REIMPLEMENTATION OF THE COMPUTER AND INFORMATION SCIENCE AND ENGINEERING (CISE) DEPARTMENT WEBSITE USING A CONTENT MANAGEMENT SYSTEM

By

RIMA GERHARD

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2003

This document is dedicated to my family.

ACKNOWLEDGMENTS

I would like to thank Dr. Markus Schneider for being the chair of my committee and helping me with this thesis despite his busy schedule. I would also like to thank Dr. Joachim Hammer and Dr. Douglas Dankel for agreeing to serve on my committee. I would also like to thank Mr. John Bowers for his continuous assistance throughout my years in graduate school. Finally, I would like to thank my family and friends for their support.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

REIMPLEMENTATION OF THE COMPUTER AND INFORMATION SCIENCE AND ENGINEERING DEPARTMENT WEBSITE USING A CONTENT MANAGEMENT SYSTEM

By

Rima Gerhard

August 2003

Chair: Markus Schneider
Major Department: Computer and Information Science and Engineering

The maintenance of a website is a complex and time-consuming task. Many documents in different formats are published online by many different groups of people. The CISE Department hosts a website of medium size. This website has received a redesign in the summer of 2003. New content has been added and the integrity of the site has improved (i.e., broken links have been updated and the navigational strucure has been reimplemented). Still, the website is mainly static, and with every update to the website profound knowledge of HTML is needed. This reduces the number of people that can work on the website to a very small group of web administrators. This situation quickly turns into a bottleneck. The website is difficult to manage in terms of ensuring consistency and supporting evolution.

New tools for dynamic websites have emerged in the past year. There is, on the one hand, the group of server-side languages that allow for the creation of dynamic websites. On the other hand, there is the group of web content management systems (CMS). The

main objective of such systems is to move the developer away from the level of HTML documents and to assist in the general maintenance problem of ensuring consistency of content, reference, and presentation. CM systems are especially useful when websites are frequently updated and several authors exist, as in the case of the CISE Department's site. Another feature of content management systems is that they allow for a separation of content, style, and logic.

The goal of this thesis is to reorganize the Department's website using a web content management system and a database. All content is stored centrally in a database, which means that data are not stored in a redundant fashion any more. The layout and logic are being handled by the content management system. The resulting website0 conforms to the policies and guidelines stated by the University of Florida's Web Administration group as well as the College of Engineering. The resulting system is easy to maintain even by people who lack fundamental HTML or programming logic. With the new system, it is easy to distribute and assign work among many users.

CHAPTER 1
INTRODUCTION

The Internet steadily grows in size. This growth can be attributed to an ever-increasing amount of documents available on web servers. Due to the rising number of online documents, the management of websites becomes a difficult and time-consuming task. The Computer and Information Science and Engineering Department hosts a website and is currently experiencing the complexities involved in maintaining a current website of medium size. In the fall of 2002, when this project was initiated, an older version of the Computer and Information Science and Engineering Department website was in use. Many times website users were frustrated because they were not able to locate the information they are looking for. In the summer of 2003, a new website went online, which features a new design and some new capabilities. However, there is still no support for the frequent updates and developers need profound knowledge of HTML to publish content. Therefore, the number of possible authors is greatly restricted and most work falls back on the site's main administrators, a situation that quickly turns into a bottleneck. The website is difficult to manage in terms of ensuring consistency and supporting evolution.

The main problem is that maintenance for a site of relatively medium size remains very complex and time-consuming. To support maintenance, a large variety of new products and tools for website engineering have been developed. These so-called web content management systems (CMS) range from full-scale enterprise platforms, large upper tier and mid-market packages to various forms of open-source tools. The main

objective of such systems is to move the developer away from the level of HTML documents and to assist in the general maintenance problem of ensuring consistency of content, reference, and presentation. CM systems are especially useful when websites are frequently updated and several authors exist, as in the case of the Computer and Information Science and Engineering Department's site. In the course of this project, several CMS have reviewed. One of them fulfilled most requirements and was chosen for the implementation of the new website.

The goal of this thesis is thus to reorganize the Department's website, according to the following constraints: first of all, content shall be separate from presentation. Second, the website should be conform to the policies and guidelines stated by the University of Florida's Web Administration group. Finally, website data shall be managed by a database and editing of content as well as creation of new documents shall be simplified.

We begin in Chapter 2, with a description of related work. First of all, we will review general design principles and specific guidelines imposed by the University of Florida. Second, we review the major website implementation technologies available today and make a selection of the technologies deemed appropriate for the implementation of the new website. Third, we will list a number of existing open source content management systems. Chapter 2 concludes with which system was chosen along with the reasons for doing so. In Chapter 3, we will present an overview of Zope, the CMS chosen for this thesis. Chapter 4 identifies the requirements that have been fixed for the website. In Chapter 5, a description of the implementation follows. Chapter 6 will give an overview of the user interface of the final website. Chapter 7 states the conclusion

for this project as well as what possible additional work could be undertaken in the future. The last chapter lists all references used throughout this thesis

CHAPTER 2
RELATED WORK

In this Chapter, we will give an overview of the topics that are relevant for this project.

The first area that is important for websites is the area of website design. Graphical User Interface (GUI) design methods are quite different from Web User Interface (WUI) design methods. Section 2.1.1 and 2.1.2 are dedicated to this area, detailing the guidelines for website design as imposed by the University and general design principles. Most work relevant for designing and evaluating user interfaces has come out of the Human-Computer Interaction (HCI) community.

There is another meaning to the word "design." There is the graphical and user interface aspect of website design as treated in subsection 2.1.1 and 2.1.2. But there is also the design of the structure of the site. This area is closely related to the field of database design. Any database designer knows that a database needs a well-defined structure; otherwise, many maintenance problems will occur. Websites are certainly comparable to databases – both provide (sometimes large) amounts of information that needs maintenance. Databases have design methods such as the well-known ER-diagram [CHE76]. Therefore, websites need design methods similar to the design methods developed for databases. We will present some related work in the area of website design methods, which are lately also referred to as "Hypermedia Development Methods", in subsection 2.1.3.

Section 2.1.4 gives an overview of the website implementation technologies currently in use. In addition to giving a brief explanation of each technology, the choice is made whether the technology should be adopted for this project or not.

The next area of related work is CM Systems. They were developed to support website builders in the task of managing the content, style and structure of a site. We have selected 5 systems, and describe their features in subsection 2.2. At the end of the section, a choice is made for one of the CMS.

2.1 Web User Interface Design and Implementation

Web User Interface Design refers to the area of the actual visual presentation of the website. This includes the graphical user interface. It defines the first impression a user gets after visiting a page, therefore, we will list related work of this area before we treat all other aspects of a website.

The most important goal of WUI design is usability. Nielsen [NIE00] defines usability as follows: “Usability is the measure of the quality of the user experience when interacting with something – whether a website, a traditional software application, or any other device the user can operate in some way or another.” In the context of the World Wide Web, this means, “how easy it is to find, understand and use the information displayed on a Website” [KEE98].

To give some concrete examples, the following problems could be found on a website with poor usability: first of all, websites might suffer from information overload. Typically, a user does not want to browse through a lot of pages to find the piece of information that he is looking for. The attention span of a surfer is likely to be short. Second of all, ill-structured websites might lead the user to experience the lost-in-hyperspace syndrome. This syndrome appears when the navigation process is not very

well structured and guided. It therefore becomes hard and time-consuming for the user to locate the desired information. The user is then said to be "lost" within the hierarchy of hypertext.

The usability, appearance, accessibility and uniformity of a web pages projects the image of the CISE Department to students, alumni, visitors, and prospective employees. If a website has poor usability, it will discourage exploration and waste user time and leave a negative image of the CISE Department with the user. Therefore, it is important to adhere to methodologies for designing usable websites. There is a number of researchers who have treated the subject of graphical and user interface aspects of website design. Nielsen [NIE00] has suggested methods for designing pages with good usability.

Another alternative for most page designers is to provide them with a set of simple guidelines for designing pages. These guidelines should be based on fundamental principles of user interface design.

2.1.1 General User Interface Design Guidelines

The WWW hosts a wealth of guidelines for designing web pages. Conference articles that address the issue are few, however. Borges proposes simple design guidelines [BOR96][BOR98], such as

- Verifying that links connect to existing pages,
- Including the date the page was last modified, the mail address of the person that maintains the page and the URL address of the page on a footer,
- Avoiding pages become overcrowded with links,
- Keeping pages short (about a letter size page),
- Providing clear navigation from anywhere to anywhere, and do it on every page,

to name but a few. Most guidelines have a basic idea in common: the key to successful Website design is not sophistication, it is simplicity.

2.1.2 Specific Guidelines imposed by UF and the COE

The University of Florida Web Administration group has imposed a set of guidelines. Their guidelines can be broken up in two sets of policies. First, they propose the "Acceptable Use of Computing Resources Policy" which consists of general rules for web pages, e-mail, and University affiliation. The key idea here is that the following information must be readily accessible on the main page:

- The name of the unit or group represented by the page;
- A means of contacting the person(s) responsible for maintaining the page content;
- The date of last revision;
- The university word mark; and
- An active link to the UF home page.

Second, they have formulated the Accessibility Recommendations for UF Websites - under the ADA (Americans with Disabilities Act), the University is required to accommodate individuals with disabilities. A similar, more expansive set of recommendations has been put forward by the W3 consortium [CHR99]. These documents encompass areas such as recommendations for colors (ensure that all information conveyed with color is also available without color, for example from context or markup), graphics (provide a text equivalent for every non-text element because Screen readers for the visually impaired will skip over images, image maps, buttons, bullets, etc. "Alt" tags give a written description of the image.) and language (use the clearest and simplest language appropriate for a site's content because screen readers will translate simple language more accurately than complex text).

The Office of Public Relations of the University of Florida has put together a set of graphic policies for the University of Florida. To achieve a look, a consistent impression, they have defined rules concerning the University Word mark. A word mark, words used as a logo, provides a strong, distinctive graphic symbol unique to the University of

Florida. The word mark consists of the seal (based on the state seal with the university motto and founding date), a word mark using the university's name and a positioned rule. The university mark should be used as a signature in visual presentations about the university, including websites. Furthermore, the University has established rules concerning colors and typeface. The official university colors are orange and blue. The exact shades for use on the web are Orange Hex={FF,4A,0} and Blue Hex={00,21,A5}. The university's official typeface is Palatino (called Palatino, Book Antiqua, or other names by various companies).

Finally, the College of Engineering (COE has established a "Web Page Guidance Document" [COL03] which outlines the common thematic and structural elements that will brand the web pages within the COE at the University of Florida. Websites subject to the requirements include those maintained by the COE and the Departments within the COE. The documents stated that departmental home pages should have a uniform layout comprised of the following three elements: a header, a left-side menu box, and footer.

Other guidelines in this document include the following areas:

Layout. Web pages shall be standardized upon a minimum 780-pixel width. The height of the page is variable to allow greater flexibility for the departments' free space. The intent is to have an effective presence over the broad range of browser and screen combinations present worldwide.

Color Schemes. Each department can choose the color scheme for its website. Light lettering on dark backgrounds will be used for headers and footers. Menu boxes on home pages will be encapsulated in the same dark background color. The menus will be dark lettering on a light background. Above this menu box, but within the encapsulating

"frame," a search box, contacts, sitemap, or other similarly appropriate links may be included at the department's discretion.

Headers. Headers will be designed and used as the primary branding elements to instill a sense of community within the cyber-presence of the COE. Headers on home pages will consist of three components:

1. The upper strip (height 35 pix) in which the COE and UF wordmarks are placed in left- and right justified positions, respectively.
    a. The UF wordmark should be 20 pixels high and 80 pixels wide. Graphics in the desired color scheme can be requested from UF Web Administration.
    b. The gator engineering wordmark should be 25 pixels high and 120 pixels wide
2. A larger middle strip (height 45 pix) in which the text "Department of" (Arial 10 bold) above the department name (Arial 30 bold max, smaller to incorporate longer department titles) appears both left and middle justified.
3. A navigation bar (height 20 pix) below that having a prescribed font (Arial 16), but department defined functions

Department names shall appear as "College of Engineering" for the COE and "Department of …" for the departments. No abbreviations shall be used.

The upper strip for the home page header shall be the base for the subordinate page header. The name of the department will be superposed using a prescribed font (Arial 20) in a center-justified position. For subordinate pages "Department of" should be omitted so that longer named programs may be accommodated in a consistent manner. Each department shall maintain an option of including a strip for logos or section headers below this upper strip. Finally, an optional navigation bar of the same constraint as above will be immediately below that. These navigation bars may be section or even page dependent, according to the needs of the department.

Left Justified Menu Box. Departmental home pages are required to have a left-side menu box (width 180 pix) that contains three sections: Quick Links, Information

For, and Programs. The required links cannot be hidden in a drop-down or expandable menu, but if needed each required link can be extended with a drop-down or expandable menu. These sections are to be uniformly labeled for the departments using a prescribed font (Arial Bold 3, 16). Links (Arial 2, 12) within these sections are to be uniformly labeled, if used. These sections are outlined below. The ordering of items within them should be precisely followed. Optional items that a department wishes to add should follow the prescribed list.

Quick Links
ABC Directory (where ABC = departmental initials)
College of Engineering
Graduate School
University of Florida
Career Opportunities
Other optional links

Information For
Prospective Students
Undergraduate Students
Graduate Students
Alumni
Visitors
Faculty/Staff (optional)
Corporate Partners (optional)
Other optional links

Programs
Research
International
Centers (optional)
Laboratories (optional)
Other optional links

The COE also provides 2 HTML templates for use by other COE Departments [COL03]. The template currently used by the CISE Department as well as in this project is based on the COE templates.

2.1.3 Software Engineering Methods for Websites

As mentioned in the introduction at the beginning of this chapter, databases have several similarities to websites. Both store information for users. Both need a structure to facilitate working with them. Both can suffer under maintenance problems such as redundancy, incompleteness, and inconsistency, if their structure was not designed properly. In order to avoid these problems, researchers have developed several hypermedia development methods: HDM [GAR93b], its successor HDM2 [GAR93a], OOHDM [SCH98], RMM [ISA95], W3DT [BN96], SOHDM [LEE98], and WSDM [DET98]. Koch has presented a comparative study of the most relevant methodologies for hypermedia and Web development [KOC01].

Currently, websites are often developed ad-hoc. The original CISE website seemed to be no exception. The development style can be blamed for many of the inconsistencies and poor usability that the user experiences when surfing the site. The methods listed above assist the website designer in planning a site. They provide appropriate techniques, processes, and methodologies. They fall into the area of software engineering methods and can be compared to the methods created for designing databases such as the ER-method [CHE76].

When websites are developed ad-hoc, the target audience of the site often experiences one or more of the following problems: first of all, information is needlessly repeated in several areas. On the previous CISE site, there was a textual description of the undergraduate classes offered in CISE under the area “Undergraduate Program” as well as in the link to the “CISE Graduate Brochure.” Not only is the information redundant, it is also inconsistent to store the information in an area where on expects to find material relevant for graduate students.

Second of all, websites tend to be incomplete when no design method was employed. The CISE website has some incomplete areas as well. To give only one example of this, consider an international student who seeks to pursue a degree in the CISE Department. When searching the previous CISE website, the student will not be able to find any information on what degrees from foreign countries will be accepted by UF. One might argue that such information does not have to be offered by the CISE website. However, that information might be provided elsewhere, on another site affiliated with UF. Even if the CISE Department does not host the information, it should at least link to it, to provide as much service to its target audience as possible. International students in general do not find very much information geared towards them. This is surprising since the population of graduate students in CISE is currently made up of more than 80% international students!

Another problem arises from the incompleteness issue: the problem of broken links. A broken link leads to a user not finding the needed information. This will decrease the confidence of users and leave a negative over-all impression. Many broken links can be found on the current site, for example under the Student Organizations area. Three out of the seven student organizations listed have either dissolved or else changed their URL.

We have not implemented a specific software engineering method in this project. This is due to the use of a database instead. The content management system used also solves the majority of the problems described above (broken links, redundancy, incompleteness).

2.1.4 Website Implementation Technology

In this Section, we present an overview of the technology available for website creation to date. Before however giving more detail of the implementation languages and

technologies available to date, we will give some background on the mechanics of website publishing.

2.1.4.1 Web publishing

Most of today's web publishing is happening through the Hyper Text Transfer Protocol (HTTP), or its variant, the secure HTTPS, which combines HTTP and the Transport Layer Security (TLS) or the Secure Socket Layer (SSL). The inventor of HTTP is Tim Berners-Lee. He implemented the protocol in 1990 at CERN, the European Center for High-Energy Physics in Switzerland.

HTTP's functionality is quite simple. A client is sending a request to a server. The server then processes that request and answers with a response. Both the request and response are sent as messages over a TCP connection. The Multi purpose Internet Mail Extension (MIME) format is used for these messages. A MIME message is made up of a body and a set of headers. For HTTP messages, the body is optional, and the message has been termed as the HTTP entity. Additionally, a response (or request) line is prepended to the message. The request line has information about the request method, the resource locator, and the protocol version. The resource locator identifies resources in a hierarchical structure (such as the file system). The resource itself could be anything from an HTML page, to an image, a file, or an application. The HTTP resource locator makes use of the URL syntax. The server must determine what resource is actually requested by the resource locator.

The most popular request methods employed by HTTP are GET, PUT, POST, and HEAD. The GET method is used by a web browser to request a specific document. A web browser's job is to formulate the actual GET request, send it to the server, receive an HTML document in return, and finally display the HTML document according to the

HTML instructions. We discuss HTML in further detail later in this chapter. The POST message is used by web browsers to send data to a web server. The web browser can attach the data to an HTTP request as the message body. The message body would normally be the URL String after the question mark (?), e.g. http://www.cise.ufl.edu/inde.html?name=Markus+Schneider&phone=1234567. As with the GET method, the browser is responsible for preparing the POST headers and request body. The PUT request sends information to the server that will be used to create a resource as specified by the resource locator. If the resource already exists at that location, it can be overwritten. The HEAD method is used to get information about a document. Since a much smaller of data is transferred (not the whole document itself), HEAD is a lot faster than GET. The HEAD method is often used by clients who use caching to determine whether a document has changed since the last time it was accessed by the client. A local, cached copy can be used again if it has not changed. If it has, the document needs to be retrieved again using the GET method.

A final important property of HTTP is that it is a stateless protocol. This implies that a request must have enough information in it to process it. The web server does not have any saved state information from previous requests that might be useful when processing the current request. This feature of HTTP makes it quite complex to build web applications. User might expect from an application that it remembers their preferences and some facts from previous interactions. Many workarounds have been devised to cope with this limitation: cookies, session products, authentication headers, and hidden form variables. Future HTTP versions could quite possibly remove this limitation.

2.1.4.2 Web technologies

Before listing the web technologies available today, it is important to note the difference between client-side and server side technologies. Figure 2.1.4.2 shows how client-side and server-side languages differ when a website is called.

The client-side is basically represented by the browser, the front-end of any website. The browser displays HTML sites in frames. Technologies that are termed "client-side" are downloaded along with the HTML code and are employed on the computer of the user (the "client").

The server-side languages deal with resources of the server. Connections to databases are established, information between several users of an application is shared, or the file system of the server is being manipulated.

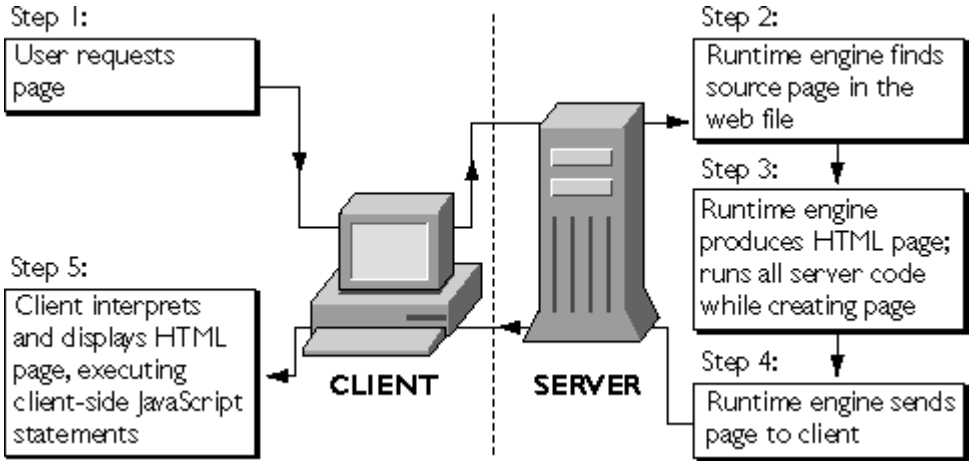


Figure 2-1 Client/Server-side technologies

HTML/XHTML. The layout of text and images that show up in a user's web browser when the user visits a website is usually composed using a simple language known as Hyper Text Markup Language (HTML). When a user visits a typical website, a chunk of text that is "marked-up" with HTML is transferred between the web server and

the user's browser. The browser interprets the chunk of text, showing text and images to the user. The chunk of text, which is transferred, is typically referred to as a *page*.

With HTML, website developers can create headers, paragraphs of text, lists, and tables. They can create links, which refer a user to another website or other data in the web. HTML can be extended with CSS Style Sheets and JavaScript, which is discussed in the next Sections.

HTML is actually a simpler and easier-to-use subset of the Standard Generalized Markup Language (SGML). The W3 Consortium [WOR03] is responsible for the standardization of HTML. No specific editor or software is needed to create HTML files. However, there are powerful programs for HTML editing. The CISE Department has purchased a license for Microsoft FrontPage [MF4], whose main feature is that pages can be created in Normal (WYSIWYG) View, or code can be written in HTML View. FrontPage users can, therefore, create websites without knowing any HTML. Also, FrontPage has HTML, ASP, and XML source-code preservation capabilities in that allow a user to import code created in a text editor or other HTML editing tool—without rearranging or changing it.

XHTML is a new HTML derivate. In version 1.0, XHTML is a redefinition of HTML with the addition of XML.

One of the most important features of HTML is the ability of defining links (short for “hyperlinks”). Links can refer to other pages within a site, or any address in the World Wide Web. This simple feature turns out to be quite powerful, since machines, that are physically miles apart, can now be reached through modern web browsers with nothing but a mouse click. The World Wide Web basically relies on this feature.

One kind of web design that is possible with HTML is websites that are built based on “frames.” There are many issues that must be considered before frames are implemented on a site. One problem arises with search engines. Search engines can link directly to framed content documents, but they cannot link to the combinations of frames for which those content documents were designed. Many framed content documents are difficult to use when accessed directly (outside their intended frameset), so there is little benefit if search engines offer links to them. Therefore, many search engines ignore frames completely and go about indexing more useful (non-framed) documents.

Another issue is bookmarks. In current browsers, if a user bookmarks a page, the browser actually only bookmarks the parent frameset. When that user later calls up that bookmark, she will get the home page or equivalent. This can quickly become frustrating when a user realizes that she has to retrace their path through the site when the user calls up the bookmark.

To conclude, the Accessibility Guidelines suggests to avoid using frames altogether. The reason for this is that there are still browsers out there that do not use frames, browsers that are brand new and make a conscious decision to not offer that traditional support. Some of these are browsers for the blind, or other handicapped users.

CSS style sheets. Style sheets [BOS98] can be used to manage, update, and change the layout of large sites easily [BOS98]. With traditional HTML, the presentational information is scattered throughout the document. In CSS, the presentational information is collected in one place, and it is easy to read and edit. CSS collect the style information for an entire website into one file, which is in turn included in the header of all HTML

files that the style should be applied to. The editing of that file affects the presentation of the entire site.

The Extensible Style Language [CLA99] is supposed to replace CSS. XSL defines stylesheets, which in turn define how XML documents are being displayed in a web browser. Through transformations, an XML file can be turned into another XML file, i.e. by leaving out or adding on certain elements, or by changing the order of elements.

Disadvantages of style sheets are that support is often limited, buggy, or sometimes completely nonexistent within certain versions of popular Web browsers. However, browser support is getting better with newer versions of web browsers.

The University of Florida Web Administration also has their take on CSS. They simply recommend organizing documents so that they may be read without style sheets. For example, when an HTML document is rendered without associated style sheets, it must still be possible to read the document.

JavaScript. JavaScript is a general-purpose programming language that, when used for Web documents, goes directly inside the HTML documents and is downloaded to the browser with the rest of the HTML tags and content. Scripts can make a page interactive. Traditional web pages tend to be lifeless and flat unless one adds animated images or more bandwidth-intensive content such as Java applets or other content requiring plug-ins to operate (ShockWave and Flash, for example).

A popular example of the use of JavaScript found on the web is image rollovers: roll the cursor atop a graphic image and its appearance changes to a highlighted version. Other things JavaScript can be used for includes:

- Automatically change a formatted date on a Web page
- Cause a linked-to page to appear in a popup window

Furthermore, interactive forms validation is an extremely useful application of JavaScript. While a user is entering data into form fields, scripts can examine the validity of the data, i.e., “Did the user type any letters into a phone number field?.” JavaScript could be included on a web form to perform interactive error checking. This is useful for example for forms that transmit data to a database. Before inserting the actual data, one could check with JavaScript whether all fields of a form were filled out and whether they were filled with the right data type.

