NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

WEB-BASED TESTING TOOLS FOR ELECTRICAL ENGINEERING COURSES

by

Jaime Briggs

September 2001

Thesis Co-Advisors: Roberto Cristi
Thomas Wu

Approved for public release; distribution is unlimited.

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
30 Sep 2001	N/A	-

Title and Subtitle Web-Based Testing Tools for Electrical Engineering Courses	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Briggs, Jaime	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Research Office Naval Postgraduate School Monterey, Ca 93943-5138	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 134	

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Title (Mix case letters) Web-Based Testing Tools for Electrical Engineering Courses	**5. FUNDING NUMBERS**
6. AUTHOR(S) Briggs, Jaime	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

This thesis presents a distance-learning tool, which provides a self-sufficient application that allows one to implement online courses for electrical engineering. A major emphasis is placed on replacing simplistic multiple-choice or true-false test questions. A system named, Distance Learning Tools for Online Tests (DLTOT) is designed, modeled and implemented.

The implementation is based on the Java programming language, using Servlets and Java Server Pages (JSP), three-tier technology and Commercial Off-the-Shelf (Costs) products, namely, an Apache web server, Tomcat Application server, Microsoft Access, Mathematica, WebMathematica and JSP/ Servlet technology.

DLTOT is able to control student access, to allow interaction with the student during the course, and to present a challenging test, which is easily graded by the application itself.

14. SUBJECT TERMS Web-based learning, test online, multi-tier architecture	**15. NUMBER OF PAGES** 134
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

WEB-BASED TESTING TOOLS FOR ELECTRICAL ENGINEERING COURSES

Jaime C. Briggs
Lieutenant, Chilean Navy
B.S. in Electrical Engineering, Naval Polytechnic Academy, 1992
B.S. in Information Systems, Naval Polytechnic Academy, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
and
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: _____
Jaime C. Briggs

Approved by: _____
Roberto Cristi, Thesis Co-Advisor

_____
Thomas Wu, Thesis Co-Advisor

_____
LCDR Chris Eagle, Chairman
Department of Computer Science

_____
Jeffrey B. Knorr, Chairman
Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis presents a distance-learning tool, which provides a self-sufficient application that allows one to implement online courses for electrical engineering. A major emphasis is placed on replacing simplistic multiple-choice or true-false test questions. A system named, Distance Learning Tools for Online Tests (DLTOT) is designed, modeled and implemented.

The implementation is based on the Java programming language, using Servlets and Java Server Pages (JSP), three-tier technology and Commercial Off-the-Shelf (Costs) products, namely, an Apache web server, Tomcat Application server, Microsoft Access, Mathematica, WebMathematica and JSP/ Servlet technology.

DLTOT is able to control student access, to allow interaction with the student during the course, and to present a challenging test, which is easily graded by the application itself.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This thesis is dedicated to my family, my wife, Alejandra, for her invaluable support and understanding and to my two sons, Javier and Rodrigo, for sharing the computer with “Dad.” I would also like to thank Dr. Roberto Cristi for his patience, understanding and support. He played a key roll in this development and his guidance was always a light to follow. Also I will like to thanks Dr. Thomas Wu, for teaching me the technology to implement this solution. I would like to mention my gratitude to Ron Russell for his friendly and accurate advice during the editing of this thesis.

Finally I would like to thank God for the opportunity to obtain this outstanding education and to ask him for his wisdom when needed.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The concept of distance-learning education has become a major area of research over the past decade, not only in Defense Institutions, but also in civilian universities. Presently, an increasing number of universities offer specific distant-learning courses, and even offer a Master's degree with most of their courses available on the web.

Distance-learning education has developed over the years reaching a mature technology by which web-based courses can now be efficiently presented. Unfortunately, evaluating a student's work in such courses is still in its early stages. An efficient method for evaluating the progress of a distance-learning student is essential. Currently most online tests are based on a limited multiple-choice model for which the student must select the correct answers. Civilian institutions that offer distance-learning courses face the same problems when presenting a test on a web-based course. Professors are obligated to test students by using traditional "paper and mail" techniques. Naturally, these dated methods are labor intensive and correcting such material absorbs much of the professors' time.

The Naval Postgraduate School (NPS) is supporting research to create a tool that can construct more challenging online tests. This tool will enable professors to create a meaningful test for a Digital Signal Processing course for the Department of Electrical and Computer Engineering (ECE) at the NPS. This thesis investigates and proposes solutions for the issues summarized in the following three research questions:

- Based on new available technologies, what is the most appropriate solution for an online test for a Signal Processing course?
- Is it possible to create a student-assessed test online that is not multiple-choice based?
- Is it possible to evaluate the student set of answers and to give a grade that accurately reflects the students' knowledge of the subject?

This present research covers the complete cycle in developing the application. The technical specifications, the model, and the implementation of a tool that solves the problem of teaching electrical engineering classes online are revised and discussed. We created an application named, "Distance Learning Test Online Tool" (DLTOT). The DLTOT functionality is represented using four modules. These main components are: the Knowledge Database, the General Database, the Mathematical Kernel, and the Coordinator.

Using this component organization, one can highlight the desire functionality of each module and based on this, one can define the most effective technology to create the implementation. We found that the three-tier technology, based on the Java programming language is best suited for the implementation. We specified a set of Commercial Off-the-Shelf (COTS) software that allowed us to implement the application. Using this software, we modeled the application, and based on the generated models, we implemented the final application. The DLTOT was written in Java Servlets, Java Server Pages (JSP) and Mathematica Server Pages (MSP). We used an Apache Web Server and Tomcat Application Server for the interaction with the students. We also used Microsoft Access Database Management System (DBMS) as the General Database, and Mathematica and WebMathematica as the Mathematical Kernel. Using Servlets, JSPs and MSPs we implemented the complete model for the DLTOT and we developed a set of courses and tests for Signal Processing classes for the ECE department at NPS.

The DLTOT considers security as one of the important components of the application. The in-place security allows the system to control the access to the course notes. The set of courses in a secure environment allows us to offer an online test that is based on mathematical questions and challenges the student to "create" the answer, not merely select it from a list of choices. This answer neither selected from a set of presented alternatives nor from a set of true or false alternatives is far more revealing of a student's abilities. The test is based on a set of questions that are presented to the student in a browser and the answer is evaluated using the Mathematical Kernel. Logic for the grading is implemented in the test, and a grade based on the result obtained on the test is

presented at the end of the question. Using the developed logic, one can obtain grades ranging from zero to hundred, in steps of five points.

Since July 2001, the DLTOT has been working behind the NPS firewall allowing us to test its functionality. Results show the possibility of implementing a distance-learning course based on the DLTOT system. The DLTOT allows one to give tests online based on the material presented in the courses and to create rather new and challenging test questions. Some of the problems that ensued during design modeling and implementation are highlighted and all required documentation has been gathering in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. AREA OF RESEARCH

Dynamic Web as a complement to Static Web has become a major field of research. The Dynamic Web concept provides a more interactive behavior to a web site, and it allows a more intelligent response through a Hyper Text Transport Protocol (HTML) page. Dynamic Web responds to the user command, as opposed to following a predetermined path. This concept can be used to create distance learning with a capability of rich user interaction.

The Naval Postgraduate School (NPS) has been designing and supporting research on Distance Learning education, in addition to the actual Video Tele Education (VTE) for off-campus students. In classifying distance-learning education one should distinguish "synchronous education" from "asynchronous education." Synchronous augments face-to-face interaction. An example of synchronous education is text-based chat and video conferencing, such as VTE. Asynchronous education, on the other hand, allows users to access the information when they are logged-on at different times. This latter type of education has received most of the attention from researchers and is the approach used in this thesis.

The concept of distance-learning education has become a major area of research over the past decade, not only in Defense Institutions, but also in civilian universities. Presently, an increasing number of universities offer specific distant-learning courses, and even offer a Master's degree with most of their courses available on the web.

NPS has historically offered graduate education for the Naval Forces and for other Department of Defense organizations including international forces. NPS has half a century of experience and a superior faculty, which makes it a leader in postgraduate education. Over the next few years, NPS aims to take the lead in graduate distance-learning education.

In keeping with this objective, the Department of Electrical and Computer Engineering (ECE) at NPS has already designed a number of online courses with the same content as classroom courses. To achieve these goals an efficient method for evaluating the progress of a distance-learning student is essential. Currently most online tests are based on a limited multiple-choice model for which the student must select the correct answers.

Civilian institutions that offer distance-learning courses face the same problems when presenting a test on a web-based course. Professors are obligated to test students by using traditional "paper and mail" techniques. Naturally, these dated methods are labor intensive and absorb much of the professors' time to correct.

The NPS is supporting research to create a tool that can construct more challenging online tests. This tool will enable professors to create a meaningful test for a Digital Signal Processing course for the ECE Department at NPS.

B. RESEARCH QUESTIONS

New available technology for Dynamic Web development allows more effective interaction between the user and the distance-learning web site, directly from the browser. This technology could be used specifically to create test tools required for online courses and to present more appropriate test questions for a higher caliber of education.

This thesis investigates and proposes solutions for the issues summarized in the following three research questions:

- Based on new available technologies, what is the most appropriate solution for an online test for a Signal Processing course?
- Is it possible to create a student-assessed test online that is not multiple-choice based?

- Is it possible to evaluate the student set of answers and to give a grade that accurately reflects the students' knowledge of the subject?

C. STATE OF THE ART IN ONLINE TESTS

Developing an automated testing procedure is certainly an essential objective for online education. Unfortunately, to date, most testing has implemented a multiple-choice format, which is not appropriate for all applications. These tests are the most common because they are the easiest to implement. Since the correct answer is one of the choices displayed, the software can easily evaluate the score. Yet multiple-choice tests clearly lack the depth required for scholarly assessment, especially in electrical engineering courses.

Some variations of the multiple-choice format, called the "adaptive approach" have been created to address the inherent shortcomings of these multiple-choice tests. These variations are actually a sub-set of the multiple-choice models, but they add more levels of complexity to the test. For instance, the set of possible answers are divided into two or more levels of difficulty. If the test-takers answer the lower level correctly, they are promoted to the next level. In this way, it is theoretically possible to reach a point in which the student is exposed to a test progressively challenging and more meaningful.

