html

[CSS] Cascading Stylle Sheets. W3C.
http://www.w3.org/Style/CSS/

[DLFed] Digital Library Federation. A working definition of the Digital Library
http://www.clir.org/diglib/dldefinition.htm

[Drabenstott] Drabenstott, Karen M. (1994) *Analytical review of the library of the future*, Washington,
DC: Council Library Resources.

[Duguid] Duguid, Paul, & Atkins, Daniel E. (Eds.) (1997, September 20). Report of the Santa Fe planning workshop on distributed knowledge work environments: Digital libraries.
http://www.si.umich.edu/SantaFe/

[Fab] Balabanovic, M., and Shoham, Y. 1997. Fab: Contentbased, collaborative recommendation. Communications of the Association for Computing Machinery 40(3):66--72.

[FastCGI] FastCGI Project
http://www.fastcgi.com

[Ginsparg] Ginsparg, P. (1994). First steps towards electronic research communication, Computers in Physics 8(4), 390-40

[GLIMPSE] Manber, U., and S. Wu. 1994. GLIMPSE: A tool to search through the entire file systems. In Proc. Of the USENIX Winter 1994 Technical Conference, pages 23-32, San Francisco CA.
http://www.webglimpse.org/

[Greenstone] http://www.nzdl.org

[Ian] Witten I.H., Nevill-Manning C.G. and Cunningham S.J. (1996) ProcWebnet '96, San Francisco, October (invited paper); also available as Working Paper96/14, Department of Computer Science, University of Waikato.
http://www.cs.waikato.ac.nz/~nzdl/publications/1996/IHW-SJC-MDA-Project96.pdf

[ISAPI] Internet Information Server API
http://www.microsoft.com/Mind/0197/ISAPI.htm

[Java 2 API] Java ™ 2 Platform Std. Ed. v1.3

[JSPSpec] JavaServer Pages ™ Specification, Version 1.1

[Nurnberg, et al] Nürnberg, Peter J., Furuta, Richard, Leggett, John J., Marshall, Catherine C., & Shipman, Frank M., III. (1995, June). Digital libraries: Issues and architectures. In Digital Libraries '95: The second annual conference on the theory and practice of digital libraries, June 11-13, 1995 - Austin, Texas, USA.
http://csdl.tamu.edu/DL95/papers/nuernberg/nuernberg.html

[MARC] MARC Documentation Overview
Network Development and MARC Standards Office
Library of Congress
http://lcweb.loc.gov/marc/status.html

[MG] Ian H. Witten, Alistair Moffat, and Timothy C. Bell,
Managing Gigabytes: Compressing and Indexing Documents and Images
Second edition, Morgan Kaufmann Publishing, 1999

[NSAPI] Netscape API
http://developer.netscape.com/docs/manuals/enterprise/nsapi/contents.htm

[RFC1867] Network Working Group.
Request for comments 1867
Form-based File Upload in HTML
E. Nebel, L. Masinter
Xerox Corporation, November 1995
http://www.cis.ohaio-state.edu/htbin/rfc/rfc1867.html

[Rothenberg]. Jeff Rothenberg. Avoiding technological quicksand: Finding a viable technical foundation for digital preservation. Technical report, Council on Library and Information
Resources(CLIR), Washington DC, 1999.

[Saffady] Saffady, William (1995). Digital library concepts and technologies for the management of library collections: An analysis of methods and costs. Library Technology Reports, 31223-224.

[ServletAPI] Java ™ Servlet Specification, v2.2 Final Release

[Waters] Waters, Donald, and John Garret. 1996. Preserving Digital Information.
Report of the Task Force on Archiving of Digital Information. Commissioned by the Commission on Preservation and Access and the Research Libraries Group, Inc. (May 1).

[William] Arms, William Y. Digital Libraries. Cambridge, Mass: London: MIT, 2000

Appendix A

A.1 Web application descriptor (web.xml)