XML. XML stands for Extendible Markup Language. It is a flexible way to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere. Any individual or group of individuals or companies that wants to share information in a consistent way can use XML. It is a formal recommendation from the World Wide Web Consortium [WOR03] and is similar to the Hypertext Markup Language. Both XML and HTML contain markup symbols to describe the contents of a page or file. HTML, as described above, describes the content of a Web page (mainly text and graphic images) only in terms of how it is to be displayed and interacted with. For example, the letter "p" placed within markup tags starts a new paragraph. XML describes the content in terms of what data is being described.

XML-based files contain nothing but logic markup (also called semantic markup). A markup such as <description>…</description> only contains information about the meaning of the data inside of the tag. It does not contain any information about how such data should be displayed. The data is entirely independent of the output medium (such as monitor, printer, …) and does not include any formatting instructions. As opposed to HTML data, which is displayed by a browser using default values, a browser does not

know how to display XML data per se. Therefore, before such data can be presented, one needs to specify the formatting with a using a style language.

Two formatting languages are available for this purpose as of today: CSS and XSL. CSS, on the one hand, was already discussed above. It can be used to tell a web browser how to display elements of an XML file. XSL, on the other hand, is more powerful. Especially important is the transformation component XSLT, which transforms XML data into HTML on the server side. The advantage of this is that XML in combination with XSL can be used on old browser versions, which do not recognize pure XML. The disadvantage is that this only works within an HTTP environment. Furthermore, the web server needs to have a corresponding interface, that allows for the integration of an XSL/XSLT software module.

XML is "extensible" because, unlike HTML, the markup symbols are unlimited and self-defining. XML, as HTML, is also a subset of the Standard Generalized Markup Language. XML markup, for example, may appear within an HTML page.

The major problem with XML is the capability of software to represent XML-based data adequately. Not all web browsers have the same capability of interpreting XML. The MS Internet Explorer interprets XML-based markup languages since product version 5.0, the Netscape browser since version 6.0. “Interprets” means that the browser is able to recognize XML data as such and can parse it. This does not mean, however, that the XML data is also displayed nicely – formatting is done with CSS or XSL or translation with XSLT to HTML.

Since the goal of a web designer is always to make the web page viewable in most browsers, it is not recommended at this point to use XML-based languages for web

design. Clearly, when XML is used in combination with XSLT, this restriction does not apply.

CGI. The Common Gateway Interface (CGI) is not a programming language. In fact, CGI applications can be written in just about any programming language in use today. CGI programmers write scripts and programs for the web that are called from HTML files. Actually, HTML and CGI communication works both ways: a CGI script could also transfer HTML code to a web browser.

The difference to JavaScript is that those scripts or programs are not executed after the website has been transferred to the user's browser, but before the browser receives the data from the server. CGI scripts and CGI programs are so-called server-side scripts. A CGI script could, for example, query a database that is installed on the server machine and display the query results on the client browser with dynamically generated HTML. The CG Interface has to be supported by the web server software. Most of the time, a directory with the name cgi-bin is used for this purpose.

There are no requirements for the language in which a CGI program is written. Since the program has to be executable on the server, the program must have been compiled to the operating system environment of the server. Alternatively, the server could have a runtime interpreter to execute the program. If the server is running on Linux, for example, the server machine is executing C programs that were compiled by a Linux-C-compiler to an executable file.

Most of today's CGI programs, however, are not compiled programs, rather just scripts, that are executed by an Interpreter when called. The best-known and most popular Interpreter is the Perl Interpreter. The Perl-Interpreter is available for most

operating systems as freeware and is installed on almost all servers in the web. More information about Perl itself follows later in this Chapter.

Alternatives to a CGI applications are Microsoft's Active Server Page (ASP), JSP, and PHP in which a script embedded in a Web page is executed at the server before the page is sent. They are discussed in further detail in the next Section.

ASP. ASP stands for Active Server Pages. It is another alternative to PHP and CGI/Perl. Consequently, ASP is used to create server-side dynamic websites. As opposed to CGI/Perl, scripting code in ASP is embedded in HTML code (as is the case with PHP and JavaScript). ASP is tightly coupled with the world of Microsoft and Windows, even though ASP exists for Linux and other Unix-based operating systems and the Apache web server is supported along with Microsoft web servers. However, the ASP integration is optimized for MS web servers and is therefore mostly used under Windows NT.

ASP is not a scripting language, as opposed to PHP. It is considered an environment; the scripting languages that can be used within this environment are Jscript and VBScript (the standard).

The principle of ASP is as follows: due to certain conventions the web server recognizes that a HTML file is an ASP file. The server then implements the script code contained in the file and sends the entire chunk of generated HTML code back to the calling browser.

A simple example for ASP is given in the following two documents. The first one, “search.html,” calls a script when an HTML form is submitted. The second script, called “db_script.asp,” executes a database query and displays the result. VBScript was chosen for the implementation.

1. File “search.htm”

```
<html>
  <body>
  <form ACTION="db_script.asp" METHOD="post">
   <input TYPE="text" NAME="SomeName" LENGHT="10">
   <input TYPE="submit" VALUE="Suchen">
  </form>
  </body>
</html>
```

2.File “db_script.asp”

```
<%@ LANGUAGE = "VBScript" %>
<% SomeName = Request.Form("SomeName") %>
<html><head></head>
<body>
<%= "You are looking for: " & SomeName & "<p>" %>

<%
    Dim db_conn, db_query
    Set db_conn = Server.CreateObject("ADODB.Connection")
    db_conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data
                  Source=D:\Inetpub\wwwroot\cise.mdb"
    Set db_query = db_conn.Execute("SELECT * FROM Professor
                  WHERE Name LIKE '%" & SomeName & "%'")

   Do While Not db_query.eof
    Response.write db_query.Fields("firstName")
    Response.write db_query.Fields("lastName")
    Response.write db_query.Fields("email")
    Response.write "<br>"
    db_query.MoveNext
   Loop
   db_query.close
   db_conn.close
%>
</body>
</html>
```

The ASP environment is available under a license and needs to be purchased.

DHTML. Dynamic HTML (DHTML) is not a new language, it is rather an extension of HTML. It does not introduce any new HTML tags, however. It makes HTML “dynamic” by allowing website authors to alter elements of a website when it is displayed in a browser. Some content can change without, however, having to reload an entire page. A website therefore acts like an application that needs to be loaded only once. The interaction with the user decides what will happen on the screen next. To give a concrete example of DHTML, one could create a website where users can select one item from a drop down box. Depending on what item of the box was chosen, the website

would then display another drop down box or a text field. This is just a simple example – more complex examples of DHTML include animated graphics that glide over the screen, dynamic sitemaps, or clock counting up seconds in real time.

When programming DHTML, website creators need a profound knowledge of JavaScript to begin with. Some ready-made DHTML solutions are available on the web, but whenever a custom-made solution is needed, one needs to have a strong background in JavaScript programming.

Another problem with DHTML is that the two most common web browsers, MS Internet Explorer and Netscape, are only able to interpret it since version 4.0 (in both cases). Unfortunately, DHTML is one of the areas where these two browsers have decided to implement entirely different strategies. When writing DHTML for both browsers, one often has to write twice as much code – one version for each browser. Many features will only work with one of the browsers.

Finally, the major disadvantage of DHTML is that it could pose problems for people with visual impairments. The Accessibility Recommendations for UF Websites by the University of Florida Web Administration does not state any explicit recommendations as far as DHTML is concerned, but it is safe to say that overly dynamic sites with moving elements would not be deemed acceptable.

Perl. Perl stands for Practical Extraction and Report Language [PER03]. The language stems from the Unix operating system world and was first published in 1987. Perl was invented by its only creator, Larry Wall. Following the Unix philosophy, Perl is an open language. Since Version 5, Perl supports elements of object-oriented

programming. It is a scripting language that really started becoming popular in conjunction with the CGI environment.

Perl has reached a certain level of popularity for being the language of hackers. One can achieve complex instructions with just one line of code. The code itself seems almost unreadable to the uninitiated. The syntax is based on C. Files that contain Perl code aren't compiled to executable code. The Perl Interpreter is responsible for checking syntax and compiling Perl code whenever a file with Perl code is called. The obvious disadvantage is that due to this extra step, scripts don't execute as fast as compiled and linked scripts. The advantage is that scripts don't have to be compiled and linked for every new operating system environment. The scripts are therefore portable and can run anywhere as long as the Perl Interpreter is installed.

The disadvantage of CGI in combination with Perl is that Perl Scripts are separate files, that reside (due to the CGI architecture) in different directories than the HTML files. Another problem is that Perl was devised as a universal programming language and not specifically for dynamic website development.

Server side includes. Server Side Includes (SSI) are used to insert bits of dynamic information inside of HTML file. Popular examples are the inclusion of the current date and time. SSI can also start CGI programs and include their output (for example, the number of times a file has been accessed) in the HTML file.

SSI are only executed when a web browser calls the HTML file over a web server (e.g. with a URI of the type http://…). Another condition is that the web server supports SSI. Not all web server do so, some only interpret a part of the instructions. In case the web server doesn't know SSI, the instructions won't function.

To signal the web server that SSI are present in an HTML file, it has become standard to give them a special extension: .shtml, .shtm, or .sht. Most web servers will ignore SSI if they are written in a file ending with .html or .htm.

PHP. HTML by itself cannot be used to generate dynamically generated content. A scripting language is needed for this purpose. JavaScript and CGI/Perl have already been discussed above, along with their respective disadvantages. PHP was developed to remedy some of these problems. PHP originally stood for *Personal Home Page* Tools. PHP's core concept is that PHP code resides directly in HTML files, as is the case for JavaScript and PHP. When a web browser calls the HTML file, the web server recognizes that the requested file contains PHP code and is more than a static HTML file. The web server proceeds by having the file translated by the PHP Interpreter, which is installed on the server. The Interpreter executes the PHP code embedded in the HTML and generates the final HTML code, which is sent back to the browser.

Whatever can be done with CGI and Perl can be achieved with PHP. Some things are easier though, since the PHP Interpreter is better fitted to web publishing demands. It is, for example, possible to generate PDF files dynamically. PHP has enjoyed a wide success and is used by many websites. A simple example of PHP code in an HTML file is the following set of files. The first file ("search.htm") simply contains a form, which calls a script called "db_script.php" upon submission. The second file is the db_script.php. We reprise therefore the same example as the one given for ASP.

1.File "search.htm"

```
<html>
<body>
<form action=”db_script.php” METHOD="post">
<input TYPE="text" name=”SomeName” LENGHT="10">
<input TYPE="submit" VALUE="Suchen">
</form>
</body>
```

```
</html>
```

2.File “db_script.php”

```
<html>
<body>
<?php echo("You are searching for: $SomeName<p>\n");?>
<script language="php">
/*Search in the db */
  $db_conn=mysql_connect("db.server.net","markus","secret");
  mysql_selectdb("cise",$db_conn);
  $db_query=mysql_query("SELECT * FROM Professor
           WHERE name LIKE '%".$SomeName."%';",$db_conn);

  // output all results
  while ($db_res=mysql_fetch_array($db_query))
  {
   echo $db_res["firstName"]."\n";
   echo $db_res["lastName"]."\n";
   echo $db_res["email"]."\n<br>";
  }
 </script>
 </body>
</html>
```

PHP and its Interpreter are available for free. The software was developed by the PHP Group, a consortium of programmers [PHP03].

Java. Java is a platform-independent programming language developed by Sun Microsystems [SUN03]. The language is structurally and syntactically close to C and C++. Java was developed as a programming language for the Internet, but it turned out that Java is suitable for universal programming purposes. It did not reach the dominance in the Web that the Java developers were hoping for.

Today, Java is particularly used in connection with on-line banking, on-line broking, and web-based chats. As far as animations, effects, and on-line plays is concerned, Java is less popular than Flash (described in the next Section)

One kind of Java programs for the Internet is called applets. Java applets can only be executed when users have explicitly enabled their browser to support them, since they could possibly be used to change files on the user’s computer or even delete them.

As far as the Client-Server model is concerned, Java can be used on both sides. Applets running in a browser are client-side. Oftentimes, those applets communicate with server-side programs. Consider for example a chat application. On the one hand there is an applet, which runs in the browser representing the chat interface to users. On the other hand, since chats usually involve more than one person, naturally some program must be running on the server to administer the chat participants.

Another kind of Java program for the Internet is servlets. Servlets are Java classes that are typically used for tasks such as security checking, control over downloads and uploads. Java Server Pages (JSP) is yet another option similar to PHP, ASP, and CGI/Perl. The server page is an HTML file with embedded Java code. JSPs are a good choice for smaller web applications. Their disadvantage is that they become hard to maintain since code and HTML are intermingled. The example used for PHP and ASP can be rewritten for JSP as follows. Again, two files are needed, the first one (“search.jsp”) containing a from which calls the second file (“db_script.jsp”).

1.File “search.htm”

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function submitForm(input) {
   res = "db_script.jsp?";
   res = res + "inpName="+ input;
  //window.location.href is the current window
  window.location.href = res;
  }
}
</SCRIPT>
</HEAD>
<%@ page language = "java" import="java.sql.*" %>
<BODY>
<%@ include file = "default_frameless.jsp" %>

<form name="userInputForm" >
<input TYPE="text" name=”SomeName” LENGHT="10">
</div>
<div id="submitButton">
            <input name="pagemode" type="hidden" value="submit">
            <input type=button value="Search"
onClick="submitForm(document.userInputForm.SomeName.value)">
```

```
</div>
</form>
</BODY>

</HTML>
```

2.File “db_script.jsp”

```
<%@ page language="java" import="java.sql.*, java.lang.*,java.net.*" %>
<html>
<head></head>
<body>
<%
/* set up database connection */
Class.forName("oracle.jdbc.driver.OracleDriver");
String user = "user";
String password = "secret";
Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@oracle:1521:oradb", user,
password);
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
String searchName = request.getParameter("inpName");
/* run query */
ResultSet rs = stmt.executeQuery("select * from Professor WHERE name LIKE
'%"+searchName+"%'");
  if (rs != null)
  {
    rs.last();
    rs.beforeFirst();
  }

  if (rs != null) {
     while (rs.next())
     {
            String displayName = rs.getString("name");
%>
<P><%=displayName %>
<%
     } /* end of while */
  } /* end of if */
stmt.close();
conn.close();
%>
</body>