Currently this adaptive approach is the cutting-edge technology in automated testing. Examples of the adaptive format are found in the preparations kits for the Cisco Certification [1]. This kit presents tests, which are based on the tester's previous knowledge and mastery of the subject matter. This approach in turn reduces preparation time allowing students to focus on specific areas rather than on the complete set of multiple topics covered on the formal test. The test questions become more challenging as the test takers prove their knowledge, so students advance to a more complex level only when they have scored well on the lower levels. The Adaptive format is also employed in the Test of English as a Foreign Language (TOEFL®), an examination designed to measure one's level of fluency in the English language. The following figure illustrates a page of the TOEFL examination. Note the multiple-choice alternatives:

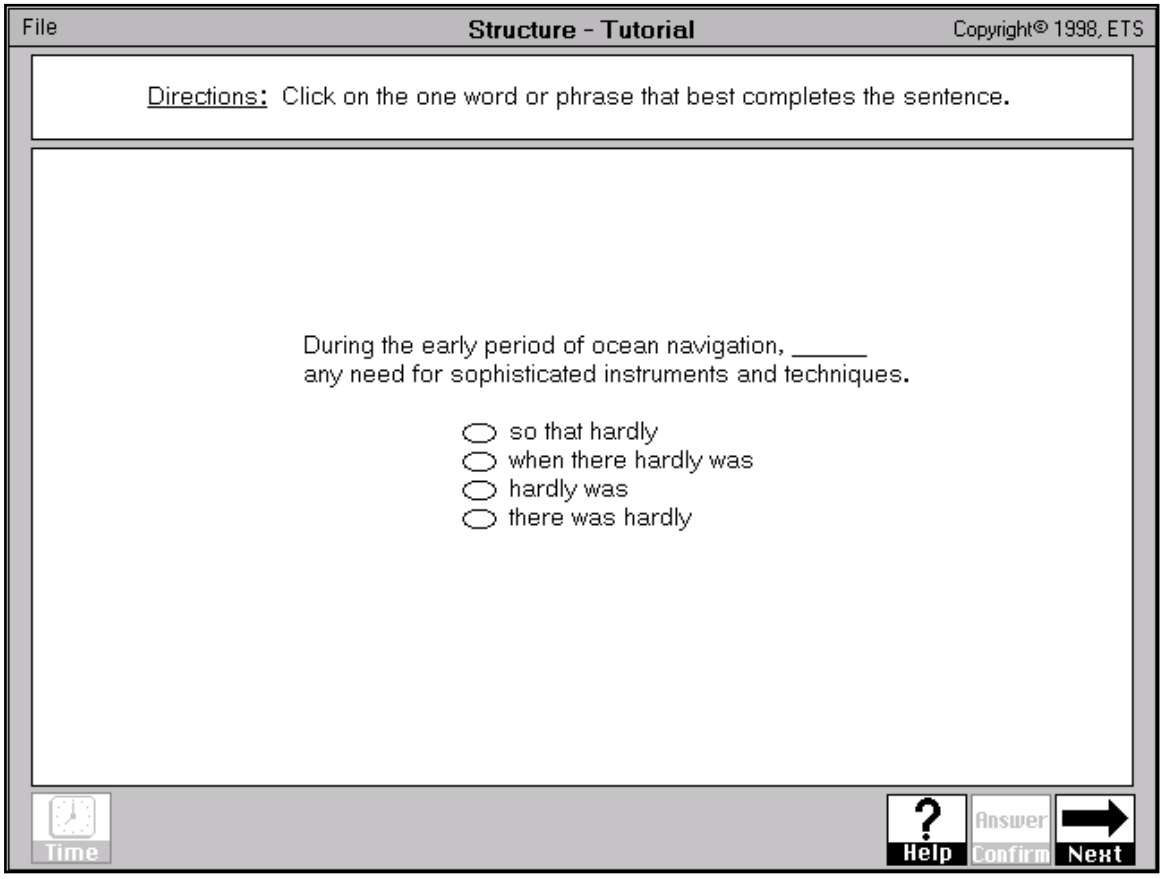

Figure 1. TOEFL Test, Administered by a Computer.

During the research for this thesis, we discovered other assessment approaches besides the adaptive approach, which are published in the Institute of Electrical and Electronics Engineers (IEEE). In those papers the ideas of distance learning are based on forming "virtual teams" [2]. Each team is equipped with the same software packages and share information via networks and video. These other approaches are based merely on practice and group effort, so no true testing components exist. Although these approaches are absolutely valid, they cannot assess the students' mastery of the subject.

One crucial aspect of online testing technology has been neglected: in many areas of science and technology monitoring the progress of the student by emulating a typical

drill problem assignment would be highly desirable and informative. In such cases, the student must answer a question by writing a mathematical formula instead of merely selecting it from a set of multiple-choice alternatives.

Certainly, a Master of Science program requires a deeper understanding of a subject matter to solve complex problems, a skill that cannot be evaluated by a multiple-choice approach. This paper addresses the problem of student progress assessment and offers solutions to the problem of online-testing for a Master's of Science in technical areas, such as electrical engineering.

D. ADVANCED MODELS

Very few current applications present approaches more complex than multiple-choice alternatives or adaptive approaches. Basically these current approaches use the web as a communication media and use an evaluation engine that could be custom-made or could use Commercial Off-the-Shelf (COTS) Software. Initially these applications were difficult to implement not because of their design but because they were implemented as stand-alone applications. The technology of communication between two applications had always been complex and challenging. Finding solutions to such problems was difficult because most applications were designed with a lack of interaction.

Until approximately 1996, networking programming was a difficult task because programming languages were not well suited to the networking environment. Also the interaction between applications was performed by a third-party code that attempted the interaction in a local and restricted way. The commercial application approach did not provide any external control or any communication between the various applications.

Due to such problems, the solutions to test online software prior to 2000 basically supported limited access to information over the network. An example was a project funded by the Department of Electrical and Computer Engineering of the University of Illinois, which created an application called Mallard™ [3]. This application

was to be used on the web as an interactive learning environment and was presented as "an educational tool for Digital Signal Processing" [3]. It also had support for testing online. The questions were written on extensions of HTML and the evaluation was performed using the PERL 5 programming language.

Mallard, as a tool, still exists but the orientation on Digital Signal Processing is no longer available. The actual Mallard demos give a solid impression on the excellence obtainable with enough time [4]. We considered Mallard as an inspiration to our work because some functional ideas inherent in Mallard guided this thesis. For example Mallard can:

- Control the test results, student by student.
- Work on a user-restricted environment
- Provide an integral solution by using class notes, as well as tests on the topics presented.
- Allow multiple repetitions of a test until the students obtain an acceptable score.
- Change the questions for each test.
- Give a detailed report on the students' progress in the course.
- Does not require any interaction from the professors, except to create the questions and to check student progress.

E. ORGANIZATION OF THE THESIS

This thesis is organized as follows:

Chapter I outlines the scope of this thesis and presents its organization. Chapter I also reviews previous research in this area and also presents the current online tests and specifically on Digital Signal Processing tests.

Chapter II analyzes the functional definitions that the tool requires in order to be constructed. In this chapter, we define all requirements necessary to obtain the desired functionality.

Chapter III analyzes the available technologies to implement the functional requirements outlined in Chapter II.

Chapter IV shows the modeling and implementation aspects of the solution using the technology specified in Chapter III. Chapters III and IV are intended for the next developer who may construct or extend the research.

Chapter V presents the applications of this system to a Signal Processing case. A set of course materials and the tests on these materials are analyzed. Then the structures of the test questions are discussed.

Chapter VI contains the conclusions and recommendations.

THIS PAGE INTENTIONALLY LEFT BLANK

II. GENERAL SPECIFICATIONS FOR THE APPLICATION

A. INTRODUCTION

In Chapter I we introduced existing online testing solutions; in Chapter II we specify the type of application needed to solve the problem outlined in the previous chapter. First we describe the application at a high level of abstraction looking for its modular functionality, and later we analyze its interaction. At the end of this chapter, the reader will have a complete idea of what the modules of the application are and how they work.

B. GENERAL SPECIFICATIONS

This application will have the name "Distance Learning Test Online Tool" (DLTOT). We consider "autonomy" a main characteristic of DLTOT. Autonomy means that the system must perform its tasks with the minimal intervention from a human. Except for the server maintenance or required upgrades, the application must be auto sufficient.

Since this system will serve the purpose of distance-learning worldwide, the application must be functional through the Internet, seven days a week, twenty-four hours a day. Also it must be available for student's access through a wide variety of workstations and speeds, ranging from home personal computers (PCs) through a standard telephone line to workstations at the workplace in a local area network.

In order to accommodate all users, this application has to support 56 Kbps telephones lines and be compatible with the most popular Operating Systems (OS), such as Windows, Macintosh, Unix, and Linux.

C. APPLICATION COMPONENTS AND REQUIRED TASKS

The DLTOT is intended to be a complete distance-learning tool with emphasis on the testing tools developed. Following this idea, we define a set of components that cover the complete functionality of a distance-learning site with testing online capabilities.

The figure below shows the main components of the application:

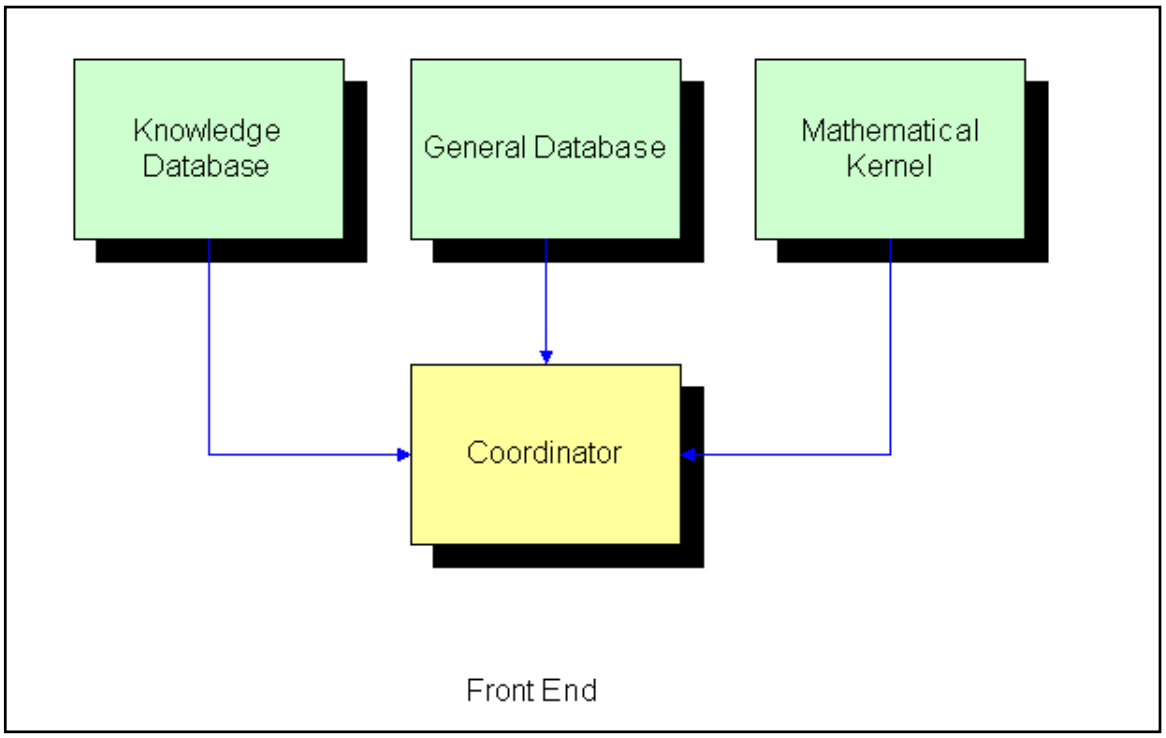

Figure 2. DLTOT Application Architecture.

1. The Knowledge Database

This database is a set of preprocessed information in digital format that will serve as the base for the online courses. It will be the responsibility of the course coordinator to generate it. From the Knowledge Database a set of processed information, which can be distributed to the students, must be constructed. This will be the material for a set of courses in which the students will be enrolled.

2. The General Database Engine

This is a Database Management System (DBMS) that holds the General Database. This database will control the students' course enrollment, the course progress and course results. It will also store the set of tests questions, and it will be able to extract statistics about a certain course or a particular student. Other required capabilities: it must be able to control the courses that a student is taking and the pre-requisites for the courses. The Coordinator component (to be defined below) must be able to retrieve information from the General Database, for example the test data for a particular student.