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
    <display-name>Eircom's RD Digital Libray</display-name>
    <description> Web application usr servlets/jsp to implement a
                  Digital Library for Eircom's RD Dep as a Msc.
                  in Networks and Distributed systems project.
    </description>
    <context-param>
        <param-name>librarian</param-name>
        <param-value>sanchezo@tcd.ie</param-value>
        <description>The EMAIL address of the administrator
                     to whom questions and comments about
                     this application should be addressed
        </description>
    </context-param>
    <context-param>
         <param-name>mailserver</param-name>
         <param-value>mail.tcd.ie</param-value>
         <description>Mail server through which email notifications
                       are sent to the users
         </description>
    </context-param>
    <context-param>
             <param-name>emailbody</param-name>
             <param-value>This email is to let you know that a new
                          document that might be of your interest has
                          been uploaded to the digital library
             </param-value>
             <description>Body of the notification email
             </description>
    </context-param>
    <session-config>
       <session-timeout>30</session-timeout> <!-- 30 minutes -->
    </session-config>
    <servlet>
        <servlet-name> CheckP </servlet-name>
        <servlet-class>CheckP</servlet-class>
      <!-- It has to be loaded on startup of the application
           so that ChangeProfile.jsp can use it-->
       <load-on-startup>-1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name> CheckP </servlet-name>
        <url-pattern> /CheckP </url-pattern>
    </servlet-mapping>

</web-app>
```

A.2 DHTML code for login page

```
//-JavaScript File included by the JSP----------------------

function init(){
  if (ns4) {
      pass = document.layers["passDiv"]
      pass.left = 770
```

```
        pass.top = 390
        pass.xpos = parseInt(pass.left)
            pass.ypos = parseInt(pass.top)

      }
      else if (ie4){
        pass = passDiv.style
        pass.offsetX = 770
        pass.offsetY = 390
        pass.xpos = parseInt(pass.offsetX)
            pass.ypos = parseInt(pass.offsetY)
      }
   }


//- The body of the JSP----------------------------------------------------------------------------------------

<body bgcolor="#ffffff" text="#000000" link="#006666" alink="#cc9900" vlink="#666633"
marginwidth="0" marginheight="0" leftmargin="0" topmargin="0" onload="init();
movePassByToAt(-20,-10,350,150,2);">

 <DIV ID="passDiv">
  <TABLE CELLSPACING=0 CELLPADDING=3 BORDER=0 width="80%">
   <TR>
     <TD BGCOLOR=#305073>Enter the library</TD>
   </TR>
   <TR>
      <TD BGCOLOR="#305073">
        <TABLE CELLSPACING=0 CELLPADDING=3 BORDER=0 WIDTH="100%">
            <TR>
              <TD BGCOLOR="#B8C8E0" >
                    You must enter your <BR>
                    username and password
                  <FORM NAME="FormP" ACTION="<%=strServChec%>" METHOD="POST">

                     <TABLE>
                        <TR> <TD ALIGN ="RIGHT">Username: </TD><TD> <INPUT TYPE="TEXT"
                                NAME="Username" VALUE="user1" SIZE=8 MAXLENGTH=8>     </TD>
                         </TR>
                         <TR> <TD ALIGN ="RIGHT">Password: </TD><TD> <INPUT
                                TYPE="PASSWORD" NAME="Password" VALUE="passwd1" SIZE=8 >
                             </TD>
                        </TR>
                       <TR> <TD COLSPAN = "2" ALIGN="CENTER"> <INPUT TYPE="SUBMIT"
                                VALUE = "Log in"> </TD>
                         </TR>
                     </TABLE>
                  </FORM>
            </TD>
           </TR>
       </TABLE>
    </TD>
   </TR>
</TABLE>
 </DIV>
```

A.3 Greenstone home page configuration file

```
package home

##############################################################################
# java images/scripts
##############################################################################
# the _javalinks_ macros are the flashy image links at the top right of
# the page. this is overridden here as we don't want 'home'
# links on this page

_javalinks_ {}
_javalinks_ [v=1] {}

##############################################################################
# icons
##############################################################################

_icongbull_ {<img src="_httpicongbull_">}
_iconpdf_ {<img src="_httpiconpdf_">}
_iconselectcollection_ {<img src="_httpiconselcolgr_" width=_widthselcolgr_
height=_heightselcolgr_ alt=_altselcolgr_>}

##############################################################################
# http macros
# These contain the url without any quotes
##############################################################################

_httpicongbull_ {_httpimg_/gbull.gif}
_httpiconpdf_ {_httpimg_/pdf.gif}
_httpicontmusic_ {_httpimg_/meldexsm.gif}

##############################################################################
# page content
##############################################################################

_pagetitle_ {_textpagetitle_}
_imagethispage_ {}
_imagecollection_ { }

_content_ {
<center>
<p><table border=0 cellspacing=0 width="100%" bgcolor="#FF6600">
<tr>
  <td>
     <A CLASS="mylinkdark" href="javascript://"> Available collections</A>
  </td>
</tr>
</table>
<p>_homeextra_
</center>
<center>
<p>_iconblankbar_
</center>
<p><center><h2><A CLASS="mylinkdark" href="javascript://">
_textprojhead_</A></h2></center>
}
```