</html>
```

Flash. With Flash, website developers can add multimedia effects to websites, as well as create applications such as games, simulations, or navigation. Flash is a binary format that can be included in HTML as a multimedia object. Alternatively, Flash files can be viewed directly – the newer browser versions have a plug-in for Flash and can display it directly. Flash websites usually somewhat resemble a TV screen – something is always in motion, light effects, and music is not unusual.

When creating the so-called Flash-movies, one works with the Flash Authoring software, that lets website developer's position graphics, text, and sound on a time line.

Flash's authoring software is not useful for more complex applications. But Flash comes with an integrated programming language that can be used to solve more evolved problems.

The main feature of Flash is that it is an extremely powerful tool to visualize content. Its authoring software is available for purchase from the Macromedia corporation [MAC03]. Flash-plugins to view Flash movies, however, are available for free.

2.1.5 Conclusion

Since user interface is such a crucial part of the website, it is important to stick to well-established guidelines that were tested and that can be relied on. Since usability is the declared goal of web user interface, we will adhere to guidelines established by researchers in this area. Additionally, the new website shall be conform to the set of guidelines established by the University of Florida.

As far as technology is concerned, Table 1 shows an overview of the technologies presented in this Section. The deciding factors on what technologies we consider for this project are, in descending order of importance:

1. whether a technology is available for free. If this is not the case, the technology is immediately ruled out.
2. the complexity of the technology. If the time that is needed to understand and put to use the technology exceeds the time limit for this thesis, the technology cannot be used.
3. whether the technology allows for a distinction between design and content. This means, how much is logic (as encoded in scripts) and presentation (e.g., the HTML) separated. When an HTML file that contains scripting logic is not viewable

in a standard HTML editor such as FrontPage, it is an indicator that logic and design are not very well separated. We then rate the separation level to be low.

4. whether the technology adheres to UF guidelines. If a website technology is considered inappropriate by the University of Florida Web Administration group, it cannot qualify for the CISE website.

5. whether the technology is client-side or server side. This is an important factor when considering what part of the project the technology applies to.

The last column in table 2.1.5 indicates whether the technology was deemed appropriate for the CISE website development or not. The choice was made depending on how well the technology fitted the requirements as listed above.

Table 1 Overview of website implementation technologies

	Free	Complexity	Adherence To Guidelines	Client/Server side	Separation of design and content	Considered appropriate for CISE website development
HTML	✔	Low	✔	client	low	yes
CSS	✔	Low		client	high	yes
JavaScript	✔	Low		client	high	yes
XML	✔	Low	✔	Doesn't apply	high	yes
CGI/Perl	✔	Medium	✔	server	low	no
ASP		Medium	✔	server	low	no
DHTML	✔	Low		client	medium	yes
SSI	✔	Low	✔	server	low	yes
PHP	✔	Medium	✔	server	low	no
Java	✔	Medium	✔	Server/client	low	no
Flash		Low		client	low	no

2.2 Content Management Systems

2.2.1 Introduction

The Internet offers a wide range of options for presenting information. Documents are available in different formats: HTML, XML, and sometimes-in form of text documents (txt, pdf, ps, doc). Multimedia data, such as audio data (mp3, wav) and image (jpg, gif) and video data (MPEG, Quicktime) also play an increasing role. Websites face

the difficulties of integrating all of these different data formats. A web-site builder must address this problem when managing the content and structure of sites. Content Management Systems (CMS) were created to precisely help support this complex task.

Researchers at AT&T, who developed the Strudel CMS [FER97] and its successor Tiramisu [ALW99], have suggested that management of a website involves three tasks:

1. Content: Selecting and managing the data available at the site,
2. Logic: Data has to be organized and structured in individual pages, and
3. Style: Design of the pages.

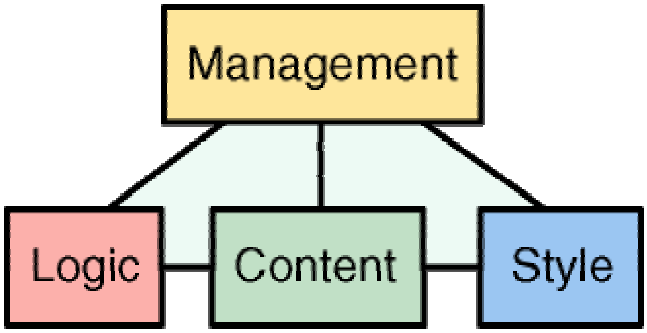


Figure 2-2 Principle of website management

Content could be managed by storing all data in a database or several databases. The data could then be retrieved through queries. Management of the logic refers to the definition of the site structure, i.e. the set of pages, the data contained in each page and the links between pages. Management of Style means the layout of the web pages. HTML is normally used for formatting the layout, but lacks some fundamental page-oriented formatting capabilities.

Many software products exist to help website developers with the task of layout management. FrontPage and Dreamweaver are part of these so-called WYSIWYG (What you see is what you get) editors. This means that pages are displayed there exactly as they will be seen later on the Internet. WYSIWYG tools are an immense help when

designing the look of a page. One could, for example, draw tables and drag-and-drop text boxes and other HTML elements in a page and the tool generates the corresponding HTML code. But is this sufficient for creating websites? Still, with every change to the look-and-feel of a site, one would have to modify every single HTML document. Since the links are hard-coded, every change would have to be manually edited. This applies to links within the site as well as links that lead to other web pages. The integrity that a site might have had when it was brand new and controlled by a few web page developers is easily lost as time passes by. This is visible to a site visitor when he discovers broken links, content that is out-of-date, and pages that look and act differently. Managing the look and organization of current, relevant content goes a long way toward looking trustworthy and professional. The use of a WYSIWYG tool is not of much help for these issues.

Solving problems with outdated content and links - both features traditionally thought of as "content management" - a Web Content Management System could help present a positive user experience. Table 3-1 presents an overview of the areas, which can be improved, with the help of a CMS.

Table 3-1 Overview of areas that can be improved using a CMS

Area	CMS Solution
Consistent Look and feel across a site	Templating: enforce design standards
Site always shows current information	Version and display new content; remove and archive old content, track broken links; eventually use of a database
Content is high quality	Workflow: pass content through review and approval cycles
Site loads quickly and is always available	Cache popular pages, deploy content to distributed servers
Navigation is up-to-date and accurate	Auto-index content, …

Throughout the writing of this thesis, the field of CM systems has seen an explosive growth. When we initially compiled a list of CMS that would come into consideration, 5 open source systems seemed to be most promising. When looking at websites that give an overview of CM systems today ([WEB03][CMS03]), one can easily find well over 50 products, many of them available for free. One german content management site [COM03] even lists 938 products (June 24th, 2003)!

Clearly, it would not have been possible to test and evaluate each of these systems. We have compared the 5 systems that were initially listed and relatively quickly opted for one of them. When the decision was made, we have not decided to research any further into the new and upcoming systems, partly due to the fact that the chosen system turned out to be a very good fit for our requirements.

In the following, we present the 5 CM systems we have originally taken into consideration. We will point out how each of the areas identified above – logic, content, and style – are being handled in the different systems. At the end of this Section, we have included a table comparing the major aspects of these systems. More detail is given on which system was chosen and what were the motives for doing so.

2.2.2 Strudel

Strudel is a website development system created by AT&T Research [FER97]. Content is being stored either in a database, external files, or in Strudel's internal data repository. All source data is converted into Strudel's own format, a labeled directed graph and wrappers translate all data into the graph model, called the data graph. The website administrator declaratively specifies the site's structure using a site-definition query in StruQL, Strudel's query language. The result of evaluating the site-definition query on the data graph is a site graph, which models both the site's content and

structure. A site graph can be rendered as a browsable website by Strudel's HTML generator, which produces HTML for every node in the site graph from a corresponding HTML template specified by the administrator. Since the HTML templates exist entirely independent of the content, we have a clear separation between the area of Content and Style. The area of Logic is being covered by the site-definition queries, which are independent of the HTML templates and the data residing in the databases. Advantages of the Strudel website management system are that there can be multiple views of the website with minimal effort – only new queries need to be created, or the HTML templates changed. Strudel allows for information to be drawn from many sources such as relational databases, Excel spreadsheets, Word documents, text documents, and other HTML files. Figure 2.2.2 gives an overview of Strudel's system architecture. The figure was taken from Strudel's website [ATT03].

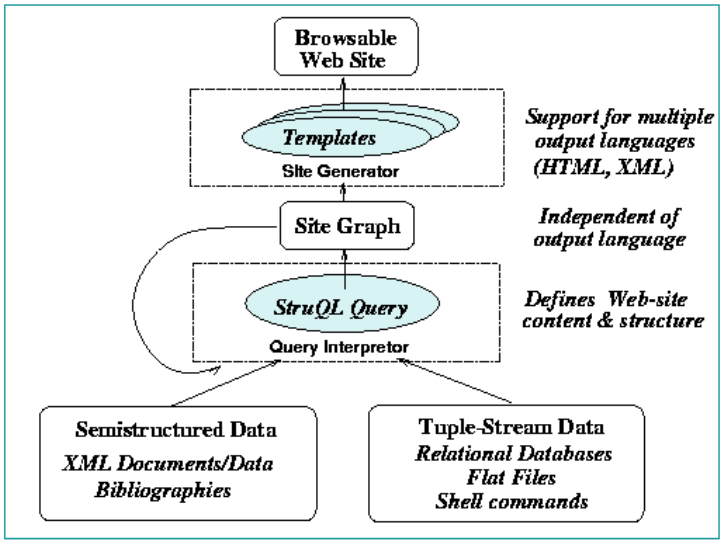


Figure 2-3 Strudel architecture

2.2.3 OpenCMS

OpenCms is an open source system that enables website builders to create and maintain websites. The software used to create the website is installed on a web server and is accessed via the Internet through a web browser. Different access permissions determine which actions users can perform and which components they see, i.e. users see only those files and directories that are relevant to them. OpenCms has an HTML editor, which is used to create content in the body. A template that ensures that the layout is uniform structures the body. OpenCms is only used to edit the body. This is how OpenCMS achieves a separation of Content and Style of a website, by assuming that the content is normally inserted in a body element. OpenCMS also supports Stylesheets to further separate content and layout. All of the files and folders of the different projects are not stored in the normal file system. OpenCms has a Virtual File System that holds the files and folders in a database. The OpenCms architecture allows using any SQL capable database that offers a JDBC connector.

2.2.4 Zope

Zope (which stands for “Z Object Publishing Environment") is an object-oriented web application server that is managed through a web interface. In Zope, one publishes "objects" – which can be plain pages, structured content, documents, images, binary files, or folders. Instead of storing its basic documents in a relational database or file system (although developers can access both with freely available programs), Zope puts everything in its Zope Object Database (ZODB). This way, every entity in the system can have properties and methods, security settings, and managed transaction support (including rollback). Zope also has functionality for XML documents; one can import or create documents within Zope and format, query, and manipulate XML. Separation of

content, style, and logic of web pages is done through the use of Zope's scripting languages DTML, Zope Page Templates, and Python Scripts. Content resides in a database or Page Templates, whereas Python Scripts define the logic. Page Templates and DTML are used to define the style of a webpage. The Zope framework is described in much greater detail in Chapter 3.

2.2.5 Ariadne

Ariadne is a web application server and a content management system. It is entirely written in the PHP scripting language [ARI03]. Ariadne can be used in conjunction with Apache, the web server also used by the CISE Department, and the MySQL and PostgreSQL database systems.

The separation of content, style, and logic is managed as follows: style is managed through Templates. They are defined over the web, in a browser and can be any kind of text type: html, xml, rtf, etc. Logic is managed through a scripting language "PINP," which consists of a subset of PHP functions. Ariadne stores content in a structured object store built on top a relational database system. The object store is very similar to the file system of the operating system. It generates files and directories and navigation is done in the Ariadne browser. The store or the database contains all the information for the website. This information is combined with class descriptions to generate objects. As is the case in traditional object-oriented programming, objects have methods associated with them, which works on the information that is contained in the database.

Content, logic, and style are merged to generate a webpage. Ariadne applies the user-defined templates (style) on these objects (content) to generate a webpage. Figure 2.2.5 gives an overview of Ariadne's architecture.

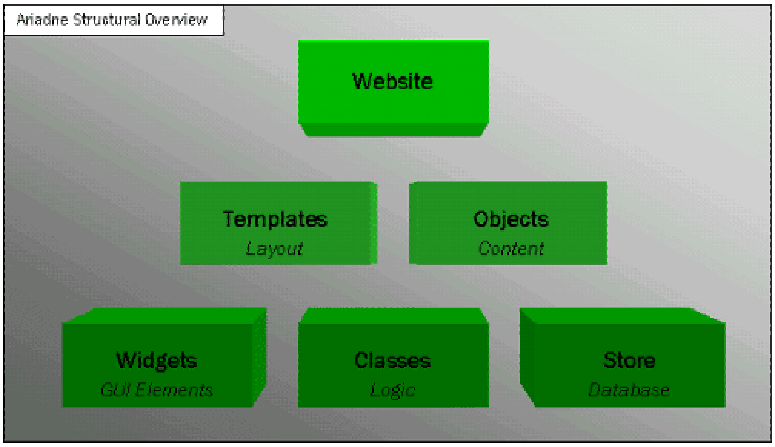


Figure 2-4 Ariadne's system architecture

2.2.6 Cocoon

Cocoon [COC03] was developed by the Apache Software Foundation as an open source project written in the Java language. In Cocoon, content is always stored in XML. The XML content is normally stored in files, but could also come from a relational or XML-database. The main idea is that XML-based data can be represented as needed in an HTML browser for a website, as PDF for printing purposes, as WML for cell phones, or RTF for use in office applications. Style is managed with XSLT. The advantage of using open standards such as XML and XSLT is that the data is not bound to a proprietary data format and proprietary systems, which are always bound to the current market and could disappear from one day to the next. Another strength of Cocoon is the simplicity, with which the same data can be represented in different output formats.

Cocoon's biggest disadvantage is the use of XSL. XSL is quite complex and cannot be mastered by someone who is used to working on the layout of websites with FrontPage and Photoshop, since some background in programming is required.

2.2.7 Evaluation

Table 3-2 gives an overview of the CM systems discussed in this chapter. The table includes the following information:

- Name of the product
- Web Server: does the product come with its own server, or does it integrate with existing web servers? This question is important since the CISE Department currently employs an Apache web server, and we assume that they intend to use that server in the future.
- Database: can the system be set up to query an existing RDBMS/ other database management system?
- Operating System: which operating system can the software be installed on?
- Input Document: what kind of document can be stored by the system?
- Output Document: what kind of document can be generated by the system?
- HTML Editor/ Generator: does the system come with an integrated HTML WYSIWYG editor, comparable to FrontPage or Dreamweaver?
- Access Rights Management: can many authors collaborate with this system?
- Programming Language used: what kind of languages will have to be learned by someone using the system?
- Documentation available: is documentation available on how the system is installed and used?
- User base: is there a community of developers to speak of?
- Installation: how hard and time-consuming will the installation be?
- Ruled out: due to which criterion was the system ruled out?

We have applied the following criterions to each of the systems, and thus singled out the one winning system, which was able to fulfill all criterions:

A. The system is not very well supported. A small user group and outdated or non-existent documentation indicate this. Another option is that the system belongs to the academic area is not yet fit for use in a concrete scenario.

B. The development of applications with this system would be very complex and time consuming. This could be due to the fact that the system is using unpopular programming languages, which would require a lot of time to learn.

C. The systems cannot be set up within the CISE environment. This could mean that the system does not allow a connection to an Oracle database, which is the one provided by the CISE Department, or that the system cannot be installed under one of the operating systems provided by CISE. Since the only operating system available for our usage is Solaris, this is an important factor. The CISE Department does provide Windows machines as well, however, administrator rights would be needed to install new software. The Solaris machines do not have this restriction.

Of all systems, we have chosen Zope for the implementation of the new CISE Department website. In the following, some additional reasons are given as to why Zope is a good fit for our requirements.

First and foremost, Zope is free of cost and distributed under an open-source license. This not only means that we can use Zope without having to pay for the software. It also means that as with most other open source software systems, the user community is fairly large – many people have made the same decision of not spending money when something free can be tested first. Whenever the user community is large, it means that

documentation must be available or is about to be created, and good support through mailing lists and web forums exist. This is definitely the case with Zope. Since there is a large community of application developers, new Zope add-on "Products" (more about this in Chapter 3) appear almost on a weekly basis. The quality of the product definitely increases when several developers are able to take a look at the source code ("peer review"). In open source systems, peer review is an important factor that ensures quality at a low price and the inclusion of new ideas are much easier to implement than in proprietary systems. Most closed-source, commercial tools do not let users extend, customize, and redistribute them. This is not the case with Zope. Another fact that arises from Zope's open source property is its continuity. Since there are many users and developers supporting this product, Zope is less likely to disappear from one day to the next than some proprietary CM software system, which is always dependent on the current market situation in the IT sector and thus constantly threatened by bankruptcy!

Furthermore, Zope itself is an inclusive platform. It comes with all the necessary components to begin developing an application. No extra license software is needed to support Zope. It also turned out that Zope was very easy to install, since no extra software had to be installed to make Zope function.

As stated in Chapter 4, one of the main requirements is the ease of use of the application. Ideally, no additional knowledge is needed when dealing with maintaining the CISE website. Zope's Management Interface is displayed in a web browser. The Internet Explorer, Mozilla, Netscape, OmniWeb, Konqueror, and Opera browsers are all known to be able to be used to display and manipulate Zope's development environment. Most everybody today knows how to use a web browser. One does not need to learn and

get used to an entirely new IDE (Integrated Development Environment). What's more, developers can delegate duties to other developers through the web using the same interface. Very few other application servers, if any, deliver the same level of functionality. Zope allows teams of developers to collaborate easily. Collaborative environments require tools to allow users to work without interfering with each other, so Zope has *Undo*, *Versions*, *History* and other tools to help people work safely together and recover from mistakes. Many other application servers do not provide these kinds of features. Some tools do not scale as well as Zope does to handle large numbers of developers and users. Zope has a consistent, powerful user management system that can scale to many users with unique, easily managed privileges.

Additionally, Zope runs on most popular microcomputer operating system platforms: Linux, Windows NT/2000/XP, Solaris, FreeBSD, NetBSD, OpenBSD, and Mac OS X. Zope even runs on Windows 98/ME (recommended only for development purposes, however). Many other application server platforms require that you run an operating system of their licensor's choosing. This gives the CISE Department a vast array of choices as to what operating system to install Zope on.

Moreover, Zope can be extended using the interpreted Python scripting language. Python is popular and easy to learn, and it promotes rapid development. Many libraries are available for Python. Many other application servers must be extended using compiled languages such as Java, which cuts down on development speed. Many other application servers use less popular languages for which there are not as many ready-to-use library features.

As described in Chapter 3, Zope and the programming language Python are entirely object-oriented. As opposed to scripting languages such as PHP and Perl, the object-orientation was not added as a feature after the language had existed for some years. Python, the base technology of Zope, was designed as an object-oriented language from scratch. If applied consequently, the OO-programming paradigm leads to better, more maintainable code. Whilst PHP was created for web applications, Python is more of a "real" programming language. This is an advantage when one has to implement functionality that is not directly related with web presentation.

Zope is being developed since 1996 and has proven itself to be extremely reliable. The Zope website [ZOP03b] lists a number of impressive, reputable clients (Navy, Nasa, and Bank of America to name but a few).

Another interesting aspect of Zope is the scalability. Zope Enterprise Objects can be added, which allow a site to increase capacity by adding computing resources to handle the load [EVE03].

Table 3-2 Comparison of CMS

	Zope	Cocoon	OpenCMS	Ariadne	Strudel
Web Server	Apache Tomcat Own server (Zserver)	Apache Tomcat	Apache (1.3.19 + up) or MS IIS 5.0	Apache Tomcat	Apache Tomcat
Database	Oracle, PostgreSQL, MySQL, Sybase, Interbase, DB2, any ODBC-compliant database, including SQL Server 2000	RDBMS, LDAP, Native XML Data bases	MySQL (3.23.40 + up) or Oracle (8.1.7 + up), MS-SQL 7.0	MYSQL, PostgrSQL	Relational Database (no further specification)
Operating System	Linux, Unix, Windows98/2000/ME/NT, MacOSX, FreeBSD, NetBSD, OpenBSD	Windows 2000, Linux, Solaris	Windows 98/2000/ME/NT, Linux, Solaris (2.7)	Windows 2000.	Linux, Unix, Windows98/ 2000/ME/NT
Input Document	XML, HTML	XML, HTML	XML, HTML	HTML, XML, RTF	XML
Output Document	PDF, HTML, TXT, DOC, PS	XML, HTML, RTF,	PDF, WAP, HTML, XML, TXT, DOC, PS	HTML, XML	HTML, XML, PDF
HTML Editor/Generator	NO	No Editor but has HTML Generator	YES	HTML Editor	No Editor but has HTML Generator
Access Rights Management	YES	XRPM	YES	YES	NO
Programming Language Used	Python, DTML	Java, Servlets, XML and XSP	Java	PHP	StruQL
Documentation Available	Online Help API available “Zope Administrators Guide”	YES	Online Help	Under Development	API available Documentation under construction (since 1999)
User Base	YES	YES	YES	8 companies	Research
Installation	Detailed installation guide setup wizard for Windows	YES	HTML setup wizard	Incomplete Installation guide	Rough installation guide on website
Ruled out		B	C	B	A

CHAPTER 3
ZOPE

The Z Object Publishing Environment (Zope) is a comprehensive *web application* development environment, web server, and content management system written in Python. In this chapter, we will explain Zope's purpose and audience in greater detail. Most of the information has been gathered from the Zope Book [LAT01].

3.1 Introduction

To understand the purpose of Zope, it is first of all necessary to understand what a web application is. We can make a distinction between a *static* and *dynamic* website. A dynamic website has also been termed a web application.

To maintain a static website, there has to be a person with specific access rights, the Webmaster, in charge of manually updating the site's content. This entails manually visiting and updating the HTML that makes up each page. Normally, the Webmaster does so by updating a set of files on the *web server* (the machine that runs the website). Each of these files represents a single page. If a Webmaster had to change the look-and-feel of a static website he would have to visit and update every single file that composes the website. For a large website (such as the CISE site) this task can become non-trivial. It is also the case that the Webmaster might make mistakes and forgets to update or remove critical pages and links.

Dynamic websites (or web applications) are also served over a web server. The difference to static websites is, however, that they are generated each time that a user requests the page. For example, a web application might query a database and generate

some HTML to display the results of the database query in a page. Web applications are extremely common. Some popular examples of web applications are those that let users search the web, like *Google*; collaborate on projects, like *SourceForge*; buy items at an auction like *eBay*; communicate with other people over e-mail, like *Hotmail*; or view the latest news such as *cnn.com*. Users and browsers are usually not aware of the difference between contacting a web server which delivers a static website and a web server which delivers a web application.

A framework, which allows people to construct a web application, is often called a *web application server*, or sometimes just an *application server*. A web application server typically allows a developer to create a web application using some common computer programming language. This is exactly the purpose of Zope.

3.2 Zope History

In 1996 Jim Fulton (the current CTO of Zope Corporation, the distributors of Zope) had to teach a class on CGI programming, even though he knew only little about the subject. *Common gateway interface* (CGI) programming is a commonly used web development model that allows developers to construct dynamic websites (please refer to Chapter 2, Related Work, for more information on CGI). Fulton realized that there were several aspects of CGI based programming that he did not like, and so he wrote the core of Zope on the plane flight back from the class.

Zope Corporation, which was then known as Digital Creations, then released three open source software packages to support web publishing, *Bobo*, *Document Template*, and *BoboPOS*. All of these packages were written in the Python programming language. All of them provided a web publishing facility, text templating, and an object database. Digital Creations had also developed a commercial application server based on their three

open source components. This product was called *Principia*. Investor Hadar Pedhazur convinced Fulton in November of 1998 to open source Principia. Zope has evolved from the Principia components.

According to the Zope website, the "Z" in *Z Object Publishing Environment* does not really mean anything in particular. Most of Zope is written in the Python scripting language, with performance-critical pieces written in C.

3.3 Zope' s Fundamental Concepts

This subsection lists the fundamental underlying concepts of the Zope framework.

Zope is a framework. Zope combines several tools including a web server, an object database (ZODB), and two scripting languages: a tag-based scripting alternative, (DTML) analogous to XSP or JSP, and one in which special formatting attributes are embedded directly inside HTML tags, (Zope Page Templates or ZPT), for compatibility with standard HTML tools such as Dreamweaver, Netscape Composer, or FrontPage. We describe the scripting languages in greater detail in Section 3.7. It is possible to build websites by writing pure HTML or by encoding automation using Python (Perl is also supported). It is easy to connect to external databases such as Oracle or MySQL, an area we cover in Section 3.8. Zope can also be attached to Apache instead of the built-in Zope "Medusa" web server.

Object orientation. Most web scripting languages are procedural, such as Perl, PHP, or ASP. Unlike these popular scripting languages, Zope is an object-oriented application server. Object orientation is a software development pattern that is used in many programming languages (C++, Java, Python, Eiffel, Modula-2, others). We give further details in Section 3.5.

Website publishing. Zope's founders realized that the Web is fundamentally object-oriented. A URL to a Web resource is nothing but a path to an object in a set of containers. A way of sending messages to an object is provided with the HTTP protocol.

The object structure of Zope is hierarchical. This means that a Zope site is typically composed of objects, which contain other objects (which in turn contain other objects, etc.). URLs map to objects in the hierarchical Zope environment based on their names. For example, the URL "Research/index.html"could be used to access the Document object named "index.html"located in the Folder object named "Research."

Zope's duty is to "publish" the objects one creates. This works as follows: say a web browser sends a request to the Zope server. The request specifies a URL in the form `protocol://host:port/path?querystring`,e.g. `http://www.zope.org:8080/Resources?batch_start=100`.

Zope separates the URL into its component "host," "port", "path", and "query string" portions (`'http://www.zope.org'`, `8080`, `/Resources` and `?batch_start=100`, respectively).

Zope then locates the object in its object database corresponding to the "path" (`'/Resources'`). It executes the object using the query string as a source of parameters that could alter the behavior of the object. This means that the object may behave differently depending on the values passed in the query string. If it is the case that executing the object returns a value, then the value is sent back to the browser. Normally, a given Zope object returns HTML, file data, or image data, which is interpreted by the browser and shown to the user.

Mapping URLs to objects is not a new idea. Web servers like Apache and Microsoft's IIS do the same thing. They translate URLs to files and directories on a file system. Zope similarly maps URLs on to objects in its object database.

A Zope object's URL is based on its “path.” It is composed of the `ids` of its containing Folders and the object's `id`, separated by slash characters. For example, if there is a Zope "Folder" object in the root folder called *Schneider*, then its path would be `/Schneider`. If *Schneider* is in a sub-folder called *Professors* then its URL would be `/Professors/Schneider`.

There could also be other Folders in the *Professors* folder called *Hammer* and *Dankel*. One would access them through the web similarly:

```
/Professors/Hammer
/Professors/Dankel.
```

The URL of an object is composed of its `host, port`, and `path`. So for the Zope object with the path `/Schneider` on the Zope server at `http://localhost:8080`, the URL would be `http://localhost:8080/Schneider`. Visiting a URL of a Zope object directly is termed *calling the object through the web*. This causes the object to be evaluated and the result of the evaluation is returned to the web browser. This is also where Zope’s name stems from – “objects” are “published” and presented in an appropriate format to the user.

Zope’s Web Management Interface. A web browser is all one needs to access the Zope Management Interface (ZMI) since all management and application development can be done completely through the web. The Zope management interface provides a

familiar Windows Explorer-like view of the Zope object system. Screenshots can be found in Chapter 6. Through the management interface a developer can create Zope objects without requiring access to the file system of the web server. More on the ZMI can be found in Section 3.6.

Security. Usually, web applications have more than one author. This can quickly become problematic. It is important to decide how much control every author should get. The question of how this will affect the security immediately arises. Suppose, an author has access to all files. What happens if the SQL code embedded in one of the files the author is working on exposes the database login (which is usually the case with ASP/JSP/PHP)? The author could get access to the database, which might not have been planned.

The objects in Zope provide a much richer set of possible permissions than a conventional file-based system. Permissions vary by object type based on the capabilities of that object. It is therefore possible to implement strict security control such as setting access control so that authors can use "Z SQL Method" objects, but not change them or even view their source. One could also set restrictions so that a user can only create certain kinds of objects, for instance "Folders" and "DTML Documents" but not "SQL Methods" or other objects.

Zope provides the capability to manage users through the web via "User Folders," which are special folders that contain user information. The ability to add new User Folders can be delegated to users within a subfolder. This allows delegating the creation and user management of subsections of the website to semi-trusted users without having to worry about those users changing the objects "above" (in the hierarchy) their folder.

Native object persistence and transactions. The Zope Object Database (ZODB), a transactional object database, is in charge of storing all Zope objects. Each web request is treated as a separate transaction by the ZODB. Should an error occur in the application during a request, then any changes made during the request are automatically rolled back. The ZODB also provides multi-level undo, which allows a site manager to “undo” changes to the site with the click of a button. The Zope framework makes all of the details of persistence and transactions totally transparent to the application developer. We cover this area in more detail in Section 3.8

Acquisition. Acquisition is one of the most powerful aspects of Zope. The basic idea is that since Zope objects are contained inside other objects (such as Folders), objects can "acquire" attributes and behavior from their containers.

The concept of acquisition works with all Zope objects. A commonly used SQL query or snippet of HTML, for example, can be defined in one Folder and objects in subfolders can use it automatically through acquisition. If the query needs to be changed, one only needs to change it in one place without worrying about all of the sub objects that use the query. This is particularly useful when compared to other scripting languages, which normally mingle SQL code, and HTML tags such as JSP, ASP, and PHP (see Section 2.1.4). When a query changes, the programmer needs to edit the query everywhere it is executed, which can become very tedious and time-consuming.

Objects are acquired by starting at the current level in the containment hierarchy and searching upward. Therefore, it is easy to specialize areas of a site with a minimum of work. If, for example, one has a Folder named "Research" on a site containing research-related content, one could create a new header and footer document in the

Research Folder that use a research-related look-and-feel. Content in the Research folder and its subfolders will then use the specialized research header and footer found in the "Research" folder rather than the header and footer from the root folder on the site.

Extensibility. Zope is easily extensible. More advanced users can create new kinds of Zope objects, either by writing new Zope add-ons in Python or by building them completely through the Web. There are a set of Zope add-on products available on the Zope Corporation's website that provide features like drop-in Web discussion topics, desktop data publishing, XML tools and e-commerce integration. Many of these products have been written by the highly active members of the Zope community, and most are also open source. To illustrate how active this community is, one only needs to take a look at the 757 Zope free products listed on Zope's website [ZOP03a] (date: July 9th 2003)!

3.4 Zope Architecture

Zope's fundamental components are shown in Figure 3.

Zserver. Zope comes with a built-in web server that serves content to users. This web server also serves Zope content via FTP, WebDAV, and XML-RPC (a remote procedure call facility).

Web server. In case one already has an existing web server, such as *Apache* or *Microsoft IIS* and one does not wish to use Zope's, Zope can be set up to work with an existing web server. Zope works with any other web server that supports the Common Gateway Interface (CGI).

Zope core . This is the engine, which coordinates everything, driving the management interface and object database.

Object database. All objects are stored in Zope's object database.

Relational database. Not all information has to be stored in Zope's object database. Zope works with other relational databases such as *Oracle*, *PostgreSQL*, *Sybase*, *MySQL* and others.

File system. Zope works with documents and other files stored on the server's file system.

ZClasses. Zope allows site managers to add new object types to Zope using the Zope Management Interface. ZClasses are these kinds of objects.

Products. Zope also allows site managers to add new object types to Zope by installing "Product" files on their Zope server's file system.

3.5 Object Orientation in Zope

As mentioned in the Fundamental Concepts Section above, Zope is an object-oriented development environment. Therefore, we need to explain attributes, methods, classes, and inheritance to gain full understanding of Zope. This chapter provides an overview of the fundamentals of object orientation from the perspective of a Zope developer.

Objects. In an object-oriented application, programs designed around *objects*. Objects are self-contained "bundles" of data and logic. Both data and code are stored in one or more objects, each of which represents a particular "thing." In Zope, the *Control_Panel* is an object, Folders which one creates are objects, and even the Zope "root folder" is an object. When one uses the Zope "add list" to create a new item in the Zope Management Interface, one creates an object. People who extend Zope by creating *Products* define their own types of objects which are then entered in to the Zope "add list," allowing one to create objects from them. A product author might define a "Form"

object or a "Weblog" object. Basically, anything that can be described using a noun can be modeled as an object.

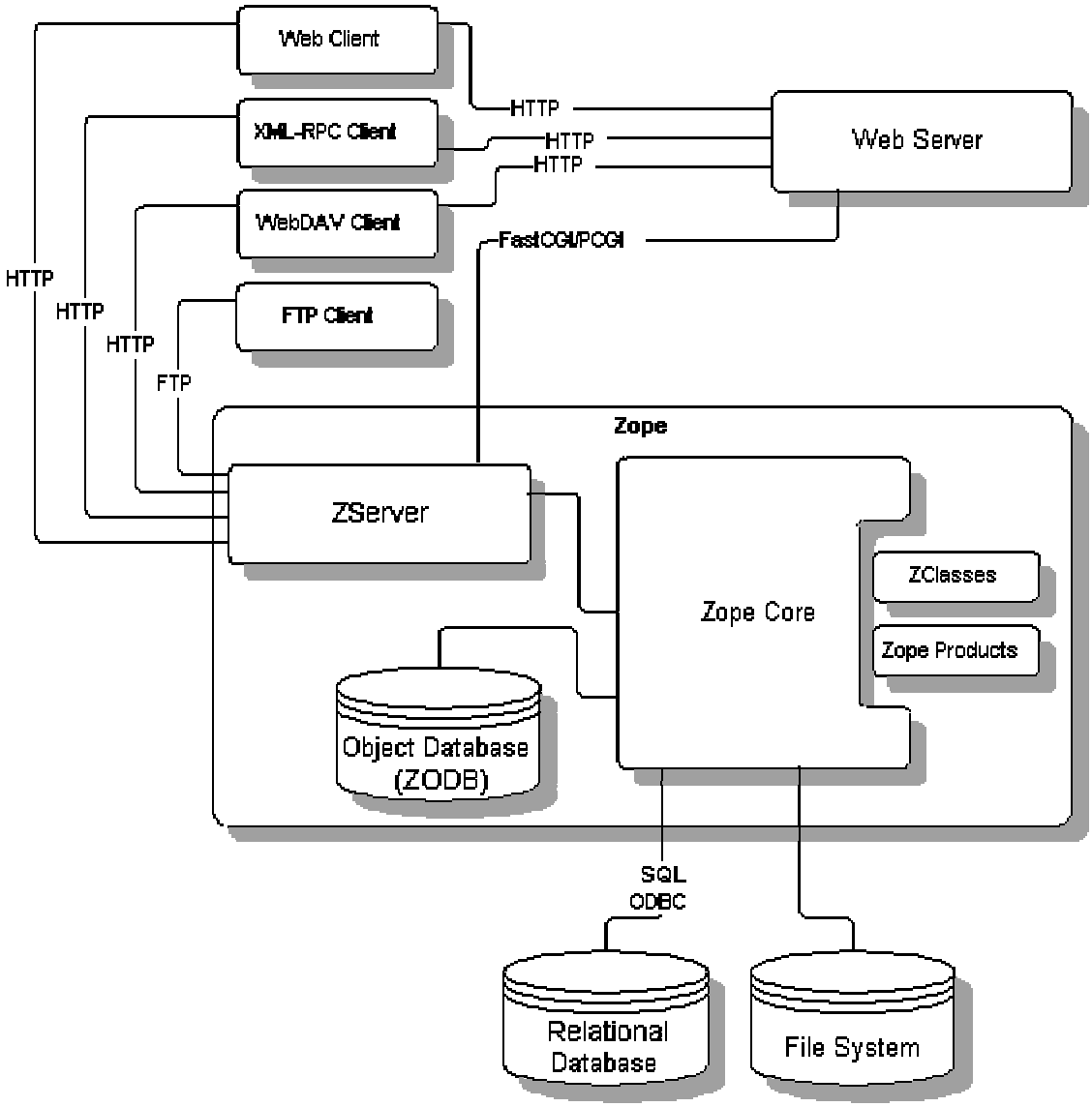


Figure 3 Zope Architecture

Attributes. The data of an object is defined by its *attributes*. Typically, an object uses attributes to store elements that describe it. The attributes assigned to an object define the object's *state*. *Properties* in Zope are special kinds of web-editable object attributes.

Methods. *Methods* define the set of actions that an object may perform. They are code definitions attached to an object, which typically perform an action based on the *attributes* belonging to the object on which the method is defined.

Some objects in Zope are actually called "methods." For example, there are *DTML Methods*, *SQL Methods*, and *External Methods*. They are "bound" to their containing Folder object by default when called, and the logic that they contain normally makes reference to their containing Folder.

Classes and instances. A class defines an object's behavior and acts as a constructor for an object. The objects that are constructed by a class are the *instances*. As an example, there could be a `Research` folder with an `id` attribute of `Research`, while another folder may have an `id` attribute of `MyFolder`, but they are both instances of the same class, and therefore behave identically. All of the objects that one deals with using the Zope management interface are instances of a class. Typically, if the class did not come by default with Zope, then it was defined in Zope *Products*, which are created by Zope developers and community members.

Inheritance. Sometimes it is desirable for objects to share the same essential behavior, except for small deviations from each other. In Zope, inheritance is used extensively. For example, the Zope "Image" class inherits its behavior from the Zope "File" class, because images are really just another kind of file, and they share many behavior requirements. But the "Image" class adds a bit of behavior which allows it to "render itself" by printing an HTML tag instead of causing a file download. It does this by *overriding* the `index_html` method of the File class.

Object lifetimes. Object instances have a specific *lifetime*. Either a programmer or a user of the system in which the objects “live” typically controls this lifetime.

Instances of web-manageable objects in Zope like Files, Folders, DTML Methods, and such have a lifetime of from when a user creates them until she deletes them. These kinds of objects are also described as *persistent* objects. They are stored in the ZODB.

Other object instances have different lifetimes. There are object instances in Zope, which last for a "programmer-controlled" period of time. For instance, the object that represents a web request in Zope (also called REQUEST), has a well-defined lifetime. Its lifetime lasts from the moment that the object publisher receives the request from a remote browser until the response is sent back to that browser. It is then destroyed automatically. Zope "session data" objects have another well-defined lifetime. These objects last from the time that a programmer creates one on behalf of the user via his code until the system (on behalf of the programmer or site administrator) deems it necessary to throw away the object in order to conserve space or indicate an "end" to the user's session. This is defined by default as 20 minutes of "inactivity" by the user for whom the object was created.

3.6 Zope Management Interface

Upon logging in to Zope over a web browser, one is presented with the Zope Management Interface (ZMI). The ZMI is a management and development environment that allows users to control Zope, manipulate Zope objects, and develop web applications.

The ZMI represents a view into the Zope *object hierarchy*. Almost every link or button in the ZMI represents an action that is taken against an *object*. Objects can be dropped in anywhere in the object hierarchy. Site managers can work with their objects

by clicking on tabs that represent different "views" of an object. These views vary depending on the type of object. A "DTML Method" Zope object, for example, has an "Edit" tab which allows one to edit the document's source, while a "Database Connection" Zope object provides views that let one modify the connection string or caching parameters for the object. All objects also have a "Security" view that allows one to manage access control settings for that object.

The Zope management interface is broken into three frames:

- The left frame is called the *Navigator Frame.* The user can navigate around Zope much like he would navigate around a file system with a file manager like the Windows Explorer. In this frame one sees the root folder and all of its subfolders. The root folder is in the upper left corner of the tree. The root folder is the "top" of Zope. Everything in Zope lives inside the root folder.
- Above the folder tree Zope shows login information in a frame (called *Status Frame*), i.e. which user is presently logged in.
- To manage a folder, the user must click on it and it will appear in the right-hand frame of the browser window. This frame is called the W*orkspace Frame*. The workspace gives information about the current object, and lets one change it. Across the top of the screen are a number of tabs. Each tab takes the user to a different *view* of the current object. Each view lets the user perform a different management function on that object.

3.7 Zope's Scripting Languages

3.7.1 Python

So far, we have made a distinction between logic, content, and style. This Section explains in more detail how this distinction is achieved in Zope. What is the difference between *Logic* and *Style*? Style formats and displays information, whereas Logic provides those actions which change objects, send messages, test conditions and respond to events. With Zope, style is handled with DTML or Page Templates, and logic is handled with Python scripts.

Zope *Script* objects are objects that encapsulate a small chunk of code written in a programming language. Script objects first appeared in Zope 2.3 and are now the

preferred way to write programming logic in Zope. Currently, Zope comes with *Python-based Scripts*, which are written in the Python language. There is a third-party extension to Zope, which allows users to write *Perl-based Scripts* in the Perl language. However, support for Perl based scripts has almost vanished, and products for this area are not actively under development any more.

Python code comes in two variants for the Zope framework: Python Scripts and External Methods.

Python scripts. Python, a general purpose scripting language, can be used to control Zope objects and perform other tasks. These Scripts give general purpose programming facilities within Zope. To create a Python-based Script one needs to choose *Script (Python)* from the Product add list in the Zope Management Interface.

External methods. An *External Method* executes Python code like a *Python Script* does. However, this code is not stored in the ZODB but in a Python source file in the file system. As the file system is considered much safer than the web-editable ZODB, external methods are not restricted by Zope's security system. They can access the complete Python library, all Zope packages and modules and any attribute in objects, even private ones.

Sometimes the security constraints imposed by scripts, DTML, and ZPT are too strict. This could be the case when, for example, one wants to read files from disk, or access the network, or use some advanced libraries for things like regular expressions or image processing. *External Methods* would be used in these cases.

To create and edit External Methods the file system needs to be accessed. This makes editing these scripts more cumbersome since they cannot be edited right in a web

browser. However, requiring access to the server's file system provides an important security control. If a user has access to a server's file system they already have the ability to harm Zope. So by requiring that unrestricted scripts be edited on the file system, Zope ensures that only people who are already trusted have access.

External Method code is created and edited in files on the Zope server in the *Extensions* directory. This directory is located in the top-level Zope directory. To add an *External Method* in Zope, one needs choose *External Method* from the product add list in the Zope Management Interface.

Calling Python scripts. There are two general ways to call a script and provide it with a context: by visiting a URL, and by calling the script from another script or template. The following Section explains how a Python Script is called by visiting a URL or from another script. The following Sections about DTML and Zope Page Templates detail how a Python Script is called, respectively.

Calling Scripts from the web. A script can be called directly with a web browser by visiting its URL. A single script can be called on different objects by using different URLS. This works because Zope can determine the script's context by URL. This is a powerful feature that enables users to apply logic to objects like documents or folders without having to embed the actual code within the object.

To call a script on an object from the web, one needs to visit the URL of the object, followed by the name of the script. This places the script in the context of your object. To call the *add* script on the *Professor* object inside the root folder one would visit the URL *Professor/add*.

Arguments can be passed to a URL, too. They are appended as standard query strings:

```
http://zope:8080/Professor/add?name=Schneider
```

Calling Python scripts from a Python script. Calling scripts from other Python or Perl scripts works the same as calling scripts from DTML, except that script parameters must *always* be passed calling a script from Python. For example, here is how the *updateInfo* script might be called from within a Python script:

```
newName='Schneider'
context.update(name=newName, email="mschneid@cise.ufl.edu").
```

The *context* variable is used to tell Zope to find updateInfo by acquisition.

3.7.2 DTML

DTML (Document Template Markup Language) is a templating facility, which supports the creation of dynamic HTML and text. It belongs to the family of a server-side scripting languages. This means that Zope executes DTML commands at the server, and the result of that execution is sent to the web browser. By contrast, client-side scripting languages like JavaScript are not processed by the server, but are rather sent to and executed by the web browser. DTML is used for scripting in two types of Zope objects, *DTML Documents* and *DTML Methods*.

DTML is typically used in Zope to create dynamic web pages. For example, one might use DTML to create a web page, which "fills in" rows and cells of an HTML table contained within the page from data fetched out of a database.

DTML is a *tag-based* presentation and scripting language. This means that *tags* (e.g. '<dtml-var name_of_object>') embedded in the HTML code cause parts of the page to be replaced with "computed" content. Typically, DTML is mixed with HTML as follows

```
<dtml-var standard_html_header>
<h1>Hello World!</h1>
<dtml-var standard_html_footer>.
```

The above DTML methods standard_html_header and standard_html_footer are either DTML documents or DTML methods. They would, in turn contain a mixture of DTML and HTML. The resulting page consists of the generated HTML only. The basic idea is that DTML Methods can act as templates tying reusable bits of content together into dynamic web pages.

DTML is similar in function to "HTML-embedded" scripting languages such as JSP, ASP and PHP. It differs from these facilities by not allowing users to create "inline" Python *statements* (if... then.. else..) in the way that JSP or PHP will allow embedding a block of their respective language's code into an HTML page. DTML provides flow control and conditional logic by way of "special" HTML tags (<dtml-if>, <dtml-else>, …). It can also be compared to the web server facility of Server Side Includes (SSI), but with far more features and flexibility.

Zope has a technology named Zope Presentation Templates, which has purpose similar to DTML. They are presented further in the next Section. DTML and ZPT are both scripting languages which allow users to create dynamic HTML. However, DTML is capable of creating dynamic text, which is *not* HTML, while ZPT is limited to creating text which is HTML (or XML). DTML also allows users to embed more extensive "logic" in the form of conditionals and flow-control than does ZPT. While the source to a ZPT page is almost always "well-formed" HTML through its lifetime, the source to DTML pages are not guaranteed to be "well-formed" HTML, and thus do not play well in many cases with external editing tools such as FrontPage or Dreamweaver.

Calling Python scripts from DTML. Python scripts are called from DTML with the *call* tag. For example:

```
<dtml-call scriptName>.
```

DTML will call the *scriptName* script, whether it is implemented in Perl, Python, or any other language. Other DTML objects and SQL Methods would be called the same way.

If the *scriptName* script requires parameters, one can simply pass in any variables that are valid in the current DTML namespace. For example, if *newName* and *newEmail* are defined using `<dtml-let>`, the variables can be passed as parameters like this:

```
<dtml-call expr="scriptName(name=newName, email=newEmail)">.
```

3.7.3 Zope page templates

Page Templates are a web page generation tool. They help programmers and designers collaborate in producing dynamic web pages for Zope web applications. Designers can use them to maintain pages without having to abandon their layout tools.

The goal of Page Templates is to allow designers and programmers to work together easily. A designer can use a WYSIWYG HTML editor to create a template, then a programmer can edit it to make it part of an application. If required, the designer can load the template *back* into his editor and make further changes to its structure and appearance. By taking reasonable steps to preserve the changes made by the programmer, the designer will not disrupt the application.

Page Templates aim at this goal by adopting three principles:

1. Play nicely with editing tools.
2. What you see is very similar to what you get.
3. Keep code out of templates, except for structural logic.

A Page Template is like a model of the pages that it will generate. In particular, it is a valid HTML page.

Page Templates use the Template Attribute Language (TAL). TAL consists of special tag attributes. For example, a dynamic page title might look like this:

```
<title tal:content="here/title">Page Title</title>.
```

The `tal:content` attribute is a TAL statement. The name `content` indicates that it will set the text contained by the `title` tag, and the value `"here/title"` is an expression providing the text to insert into the tag.

Since it has an XML namespace (the `tal:` part) most editing tools will not complain that they do not understand it, and will not remove it. It will not change the structure or appearance of the template when loaded into a WYSIWYG editor such as FrontPage or a web browser. This makes TAL quite unique, when compared to other scripting languages. Most popular scripting languages such as ASP, PHP, JSP, and CGI/Perl modify the HTML document in a way that a traditional HTML editor cannot correctly display it. This immediately means that design and content are not clearly separated any more. All TAL statements consist of tag attributes whose name starts with `tal:` and all TAL statements have values associated with them. The value of a TAL statement is shown inside quotes (e.g., “here/title”).

To the HTML designer using a WYSIWYG tool, the dynamic title example is perfectly valid HTML, and shows up in their editor looking like a title should look like. In other words, Page Templates play nicely with editing tools.

This example also demonstrates the principle that "What you see is very similar to what you get.” When the template is viewed in an editor, the title text will act as a placeholder for the dynamic title text. The template provides an example of how generated documents will look.

When this template is saved in Zope and viewed by a user, Zope turns the dummy content into dynamic content, replacing `"Page Title"` with whatever `"here/title"` resolves to. In this case, `"here/title"` resolves to the title of the object to which the template is applied. This substitution is done dynamically, when the template is viewed.

3.7.3.1 Path Expressions

In the above example, “here/title” is a path expression. There are many other types of path expressions define by the TALES (TAL Expression Syntax) specification. The “here/title” path expression in the above example fetches the title property from the object to which the template is applied. Another common path expression is container/objectIds, which returns a list of Ids of the objects in the same folder as the template.

Every path expression starts with a variable name. To get to sub-objects or properties, one has to add a slash (‘/’) at the end of the variable name.

In the following, some basic tal tags are explained.

tal:replace. Zope replaces the entire tag with the value of the path expression.

For example, `<b tal:replace="here/title"></b>` would evaluate to the title being displayed in the normal format, not bold, because the entire `<b>` tag was replaced. To place dynamic text inside of other text, one would usually use `tal:replace` on a `span` tag rather than on a bold tag. For example the code:

```
<html>
The URL is <span tal:replace="request/URL">http://www.example.com</span>.
</html>
```

would translate to (assuming that the id of the Page Template containing the line of code is “simple_page):

```
<html>
```

```
The URL is http://localhost:8080/simple_page.
</html> .
```

This is because the span tag is structural, not visual.

tal:content. If the goal is to insert text into a tag but leave the tag itself alone, one would use the `tal:content` statement. To set the title of a page to the template's title property, one would write:

```
<head>
  <title tal:content="template/title">The Title</title>
</head>.
```

Assuming that the title of the page template is “Simple Page,” the source of the rendered page would then be:

```
<html>
  <head>
    <title>Simple Page</title>
  </head>.
```

As opposed to the tal:replace tag, the <title> tag was not removed, it is only the content of the tag that has changed.

tal:repeat. The repeat statement is used to generate loops. To illustrate with an example, repeat could be used to generate a list of the objects that are in the same folder as the template. One could create a table that has columns for the id and title of each object as follows:

```
<table border="1" width="100%">
  <tr>
    <th>Id</th>
    <th>Title</th>
  </tr>
  <tr tal:repeat="item container/objectValues">
    <td tal:content="item/getId">Id</td>
    <td tal:content="item/title">Title</td>
  </tr>
</table>.
```

The `tal:repeat` statement on the table row means “repeat this row for each item in the container's list of object values.” The repeat statement puts the objects from the list into the `item` variable one at a time (item is called the repeat variable) and makes a copy

of the row using that variable. The value of “item/getId” in each row is the Id of the object for that row, and likewise “item/title.”

The above could would for example translate to:

```
<table border="1" width="100%">
  <tr>
    <th>Id</th>
    <th>Title</th>
  </tr>
  <tr>
    <td>acl_users</td>
    <td>User Folder</td>
  </tr>
  <tr>
    <td>images</td>
    <td></td>
  </tr>
  <tr>
    <td>cise</td>
    <td></td>
  </tr>
</table>.
```

One can use any name for the repeat variable (“item” is only an example), as long as it starts with a letter and contains only letters, numbers, and underscores (“_”). The repeat variable is only defined in the repeat tag. If trying to access it above or below the `tr` tag one will get an error.

tal:condition. This tag is useful for dynamically querying the environment and selectively inserting text depending on some condition. A `tal:condition` statement leaves the tag and its contents in place if its expression has a true value, but removes them if the value is false. Zope considers the number zero, a blank string, an empty list, and the built-in variable `nothing` to be false values. Nearly every other value is true, including non-zero numbers, and strings with anything in them, including spaces.

tal:attributes. The tal:attributes tag replaces an attribute of an HTML tag with another value. A common use of this is to change links in <a href> statements. The following code:

```
<a href="link" tal:attributes="href request/URL">Link</a>
```

would replace the “link” with the URL of the request, for example:

```
<a href="http://rain.cise.ufl.edu:8080">Link</a>.
```

tal:define. The `tal:define` attribute lets programmers define their own variables. One reason for doing so is to avoid having to write long expressions repeatedly in a template. Another is to avoid having to call expensive methods repeatedly. One could define a variable once within an element on a tag and then use it many times within elements which are enclosed by this tag. For example, this define statement declares a list as a variable and later tests it and repeats over it:

```
<ul tal:define="items container/objectIds"
    tal:condition="items">
  <li tal:repeat="item items">
    <p tal:content="item">id</p>
  </li>
</ul>.
```

The `tal:define` statement creates the variable `items`, which can then be used anywhere inside the `ul` element. The first statement assigns the variable `items` and the second uses `items` in a condition to see whether it is false (in this case, an empty sequence) or true. If the `items` variable is false, then the `ul` element (and everything contained in the <ul>.. tags) is not shown.

The `items` variable in the above example is only visible from the beginning of the <ul> tag to its closing tag. By placing the keyword `global` in front of the variable name in combination with the `span` tag, one can make the definition last from the `span` tag to the bottom of the template:

```
<span tal:define="global items container/objectIds"></span>.
```

3.7.3.2 Calling Python scripts from Page Templates

There are two ways of calling a Python script from within a ZPT. The quick way is to call a script through the tal:define tag, for example:

```
<span tal:define="dummy here/scriptName"/>.
```

This example calls the scriptName method and assigns the result to the dummy variable. In a page template, *here* refers to the current context. It behaves much like the *context* variable in a Python-based Script. In other words, *scriptName* will be looked up by acquisition. If the script that is being called requires arguments, one must use a TALES python expression like so:

```
<div tal:replace="python:here.scriptName(param='one')" />.
```

Just as in Path Expressions, the `here` variable refers to the acquisition context the Page Template is called in. The python expression above is exactly like a line of code one might write in a Python Script. The only difference is the name of the variable used to get the acquisition context. Unfortunately, the different names used in ZPT and Python Scripts (context and here) evolved independently. The ZPT variable *here* is planned to become *context* in a future version of Zope, probably Zope 3.

Another way of calling a script is through using a common pattern called the "form/action/response pattern.” The form and response should be Page Templates and the action should be a script. The form template gathers the input and calls the action script. The action script should process the input and return a response template. This pattern is more flexible than the form/action pattern since it allows the script to return any of a number of different response objects

For example here's a part of a form template:

```
<form action="action">
  <input type="text" name="name">
  <input type="text" name="age:int">
  <input type="submit">
</form>.
```

This form could be processed by this script:

```
## Script (Python) "action"
##parameters=name, age
##
container.addPerson(name, age)
return container.responseTemplate().
```

This script calls a method to process the input (“addPerson”) and then returns another template, the response (“responseTemplate”). One can render a Page Template from Python by calling it. The response template typically contains an acknowledgment that the form has been correctly processed.

The action script can do all kinds of things. It can validate input, handle errors, send email, etc. Here is a sketch of how to validate input with a script:

```
## Script (Python) "action"
##
if not context.validateData(request):
    # if there's a problem return the form page template
    # along with an error message
    return context.formTemplate(error_message='Invalid data')

# otherwise return the thanks page
return context.responseTemplate().
```

This script validates the form input and returns the form template with an error message if there's a problem. One can pass Page Templates extra information with keyword arguments. The keyword arguments are available to the template via the `options` built-in variable. So the formTemplate in this example might include a section like this:

```
<span tal:condition="options/error_message | nothing">
Error: <b tal:content="options/error_message">
  Error message goes here.
</b></span>.
```

This example shows how to display an error message that is passed to the template via keyword arguments. The expression “ `| nothing`” is used to handle the case where no `error_message` argument has been passed to the template.

3.7.3.3 Macros

One important feature of Page Templates is the ability to reuse look and feel elements across many pages. With DTML, one can achieve so by inserting DTML methods such as the standard_html_header and the standard_html_footer. ZPT offers this functionality, but in a slightly different way. With Page Templates, one can create sites with a standard look and feel. No matter what the "content" of a page, it will have a standard header, sidebar, footer, and/or other page elements. This is a very common requirement for websites.

The utility for reusing presentation elements across pages is called *macros*. Macros define a section of a page that can be reused in other pages. A macro can be an entire page, or just a chunk of a page such as a header or footer. After defining one or more macros in one Page Template, they are used in other Page Templates.

Defining macros. Macros are defined with tag attributes similar to TAL statements. Macro tag attributes are called Macro Expansion Tag Attribute Language (METAL) statements. Here's an example macro definition:

```
<b metal:define-macro="copyright">
  Copyright 2003, <em>Rima Gerhard</em>.
</b>
```

This `metal:define-macro` statement defines a macro named “copyright.” The macro consists of the `b` element (including all contained elements).

Macros defined in a Page Template are stored in the template's `macros` attribute. They are then used from other Page Templates by referring to them through the `macros`

attribute of the Page Template in which they are defined. For example, suppose the `copyright` macro is in a Page Template called “master_page.” Here is how to use `copyright` macro from another Page Template:

```
<hr>
<p metal:use-macro="container/master_page/macros/copyright">
  Macro goes here
</p>.
```

In this Page Template, the p element will be completely replaced by the macro when Zope renders the page:

```
<hr>
<b>
  Copyright 2003, <em>Rima Gerhard</em>.
</b>.
```

If the macro changes (in the above example, if the copyright holder changes) then all Page Templates that use the macro will automatically reflect the change.

Zope handles macros first when rendering Page Templates. Then Zope evaluates TAL expressions. For example, consider this macro:

```
<p metal:define-macro="title"
   tal:content="template/title">
  template's title
</p>.
```

When using this macro it will insert the title of the Page Template in which the macro is used, *not* the title of the template in which the macro is defined. In other words, when using a macro, it's like copying the text of a macro into a template and then rendering the template.

When checking the *Expand macros when editing* option on the Page Template *Edit* view, then any macros will be expanded in the template's source. When editing in the ZMI, rather than using a WYSIWYG editing tool, it is more convenient not to expand macros when editing. This is the default for newly created templates. When using

WYSIWYG tools, however, it is often desirable to have the macros expanded in order to edit a complete page.

Slots. Macros are much more useful when one has the option to override parts of them when using them. This can be done by defining *slots* in the macro that can be filled in when the template is used. For example, the following sidebar macro:

```
<div metal:define-macro="sidebar">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/research">Research</a></li>
    <li><a href="/projects">Projects</a></li>
    <li><a href="/contact">Contact Us</a></li>
  </ul>
</div>
```

can be extended by including some additional information in the sidebar on some pages. One way to accomplish this is with slots:

```
<div metal:define-macro="sidebar">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/research">Research</a></li>
    <li><a href="/projects">Projects</a></li>
    <li><a href="/contact">Contact Us</a></li>
  </ul>
  <span metal:define-slot="additional_info"></span>
</div>.
```

When using this macro the slot can be filled like so:

```
<b metal:use-macro="container/master.html/macros/sidebar">
  <p metal:fill-slot="additional_info">
    Please visit our <a href="/labs">labs</a>.
  </p>
</b>.
```

When rendering this template the side bar will include the extra information that was provided in the slot:

```
<div>
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/research">Research</a></li>
    <li><a href="/projetcs">Projects</a></li>
    <li><a href="/contact">Contact Us</a></li>
  </ul>
  <p>
    Please visit our <a href="/labs">labs</a>.
  </p>
</div>.
```

A common use of slot is to provide default presentation, which can be customized. In the slot example in the last section, the slot definition was just an empty span element. However, default presentation can be provided in a slot definition. For example, consider this revised sidebar macro:

```
<div metal:define-macro="sidebar">
  <div metal:define-slot="links">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/research">Research</a></li>
    <li><a href="/projects">Projects</a></li>
    <li><a href="/contact">Contact Us</a></li>
  </ul>
  </div>
  <span metal:define-slot="additional_info"></span>
</div>.
```

Now the sidebar is fully customizable. The `links` slot could be filled to redefine the sidebar links. However, if one chooses not to fill that slot then one will get the default links, which appear inside the slot.

This technique can be taken further by defining slots inside of slots. Following is a sidebar macro that defines slots within slots:

```
<div metal:define-macro="sidebar">
  <div metal:define-slot="links">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Products</a></li>
    <li><a href="/support">Support</a></li>
    <li><a href="/contact">Contact Us</a></li>
    <span metal:define-slot="additional_links"></span>
  </ul>
  </div>
  <span metal:define-slot="additional_info"></span>
</div>.
```

If one wishes to customize the sidebar links one could either fill the `links` slot to completely override the links, or one could fill the `additional_links` slot to insert some extra links after the default links. Slots can be nested as deeply as desired.

Whole page macros. Macros can also be used to define entire pages. Here's an example macro that defines an entire page:

```
<html metal:define-macro="page">
  <head>
    <title tal:content="here/title">The title</title>
  </head>

  <body>
    <p metal:define-slot="body">
      This is the body.
    </p>

    <span metal:define-slot="footer">
      <p>Copyright 2003 Rima Gerhard</p>
    </span>

  </body>
</html>.
```

The above macro defines a page with two slots, body, and footer. This macro can be used in templates for different types of content, or different parts of a site. For example here is how a template for research items might use this macro:

```
<html metal:use-macro="container/master.html/macros/page">
  <p metal:fill-slot="body">
    This page contains information about research conducted here.
  </p>
</html>.
```

The above template redefines the body slot to contain information about a research project.

The powerful thing about this approach is that one can now change the page macro and the research item template will be automatically updated. For example one could put the body of the page in a table and add a sidebar on the left and the research item template would automatically use these new presentation elements.

This is a much more flexible solution to control page look and feel than the DTML standard_html_header and standard_html_footer solution.

3.7.4 DTML vs. ZPT

DTML can do things that Page Templates cannot, such as dynamically generate email messages (Page Templates can only generate HTML and XML), and so DTML is not a "dead end." The Zope Corporation website states, however, that it is probable that Page Templates will be used for almost all HTML/XML presentation by Zope Corporation and many members of the Zope community have abandoned DTML to work with ZPT instead. The Zope mailing list agreed that in Zope 3, which will be the upcoming Zope release, DTML is only used in areas where an XML-like language is not necessary, such as ZSQL and Mail. The ZPT concept of putting logic in tag-attributes is such a good idea that it can be found in Java tag-libs and in the .NET framework. The language chosen for this project is therefore ZPT.

3.8 Connecting to a relational database with Zope

3.8.1 Difference between ZODB and a relational database

The Zope Object Database (ZODB) is used to store all the pages, files, and other objects users create. It is fast and requires almost no setting up or maintenance. Like a file system, it is especially good at storing moderately sized binary objects such as graphics.

Relational Databases work in a very different way. They are based on tables of data. Information in the table is stored in rows. The table's column layout is called the *schema*. A standard language, called the Structured Query Language (SQL) is used to query and change tables in relational databases.

Relational databases and object databases are very different and each possesses its own strengths and weaknesses. Zope allows users to use either, providing the flexibility to choose the storage mechanism, which is best for the data. The most common reasons

to use relational databases are to access an existing database or to share data with other applications. Most programming languages and thousands of software products work with relational databases.

By using relational data with Zope we can retain all of Zope's benefits including security, dynamic presentation, and networking. The database we have used in our implementation is Oracle 9i. Oracle is arguably the most powerful and popular commercial relational database.

A database can only be used if a Zope Database Adapter is available. We have used the DCOracle2 package from the Zope Corporation [KRO03].

3.8.2 Z SQL Methods

Z SQL Methods are Zope objects that execute SQL code through a Database Connection. Z SQL Methods can both query and change database data. Z SQL Methods can also contain more than one SQL command.

A ZSQL Method has two functions: it generates SQL to send to the database and it converts the response from the database into an object.

The characteristics of ZSQL methods are as follows: Generated SQL will take care of special characters that may need to be quoted or removed from the query. This speeds up code development. Results from the query are packaged into an easy to use object, which will make display or processing of the response very simple. Transactions are mediated (Transactions are discussed in more detail in Section 3.8.4).

Examples of ZSQL methods. ZSQL methods can be used to query, update, or delete from the database. Examples of these operations are ZSQL methods containing the following body:

Q1(“list_all_research”): `select * from Research`

Q2(“insert_research”):
```
insert into Research (res_id, area, title,
lab_id) values
(<dtml-sqlvar res_id type="int">,
<dtml-sqlvar area type="string">,
<dtml-sqlvar title type="string">,
<dtml-sqlvar lab_id type="int">
)
```

Q3(“delete_research”): `delete from Research where res_id = <dtml-sqlvar res_id>`

Queries Q2 and Q3 take arguments. Just like Scripts, Z SQL Methods can take arguments. Arguments are used to construct SQL statements. Q2 and Q3 contain DTML that is evaluated when the method is called. This DTML can be used to modify the SQL code that is executed by the relational database.

One way of providing the arguments is through calling a Z SQL Method without arguments from DTML or ZPT; the arguments are then automatically collected from the REQUEST. The other way is that Z SQL Methods can also be called with explicit arguments from DTML or Python (e.g., `<dtml-in expr="research_by_id(res_id=42)">`). Yet another possibility of getting arguments is through Zope’s mechanism of acquisition.

Querying a relational database returns a sequence of results. The items in the sequence are called *result rows*. SQL query results are always a sequence. Even if the SQL query returns only one row, that row is the only item contained in a list of results.

Somewhat predictably, as Zope is object oriented, SQL methods return a *Result object*. All the result rows are packaged up into one object. For all practical purposes, the result object can be thought of as rows in the database table that have been turned into Zope objects. These objects have attributes that match the schema of the database result.

Result objects can be used from DTML to display the results of calling a Z SQL Method. For example, one might add a new DTML Method called `listResearch` with the following DTML content:

```
<dtml-in list_all_research>
<li><dtml-var res_id>: <dtml-var title>, in the area of
                       <dtml-var area> has a lab with ID
   <dtml-var lab_id>
</li>
</dtml-in>.
```

This method calls the `list_all_research` Z SQL Method from DTML. The *in* tag is used to iterate over each Result object returned by the `list_all_research` Z SQL Method.

The body of the *in* tag is a template that defines what gets rendered for each Result object in the sequence returned by *list_all_research*. In the case of a table with three research fields in it, *listResearch* might return HTML that looks like this, where the *in* tag rendered an HTML list item for each Result object returned by `list_all_research`.

```
<li>1: Spatio-temporal data modeling, in the Area of Database
       and Information Systems has a lab with ID 3.
</li>
<li>2: Knowledge Management, in the Area of Database and
       Information Systems has a lab with ID 4.
</li>
<li>3: Data Warehousing, in the Area of Database and
       Information Systems has a lab with ID 4.
</li>
```

An important difference between result objects and other Zope objects is that result objects are not persistent. They do not get created and permanently added to Zope. They exist for only a short period of time; just long enough for you to use them in a result page or to use their data for some other purpose. As soon as one is done with a request that uses result objects they go away, and the next time one calls a Z SQL Method one gets a new set of fresh result objects.

In summary, the concept of dividing the database access from the HTML design template as declared in ZPT or DTML allows us to neatly differentiate between style and content, one of the goals of website development as declared in Chapter 2.

3.8.3 Caching

Another functionality Zope provides in the area of relational database connectivity is caching results. Users can increase the performance of SQL queries with caching. Caching stores Z SQL Method results so that if users call the same method with the same arguments frequently, they will not have to connect to the database every time. Depending on the application, performance can be improved through caching.

Zope users can manually edit the *Maximum number of rows received* field, which controls how much data to cache for each query. The *Maximum number of results to cache* field controls how many queries to cache. The *Maximum time (in seconds) to cache results* controls how long cached queries are saved for. In general, the larger these values are set, the greater is the performance increase, but the more memory Zope will consume.

In general, it is advised to set the maximum results to cache to just high enough and the maximum time to cache to be just long enough for an application. For site with few hits one should cache results for longer, and for sites with lots of hits one should cache results for a shorter period of time. For machines with lots of memory one should increase the number of cached results. To disable caching one needs to set the cache time to zero seconds. For most queries, the default value of 1000 for the maximum number of rows retrieved will be adequate. For extremely large queries one may have to increase this number to retrieve all results.

3.8.4 Transactions

A transaction is a group of operations that can be undone all at once. As was mentioned in Section 3.4, all changes done to Zope are done within transactions. Transactions ensure data integrity. When using a system that is not transactional and one web actions changes ten objects, and then fails to change the eleventh, then data is now inconsistent. Transactions allow us to revert all the changes made during a request if an error occurs.

Most commercial and open source relational databases support transactions. If the relational database supports transactions, Zope will make sure that they are tied to Zope transactions. This ensures data integrity across both Zope and the relational database.

It is guaranteed that operations in this transaction are either all performed or none are performed even if these operations use a mix of Zope Object Database and external relational database.

CHAPTER 4
REQUIREMENTS ANALYSIS

In this chapter, project-relevant information gathered during an analysis phase is gathered.

4.1 Functional Requirements

The functional requirements for this system are:

4. The new website should be based on widely spread and reliable web techniques such that most browsers can access the site.
5. The new website embedded in the CMS must adhere to the COE and UF Webadmin standards.
6. The data and the graphical user interface should be separated. This way, alternative front ends could be used and the DBMS could be exchanged.
7. Faculty members and other appointed staff should have the option to access certain areas of the site and create content.
8. Administrative tasks such as adapting the design and content and updating and maintaining the database shall be possible without further knowledge of a markup- or programming language.
9. The database should be accessible over an easy-to-use graphical interface, providing forms for adding, updating, and deleting data in the database tables. Furthermore, there shall be forms for creating new tables in table.
10. The database transactions must be secure and the application should have multi-user functionality.
11. The application should be easily extendible (e.g., adding a calendar of events, a forum, etc.)

The list of requirements proves that Zope is a good fit: as far as the first functional requirement is concerned, Zope allows for any kind of HTML code and client-side scripting languages (e.g., JavaScript, CSS, Frames, etc.). All static HTML is rendered as usual, Zope simply interprets any existing DTML code or Python Scripts and inserts the result in the HTML page.

The third requirement is fulfilled within Zope through the use of style sheets and DTML methods or ZPT macros that can be included to build a page. The fourth

requirement is also simplified with Zope since comes with a set of elaborate authentication and authorization mechanisms. Authentication means finding out who someone is, and authorization means determining what that someone can do. Zope provides separate facilities to manage the processes of identifying users and granting access to controlled actions. Zope is also able to handle transactions, which ensure data integrity. Zope transaction boundaries are demarcated by a single request. Zope will integrate with third party RDBMS that support transactions, by acting as a transaction authority via a two-phase-commit protocol, such that if Zope's transaction succeeds the RDBMS will commit.

As far as requirement 8 is concerned, the extensibility of Zope is obvious since it offers a number of prefabricated extensions on the website. We have tested some of the available products and have found the installation process straightforward. This said, problems were encountered but the Zope mailing lists have always provided immediate feedback and help.

We note that the functional requirements to not impose one single DBMS to use (Zope can be set up with Oracle, MySQL, postreSQL and many more). We could have therefore made the decision to store all data within the Z Object Database that comes with Zope. However, requirement 3 dictates that an RDBMS is preferable in case the DBMS is exchanged.

4.2 Non-Functional Requirements

1. A flexible database schema that allows for further changes and additions.
2. Consideration of data security.
3. Clear, and simple user interface for the maintenance staff.
4. Platform-Independence: the application should be available under any operating system, on any machine.

CISE staff assigned with maintaining the CISE website are not necessarily a professional in the area of PCs and the Internet. Therefore, the GUI should have a simple, and clear layout. Since the application is web based, pure HTML is preferable to many graphical elements and strong usage of JavaScript. This is also important when we consider that the application should be compatible to all browser types (requirement 4). It might be the case that the maintenance person uses uncommon browsers (such as Mozilla or Opera or even old versions of Netscape or the Internet Explorer) or has strong security settings that disable Java/JavaScript/Cookies.

CHAPTER 5
IMPLEMENTATION

5.1 Software and Hardware Specifications

This Chapter details the specific hardware and software used for the implementation. First, the hardware is listed. Then, the software needed for the project, especially some add-on Zope products, is given.

The development environment is the rain server of the CISE Department. It was manufactured by Sun Microsystems and the operating system distribution is Solaris 8 SPARC.

Additionally, we are using the "orcl" Oracle 9i database instance. We have migrated to this instance after the department decided to deprecate the old "oradb" 8i instance. Zope resides in the research disk space /cise/research/SpaceTimeUncertainty/workspace/rima/. The version of Zope used for this project is Zope 2.6.0 Solaris 2.8 sparc. This was the current stable release at the time the project started. Ever since, a new version of Zope (2.6.1) can be downloaded from the Zope website [ZOP03a].

As far as Zope specific software is concerned (e.g., products), the following installations were added to the basic Zope installation:

- To connect to the Oracle database, it was necessary to install a database adaptor. The appropriate Oracle database adapter is DCOracle2 [KRO03].
- The Photo Product [BIC03] was installed in order to be able to upload pictures and have certain photo versions created automatically. The Photo Product creates several version of an image, e.g. a thumbnail version, or an xsmall/small/medium/large/xlarge version.

- To use the Photo product, the Image Magick [IMA03] had to be installed as well. ImageMagick is a set of tools and libraries to read, write, and manipulate an image in many image formats including popular formats like TIFF, JPEG, PNG, PDF, PhotoCD, and GIF. ImageMagick can resize, rotate, sharpen, color reduce, or add special effects to an image. Version 5.5.7 was used.
- The ZStyleSheet Product was installed and tested [HAQ03]. It was used throughout the dtml-based implementation of the project, but after the switch to ZPT, its use was discontinued. However, the product still resides in Zope and could be used for future projects.

All products except for Image Magick mentioned above reside in the /lib/python/Products directory of the Zope installation.

Image Magick is installed under /cise/research/SpaceTimeUncertainty/workspace/rima.

Zope comes with its own web server (as described in Chapter 3). Currently, this is the web server used for the website. It is listening to port 8080 on the rain server. Zope can also be set up to listen to the standard port 80.

5.2 Database Design

The database ER diagram is shown in Figure 5-1.

Not all of the entities identified in Figure 5-1 have been adapted for this project. The description of the chosen, website-relevant entities follows in alphabetical order.

Administrative. This table stores information about employees of the CISE Department that work in the administrative field. This encompasses employees such as the secretaries, the graduate und undergraduate academic advisors, the graduate senior clerk as well as the graduate program assistant. A field in this table stores whether the employee is working with undergraduate students, graduate students, both, or none. The title field indicates the job title of the employee.

Committees. The Committees table stores all departmental committees. The only information stored about a committee is its title and its email address, in case it exists (this field could be empty).

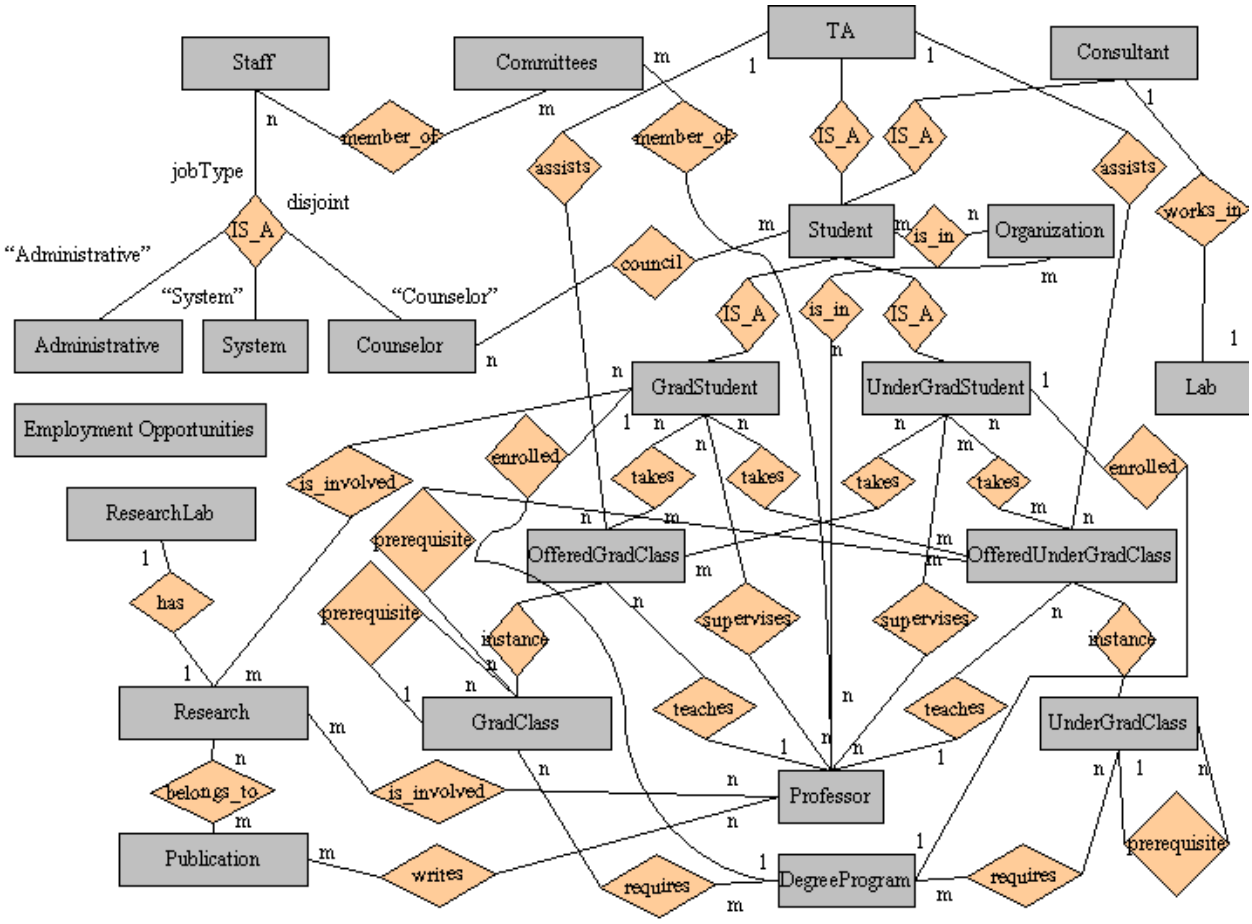


Figure 5-1 ER diagram of CISE Department

Event. Any upcoming event is stored in this table. Information about the presenter, their name, affiliation, as well as about the event itself (such as time, location, and content) are the attributes stored for an event. An event could be associated with up to one Professor, but it might not have any faculty involved with it.

GradClass. The GradClass table stores general information about the graduate classes offered at CISE. It stores the title, course number, a description, the number of credit hours the class is worth, and whether the class is a core class for the CISE

curriculum. The description also contains information about any prerequisites. The prerequisites might be other classes (undergraduate or graduate) or general prerequisites such as “knowledge in the field of numerical analysis” or “proficiency in at least one programming language.”

OfferedGradClass. An offered graduate class stores information related to the specific instance of the graduate class. This includes the year, the semester, and a link to a web page, the class period and the room number. All of these fields could be left empty, in case the information does not exist. An offered graduate class is an instance of exactly one graduate class. Exactly one Professor can be teaching the class.

OfferedUnderGradClass. Parallel to the offered graduate class, an offered undergraduate class stores information related to the specific instance of the undergraduate class. The year, the semester, a link to a web page, the class period, and the room number are stored in case they exist. An offered undergraduate class is an instance of exactly one undergraduate class. Exactly one Professor can be teaching the class.

Organizations. The Organizations table stores information about the CISE related student organizations that operate at the University of Florida. The title, a link to their web page in case they have one, as well as a short description is stored for each student organization.

Overall Links. This table records all links to outside web pages. Outside web pages are all web pages that are not stored on the CISE web server. For every such page, the link itself, the title, and the area that the link belongs to, are recorded.

ProfCommittee. This table stores the relationship between a professor and a committee, i.e., their primary keys will appear as a tuple in this table when a professor is involved in a departmental committee. Additionally, a professor could hold a specific position in that committee (e.g., “chair”).

Professor. The Professor table stores information about every professor teaching, researching, or otherwise active in the CISE Department. In addition to the personal and contact information such as their full name, office number, telephone number, email address, and degree, professors can enter details about their interests, which could be current or past research interests. The degree can be anything between “M.S.” or detailed information such as “Ph.D., University of Pennsylvania, 1986.” The title field holds the job title the professor holds at the CISE Department, e.g. “Assistant Professor,” “Lecturer,” etc. Finally, each professor has a password. This password is used by the professor to log in to the administrative area of the website and to update information about himself, his involvement in research and committees or the graduate/undergraduate classes that he is teaching.

ProfResearch. Similarly to ProfCommittee, the ProfResearch table stores the relationship between a professor and a research area, i.e., their primary keys will appear as a tuple in this table when a professor is involved in a research area.

Project. The Projects table stores all research projects that are currently active or have been active at the CISE Department. The data stored about a Project are the title, a link to a web page if existent, and a short description.

Research. This table stores the research areas of the CISE Department. A research area consists of its research area name, a corresponding research lab if it exists, a link to a web page if one exists, and a short description.

ResearchLab. This table stores the research labs and centers of the CISE Department. Information stored for a lab is the title, the link to a web page if there is one, and a descriptive text. A research lab can have one professor associated with it (the director of the research lab/center).

Seminar. A seminar is a club or a group meeting taking place at regular intervals at the CISE Department. One Professor can be associated with it; that professor then is the coordinator of the seminar.

Staff. The Staff table stores all employees of the CISE Department, whether they are work in the administrative area or the system administartion area.

System. This table stores information about employees of the CISE Department that work in the system administration field. This encompasses employees such as the senior system programmer and the system maintenance staff. The title field indicates the job title of the employee.

UnderGradClass. Similarly to the GradClass table, the UnderGradClass table stores general information about the undergraduate classes offered at CISE. It stores the title, course number, a description, the number of credit hours the class is worth, and whether the class is a prerequisite for a graduate class of the CISE curriculum. The description also contains information about any prerequisites for this class. The prerequisites might be other classes (undergraduate or graduate) or general prerequisites.

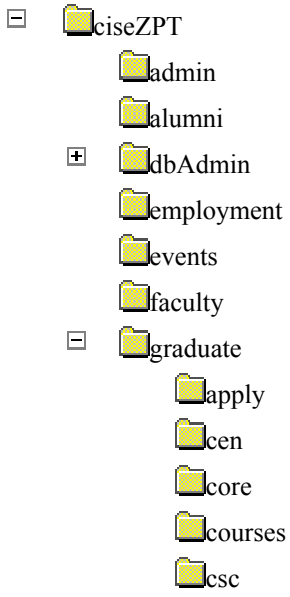
5.3 Site Architecture

5.3.1 Folder Hierarchy

One important step in the implementation of the website was to create the folder hierarchy. Since Zope maps the folder hierarchy directly to URL paths (see Chapter 3.3 for more information on this mapping is done), the hierarchy needed to be logical and also related to the schema of the underlying database. For the CISE website, the hierarchy of folders is as shown in figure 5-2 (only the first three levels are shown).

Not visible in the site architecture figure below are the other folders located inside Zope's root folder. The root folder holds, among others, the Control Panel and acl_users (included by default), and the images, sql, and ciseZPT folders (included for this project). The images folder holds all images displayed throughout the site. The sql folder holds all ZSQL methods needed throughout the project.

The site itself resides in the ciseZPT folder, which is in turn located inside Zope's root folder. To view the site, the required URL is http://rain.cise.ufl.edu:8080/ciseZPT.



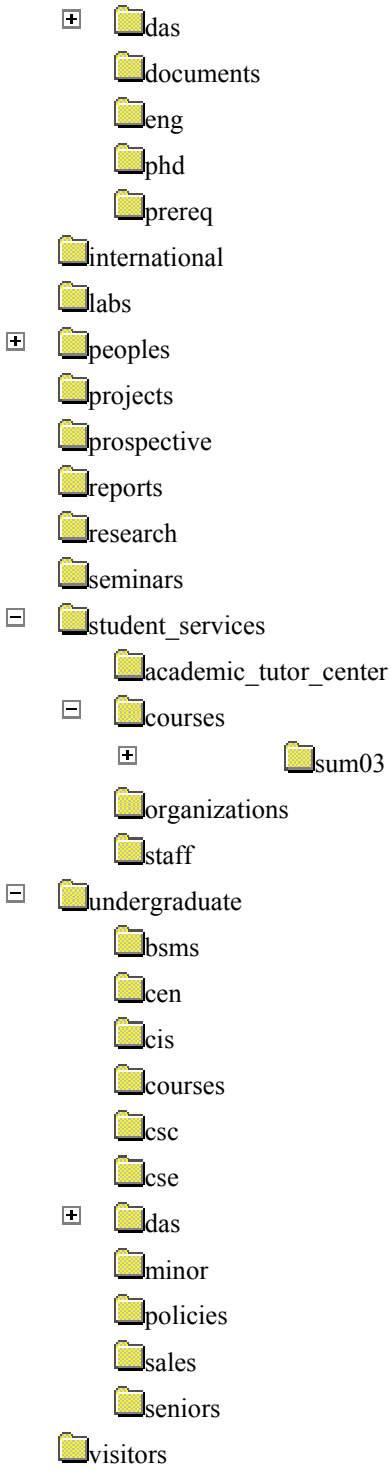


Figure 5-2 Folder Hierarchy of website

5.3.2 Folder Properties

Every folder in figure 5-2 has a specific function. The function is indicated by the folder's properties. As in common in object-oriented programming, every object in Zope

has a set of properties. Some properties are included by default, e.g. every object has a title, which could however be empty. The following properties have been defined in addition to the default folder properties:

- QuickLinks : ever folder that should be listed in the menu “QuickLinks” on the main CISE web page has the QuickLinks property (and it is set to the value “true”). This is the case for the folders with the IDs: employment, student_services, peoples, and events.
- Programs: : every folder that is included in the menu “Programs” on the main CISE web page has the Programs property (and it is set to the value “true”). This is the case for the following folders: graduate, undergraduate, faculty, visitors, and prospective.
- InformationFor: : ever folder that should be listed in the menu “Information For” on the main CISE web page has the InformationFor property (and it is set to the value “true”). This is the case for the folders with the Ids: international, seminars, reports, projects, labs, and research.

The idea behind adding properties to some folders is that a Python script can be used to check all folders and return only the ones with the given property set. When a new folder is added to the Zope hierarchy, it can be made invisible to website visitors by not setting any properties. The Python script would then not return that particular folder. It will be shown on the website, however, when one of the above properties is added to the folder. This is convenient because a web developer doesn’t have to manually include a link to a new folder (i.e., are of the site) in every HTML source code where the area is supposed to be added, but only has to set a property once.

The Python script used to return folders with a given property set is shown in Code Listing 5-3.