3. The Mathematical Kernel

This component accepts inputs from the Coordinator, evaluates them, returns the answer, manages and manipulates algebraic expressions and computes accurate solutions from a wide range of possible answers to the same question. For example, we can ask a test-taker to give an equivalent expression to $(x+y)^2$ and the student can answer this with many different expressions like x^2+y^2+2xy or $y^2+2yx+x^2$, both answers are correct and the Mathematical Kernel must know how to evaluate these kinds of expressions.

As supportive requirements it is necessary to build an easy-to-maintain Mathematical Kernel, one that permits variations of possible test questions. Also the Mathematical Kernel will serve as a mathematical tool that presents mathematical examples and concepts during classes.

4. The Coordinator Level

This component will serve as the confluent point for the information managed by the system. The Coordinator will be in charge of the interaction between components. Each student will interact with the Coordinator, which will request information or will place new data into the different components. The Coordinator will perform the hardest

work by requesting information from each of the other components and giving the students' results.

D. FUNCTIONAL INTERACTION

Now that we have examined the different application components, it is necessary to investigate the functional interaction of each component. This is presented as an example of a student who enters the system and subsequently takes a course and a test.

The first step involves validating the student through identification and a password. This is done by presenting a login window and requesting the student's ID and password and comparing this information with the data stored in the database. The student is then accepted or rejected. The Coordinator and the General Database are involved in these steps.

Once the student is validated, a list with the currently enrolled courses is presented. The student selects a particular course. Here again the Coordinator and General Database are interacting. When the course page is displayed, the Knowledge Database and the Coordinator are interacting in every step.

Once the student decides to take the test, making an appointment is required to ensure availability of server resources and to control the test time. The interrelation of the General Database and the Coordinator produces the test. During the test, the Mathematical Kernel is interacting by receiving and evaluating information from the Coordinator. The Mathematical Kernel will receive both the student's answer and the correct answer from the Knowledge Database, both originated from the Coordinator module. The results will be sent to the Coordinator, which depending on the result will be able to accept the answer or present the student with other equivalent question. In either case the answer will be graded.

E. SUMMARY

Chapter II has outlined the architecture and functionality of the distance-learning test online tool, DLTOT, showing the intended functionality of each module from a general to a more detailed description. Four main components have been explained: the Knowledge Database, the General Database, the Mathematical Kernel, and the Coordinator. Each of these four components has a clear and distinct functionality.

In the next chapter, we address the existent technologies for implementing the application. The election of each specific technology will be discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

III. CURRENTLY AVAILABLE TECHNOLOGY

A. INTRODUCTION

In the previous chapter we defined the functional behavior of the DLTOT based on: the Knowledge Database, the General Database, the Mathematical Kernel and the Coordinator. In this chapter we review existing technology that can integrate all the required components to construct the DLTOT. We review the best technology to be used in the future implementation, by discussing the main characteristics of each component and by analyzing the integration of each one with the rest of the DLTOT components.

B. SELECTION OF MATHEMATICAL KERNEL TECHNOLOGY

Our research revealed that the Mathematical Kernel was the most restrictive component in terms of being able to be controlled by an external application. Considering the kernel's restrictions, the first step toward a solution was to determine if the kernel had to be custom-made or if an existing COTS product could be used.

As expressed in Chapter II, the existing complexity of working with multiple equivalent inputs, like: $(\cos(x)+\sin(y))^2$ or $1-\sin^2(x)+\sin^2(y)+2\cos(x)\sin(y)$, makes the problem difficult for a program to handle. After much consideration, the custom-made kernel was evaluated as a less efficient alternative than the COTS. A second important aspect was the software support capability: if the program does not run as it is supposed to or if it requires a new capability when it is employed, the COTS solution is definitely better suited for this application. The COTS considered in this evaluation were:

- Matlab
- Mathcad
- Maple
- Mathematica

Research on the functional requirements showed that Mathematica was the only available kernel that accepts external control using an object-oriented language with networking support. The next table summarizes the obtained results.

Product	Accept External Control	Can be Extended to Other Languages	Accept Symbolic Input and Manipulation
Matlab V 6.0	Yes, limited using C	Can call Java programs	Yes, with toolbox
Mathcad V 2000	Yes, limited using C	Can call Java only externally	
Maple V 7	Yes, limited using C	Can call C and Fortran	
Mathematica V 4.1	Yes, unlimited using J/link and Java	Can call Java using J/link	Yes, built-in

Table 1. COTS Comparison.

The external control capability, listed in the table above, allows any program to solve a problem using the application's mathematical kernel. The result is returned to the external calling application. This is an excellent capability for developing applications that require reliable mathematical support. Unfortunately, COTS providers have not adopted this approach. One advantage that all published mathematical programs possess is their ability to extend the COTS application to external resources. Most of them offer Java support to connect to a remote resource, like a network device and advertise the advantages to extend the functionality of their products by allowing access to programming language libraries.

Mathematica was an exception to the internal-only control approach. It offers support to internal and external control. Two main design aspects allow Mathematica to offer this extra feature. First, the internal design of the application: Mathematica was

created as a stand-alone kernel and the graphical interface was written as an external program, so by design connecting to the kernel with a program is easy. The second design feature is the release of a free program from Wolfram Research Inc. that allows the user to call the kernel and to simplify the development of new applications by taking Java expressions and converting them to kernel expressions: This program is called J/link. Since the kernel can communicate to other programs using the Mathlink language, J/link uses this language to interact with the kernel.

1. J/link

Wolfram Research Inc. provides J/link as a software interface that allows the integration of Mathematica and Java [5]. This support is bi-directional because Mathematica users can access Java classes, and Java programmers can control the Mathematica kernel.

This product makes a vast difference when comparing the rest of the candidates to the Mathematical Kernel. Only Mathematica allows control from a Java application to its kernel. The Mathematica version 4.1 and J link 1.1.2 were selected as a mathematical kernel, due to the unique feature that allows the control of the kernel using an external control from Java Programming language. Since the mathematical kernel imposed the programming language for the application, the rest of the implemented solution was based on the Java programming language.

2. WebMathematica

When the solution based on Mathematica and J/Link was selected, a new product from Wolfram Research Inc. appeared in beta version: WebMathematica [6]. This web-based product uses Mathematica, J/Link and Java by providing a web interconnection with Mathematica using a browser.

When Wolfram Research Inc. learned of the Naval Postgraduate School's interest, they offered NPS to become a Beta tester for their new product. NPS accepted the offer

and during the beta testing period we found that the product was better suited for online courses than for Mathematical Kernel. During the beta-test period a set of reports with the results were conveyed to Wolfram Research Inc.

WebMathematica provides interaction with the client (user) and with Mathematica by providing an HTML page. The client sends the information. Then WebMathematica evaluates it, and sends it back as a web page to the user. The main drawback of this approach is that all interactions with the kernel are made using web pages with no intermediate control allowed. This disallows any possibility to control the result from a third program, like the Coordinator in our design.

A preliminary attempt was made using an equation editor placed at the client side. This editor would be connected through J/link to Mathematica. This approach gave full control over the kernel response by using a Java code. The solution was feasible but had a security drawback: for proper interaction with Mathematica, the client using a Java application had to be able to connect directly to the web server without security controls. The application required running the Mathematica kernel in the server side. This problem was later posted on the discussion sites of Mathematica and J/link as a possible source of a security attack.

Web Mathematica was implemented using Servlet technology and J/link. The main advantage of this approach was that any request from a client had to be made via HTML to the application server. The application server, which has security implemented, sends the requirement to J/link and this last one controls the Mathematical kernel. In this approach, J/link and Mathematica reside in the server side and all the external communication is between the browser and the server.

This approach did not present the problem of security observed in NPS's implementation. Because of this, WebMathematica was reconsidered and a solution to the interaction between the evaluation request and response was found and implemented as a test. The main idea was to be able to evaluate an expression without the web Mathematica front end. This new use of the product required changes in the

WebMathematica product, the next version (V0.93) still came in beta with support to session objects, which is the key element in the evaluation manipulation.

To present information, WebMathematica uses Mathematica Server Pages (MSP). The MSP is a type of HTML code where Mathlets are embedded. Mathlets are special tagged code processed by the server. This code is presented to WebMathematica and by using J/link, the kernel is triggered to evaluate the request. The result of the evaluation is sent as an HTML page to the client, in this case using MSPs. Figure 3 shows the code in a MSP. Note the similarities with HTML and the tags used to signal a Mathlet in this case highlighted in bold and in color blue. In this example and expression $(x+y)^4$ is expanded and the result presented to the user.

```
<HTML>
<HEAD>
<TITLE>Polynomial Expansion</TITLE>
</HEAD>

<BODY text="#171717" bgcolor = "#6e9fa6">
<br>

<H1>Polynomial Expansion</H1>

<FORM ACTION="Expand1" METHOD="POST">
Expand
<INPUT TYPE="TEXT" NAME="expr" ALIGN="LEFT" SIZE="10" VALUE =
"<%Mathlet MSPValue[ $$expr,  "x+y"] %>"
>
to the
<INPUT TYPE="TEXT" NAME="num" ALIGN="LEFT" SIZE="3" VALUE =
"<%Mathlet  MSPValue[ $$num, "4"] %>"
>
power.

<%Mathlet
MSPBlock[ {$$expr, $$num},Expand[$$expr^$$num], ""]
%>

<HR>

<INPUT TYPE="Submit" NAME="submitButton" VALUE="Compute">

<INPUT TYPE="Submit" NAME="submitButton" VALUE="Printable">

</FORM>
```

Figure 3. MSP Code.

Since an arbitrary Java program could not control MSPs, it was necessary to implement a top layer, which can use Java code and HTML. The technology that allows this is the Java Server Pages (JSP), which will be discussed later in this chapter. At present we are only interested in the result as a way of giving a general description of the behavior. A JSP was used to present a question to the student, recover the answer, and send the information to an MSP, which allows the evaluation. This type of MPS's was specially created for the case where there is no graphical output. The MSP's take care of the evaluation and instead of displaying the information to the student, it stores it in a session object that can be captured by the JSP and controlled by the Coordinator. This final approach does not have any security vulnerability and allows the Coordinator to control the application.

C. SELECTION OF THE GENERAL DATABASE TECHNOLOGY

Nowadays almost all commercial DBMS are equipped with Java support, which means that when using the appropriate driver, a Java program can access the data within the DBMS. The part of the Java Application Programming Interface (API) that provides a standard library for accessing relational databases is known as JDBC (incidentally, JDBC does not stand for an abbreviation or an acronym.) JDBC is the technology that enables the Standard Query Language (SQL) to send queries and to commute transactions to a DBMS. The JDBC specifications can send any type of SQL that the particular DBMS can understand. This means that the use of standard SQL in the query construction makes the DBMS independent. Since the main technology in JDBC [7] is already well known, we do not cover these details in this chapter.

Because the General Database only requires the storage of the tests results and the basic information about the students, a basic Database Management System (DBMS) with Java support is sufficient. Microsoft Access was selected as a testing platform because we had previous experience in this DBMS [8], it is easy to design and to run Relational DBMS. These characteristics allow rapid implementation.

D. THE KNOWLEDGE DATABASE TECHNOLOGY

The Knowledge Database, as applied in our research, only requires being in a digital format. Since this database will be converted to a final format, it only requires a Microsoft Windows compatible format.

E. SELECTION OF COORDINATOR TECHNOLOGY

The Coordinator component is a Java application that uses Dynamic Web Programming, which is the up-to-date technology when designing interactive applications on the web. The Coordinator capabilities can display dynamic web contents based on user behavior. A dynamic web can be obtained with Java by implementing a multitier technology based on Servlets and Java Server Pages (JSP).

1. The Multi-tier Architecture

The multi-tier architecture is an emerging technology based on a client and server approach in which the client does not require any installed application and all the transactions and logic are implemented at the server level.

The multi-tier model has many characteristics that make it a good choice in this case. First it allows a better scalability since it does not tie the client to particular software, and it can increase the efficiency of the network by just sending the client data in the HTML format. Finally, security is managed at the server layer with a good support from the application server and the web server software.

In the multi-tier architecture, a middleware component handles the requests from the client and by using a connection with different data sources, it can retrieve an answer for the client. Normally this response is HTML based, and therefore it does not require a particular application running at the client side. In a Java environment, this middleware component handles the connection with the data sources by using Java Servlets and Java Server Pages (JSP). Those Java components require an application server, which can understand the request from the client and give it to the appropriate Servlet or JSP. For

our application, a web server was implemented to handle static content. The intention was to give the final application complete support for static and dynamic web content.

2. Servlets

Servlet technology implements Java code that interacts with an application server and a client. Servlets are programs that run on the web server, more specifically in the application server, connecting the browser call to a database or application that could reside in the server or in a different place, as long as the server knows how to reach it. The client sends information in HTML to a Servlet through the application server. Then the Servlet queries or executes the application on the other side and responds using HTML code. The result is a web page. Usually this page does not have rich content in terms of presentation because the HTML code must be written inside the Servlet line by line, making the process long and error prompt. An alternative approach is by using JSPs.

3. Java Server Pages JSPs

JSP is a relatively new technology in which the Java code is embedded in the HTML code. Once the client calls a JSP, the application server extracts the Java code from the JSP, then compiles and runs it as a Servlet. The HTML part goes to the client and then the evaluation is embedded in the HTML code. This compiled version of the JSP stands at the server as long as the server is running, which means that no recompilation is required during the next call.

The main practical difference between Servlets and JSPs is the higher complexity supported by Servlets. Servlets are mainly Java code with HTML embedded. JSPs are used as a presentation layer. Although both technologies, Servlets and JSPs, can do the same work, in practice Servlets are the workhorses in this architecture, and JSPs present the HTML with the response to the client. JSPs also have another way to work with code, called “beans.” These beans are java code associated to the JSP for a particular format that allows the JSP a rapid interaction with their beans.

4. Web Server

A web server is not a required component for this solution but since any web site must support static content, it is considered part of the solution. For this implementation the "Apache Web Server" was selected mainly because of its well-known stability, good integration with the selected application server "Tomcat" (developed below) and zero cost. The security in any web server is an important factor when connecting the server to the Internet. Apache integrates security in a well-documented way, and "@Stake," a security consultant group, recommended this web server during a presentation at NPS [9] in 2001.

5. Application Server

"Tomcat" was selected as the application Server because it is one of the recommended servers to run WebMathematica. Also the Jakarta project [10] that creates "Tomcat" is part of the Apache organization, which represents good integration with Apache as a web server and Tomcat as an application server. Another consideration was its availability at no cost.

6. Java Language

Java is an object-oriented programming language, which allows the integration between all the components of this application. Java has an excellent documentation in computer literature and on the web; because of that, it will not be discussed in this thesis. The only required component to run Java is a Java Development Kit (JDK) which can be downloaded free from Sun web site [11]. In this application we used the JDK 1.3.1 from Sun.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. FORMAL MODEL AND IMPLEMENTATION FOR THE TESTING TOOL

A. INTRODUCTION

Implementing an application without proper modeling can create serious problems during the development phase and will certainly create problems when upgrading or extending capabilities. In this chapter the solution is modeled using standard, available methods. This phase allows the system to be analyzed for inconsistencies before implementation. The documentation produced in this phase is extremely important because it allows maintenance and highlights functionality to programmers when upgrading the system.

The modeling technique used for Java code was the Unified Modeling Language (UML). For databases we used standard table definitions and relations diagrams. For HTML, MSP, and JSP we used functional diagrams.

B. MODELING WITH UML

Before implementing any application, we needed to build a model that could most accurately represent the problem. Since our application can be abstractly conceptualized based on an object-oriented approach, using UML, a language for the modeling phase of an object-oriented problem is valid. The resulting model in UML does not relate to any particular language, because the conversion to the language is performed in the implementation phase.

UML allows modeling a system using a set of graphical notation and documentation permitting the correct representation of complex systems during the modeling phase. UML describes notation for classes, components, nodes, activities, workflow, use-cases, objects, states and relationships between elements. The ones used in this phase were use-cases, interaction diagrams called sequence diagrams and class diagrams. More information on this topic can be found in [12].

Use-cases describe functionality in a new system. The interaction described is between a human or machine, called an actor and the system. Each use-case is a single unit of meaningful work, and it has a description of what functionality will be built in the system. A use-case may include other use-cases or extend their behavior. The first part will be represented using use-case diagrams. The four-component model can be considered a general model that represents the complete system.

C. THE GENERAL MODEL

Figure 2 illustrates the four-component architecture for the system. Using UML and three-tier technology we can now introduce the general concept of notation in modeling our system. Figure 4 uses a simplified use-case diagram to indicate the requirements for controlling: the General Database, the Mathematical Kernel, and the Knowledge Database. An actor, the student, can interact with the application after the login module, and if the student is taking the course, he or she can read material or can take the test.

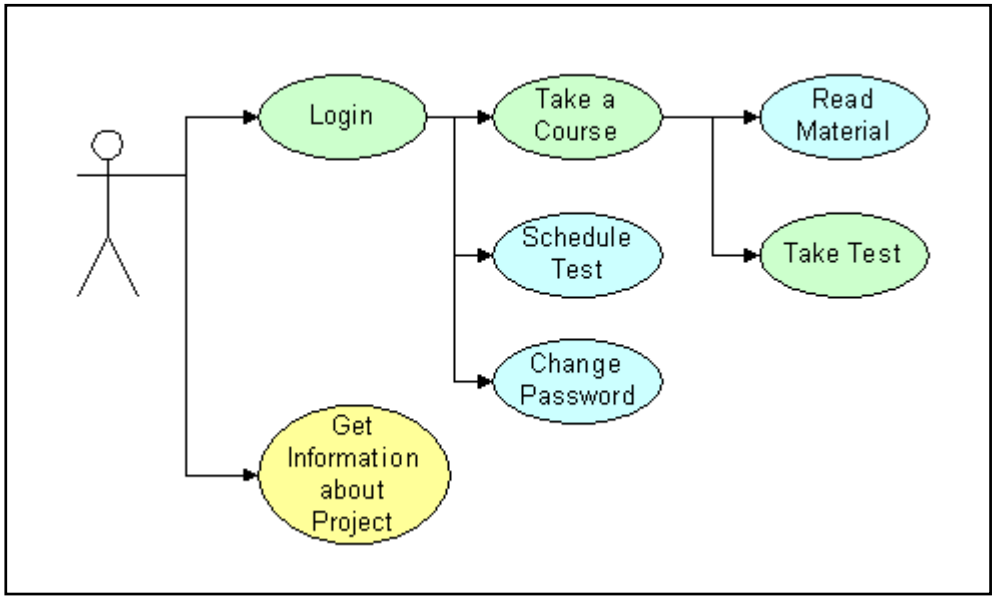

Figure 4. Generalization of UML Use-Case for DLTOT Application.

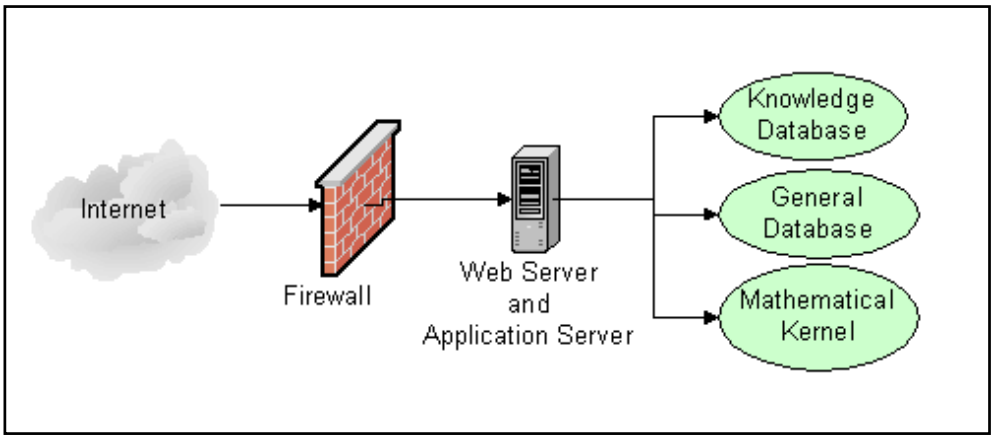

Figure 5. Multi-tier Architecture.

Figure 5 introduces the concept of multi-tier architecture in which the Coordinator level connects to the students (clients) via the Internet, using as input/output HTTP protocol. Note the presence of a firewall. We include this component to model a military site where security is part of the design. During the implementation phase, all this functionality will be converted to programming language code and COTS software and hardware configuration.

1. Modeling Database

General Database was modeled using relational database notation. The minimal set of required tables is specified below:

- **Students**: This table stores personal students' data.
- **Courses**: It stores a set of courses available and the corresponding physical location at the application server.
- **Tests**: It stores information about each question on each test.
- **Professors**: It stores data identifying the professors in charge of each course.

- **Student/Course**: Since many students can take many courses, a buffer table is required to represent the information.

Each student must be enrolled in at least one course. Once students are individualized in a course, they can be assigned a test. Professors can change the questions on the test. The next figure represents the set of minimal tables required and their relations.