```
#this python script returns all folders situated in the root folder
#that have a String property with a value of the parameter "property"
#it does not return the calling folder itself

#this script check the property
def okToList(obj):
      if (hasattr(obj.aq_explicit,property) and
          (obj.getId()!='private')):
        return 1
```

```
#this script gets all the folders in the root folder
def list_top_folders():
      # Find folders in the root folder with a menu_entry property
      objects=filter(okToList,context.getSiteRoot().objectValues())

      return objects


# Get list of top-level folders
folder_list=list_top_folders()

#construct dictionnary
out = [ {'id':obj.getId(),
             'obj':obj,
             'url':obj.absolute_url(),
             'title':obj.title_or_id() }
             for obj in folder_list ]
return out
```

Code Listing 5-3 Python Script getFolders

When testing the Python Script with, for example, the property "Programs," the script returns the following dictionary:

```
[{'title': 'Research Areas', 'id': 'research', 'obj': <Folder instance at
fc0c28>, 'url': 'http://rain.cise.ufl.edu:8080/ciseZPT/research'}, {'title':
'Centers and Labs', 'id': 'labs', 'obj': <Folder instance at f6e940>, 'url':
'http://rain.cise.ufl.edu:8080/ciseZPT/labs'}, {'title': 'Projects', 'id':
'projects', 'obj': <Folder instance at fc1360>, 'url':
'http://rain.cise.ufl.edu:8080/ciseZPT/projects'}, {'title': 'Technical
Reports', 'id': 'reports', 'obj': <Folder instance at fc0ec0>, 'url':
'http://rain.cise.ufl.edu:8080/ciseZPT/reports'}, {'title': 'Seminars', 'id':
'seminars', 'obj': <Folder instance at fc0de0>, 'url':
'http://rain.cise.ufl.edu:8080/ciseZPT/seminars'}, {'title': 'International',
'id': 'international', 'obj': <Folder instance at f6e5f0>, 'url':
'http://rain.cise.ufl.edu:8080/ciseZPT/international'}].
```

As the example shows, every object has an absolute_url associated with its location within the Zope hierarchy. When the object moves to a different location, the abolute_url changes accordingly. This is convenient for web developers, since the link to a location does not have to be hard coded any more. One would simply insert a link to the abolute_url of an object, and the correct location would we resolved by Zope.

Also, the above example shows that folders have an "id" as well as a "title." The id could be short and could be not descriptive enough. Therefore, the title property was added whenever the id itself was not sufficient. This is for example the case for the research folder. The additional title "Research Areas" was added to clarify what type of information can be found in the folder.

The following code listing illustrates the use of the python script from within a page template. The HTML code defines a table. The Python script getFolders is called to fill the rows of the table with folders that have the QuickLinks property set (see Code Listing 5-4).

```
<TABLE WIDTH=160 BORDER=0 CELLPADDING=0 CELLSPACING=0>
  <TR>
    <TD CLASS="menu-section-title" COLSPAN=2 bgcolor="#6698CB">
    <P>Quick Links
    </TD>
  </TR>
  <TR>
    <TD WIDTH=5></TD>
    <TD CLASS="menu-links">
    <div tal:repeat="item python:here.getFolders('QuickLinks')">
    <div align="left" class="menu"
     tal:condition="python:here!=item['obj'] or
     template.id!='index_html'">
    <a href="link url" tal:attributes="href item/url"
     tal:content="item/title">link to information folders</a>
    </div>
    </div>
    </TD>
  </TR>
</TABLE>
```

Code Listing 5-4 Usage of getFolders from Page Template

When Code Listing 5-4 is viewed through a web browser, Zope interprets the Python script and generates the HTML code in Code Listing 5-5.

```
<table width="160" border="0" cellpadding="0" cellspacing="0">
  <tr>
    <td class="menu-section-title" colspan="2" bgcolor="#6698CB"><p>Quick
Links</p>
    </td>
  </tr>
  <tr>
    <td width="5"></td>
    <td class="menu-links">
    <div>
      <div align="left" class="menu">
      <a href="http://rain.cise.ufl.edu:8080/ciseZPT/employment">Employment
       Opportunities</a>
      </div>
      <div align="left" class="menu">
      <a href="http://rain.cise.ufl.edu:8080/ciseZPT/student_services">Student
Services
       Center</a>
      </div>
      <div align="left" class="menu">
      <a href="http://rain.cise.ufl.edu:8080/ciseZPT/peoples">CISE Faculty
&amp;
```

```
       Staff</a>
      </div>
      <div align="left" class="menu">
      <a href="http://rain.cise.ufl.edu:8080/ciseZPT/events">Events</a>
      </div>
    </div>
    </td>
  </tr>
</table>
```

Code Listing 5-5 Generated Code from the getFolders script

This concept of giving properties to folders has been applied throughout the implementation of the website. Whenever there had to be a difference between visible (to website visitors) and internal folders, they have been marked with a property, which is then checked by the getFolders Python Script before the folder is displayed.

Another script that deals with folder properties is the breadcrumbs script. In the top area of the site, a breadcrumbs navigation shows the current location in the site. It walks the REQUEST.PARENTS list of parents and creates a hyperlink for each parent. It stops at the ciseZPT folder. If the folder has a nickname property, the nickname will be displayed. Otherwise, it will display the Id of the Folder. The script used to generate the links follows:

```
links=[]
for parent in context.REQUEST.PARENTS[:-1]:
    if parent.hasProperty('nickname'):
        links.insert(0, """<a href="%s">%s</a>""" % (parent.absolute_url(),
parent.getProperty('nickname')))
    else:
        links.insert(0, """<a href="%s">%s</a>""" % (parent.absolute_url(),
parent.getId()))
    if parent.getId() == "ciseZPT":
        break
return "/".join(links).
```

5.4 Definition Of The Website Style

The User Interface is largely dictated by the guidelines from the COE (as outlined in Section 2.1.2), leaving almost no creativity of design to the author. In addition, the look of the new version of the CISE has been adapted. This decision has been made

because a totally new look would have been confusing to users who are by now familiar with the “new” CISE website.

The use of Macros has been explained in Chapter 3.7.2 (Zope Page Templates). The CISE website has been created to use one general design template. Code Listing 5-6 illustrates how the site has been set up with macros and slots that can be filled with content by other page templates referencing this macro. The page template has been defined in the index_html page template within the ciseZPT folder (i.e., the main page template displayed to visitors of the site).

```
<html metal:define-macro="page" tal:define="global vhroot here/getSiteRoot">
<head>
</head>

<BODY TOPMARGIN=0 RIGHTMARGIN=0 BOTTOMMARGIN=0 LEFTMARGIN=0 MARGINWIDTH=0
MARGINHEIGHT=0>

   <!-- begin "upper strip" -->
   <div metal:define-macro="upper_strip">
   ...
   </div>
   <!-- end "upper strip" -->

   <!-- begin "large strip" -->
   <div metal:define-macro="large_strip">
   ...
   </div>
   <!-- end "large strip" -->


  <!-- begin "navbar" -->
  <div metal:define-macro="navbar">
  ...
  </div>
  <!-- end "navbar" -->
  <span metal:define-macro="head_leftmenu">
  ...
  <!--the head_leftmenu macro ends here-->
  </span>



<!-- start macro content_area-->
<div metal:define-macro="content_area">

   <!-- start slot welcome_strip-->
   <div metal:define-slot="welcome_strip">
   <!-- end slot welcome_strip-->
   </div>
```

```
    <!-- start slot body-->
    <div metal:define-slot="body">
    ...
    <!-- end slot body-->
    </div>

    <!-- start slot body_title-->
    <div metal:define-slot="body_message">
    ...
    <!-- end slot body_message-->
    </div>


  <div metal:define-slot="events">
  <!-- start slot more_events_title-->
  <div metal:define-slot="more_events_title">
  <!-- end slot more_events_title-->
  </div>
  <!-- start slot more_events_text-->
  <div metal:define-slot="more_events_text">
  <!-- end slot more_events_text-->
  </div>

  <!-- end slot events-->
  </div>


<!--end of content_area macro -->
</div>


<!-- begin macro footer -->
<div metal:define-macro="footer">
...
<!-- end macro footer -->
</div>

</body>
```

Code Listing 5-6 The index_html page macro

The Code Listing in 5-6 only shows the logical slot and macro layout of the site. The actual look (i.e., the HTML code part) has been removed.

When reusing the whole page macro defined above, one simply needs to reference it in the page template using it. For example, consider the research folder. When displaying information about the research areas active at the CISE Department, we want to keep the same look and feel as the main CISE web page. Code Listing 5-7 shows how this is achieved.

```
<html metal:use-macro="root/ciseZPT/index_html/macros/page">
  <div metal:fill-slot="welcome_strip">
```

```
  </div>

  <div metal:fill-slot="body">
  Research related content will be displayed on this page.
  </div>

  <div metal:fill-slot="body_title">
  </div>

  <div metal:fill-slot="body_message">
  </div>

  <div metal:fill-slot="events">
  </div>
</html>
```

Code Listing 5-7 Using the whole page macro in another page

The first line indicates which macro shall be used. By filling certain slots, we override the code that would otherwise be inserted by the page macro. In the research folder, for example, we do not wish to show the welcome_strip image that can be seen on the main page. Therefore, in the slot "welcome_strip," we replace the content. The same counts for the "body_title," "body_message," and "events" slot. By filling them with basically nothing, we override the content. The "body" content, however, has been filled to contain the line "`Research related content will be displayed on this page.`"

When displaying this web page in a browser, the look and feel of the main page has been reused consistently. The convenience of this approach is that web developers do not have to retype whole chunks of HTML just to get the same look and feel on different pages. All one has to do is reuse the macro and fill slots that change from one page to another.

The `root/ciseZPT/index_html/macros/page` whole page macro is the one mostly used throughout all CISE web pages. Only the undergraduate students are has been modified to use a slightly different macro. The undergraduate macro differs from the traditional `root/ciseZPT/index_html/macros/page` whole page macro by adding an extra notice at the end of each page:

```
<div metal:fill-slot="body_message">
<table border="0" cellpadding="2" cellspacing=0 width="90%" align="center"
valign="center">
<tr>
   <br>
   <td class=common align=center>Interested students should contact a <a
href="link" tal:attributes="href
string:${vhroot/absolute_url}/student_services/staff/">CISE Academic
Advisor</a> or visit the</td>
</tr>
<tr>
   <td class=common align=center><a href="link" tal:attributes="href
string:${vhroot/absolute_url}/student_services">CISE Student Services
Center</a><br>
                E405 CSE Building
   </td>
</tr>

</table>

</div>.
```

This extra bit of information is found on most pages in the undergraduate area (in particular on all pages containing information about degree programs). The pages that use this macro contain the following line

```
<html metal:use-macro="root/ciseZPT/undergraduate/index_html/macros/undergrad">
```

rather than the normal macro. The slot that holds the main body content has been renamed to "undergrad_body," so the pages include the following line where body content is inserted:

```
<div metal:fill-slot="undergrad_body">.
```

5.5 SQL Queries

SQL queries serve the following purposes on the site:

- Display information: SQL queries are called to query the database. The result is embedded in HTML code for direct display on the site. These queries are typically of the structure `SELECT * FROM …`
- Update information: SQL queries are called to update values stored in the database. These queries usually are of the `UPDATE Professor SET … WHERE …` format
- Add information: SQL methods insert data in the database (`INSERT INTO Professor(…) VALUES …`)
- Delete information: When data has become obsolete, SQL queries are called to delete it (`DELETE FROM Professor WHERE …`)

Most of the queries of the first category reside in sql folder inside Zope's root folder. The reason behind this choice is that the queries are frequently reused in different parts (and thus, folders) of the web page. Due to Zope's Acquisition mechanism (see Section 3.3), the ZSQL methods are inherited in any subfolder of the root folder.

Queries of type 2, 3, and 4 are locally declared in the folder where they are needed. The maintenance of the database is entirely residing in the dbAmin folder located inside the ciseZPT folder. Every table in the database is represented with a folder, which holds all operations that can be performed on a table. Since the updating, adding, and deleting differ from one table to another, the corresponding SQL queries are stored locally within the folder for the table, rather than in a central place such as the sql folder.

```
<table border="0" cellpadding="2" cellspacing=0 width="90%" align=center>
  <tr>
    <td class=topics bgcolor="#0D1958" width="35%">Name
    </td>
    <td class=topics bgcolor="#0D1958" width="20%" align=center>Phone
    </td>
    <td class=topics bgcolor="#0D1958" width="20%" align=center>Office
    </td>
  </tr>
  <span tal:repeat="in root/sql/allAdministrativeStaff">
  <tr tal:define="odd repeat/in/odd" tal:attributes="bgcolor python:test(odd,
'white',
    '#CDD4DE')">
    <td class=common width="35%">
    <a href="link" tal:attributes="href
     string:displayStaff?Staff_ID=${in/staff_id}"><span
tal:replace="in/firstname"/>
    <span tal:content="in/lastname">Beck, Sullivan</span></a></span></td>
    <td class=common width="20%" align=center tal:content="in/telephonenumber">
(352)
     392-1057 </td>
    <td class=common width="20%" align=center tal:content="in/officenumber">
314E CSE
    </td>
  </tr>
  </span>

</table>
```

Code Listing 5-8 Calling a ZSQL method in embedded HTML

Queries of type 1 are usually included in a format resembling the one in Code Listing 5-8, taken from the index_html page template of the ciseZPT/peoples/office folder.

Code Listing 5-8 illustrates how the ZSQL method is called from within a page template. The ZSQL method is called “allAdministrativeStaff” and is one of the methods that is located in the root/sql folder. To get a good looking result web page, the result rows are alternating in color. This is achieved through a call to the Python “test” method. The “odd” repeat variable is true for odd-indexed repetitions (1, 3, 5, ...). Whenever odd is true (i.e., for every other row), the background color attribute is set to “white,” otherwise, the color is “#CDD4DE”.

Another SELECT query frequently used throughout the site is the ZSQL method selectOverallLinksByTitle. All links that lead to external pages are stored in the OverallLinks table of the database. Whenever the link is used on the site, the link is not hard coded as in traditional HTML pages. The problem with hard-coded links is that they aren’t guaranteed to be consistent. A link can appear at several areas throughout the site. The idea was to retrieve the actual link from the database whenever it is used, so that only one link has to be kept updated (which is the one in the database). As long as the database is kept current, all links on the website will be up-to-date as well. This was implemented with the following tal statement:

```
<span tal:repeat="in python:root.selectOverallLinksByTitle(title='XXX')"><a
href="link" tal:attributes="href in/linktowebpage">XXX</a></span>.
```

“XXX” in the above code snippet would in reality be the name of the link that is to be retrieved (e.g., “Undergraduate Catalogs,” “College of Engineering,” “Critical Dates,” etc).

Queries of type 2, 3, and 4 are further explained in the Section 5.6.1.

5.6 User Groups

5.6.1 Authentication to the dbAdmin area

The database administration can be performed by a set of people. The first user group that has the privilege to perform database operations are the Database Administrators, who have the right to modify every table in the database. The second group of people who are allowed to perform some database administration are the professors of the CISE Department. In the following section, we explain what action can be performed by whom.

Whenever database operations are performed, the person wishing to perform the update has to point their browser to the /ciseZPT/dbAdmin (http://rain.cise.ufl.edu:8080/ciseZPT/dbAdmin) page, which is not linked from the CISE main page since the idea is that only a certain group of people is allowed to perform database operations.

As soon as the page is being accessed, Zope requires the user to log in, since this part of the site has been set to restricted access. This means that for the index_html page of the dbAdmin folder, the “View” permission has been set to allow certain authenticated users only (rather than the default “Anonymous”). Before the index_html page is displayed, Zope prompts the user to log in by presenting an authentication dialog. Once the dialog has been "filled out" and submitted, Zope will look for the user account represented by this set of credentials. Zope *identifies* a user by examining the username and password provided during the entry into the authentication dialog. If Zope finds a user within one of its user databases with the username provided, the user is identified. Once a user has been identified, *authentication* may or may not happen. Authentication

succeeds if the password provided by the user in the dialog matches the password registered for that user in the database.

Different things can happen with respect to being prompted for authentication credentials in response to a request for a protected resource depending on the current state of a login session. If the user has not yet logged in, Zope will prompt the user for a username and password. If the user is logged in but the account under which he is logged in does not have sufficient privilege to perform the action he has requested, Zope will prompt him for a *different* username and password. If he is logged in and the account under which he has logged in *does* have sufficient privileges to perform the requested action, the action will be performed. If a user cannot be authenticated because he provides a nonexistent username or an incorrect password to an existing authentication dialog, Zope re-prompts the user for authentication information as necessary until the user either "gets it right" or gives up.

5.6.2 Professors

Professors an perform some limited database maintenance. To achieve this, we have added a user folder called “acl_users” inside the dbAdmin folder. There is already a user folder in Zope’s root folder called “acl_users” as well, but Zope can contain multiple user folders at different locations in the object database hierarchy. A Zope user cannot access protected resources above the user folder in which their account is defined. The location of a user's account information determines the scope of the user's access. The users defined inside the acl_users folder of dbAdmin may only access protected resources within the dbAdmin folder and within subfolders of that subfolder, and so on.

Zope users have *roles* that define what kinds of actions they can take. Roles define classes of users such as *Manager*, *Anonymous*, and *Authenticated*. These roles are

controlled by the Zope system administrator. Users may have more than one role, and may have a different set of roles in different contexts. Zope objects have permissions which describe what can be done with them such as “View,” “Delete objects,” and “Manage properties.”

Roles are similar to UNIX groups in that they abstract groups of users. And like UNIX groups, each Zope user can have one or more roles. Roles make it easier for administrators to manage security. Instead of forcing an administrator to specifically define the actions allowed by each user in a context, the administrator can define different security policies for different user roles in a context. Since roles are classes of users, he needn't associate the policy directly with a user. Instead, he may associate the policy with one of the user's roles.

Zope comes with four built-in roles:

Manager. This role is used for users who perform standard Zope management functions such as creating and edit Zope folders and documents.

Anonymous. The Zope `Anonymous` user has this role. This role should be authorized to view public resources. In general this role should not be allowed to change Zope objects.

Owner. This role is assigned automatically to users in the context of objects they create.

Authenticated. This role is assigned automatically to users whom have provided valid authentication credentials. This role means that Zope "knows" who a particular user is. When Users are logged in they are considered to also have the Authenticated role, regardless of other roles.

We have defined a global role for Professors called “prof.” Roles can be used at the level at which they are defined and "below" in the object hierarchy. For example, since we created a role in the ciseZPT folder, that role cannot be used outside of the ciseZPT folder and any of its superfolders and super objects.

All Professors of the CISE Department have already been added as users with the role “prof.” The login and password have been set to the current login of a professor. As soon as the professor logs in for the first time, it is advisable that he or she change the password. The login cannot be altered.

The permissions that have been granted to the prof role are very limited. User logged in with the “prof” role cannot add, change, view, or delete any of the objects located in the root folder. This security setting is inherited in all subfolders of the root folder. There are only two exceptions to this rule. The first exception to this rule has been set in the /ciseZPT/peoples/photos folder. Since Professors can upload their own photos, the permissions for Photo objects are not acquired, but explicitly set. For the users with the role “prof” the following actions are allowed: “create class instances,” “delete objects” (since when a new photo is added for a professor, the old one is deleted first), and “add Photos.” The second exception has been set in the /ciseZPT/dbAdmin/acl_users folder. The reason that users with the “prof” role have special permissions in this folder is that Professors are able to change their password over the web interface. When the password changes, the acl_user folder has to be accessed and the password for the user needs to be altered. The corresponding permission is called “Manage users,” which has been enabled for “Manager,” “Owner,” and “prof” roles.

5.6.3 Database Administrator

The admin user has been set up to have the Manager role (explained above). The Manager allows the viewing of all objects as well as their modification and object creation. As far as the maintenance of the database is concerned, the administrator is allowed to perform actions on all tables described in Section 6.2 (i.e., Administrative, Committees, Event, GradClass, OfferedGradClass, OfferedUnderGradClass, Organizations, Overall Links, ProfCommittee, Professor, ProfResearch, Project, Research, ResearchLab, Seminar, Staff, System, and UnderGradClass).

For each of the tables, the database administrator is presented with two choices: he can either view the list of all rows currently stored in a table, or add a new row to the table. When viewing the list of rows, he can either edit or delete one of the rows. When editing a row, the administrator can modify all columns except for the column representing the primary key. When deleting a row, the ZSQL method is setup such that not only the given row is deleted, but also all rows where the primary key of the deleted row appears as a foreign key. When, for example, a professor is deleted, the ZSQL method contains the following delete commands:

```
DELETE FROM ProfResearch WHERE Prof_ID = <dtml-sqlvar Prof_ID type="int">
<dtml-var sql_delimiter>
DELETE FROM ProfCommittee WHERE Prof_ID = <dtml-sqlvar Prof_ID type="int">
<dtml-var sql_delimiter>
DELETE FROM Seminar WHERE Prof_ID = <dtml-sqlvar Prof_ID type="int">
<dtml-var sql_delimiter>
DELETE FROM Professor WHERE Prof_ID = <dtml-sqlvar Prof_ID type="int">.
```

The `<dtml-var sql_delimiter>` can be used to separate several SQL statements in a ZSQL query.

5.6.4 Additional Users and Delegation

That official CISE website is maintained by more than one webmaster. This setup can be achieved easily with Zope, since multiple user folders can be contained at

different locations. A Zope user cannot access protected resources above the user folder in which their account is defined. The location of a user's account information determines the scope of the user's access.

The following scenario illustrates how this would be handled. Consider the case of a user folder at *ciseZPT/research/acl_users*. Suppose the user XY is defined in this user folder. This user cannot access protected Zope resources above the folder at */ciseZPT/research*. Effectively XY's view of protected resources in the Zope site is limited to things in the *ciseZPT/research* folder and below. Regardless of the roles assigned to XY, the user cannot access protected resources "above" his location. If XY was defined as having the `Manager` role, he would be able to go directly to /ciseZPT/research/manage to manage his resources, but could not access /ciseZPT/manage at all.

To access the Zope Management Interface as a Manager user who is not defined in the root user folder, Zope uses the URL to the folder which contains his user folder plus `manage`. For example, if XY above has the Manager role as defined within a user folder in the *ciseZPT/research* folder, he would be able to access the Zope Management Interface by visiting `http://zopeserver/ciseZPT/research/manage`.

Of course, any user may access any resource which is *not* protected, so a user's creation location is not at all relevant with respect to unprotected resources. The user's location only matters when he attempts to use objects in a way that requires authentication and authorization, such as the objects which compose the Zope Management Interface.

It is straightforward to delegate responsibilities to site managers using this technique. One of the most common Zope management patterns is to place related objects in a folder together and then create a user folder in that folder to define people who are responsible for those objects. By doing so, the responsibilities for these object are safely delegated to these users.

For example, suppose user XY is responsible for maintaining the research area of the CISE website. The information about research is stored in the research folder of the Zope site. When an update to the research area has to be done, e.g., a new subsection needs to be added, the XY user doesn't have to ask the web master to update the site, he or she can update their own section of the site without bothering anyone else. Additionally, XY cannot log into any folder above the research folder, which means XY cannot manage any objects other than those in the research folder. This process of delegation could be used for different areas of the site. By delegating different areas of a Zope site to different users, the burden of site administration is taken off of a small group of managers and is spread around to different specific groups of users.

This information and more on delegation and security can be found in the Zope Book [LAT01].

5.7 Database Administration Issues

5.7.1 Form Checking with JavaScript

Whenever HTML based forms are used, a user could enter any kind of data in the form without any default error-checking being performed on the form. This could lead to problems with the database when a form issued to add or edit rows in the database. Some fields need to be filled out, and sometimes it is important that a column is filled with a specific data type (e.g., an integer). JavaScript can be used to check form entries before

the form action is called. When form data is missing or of the wrong type, the form action can be prevented. The Code Listing 5-9 illustrates how form checking was implemented for the database update forms:

```
<h2>Add <span tal:content="container/title_or_id"/></h2>
<script type="text/javascript">
<!--
function CheckInput() {
  if(document.f.firstName.value == "")  {
    alert("Please enter a first name!");
        document.f.firstName.focus();
        return false;
   }
  if(document.f.lastName.value == "")  {
    alert("Please enter a last name!");
        document.f.lastName.focus();
        return false;
   }
  if(document.f.email.value == "")  {
    alert("Please enter an email address!");
        document.f.email.focus();
        return false;
   }
  if(document.f.password.value == "")  {
    alert("Please enter a password!");
        document.f.password.focus();
        return false;
   }
  if(document.f.login.value == "")  {
    alert("Please enter a login!");
        document.f.login.focus();
        return false;
   }
  if(document.f.the_file.value == "")  {
    alert("Please enter a photo!");
        document.f.the_file.focus();
        return false;
   }
  return true;
}
-->
</script>
<form name="f" action="addScript" method="POST" enctype="multipart/form-data"
onSubmit="return CheckInput();">
<table border=0>
<tr>
<td>First Name</td>
<td><input name="firstName" size="40" width=30 value="" ></td>
</tr>
<tr>
<td>Middle Name</td>
<td><input name="middleName" size="40" width=30 value="" ></td>
</tr>
<tr>
<td>Last Name</td>
<td><input name="lastName" size="40" width=30 value="" ></td>
</tr>
```

```
<tr>
<td>Email</td>
<td><input name="email" size="40" width=30 value="" ></td>
</tr>
<tr>
<td>Webpage</td>
<td><input name="linkToWebpage" size="40" width=30 value="" ></td>
</tr>
<tr>
<td>Telephone Number</td>
<td><input name="telephoneNumber" size="40" width=30 value="" ></td>
</tr>
<tr>
<td>Office Number</td>
<td><input name="officeNumber" size="40" width=30 value="" ></td>
</tr>
<tr>
<td width="118">Degree</td>
<td width="346">
<textarea rows="7" name="degree" cols="30"></textarea>
</td>
</tr>
<tr>
<td width="243" > Login </td>
<td width="620" ><input maxlength="8" name="login">
</td>
</tr>
<tr>
<td width="243" > Password </td>
<td width="620" ><input type="password" maxlength="8" name="password">
</td>
</tr>
<tr>
<td>Title</td>
<td><input name="title" size="40" width=30 value="" ></td>
</tr>
<tr>
<td width="118">Research Interests</td>
<td width="346">
<textarea rows="7" name="interests" cols="30"></textarea>
</td>
</tr>
<tr>
<td>Photo</td>
<td><input type="file" name="the_file" size="30"></td>
</tr>
<input type="hidden" name="sequence" value="professor_sequence">
<tr>
<td colspan=2 align=center>
<input type="submit" name="submit" value="Add Professor">
</td>
</tr>
</table>
```

Code Listing 5-9 Form checking with Javascript

This code is used to add a new Professor. The form has some input fields and a submit button to send the form. The part that differs from a regular form is the `onSubmit="return chkFormular()"` part of the `<form>` tag. The event handler

for onSubmit is called when a user clicks on the submit button. The JavaScript function chkFormular, located just above the form, is then called. If the function finds an error, it return the false value, otherwise it returns true. When false is returned, the form action is not called.

The chkFormular function itself uses a series of if statements to determine whether a field is empty. A check such as `f(document.f.firstName.value == "")` tests whether the corresponding form input field (the field with the name "firstName" in the form with the name "f") contains a value by comparing it with the empty string "." Other forms in this project check whether the input field has been filled with an integer rather than a String. This test is done with the isNaN (is not a number) built-in JavaScript function:

```
if (isNaN(document.f.year.value ).
```

Three instructions follow for each check that fails. First of all, a message is displayed on the screen using the alert() built-in function. The message contains information about the problem encountered and tells the user how to fix it. Second, the focus() method is used to position the cursor in the field where the problem was encountered. This way, the user has the option to correct the flawed or missing input right away. Thirdly and lastly, the last instruction returns false. As a result of this, the form action is not executed.

The form action (`action="addScript"`) is calling a Python Script located on the server.

The JavaScript function has been adapted from material found in the SELFHTML online tutorial [MUE03].

5.7.2 The dbAdmin Folders

5.7.2.1 The Standard Folders

As mentioned before, the folders in the dbAdmin folder contain all HTML forms, ZSQL methods, and Python script needed for a given table. Most folders contains methods with the same name but different content. Those folders are described in the this section. The Professor folder is different since, as explained above, Professors can log in and edit their data. It is be described in the next Section. Figure 5-10 contains an overview of the objects located in every dbAdmin folder:

Figure 5-10 Objects inside ciseZPT/dbAdmin/Project

An explanation of the content of these objects follows.

"index_html" Page Template. This page template is essentially the same for almost all folders of dbAdmin. It simply presents two options: the list of all rows of the table, or the form to add a new row. This is displayed in the body slot of the page macro:

```
<div metal:fill-slot="body">
  <ul>
    <li><a href="listAll">List</a></li>
    <li>Add a new <a href="add"
tal:content="container/title_or_id">item</a></li>
  </ul>
```

```
</div>.
```

“listAll” Page Template. The listAll Page Template is also the same for most tables. It calls a ZSQL method which returns all rows of a table in alphabetical order. For each entry, a link is added that leads to the update or the delete page template. The primary key of the current row is appended as a parameter for each link:

```
<div metal:fill-slot="body">
<h2><span tal:replace="container/title_or_id"/></h2>
<ul>
  <li tal:repeat="in root/sql/allProjects">
  <b>
  <span tal:replace="in/title">Title</span>
  <br>
  <font size="-1"><a href="update" tal:attributes="href
string:update?title=${in/title}">edit</a> |
  <a href="delete" tal:attributes="href
string:delete?title=${in/title}">delete</a></font>
  <p>
  </li>
</ul>

Add a new <a href="add" tal:content="container/title_or_id">item</a>

</div>.
```

“delete” Page Template. Called from the listAll page template, this Page Template receives as a parameter the primary key of the row that is to be deleted. A call to the ZSQL method deleteSQL is done, and then the ZPT displays a message confirming the deletion. The option to add a row or to return to the list of rows is added. This page template is the same for almost every folder.

```
<div metal:fill-slot="body">
<span tal:define="query here/deleteSQL"></span>
<h2><span tal:content="container/title_or_id"/> deleted!</h2>
<p>
  <ul>
  <li>Add another <a href="add"
tal:content="container/title_or_id">item</a></li>
  <li>List of <a href="listAll"
tal:content="container/title_or_id">item</a>s</li>
  </ul>

</div>.
```

“add” Page Template. The add Page Template consists of a form where the row data is entered. The form action is usually the Page Template itself (“add”). A condition is evaluated. If the Page Template is called after a submit, a call to the “addSQL” ZSQL

method is made. If the Page Template is not called after a submit, the form with all input fields is shown (`<div tal:define="global submit request/form/submit | nothing" tal:condition="not:submit">`). As explained in Section 5.7.1, a JavaScript function is called to check the form input before the actual SQL method is executed. This Page Template is different for every table of the database.

The input values are passed to the ZSQL methods over the REQUEST.

```
<div metal:fill-slot="body">
<script type="text/javascript">
<!--
function CheckInput() {
  if(document.f.Prof_ID.value == "")  {
    alert("Please select a professor!");
        document.f.Prof_ID.focus();
        return false;
   }
  if(document.f.title.value == "")  {
    alert("Please enter a title!");
        document.f.title.focus();
        return false;
   }
  return true;
}
//-->
</script>
<div tal:define="global submit request/form/submit | nothing"
tal:condition="not:submit">
<h2>Add <span tal:content="container/title_or_id"/></h2>
<form name="f" action="add" method="POST" onSubmit="return CheckInput();">
<table border=0 width="465">
<tr>
<td width="118">Project Title</td>
<td width="346"><input name="title" width=30 value="" size="48"></td>
</tr>
<tr>
<td width="118">Webpage</td>
<td width="346"><input name="linkToWebpage" width=30 value="" size="48"></td>
</tr>
<tr>
<td width="118">Description</td>
<td width="346">
<textarea rows="7" name="description" cols="30"></textarea>
</td>
</tr>
<tr>
<td colspan=2 align=center width="470">
<input type="submit" name="submit" value="Add">
</td>
</tr>
</table>
</div>
<div tal:condition="submit">
  <span tal:define="query here/addSQL"/>
  <p>You have added <b tal:content="request/form/title">title</b>
```

```
  to the database.
  </p>
  <ul>
  <li>Add another <a href="add"
tal:content="container/title_or_id">item</a></li>
  <li>List of <a href="listAll"
tal:content="container/title_or_id">item</a>s</li>
  </ul>
</div>
</div>.
```

"update" Page Template. The update page template is essentially the same as the add page template. The only difference is that it is called from the listAll page template and therefore needs to display one particular table row that is to be edited. The primary key of the row was passed over the REQUEST. To retrieve all data from the table row, a call to the "selectOne" ZSQL is made. The values from result are then pre-filled into the input fields with tal:content= "python:inData[0].columnname."

As for the add page template, it is evaluated whether the page template is called after a submit.

```
<div metal:fill-slot="body">
<div tal:define="global submit request/form/submit | nothing"
tal:condition="not:submit">
<div tal:define="inData here/selectOne| nothing">
<h2>Update <span tal:content="container/title_or_id">item</span></h2>
<form name="f" action="update" method="POST" >
<table border=0 width="446">
<tr>
<td width="118">Title</td>
<td width="349"><input readonly name="title" width=30 value=""
tal:attributes="value python:inData[0].title" size="48"></td> </tr>
<tr>
<td width="118">Webpage</td>
<td width="349"><input name="linkToWebpage" width=30 value=""
tal:attributes="value python:inData[0].linktowebpage" size="48"></td>
</tr>
<tr>
<td width="118">Description</td>
<td width="349">
<textarea rows="7" name="description" cols="39"
tal:content="python:inData[0].description">description</textarea>
</td>
</tr>
<tr>
<td colspan=2 align=center>
<input type="submit" name="submit" value="Update">
</td>
</tr>
</table>
</div>
</div>
```

```
<div tal:condition="submit">
  <span tal:define="query here/updateSQL"/>
  <p>You have updated <b tal:content="request/form/title">title</b>
  in the database.
  </p>
  <ul>
  <li>Add another <a href="add"
tal:content="container/title_or_id">item</a></li>
  <li>List of <a href="listAll"
tal:content="container/title_or_id">item</a></li>
  </ul>
</div>

</div>.
```

"addSQL" ZSQL method. This method is customized for every folder. It contains an SQL command similar to the following:

```
insert into Project(title, linktowebpage, description) values
(
 <dtml-sqlvar title type="string">,
 <dtml-sqlvar linkToWebpage type="string">,
 <dtml-sqlvar description type="string">
).
```

Some tables have integers as primary key. The idea behind this was to auto_increment the primary key whenever a new tuple is inserted. Unfortunately, this is not supported by Oracle. The following workaround was used: every table that has an integer primary also has a corresponding Oracle sequence. Whenever a new row is added, the addSQL calls the nextval python script (`<dtml-var "nextval(sequence)">`), located in the root folder so that it can be inherited everywhere. The nextval script receives the name of the sequence to fetch from and returns the next value (`return sequence + ".NEXTVAL"`).

"selectOne" ZSQL method. Again, this is one of the methods that had to be customized for every folder. The ZSQL method receives a primary key over the REQUEST and returns the matching row, e.g., for the Project table with primary key title:

```
SELECT * FROM Project WHERE title = <dtml-sqlvar title type="string">.
```

“deleteSQL” ZSQL method. This ZSQL method is called from the listAll page template. Similar to the “selectOne” ZSQL method, deleteSQL is different for every folder and receives the primary key of the tuple to delete over the REQUEST:

```
DELETE FROM Project WHERE title = <dtml-sqlvar title type="string">.
```

5.7.2.2 The Professor Folder

The Professor folder is different from the other folder located in dbAdmin. First of all, when logging in to the dbAdmin area, a different set of options is displayed in the index_html of dbAdmin, depending on who has logged in:

```
<div metal:fill-slot="body">
<span tal:define="global loginName user/getUserName"/>
<h3>Welcome, <span tal:content="loginName"></span>, to the Database
Administrator Area!</h3>


<span tal:define="global roles user/getRoles"/>
<span tal:condition="python:'prof' in roles">
You are logged in as a Professor.
</span>
<br>
Here, you have the option to add, delete, and update the underlying database.
Please select the area that you wish to update from the following list.

<!-- display one set of options if a professor is logged in-->
<span tal:condition="python:'prof' in roles">
<!-- get Professor by login-->
<span tal:repeat="in python:root.sql.selectProfessorByLogin(login=loginName)">
<span tal:define="global Prof_ID in/prof_id"/>
</span>
<div tal:repeat="item python:here.getDBFolders('Prof')">
<div align="left" class="menu" tal:condition="python:here!=item['obj'] or
template.id!='index_html'">
<a href="url" tal:define="url string:${item/url}?Prof_ID=${Prof_ID}"
tal:attributes="href url" tal:content="item/title">link to folder</a>
</div>
</div>
</span>
<!-- display another set of options if an admin is logged in-->
<span tal:condition="not:python:'prof' in roles">
<div tal:repeat="item python:here.getDBFolders('DB')">
<div align="left" class="menu" tal:condition="python:here!=item['obj'] or
template.id!='index_html'">
<a href="url" tal:define="url string:${item/url}?Prof_ID=0"
tal:attributes="href url" tal:content="item/title">link to folder</a>
</div>
</div>
</span>
```

```
</div>.
```

As the code listing above shows, when a Professor is logged in (`<span tal:define="global roles user/getRoles"/><span tal:condition="python:'prof' in roles">`), the folders listed are only the ones visible to Professors(`python:here.getDBFolders('Prof')"`). The link to each folder is modified such that the primary key of the Professor currently logged in is passed as a parameter to the link (`tal:define="url string:${item/url}?Prof_ID=${Prof_ID}`). When a regular administrator is logged in, the links to folders are also modified to contain the Prof_ID parameter, but it is set to 0 to indicate that no Professor is logged in (`tal:define="url string:${item/url}?Prof_ID=0"`).

This parameter is now retrieved in the Professor folder. Different screens are displayed depending on whether a Professor is logged in or not. The comparison is done with the following condition statement:

```
<span tal:define="global Prof_ID request/Prof_ID" tal:condition="python:Prof_ID
== '0'">.
```

Professor Photos. Another difference to regular dbAdmin folders is the fact that a photo is stored for each Professor. Because of well-known problems with storing BLOBs in Oracle, Photos are stored centrally in the Zope Object database (in /ciseZPT/peoples/photos). The Photo and PhotoFolder product [BIC03] have been installed for this purpose. Whenever a new Photo is added to the PhotoFolder, two display sizes are generated by the Image Magick program [IMA03]: thumbnail (128 x 128) and xsmall (200 x 200). The createPhoto Python script in the Professor folder handles the photo creation:

```
# move to photofolder
p = context.ciseZPT.peoples.photos
#if( not p ): raise AssertionError, 'No Place to store Photos'
the_id = request.form['Prof_ID']
the_title = request.form['Prof_ID']
the_file = request.form['the_file']
```

```
p.manage_delObjects(the_title)
p.manage_addProduct['Photo'].manage_addPhoto( the_id, the_title, the_file)
```

The script deletes any photo that was saved for the Professor. This is why when a new Professor is added, a default photo needs to be specified (which can later be overridden).

Professor user in acl_user folder. When a new Professor is added to the database, a new user should also be added in the acl_user folder of dbAdmin. The "login" is the name that the user will be using to login to the dbAdmin area. The following lines in a Python script take care of adding a user with the role "prof."

```
lo = request.form['login']
p = request.form['password']
#add user in acl_users
context.acl_users.userFolderAddUser(lo, p, ['prof'], []).
```

Professors can also change their password over an update form. The update of the database is preformed with a regular ZSQL method. The roles for a Professor and the login however, cannot be changed. The update of the password of the user in the user folder is achieved with the following lines in a Python script:

```
p = request.form['password']
lo = request.form['login']
#set password if new one was given
if p != "":
  user = context.acl_users.getUser(lo)
  uid = user.getId()
  roles = user.getRoles()
  domains = user.getDomains()
       context.acl_users.userFolderEditUser(uid, p, roles, domains).
```

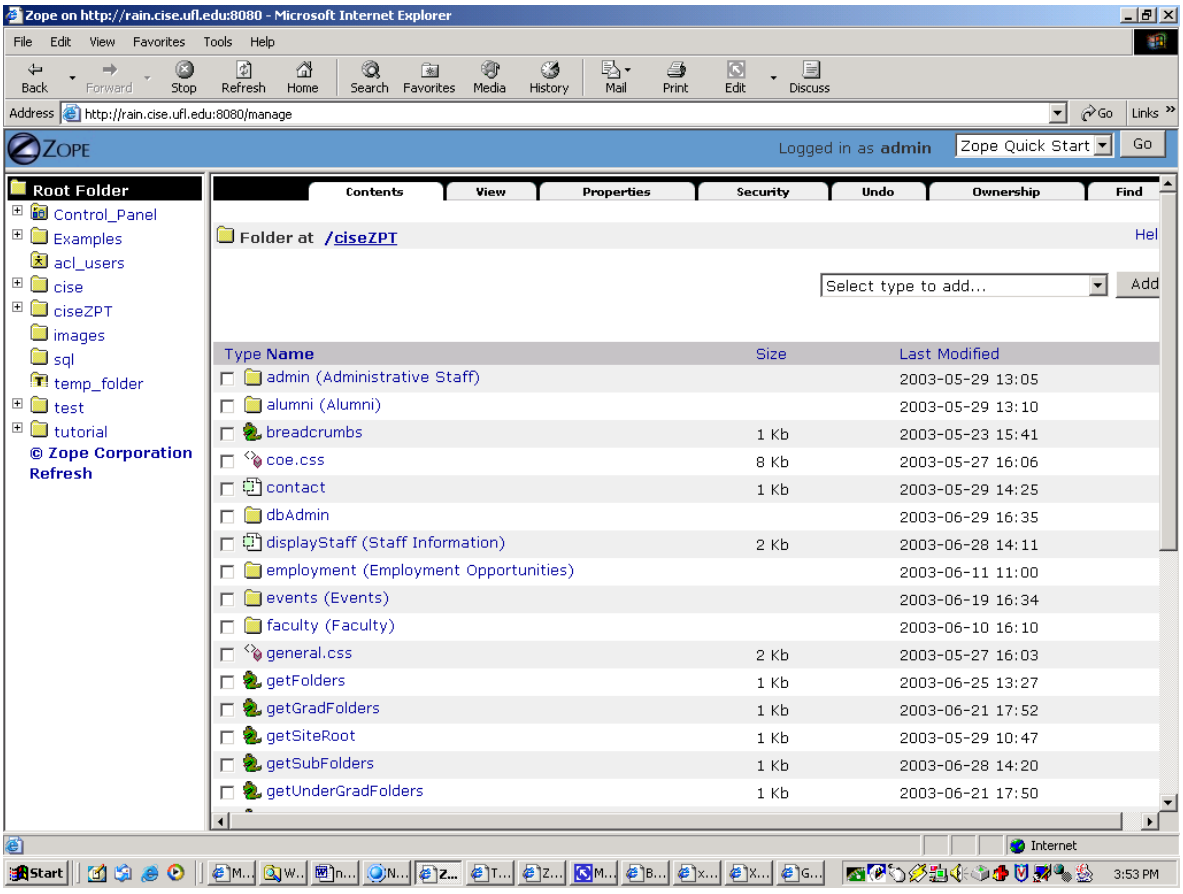
CHAPTER 6
USER INTERFACE

6.1 Overview of Graphical User Interface

The Zope Management interface when an administrator is logged into the ciseZPT folder is shown in Figure 6-1.

When viewing the index_html document of ciseZPT, the view of Figure 6-2 is produced on the screen.

Figure 6-1 Zope Management Interface



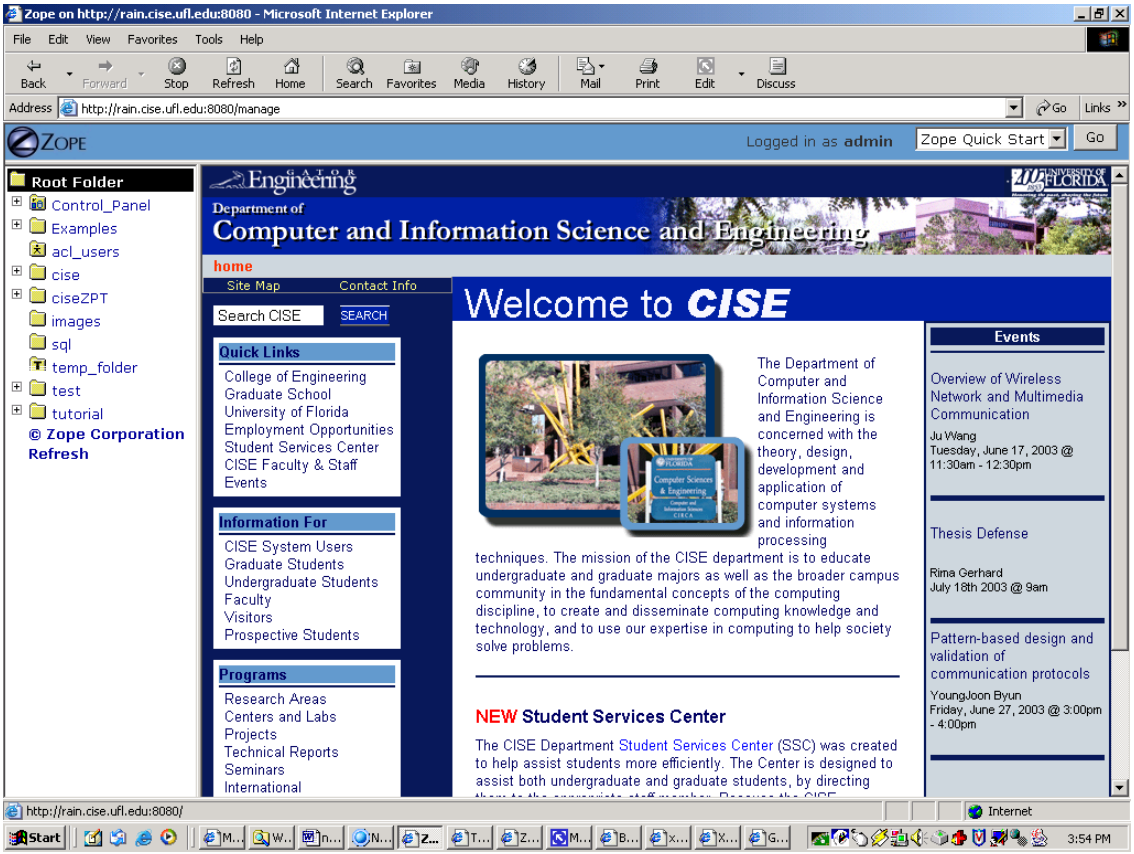


Figure 6-2 Main CISE website

6.2 Database Administration GUI

When a user tries to log in to the database admin area (/ciseZPT/dbAdmin), the Zope login dialog appears on the screen. A screenshot of this can be seen in Figure 6-3.

If an admin logs in, the dbAdmin screens presents the full set of options available, as Figure 6-4 illustrates.

If a Professor logs in, the dbAdmin shows less options for update, as Figure 6-5 shows.

A sample database dialog is shown in Figure 6-6. It is the page that lets a user add a new offered graduate class to the database. The other database dialogs have a similar design.

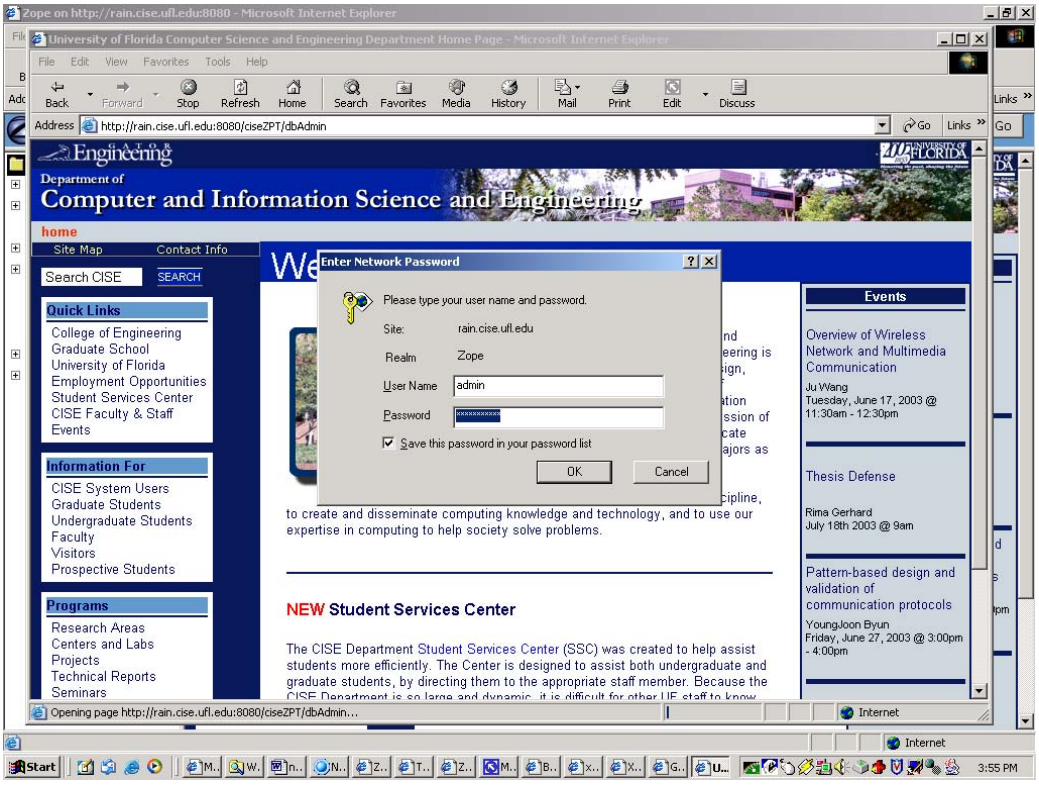


Figure 6-3 Logging in to the database administration area

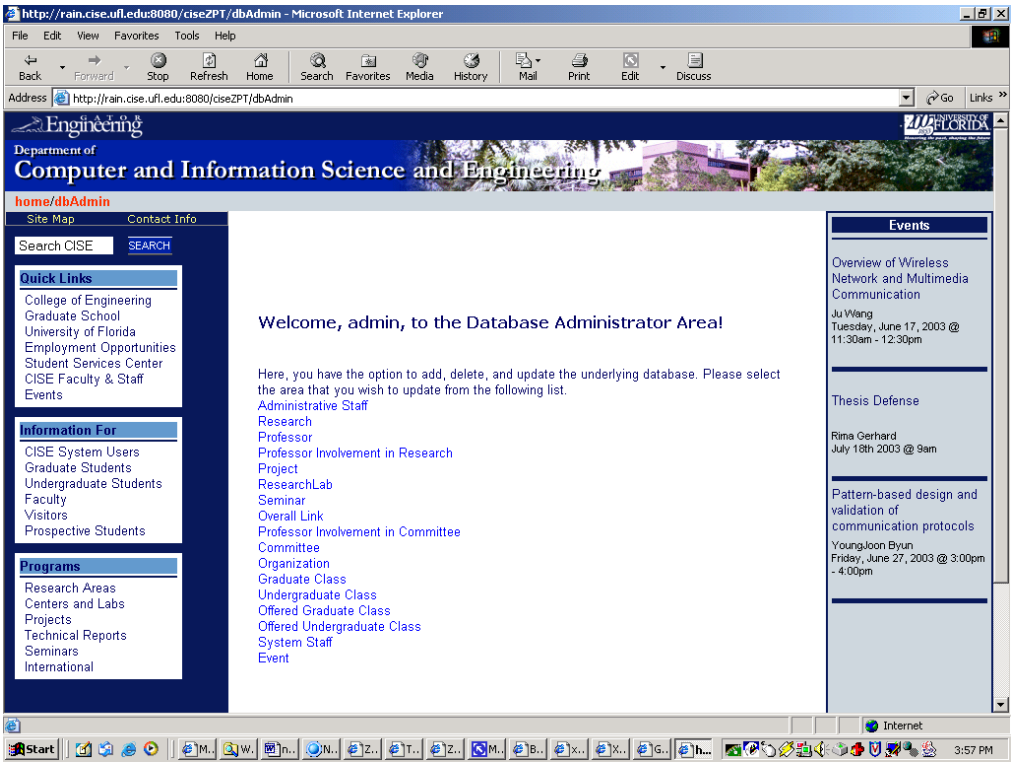


Figure 6-4 Screen presented to a database administrator

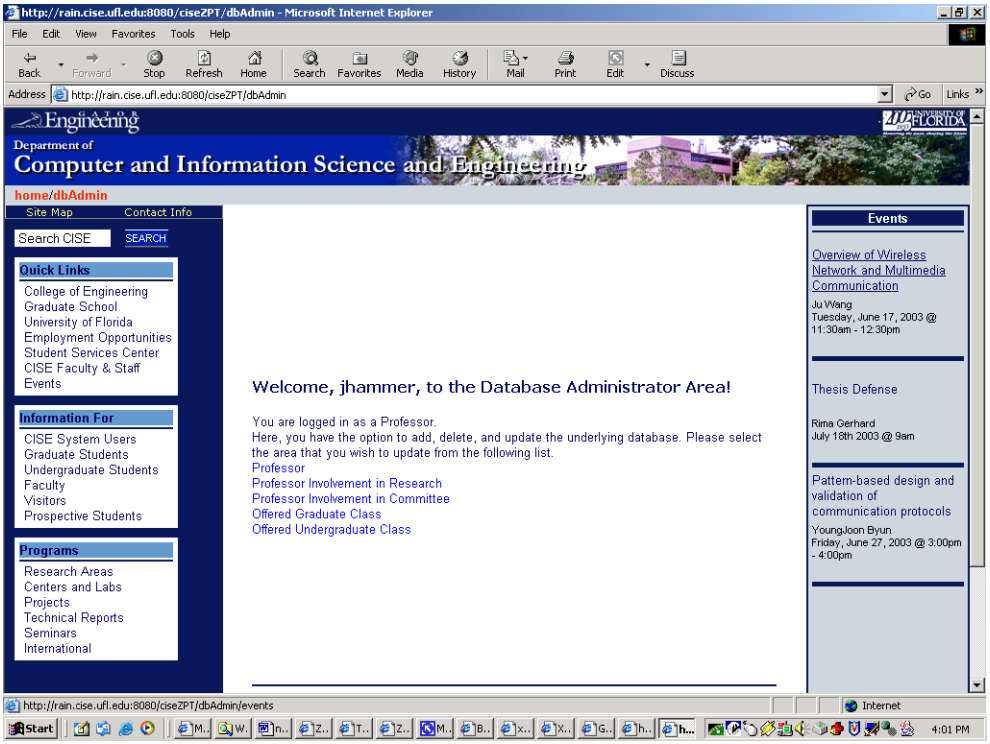


Figure 6-5 Screen presented to a professor

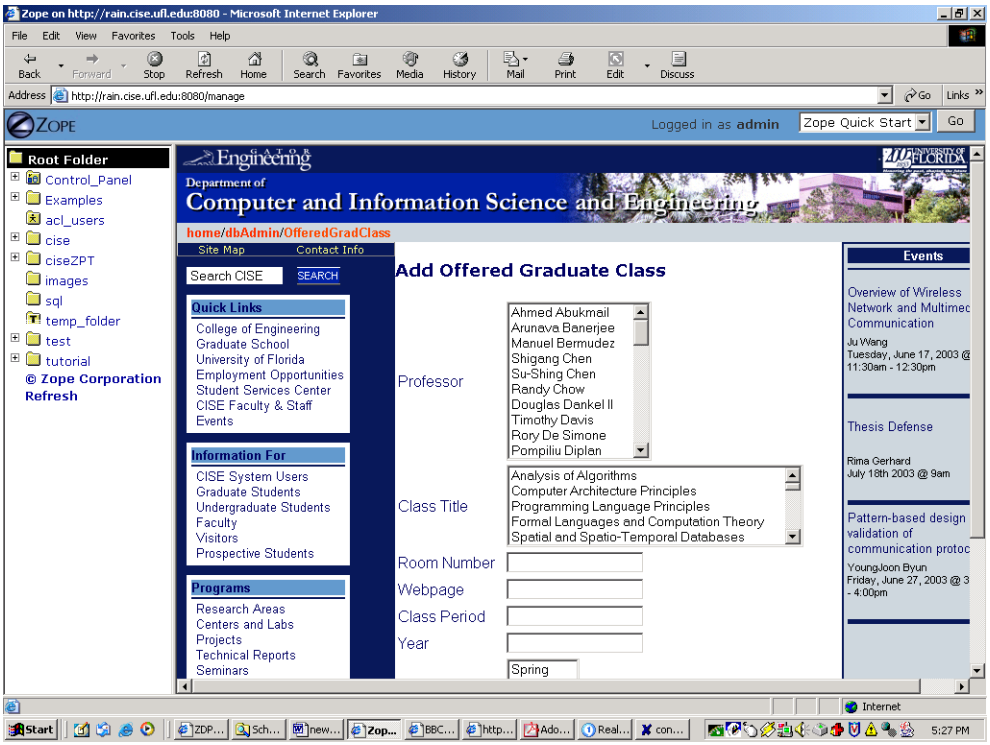


Figure 6-6 Sample database dialog

Security Settings. Figure 6-7 below shows how security settings are managed over Zope's Management Interface. The screenshot shows the security setting for the ciseZPT/peoples/photos folder. Permissions are rows and roles are columns. Checkboxes are used to indicate where roles are assigned permissions. When a role is assigned to a permission, users with the given role will be able to perform tasks associated with the permission on this item. When the *Acquire permission settings* checkbox is selected then the containing object's permission settings are used. The screenshot shows how the "prof" role has been explicitly enabled to "Add Photos"(as well as Authenticated, Manager, and Owner).

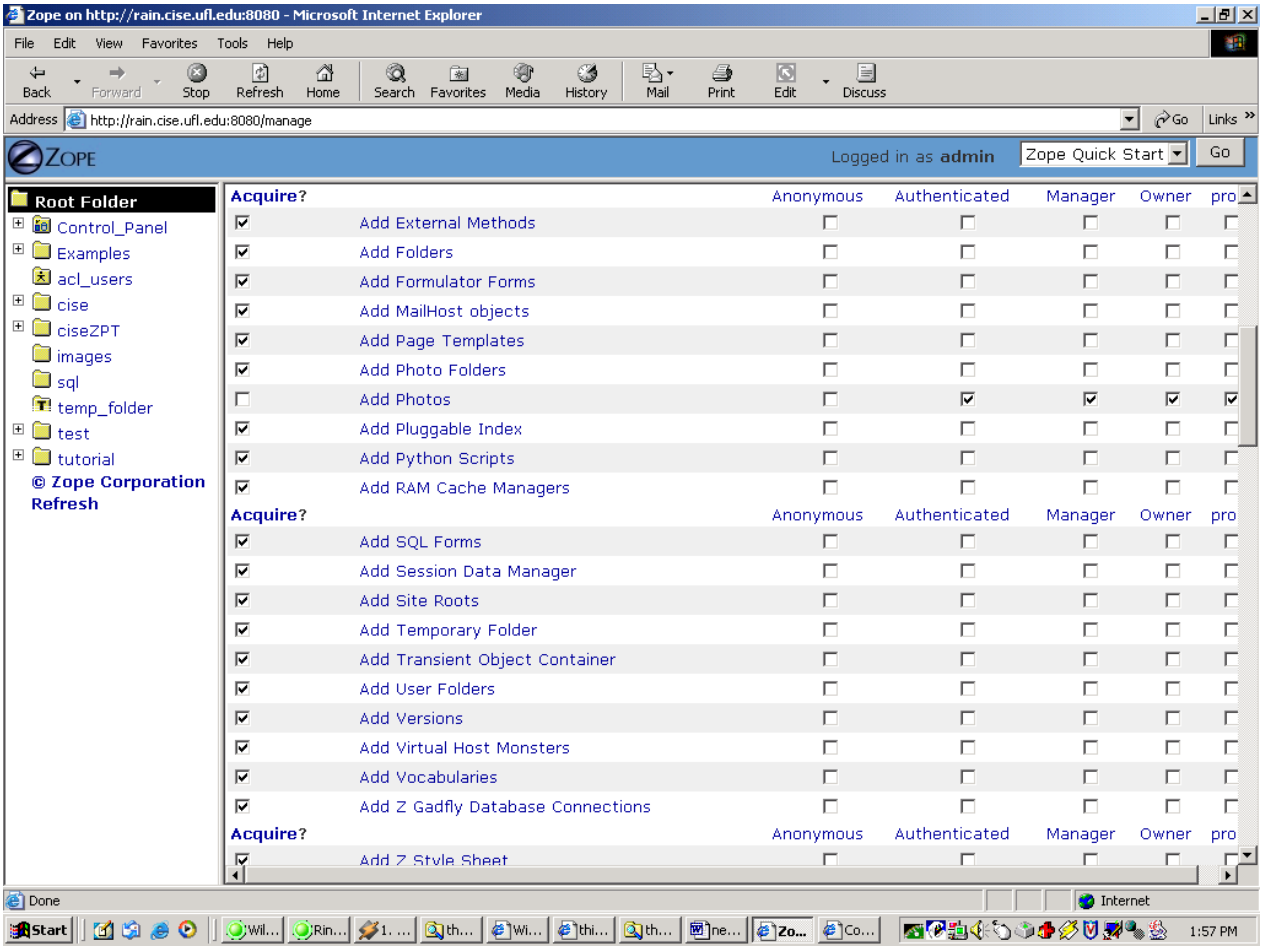


Figure 6-7 Security Management through-the-web

CHAPTER 7
CONCLUSION AND FUTURE WORK

7.1 Have the Requirements Been Met?

The first requirement, (The new web site should be based on widely spread and reliable web techniques such that most browsers can access the site) has been met. Since Zope interprets all contained DTML/ZPT/Python code, all that is output is HTML code. No proprietary data formats (Flash,etc.) has been used.

The second requirement, adherence to the COE and UF Webadmin Guidelines, has been mostly fulfilled. An exception to this is the use of JavaScript needed for form checking, which is not recommended by the UF Webadmin committee. Style sheets have also been used, but the site is still viewable even when the style sheets are disabled.

The third requirement was the separation of data and the graphical user interface. This is achieved through Zope's own technologies such as ZPT and Python scripts.

The fourth requirement (Faculty members should have the option to access certain areas of the site and create content) as well as the sixth requirement (The database should be accessible over an easy-to-use graphical interface, providing forms for adding, updating and deleting data in the database tables. Furthermore, there shall be forms for creating new tables in table.) is achieved through the dbAdmin area, where Professor can log in and edit and add data in the database. It is currently not the case that a non-Zope-programmer can easily add new tables in the database. Future work is therefore possible in this area.

The fifth requirement was that administrative tasks such as adapting the layout, design and content and updating and maintaining the database shall be possible without further knowledge of a markup- or programming language. Again, Zope provides the solution to this requirement through ZPT, which can be edited in WYSIWYG editors without further knowledge of HTML.

The seventh and eight requirement (The database transactions must be secure and the application should have multi-user functionality; the application should be easily extendible) are achieved through Zope: transactions and multi-user functionality as well as add-on products are both area of the Zope framework described in Chapter 3.

Especially the fourth requirement of taking the burden of web-site maintenance off of the shoulders of one single web developer was of importance. Zope itself offers functionality to set up several users and assign them areas of responsibility. In addition to that, the project uses an Oracle database. As long as the data in the database is kept up-to-date, the information displayed on the web site will be up-to-date, as well. To simplify the database maintenance for non-technical users, the web site contains a set of online dialogs that can be understood and used by anyone, without requiring any knowledge of the happenings “behind the scene” (such as the HTML, SQL, Python scripts, etc. involved). This aspect of database administration can therefore be handled by anybody with an Internet connection and a browser. If new areas have to be added to the site, however, a user with programming knowledge and a good understanding of Zope should be assigned to the task. It is not possible to completely eliminate the need for one person with technical background. The question whether this would be possible with other CM systems remains open. If design changes had to be implemented, the job could be handed

over to a professional web designer or anyone able to use a WYSIWYG tools such as FrontPage. The design created would then have to be reviewed by a Zope programmer (who would add Zope Page Template tal statements to the HTML). This at least eliminated the need for the web programmer to also be a good web designer.

7.2 Conclusions

The goal for this thesis was the redesign and reimplementation of the CISE Department's web site using a content management system. When researching possible implementation strategies, the Zope Framework was encountered. Zope fulfilled the requirements that were formulated, and its open source nature made it an ideal candidate for future use. When first working with Zope, the DTML scripting language was used. Most of the site was implemented when it was discovered that the newer, more powerful technology called ZPTs was a better fit and also strongly recommended by the Zope community. Since the goal of this project was to be up to date with current technologies so that it could be easily extended in the future, we have migrated the entire site to the new ZPT language. The old implementation still resides in the Zope instance for future reference.

The project has lasted for about 10 months (October 2002 – July 2003). The community support by fellow Zope developers, administrators, and users that actually help each other and share information, tips and tricks has been very impressive. Zope has proved itself an outstanding tools for the creation of dynamic web applications. This is not only the author's personal opinion – Zope's ever-growing customer list [ZOP03b], including many illustrious names – proves that his viewpoint is shared by many others. In addition to this, the existence of many Zope products is a very fascinating environment

for developers. It would take several months to explore all the possibilities of add-on products that can be downloaded for free.

If the department were to decide to use the project developed in this thesis, it would be first of all recommendable to install the latest Zope version [ZOP03a]. It appears that the latest Zope version (Zope 3) will be released in the near future. To reuse the code written for this thesis in a new installation, all one would have to do is simply copy the zope-directory/var/data.fs from the old Zope to the new Zope. All work contained in the current Z Object database is contained in Zope's data.fs. However, once the data has been copied over, one still has to copy all products installed in the old Zope version (they are not stored in Zope's data.fs). The products needed for this project have been listed in Section 5.1. Finally, one would have to decide whether Zope's own web server would be used or the Apache web server. Zope can serve content through it's own web server, or Apache can communicate with it over a certain protocol called PCGI. Using Zope and Apache with PCGI, one can serve some sites content from Apache and some from Zope. Tutorials on the web explain how this setup is done in detail [ROE03].

LIST OF REFERENCES

[ARI03] Ariadne. Ariadne World Wide Website, http://ariadne.muze.nl/, 2003, last accessed 07/18/03.

[ATT03] AT&T Research Labs. Strudel World Wide Website, http://www.research.att.com/~mff/strudel/doc/, 2003, last accessed 07/18/03.

[BIC03] R. Bickers. Photo Product, http://www.zope.org/Members/rbickers/Photo, 2003, last accessed 07/18/03.

[BOR96] J. A. Borges, I. Morales, and N. J. Rodriguez. Guidelines for Designing Usable World Wide Web Pages *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems,* pp. 277-278, Vancouver, ACM Press, 1996.

[BOR98] J. A. Borges, I. Morales, and N. J. Rodriguez. Page Design Guidelines Developed Through Usability Testing-Human Factors and Web Development, Lawrence Erlbaum Associates Publishers, Mahwah, NJ, USA, 1998.

[BOS98] B. Bos, H. W. Lie, C. Lilley, and I. Jacobs. Cascading Style Sheets Level 2 Specification, 1998.

[CHE76] P. P. Chen. The Entity-Relationship Model: Towards a Unified View of Data. *ACM Transactions on Database Systems*, pp. 471-522, 1976.

[CHR99] W. Chrisholm, G. Vanderheiden, and I. Jacobs. Web Content Accessibility Guidelines 1.0, 1999.

[CLA99] J. Clark. XSL Transformations (XSLT specification). World Wide Web Consortium, 1999.

[CMS03] CMS-List, CMS-List World Wide Website, http://cms-list.org/, 2003, last accessed 07/18/03.

[COC03] Cocoon, Cocoon World Wide Website, http://cocoon.apache.org/, 2003, last accessed 07/18/03.

[COM03] Content Manager. Content Manager World Wide Website, http://www.contentmanager.de, 2003, last accessed 07/18/03.

[COL03] College of Engineering. The College of Engineering's Departmental Branding Documentation, http://www.eng.ufl.edu/graphics/, 2003, last accessed 05/12/03.

[DET98] O. De Troyer. Designing Well-Structured Websites: Lessons to Be Learned from Database Schema Methodology. *International Conference on Conceptual Modelling/the Entity Relationship Approach*, pp. 51-64, 1998.

[EVE03] P. Everitt. Zope Enterprise Objects, http://www.zope.org/Products/ZEO/ZEOFactSheet, 2003, last accessed 07/18/03.

[FER97] M. F. Fernandez, D.Florescu, J. Kang, A.Y. Levy and D. Suciu. *Catching the Boat with Strudel: Experiences with a Web-Site Management System* SIGMOD Conference, pp. 1-22, 1997.

[GAR93a] F. Garzotto, L. Mainetti, and P. Paolini. Navigation patterns in hypermedia databases. *Proceedings of the 26th Hawai International Conference on System Science*, Maui, pp. 370-379, IEEE Computer Society Press, 1993.

[GAR93b] F. Garzotto, P. Paolini, and D. Schwabe. HDM - A Model-Based Approach to Hypertext Application Design, *ACM Transactions on Information Systems*, pp.1-26, 1993.

[HAQ03] Haqa. Zope Cascading StyleSheets, http://www.zope.org/Members/haqa/ZStyleSheet/, 2003, last accessed 07/18/03.

[IMA03] ImageMagick, Image Magick World Wide Website, http://www.imagemagick.com/, 2003, last accessed 07/18/03.

[ISA95] T. Isakowitz, E. A. Stohr, P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design *Communications of the ACM 38(8)*, pp.34-43, 1995.

[KEE98] B. Keevil. Measuring the Usability Index of Your Web Site, *SIGDOC98: Scaling the Heights: The Future of Information Technology,* 1998.

[KOC01] N. Koch. A Comparative Study of Methods for Hypermedia Development, *Technical Report 9905, LudwigMaximilians-Universitaet Muenchen*, 2001.

[KRO03] M. Kromer. DCOracle2, http://www.zope.org/Members/matt/dco2, 2003, last accessed 05/14/03.

[LAT01] A. Latteier and M. Pelletier, The Zope Book, SAMS, New York City, 2001.

[LEE98] H. Lee, C. Lee, and C. Yoo. A Scenario-Based Object-Oriented Methodology for Developing Hypermedia Information Systems, *Proceedings Of HICSS 36*, Kohala Coast, IEEE Computer Society Press, 1998.

[MAC03] Macromedia. Macromedia World Wide Website, http://www.macromedia.com/, 2003, last accessed 07/18/03.

[MUE03] S. Muenz. Selfhtml: Version 8.0, http://selfhtml.teamone.de/, 2003, last accessed 07/18/03.

[NIE00] J. Nielsen. Designing Web Usability: The Practice of Simplicity. New Riders Publishing, Indianapolis, 2000.

[PER03] Perl. Perl World Wide Website, http://www.perl.com/, last accessed 07/18/03.

[PHP03] PHP. PHP World Wide Website, http://www.php.net/, 2003, last accessed 07/18/03.

[ROE03] M. Roeder. Setting up Zope with Apache using PCGI, http://zdp.zope.org/projects/zbook/book/VII/IntegraWebServers/IntegraUnix/IntegraApacheDrafts/953556353, 2003, last accessed 07/18/03.

[SCH98] D. Schwabe and G. Rossi. Developing hypermedia applications using OOHDM. *Proceedings of Workshop on Hypermedia development Process, Methods and Models*, Pittsburgh, ACM,1998.

[SUN03] Sun Microsystems, Java World Wide Website, http://java.sun.com/, last accessed 07/18/03.

[WEB03] Webcms. Webcms World Wider Website, http://www.webcms.org/, 2003, last accessed 07/18/03.

[WOR03] World Wide Web Consortium. W3 Website, http://www.w3.org/, 2003, last accessed 07/18/03.

[ZOP03a] Zope Corporation. Zope World Wide Website. http://www.zope.org, 2003, last accessed 07/18/03.

[ZOP03b] Zope Corporation. Zope Client List World Wide Website, http://www.zope.com/ZopeClientList, 2003, last accessed 07/18/03.

BIOGRAPHICAL SKETCH

Rima A. Gerhard was born in the state of Virginia in the United States of America on October 24th, 1979. She was brought up in Germany and France and completed her Higher Secondary Certification at the German School of Paris, France. She then went on to complete her bacherlor's degree in computer engineering at the University of Ulm, in Ulm, Germany. Soon after that, she joined the Computer and Information Science and Engineering Department at the University of Florida in Fall 2001.

Her industrial work experience includes an internships at d+s online AG in Weinheim, Germany, as well as an internship at the Space and Naval Warfare Systems Center in San Diego, California. Her areas of interest include content management systems, web site pubishing and databases.