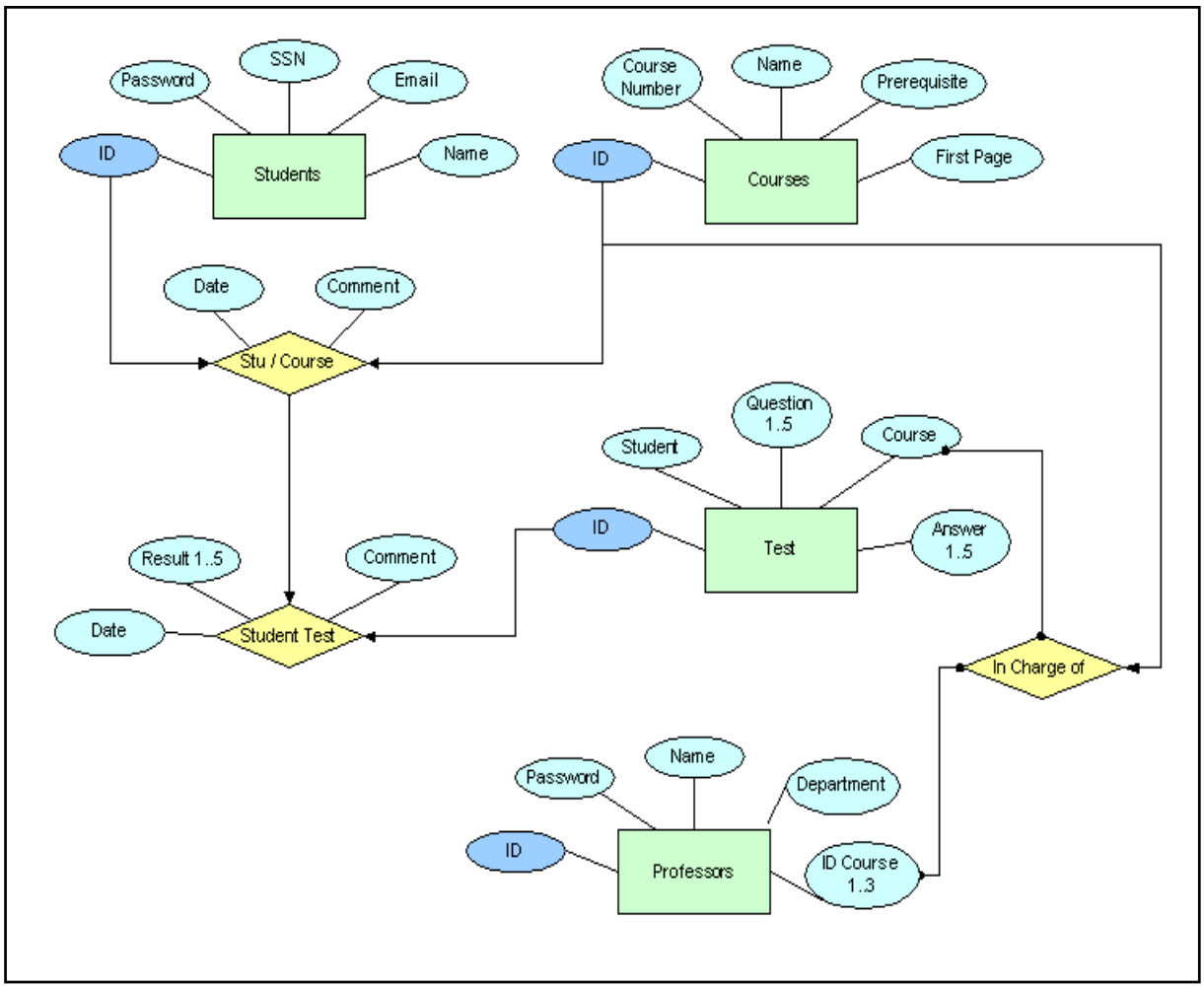

Figure 6. Database Model.

Some of the database supportive functionality was coded inside the DBMS. This approach, discussed previously, makes the database from the DBMS less independent, but also provides a level of security because only physical access to the database (at NPS) allows the use of these functions. Ideally, in a future version of the DLTOT the previously mentioned functionality must be implemented in a web-based approach. This approach will have the security challenge of allowing two types of users with the same interface, the students taking the test and the professors preparing the tests. To support future migration, the database is designed with two tables, one for students and the second for professors. By using the DBMS security setups, controlling the implementation better can be ensured.

The minimum user interfaces required are (shown in Figure 7):

- Manipulation of students and courses: namely, admission of students to a course, and deletion of students from courses
- Generation and maintenance of courses
- Generation of new tests
- Retrieval of statistical information from students and courses

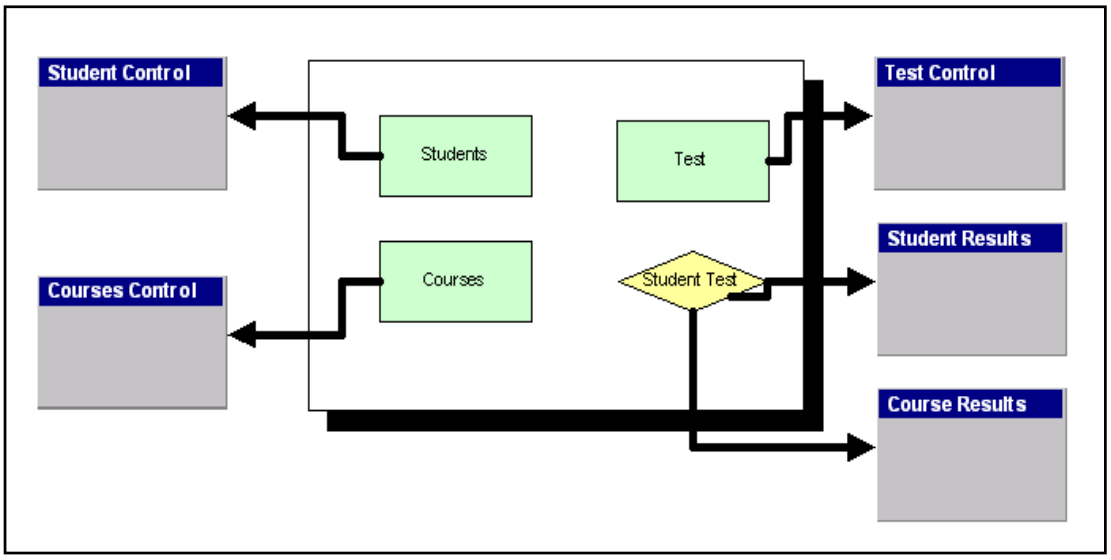

Figure 7. Interfaces Required to be Coded inside DBMS.

2. Modeling the Mathematical Kernel

The Mathematical Kernel requires evaluating the accuracy of the Coordinator-supplied answers. It must also support a set of course pages with demonstrations of concepts and a set of quiz questions.

Figure 8 illustrates this behavior in terms of sequential diagrams. The evaluation problem and the proof of concepts are displayed. The quiz can be considered as a type of test interaction.

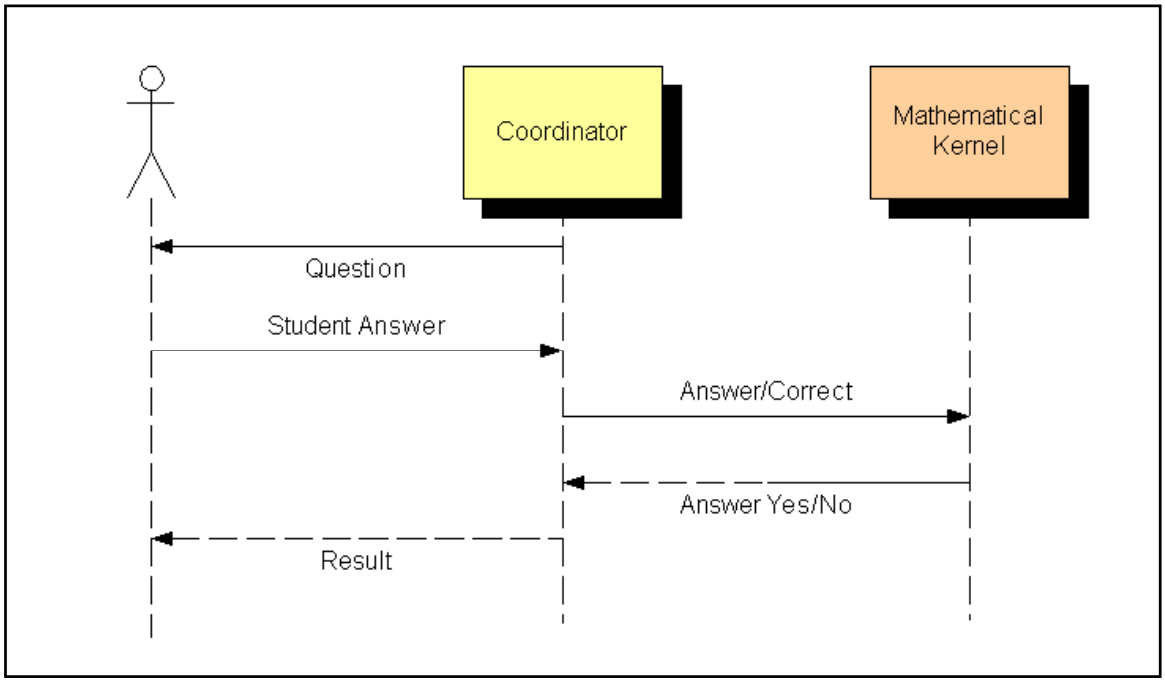

Figure 8. Test Interaction with Mathematical Kernel.

In Figure 8 the question is presented to the student, the student answers, the correct answer is collected, and the Coordinator then sends all this information to the Mathematical Kernel, which evaluates it. The answer is then sent to the Coordinator and the Coordinator presents the result to the student. Using the same behavior as in Figure 8, Figure 9 illustrates how a concept is presented to a student.

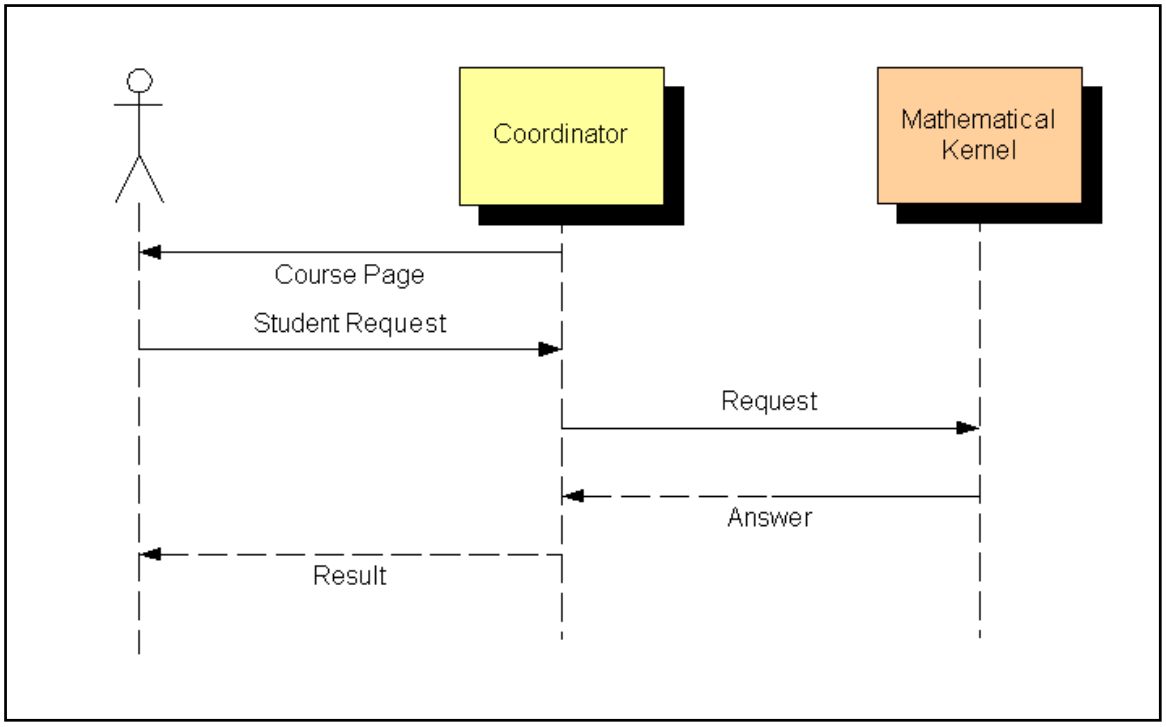

Figure 9. Demonstration Interaction with Mathematical Kernel.

3. Modeling the Knowledge Database

When modeling the Knowledge Database, we realized the importance of the interaction between the two human components. The first person is a professor who is in charge of a course. This professor provides the information and knows what must be included in a course. The second person is a web designer who can understand the functionality and limitations of a web server and an application server. These two persons have to interact to obtain a preprocessed Knowledge Database and from this database design the final course. This process depends on the Knowledge Database and its conversion to a set of protected and linked pages, which can integrate to the main application. The idea is to embed the course information in a set of pages that can only be accessed by a particular student enrolled in the course. Otherwise the page is not displayed.

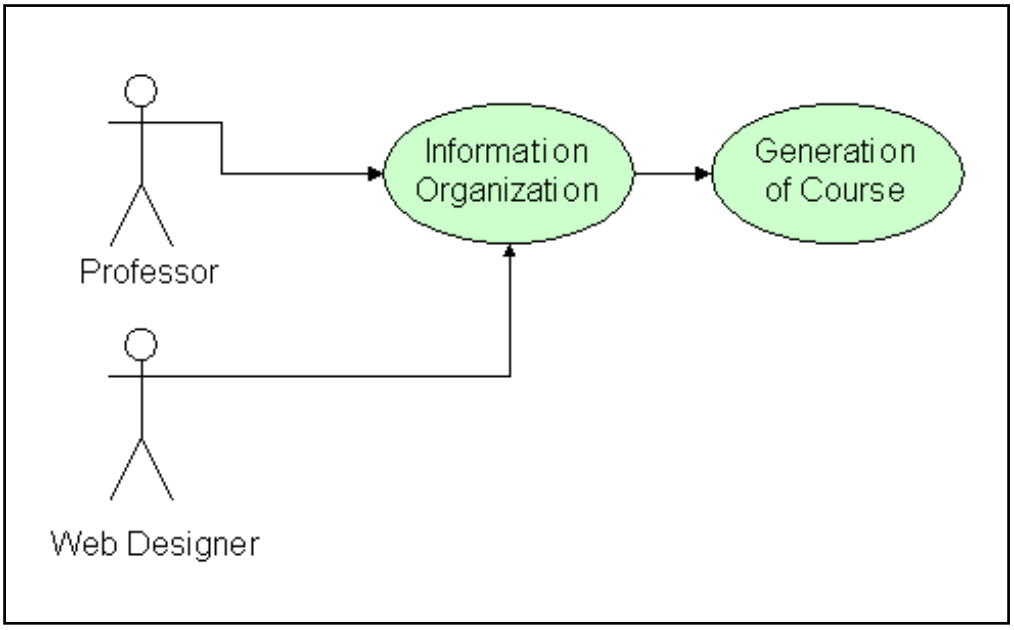

Figure 10. UML Representation of Knowledge Database.

In order to provide a more interesting and interactive experience we considered the use of Java applets to enrich the experience of the student. Applets are Java programs specially designed to work in the browser, allowing user interaction and, in most cases, they have a graphical interface. Regarding user interaction, the Massachusetts Institute of Technology (MIT) visited NPS this summer [13]. They introduced their advances in distance-learning using virtual laboratories where they can simulate a laboratory experience in a web-based environment. They can also control laboratory devices over the Internet, allowing a better access to limited resources. These approaches were not evaluated in this thesis but are an excellent alternative for increasing user interaction in a course.

When modeling the Knowledge Database, we considered the alternative of video and voice as a part of the final course. The research showed that this technology would not improve the learning experience significantly but would require a considerable amount of bandwidth on the Internet connection. MIT also confirmed our assessment of

the problem by specifying that their approach, for systems opened to the Internet, has limited audio and video implementation. MIT also indicated that the use of audio and video has been used only as a part of their internal network where they can use their high bandwidth connection; therefore, our decision was not to include this technology in the DLTOT model.

4. Modeling the Coordinator

The Coordinator was modeled as a set of objects that interact with the rest of the components. For each component, a set of rules that allow or reject the requirement is necessary. Most of the Coordinator functionality can be abstractly understood as policy enforcement in access control to the General Database, the courses, and the Mathematical Kernel. The main interface with the student is a set of static and dynamic HTML. Figure 11 illustrates this point:

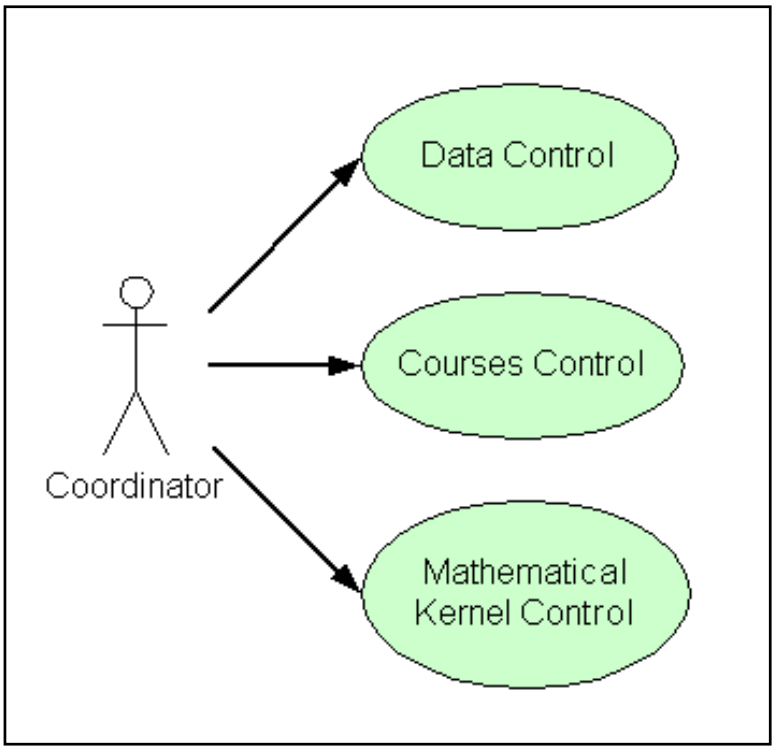

Figure 11. UML Representation of Coordinator.

In appendix "A" we include a model representation using UML class diagrams. During the rest of this subsection, we present the main class components that clarify the more complex diagram in Appendix "A." Figure 12 represents a basic Class Diagram. Note the structure of the diagram in which the first level represents the name of the class, the middle section represents a set of variables, and the lower section lists the functions available in the particular class.

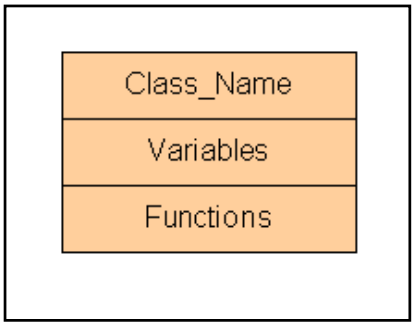

Figure 12. UML Basic Class Diagram.

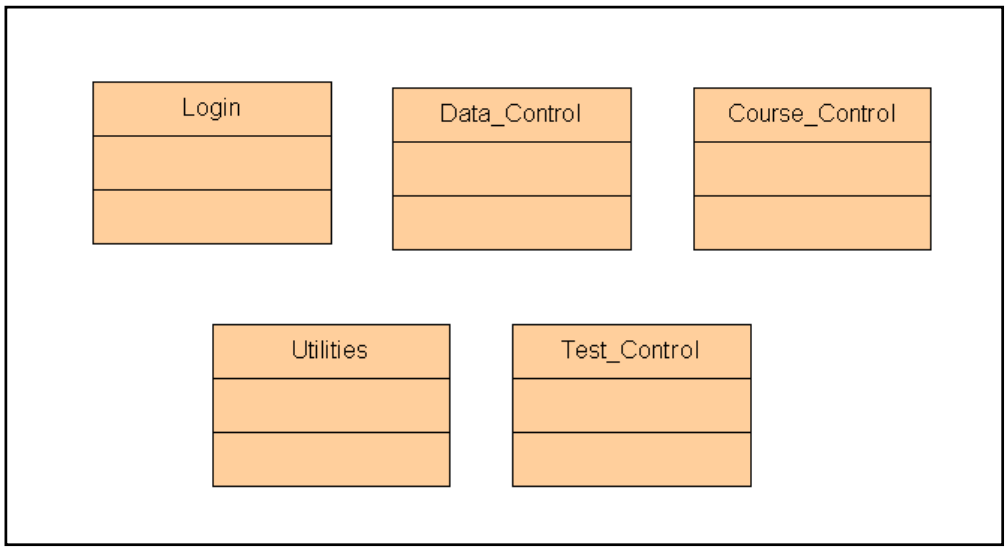

Figure 13. UML Class Diagram for DLTOT.

The classes presented in Figure 13 represent the main classes required for the DLTOT model. To simplify the understanding of the application neither the set of supportive classes nor the variables and functions for each main class are represented.

a. Security Components

Part of the Coordinator function is to establish a security monitor. This monitor will prompt for a login identification (ID), a password, and will control the information displayed. The password verification was modeled encrypted.

Figure 14. Graphical Interface for Login.

We decided to model a second security control for the Coordinator to restrict access to the different parts of the course. In a non-secure web site, any user can access any page; in this application only authorized users (students) can access the courses in which they are enrolled. However, when a student requests a page, the authorization check is performed and the student is authorized or denied access. Figure 15 represents the desired behavior. This security control is required on each page of the course and owing to this security control, we had to use JSP Technology. A different alternative on the page restriction could have been implemented using a web server or an application server access restriction. We did not consider this alternative because the administrative requirements on the configuration of the servers did not satisfy the specifications of the desired system expressed in Chapter II.

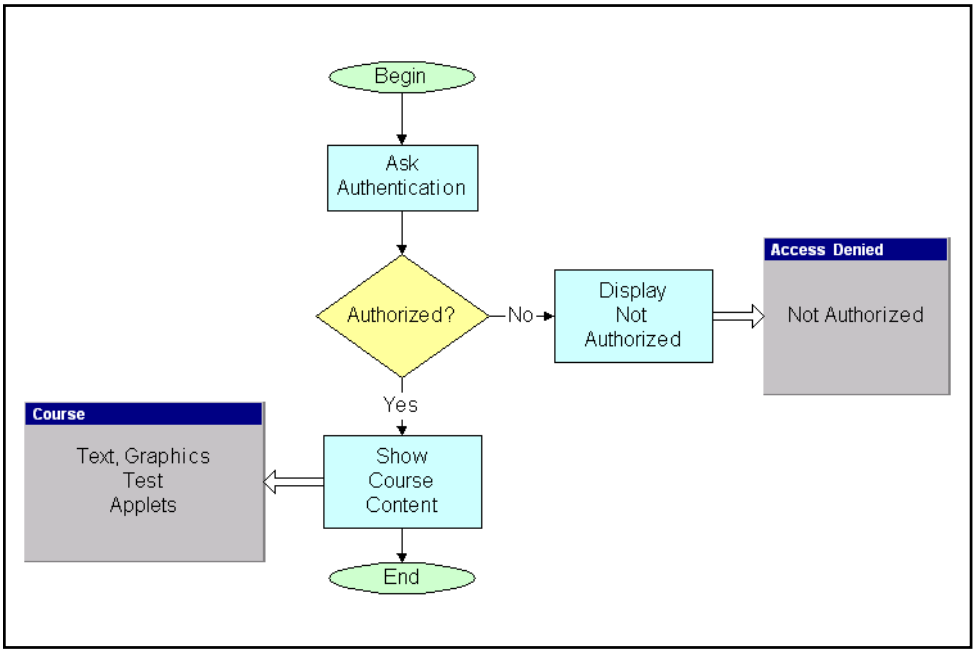

Figure 15. Security Logic for Course and Graphical Interfaces.

5. Modeling the Test

Each test consists of five questions, and each question has a follow-up question of the same type and complexity to be used in case the student cannot answer the main question correctly. In other words, each problem is composed of two components. The student has two opportunities to answer each component. During the test the first component of the question is presented to the student. If the student fails to answer it correctly, a second opportunity appears. If the second answer is incorrect, then the second question along with a hint appears on the screen. The student also has two opportunities to answer this second question correctly. This approach allows the examination to be complete and accurate in the test topics selected by the professor. The repetition of the question allows the student minor corrections, but with only two repetitions of the question "guessing" is eliminated. Two questions of the same type allow the students to be assessed on their real knowledge of the topic. A set of appointments is required during the test in order to provide enough resources at the server. Each student will be able to schedule a time slot of two hours to take the test.

6. Modeling the Grading

Any student that uses the allotted two-hour timeframe and answers all the questions correctly the first time scores 100%. If a student needs to repeat a question he or she receives a 5% penalty. If the student cannot answer correctly a second 5% penalty is applied. The second question on the same topic scores the same penalty. If a student fails all the alternative questions, a total penalty of 20% is applied.

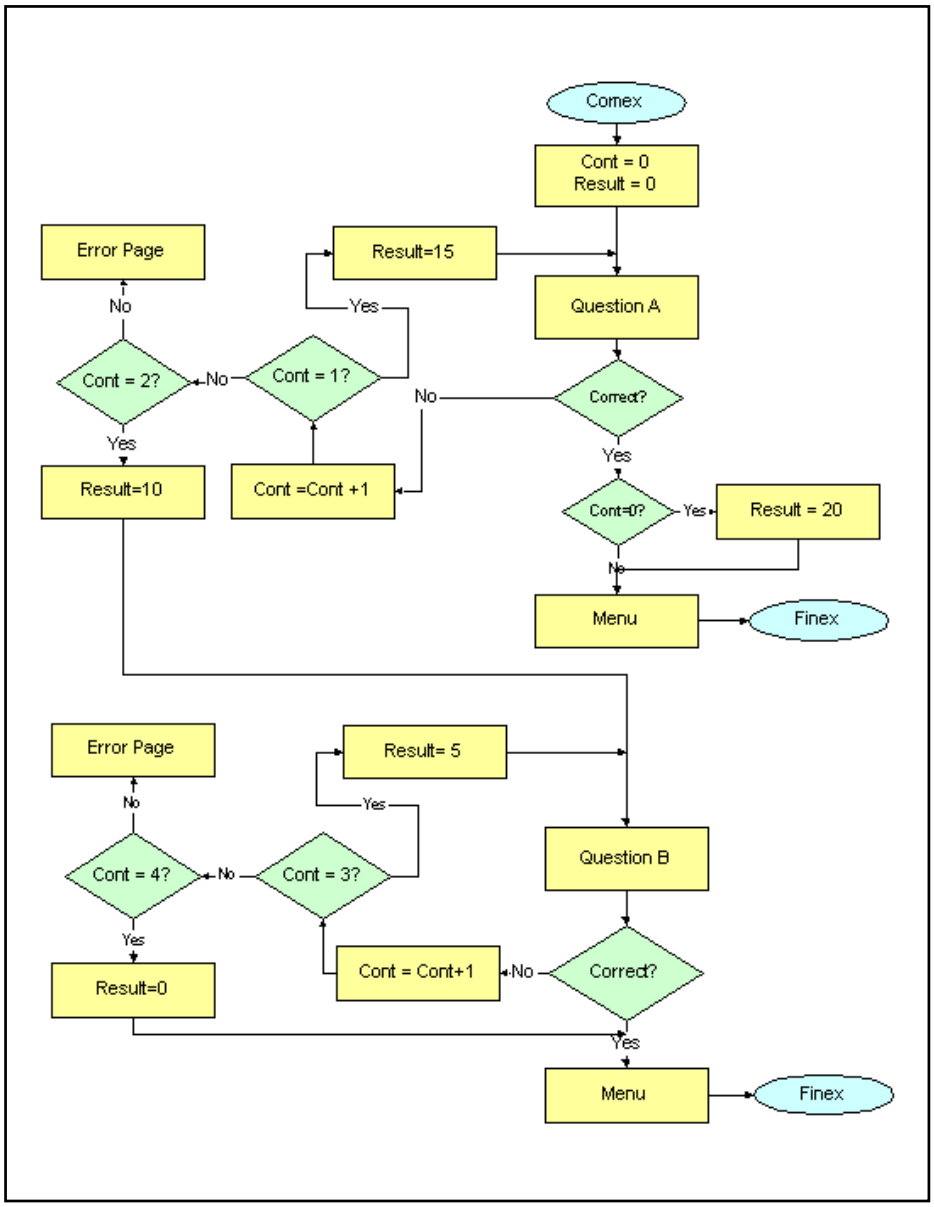

Figure 16. Grading and Question Logic for a Test.

A passing score will be 70%. Since a test has five questions, a student needs at least four correct answers and can fail one question two times and another question one time and still pass. With a two-hour time slot, students can request a second test if they are not satisfied with their results. This allows self-improvement by exposing the students to more questions and also helps them learn even more. Figure 16 shows the flow diagram used during the implementation of the test. This flow diagram represents the behavior of each question.

7. Summary of Modeling

We have seen a top-down analysis of the system by reviewing the main components in each phase, by illustrating the application design and by highlighting the security. This modeling phase shows how to build an abstract prototype, which can check for inconsistencies and can improve the design before the application is implemented.

D. IMPLEMENTATION

Before implementing, one must specify the configuration of the hardware and software used in the system. Part of the solution is based on a set of running applications that must be specifically configured. Once the running applications are in place, the last elements of the solution involve implementing the four-level architecture of the DLTOT and its interactions. We will describe the software installation because it represents a main component in the application and its complexity requires a basic explanation of each step in the process. In appendix "B" we include detailed installation instructions. In this chapter we will give enough information to understand the process and to install the software based on the hardware and the OS used at NPS.

1. Hardware and Software Implementation

In order to test the model, we used the following hardware and software configuration:

a. Hardware

- Pentium II 266 MHz
- RAM 192 Mega Bytes
- Hard disk 4 Giga Bytes using 2 partitions
- Network card 100 base T

b. Software

- Operating System Windows 2000 Professional
- Web Server Apache 2.3
- Application Server Tomcat 3.1
- Mathematica 4.1
- J/link 1.1.2
- WebMathemathica 0.93
- Java DSK 1.3.1
- Java IDE JCreator Pro 2.0
- Web Editor Microsoft Front Page 2000
- DBMS Microsoft Access 2000

2. Database Installation and Implementation

Microsoft (MS) Access 2000 is part of the Microsoft Office currently in use by all DOD workstations. Its installation does not require comments, and we adopted the software pre-installed in the application computer. The next step was to copy the developed database to the hard disk. The name of the file is "DSPTest.mdb." In this case we installed it in a directory called "D:\DSPTest".

The DBMS MS Access allows a rapid development of tables and front-end screens called Forms. According to the specified model, a set of tables and Forms were constructed. Figure 17 shows the results and presents the required relationships between the tables:

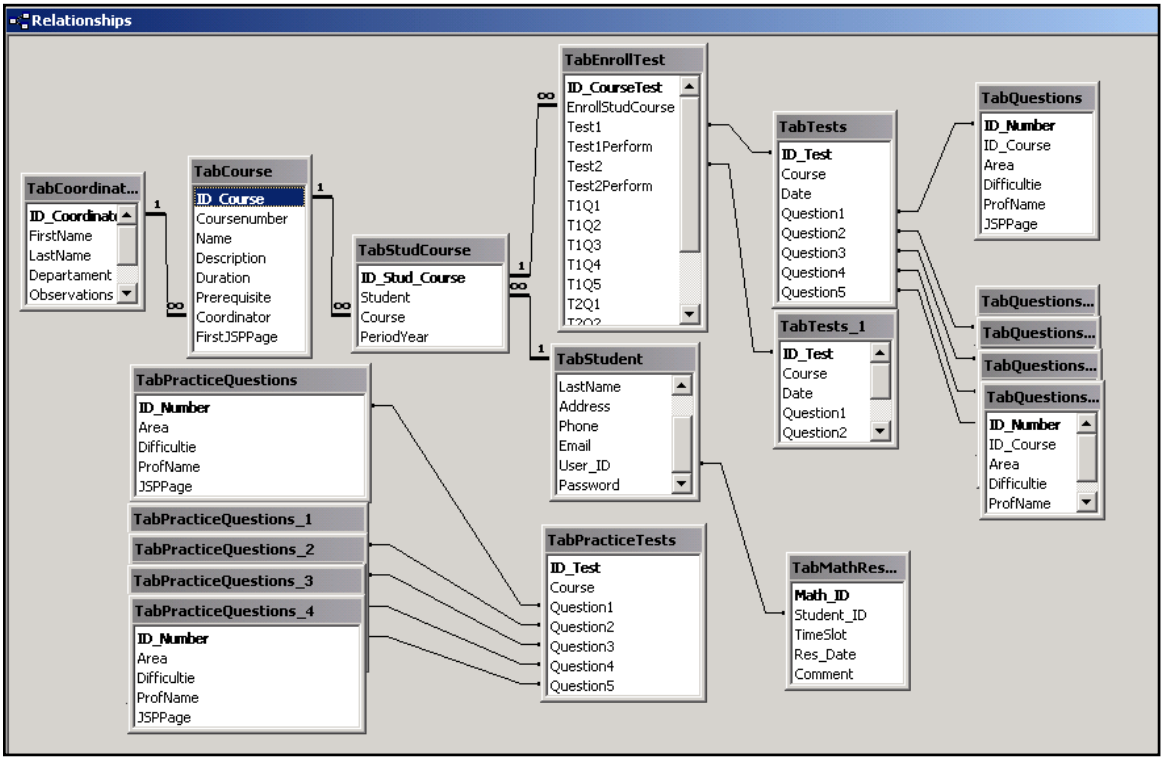

Figure 17. DSPTest Relation Diagram.

Figure 18 presents the set of tables implemented in the database. The model presented in Figure 6 was used for implementation and a set of extra tables was implemented to support the main model. Figure 19 shows the implemented Forms. The Form codes are for MS Access only.

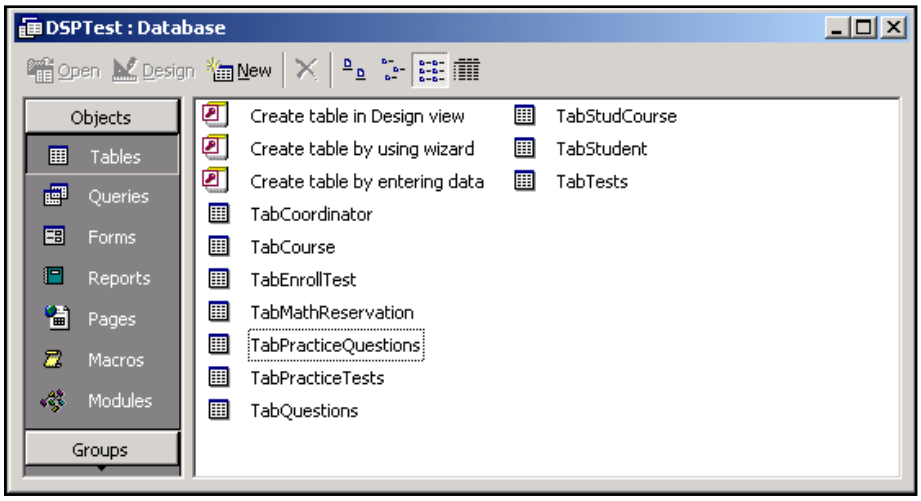

Figure 18. DSPTest Tables.

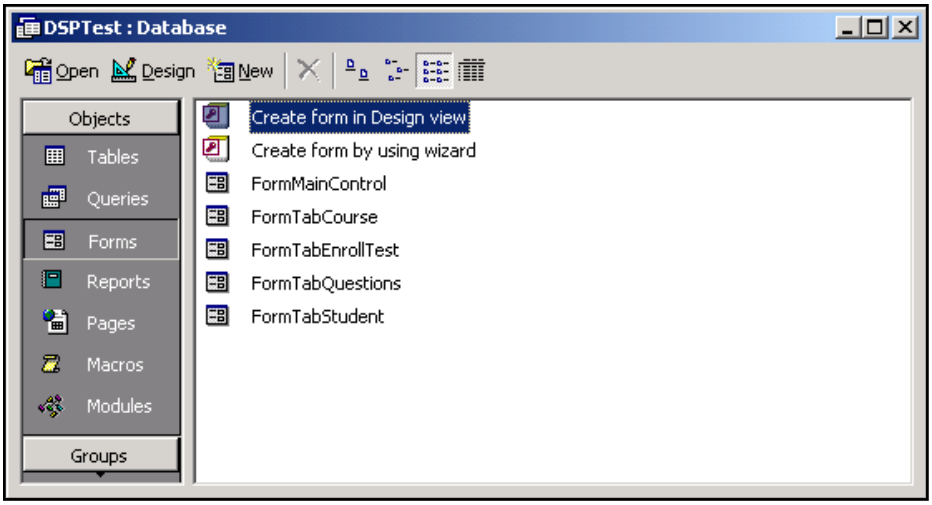

Figure 19. DSPTest Forms.

a. General Database Configuration for Java Control

To be able to access the database from Java, it is necessary to declare that the database is available in the server as an Open Database Connectivity (ODBC) data source. This procedure allows the Operating System (OS) to recognize the database. To add a data source ODBC driver from Windows 2000, one must be logged as administrator, select "the Control Panel," and click on "Administrative Tools." From the

set of administrative tools, one selects “the ODBC Data Source Administrator.” From here one can select the location of the database and give it a name. In this case the database declared is called DSPTest. Figure 20 shows this procedure:

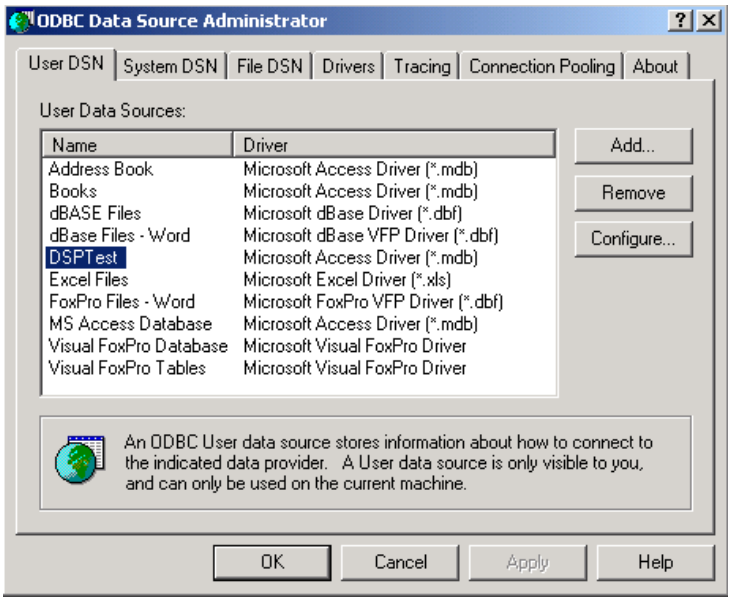

Figure 20. Declaring the General Database in the OS.

This process allows Java to control the database by just calling the JDBC/ODBC bridge driver that comes with the JDK 1.3.1 (the main distribution of Java).

3. Mathematical Kernel Installation and Implementation

In order to implement the Mathematical Kernel, Mathematica, Tomcat application server, Java, and J/link must first be installed. To install all these products, one must be logged as administrator (we are discussing the Windows 2000 installation only).

First, we installed Mathematica. The process was fully automated, and the only variation was the default driver for the installation was changed from C to D. As a second step we installed Java. This process was also fully automated except for the configuration of the CLASSPATH and PATH environment variables. We opened the Windows “Control Panel,” selecting “System,” and the “Advance” tag. Then by clicking on the

"Environment Variables" at this point, we could set the CLASSPATH, and PATH. In both cases the variables have to point to the directory where Java is installed, typically C:\JDK1.3.1\BIN. Note that in this case Java was installed in drive D instead of C. (more detailed information about the installation procedure is available at the Sun web site [11].) Figure 21 shows the configuration of the environment variables.

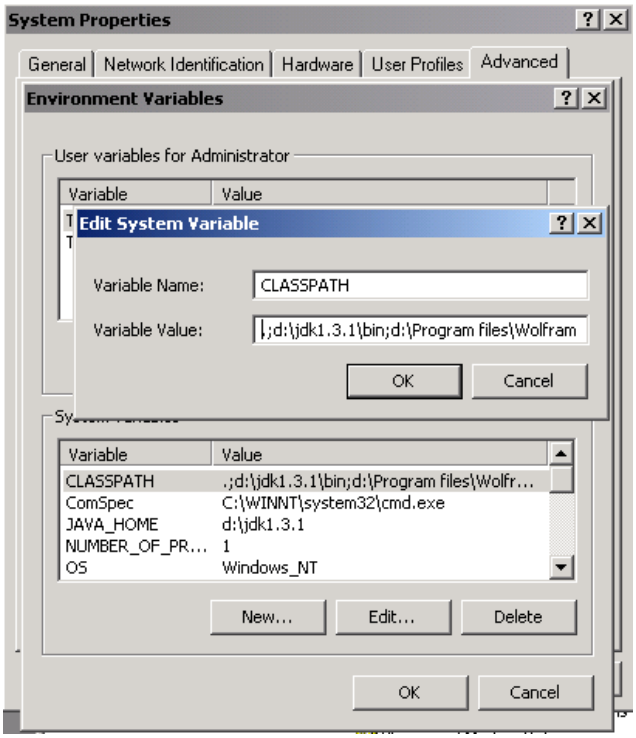

Figure 21. Configuration of Java Environment Variables.

Installing the web server was also a fully automated experience. "Apache" runs as a process in the OS. Because of that, it runs from the initial installation and does not stop unless the user specifies it.

The Tomcat application server installation requires copying the application files to a hard disk. No installation program is involved. It is necessary to decompress the files and to set or to create the environment variables TOMCAT_HOME and JAVA_ HOME, each one with the value of the respective installation directories. Figure 22 shows the appropriate configuration of the variable TOMCAT_HOME.

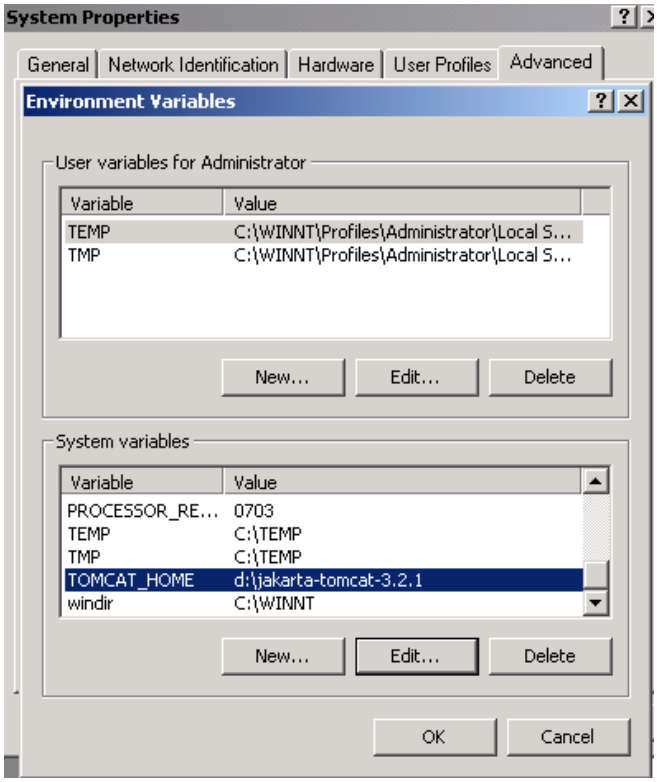

Figure 22. Configuration of Tomcat Environment Variables.

Since Tomcat was installed on "D:\Jakarta-tomcat3.2.1\" the variable TOMCAT_HOME had to point to this location. The same was valid for JAVA_HOME but it used the Java directory. Tomcat required an explicit startup procedure that was performed by running the file "startup.bat" located in the "bin" directory in the Tomcat application.

Installing J/link requires a more complicated configuration. Details of the procedure can be read on the Wolfram Research Inc. web site [5]. The procedure specifically requires unpacking the files in the Mathematica "AddOns/Applications" directory:

```
D:\Program files\Wolfram Research\Mathematica\4.1\AddOns\Applications
```

Figure 23. Where to Install J/link.

Since the distribution comes in a ZIP compressed file, unzipping it with the option "use folder names" enabled is necessary. From the files recently installed, one must find the file "JLinkNativeLibrary.dll" and copy it to the "C:\winnt\system32" directory and to set the environment variable CLASSPATH to point to the J/link directory. One must then indicate the location of the file "jlink.jar". Figure 24 illustrates the line, which must be added to CLASSPATH.

```
CLASSPATH=D:\Program Files\Wolfram Research\Mathematica\4.1\AddOns\Applications\JLink\
          SystemAdditions\JLink.jar
```

Figure 24. Line Required to be Added to CLASSPATH to Configure J/link.

The installation of WebMathematica is based on two main components: one is the servlet "WebMathematica-0.93.zip" and the second is the Mathematica "MSP-0.93.zip" component. The servlet component must be installed on the Tomcat directory under the WEB APPS directory.

```
D:\Jakarta-tomcat-3.2.1\webapps\
```

Figure 25. Where to Install WebMathematica.

Figure 25 illustrates the directory where the webMathematica-0.93.zip file is installed. To install the application it is necessary to unzip it with the option "use folder names" enabled. Once this procedure is finished, a new subdirectory called, WebMathematica will be created. The next step is to configure the deployment descriptor file "web.xml", which is found inside the WEB-INF directory in the Tomcat application server. Here one must specify the location of the Mathematica application. The next code was extracted from the actual "web.xml" configuration:

```
<init-param>
<param-name>wolfram.configuration_directory</param-name>
<param-value>d:\Program Files\Wolfram Research\Mathematica\4.1\
\AddOns\Applications\MSP\Configuration</param-value>
  </init-param>
```

Figure 26. Configuration on Web.xml File.

The Mathematica component is packed in the file "MSP-0.93.zip." This file must be decompressed in the Mathematica "AddOns/Applications" directory. Next, one must configure the way Mathematica will work. This is performed in the file "msp.conf", located in the configuration subdirectory in the new MSP application directory. In this configuration, the use of session objects with the browser is also specified. The next figure is a segment of the file "msp.conf" actually working:

```
# WebMathematica installation settings
MathLinkArguments=-linkname 'd:\\Program Files\\Wolfram Research\\Mathematica\\4.1\\MathKernel.exe' -linkmode
launch
MSPDirectory=d:\\Program Files\\Wolfram Research\\Mathematica\\4.1\\AddOns\\Applications\\MSP\\MSPScripts
ImageDirectory=d:\\Program Files\\Wolfram Research\\Mathematica\\4.1\\AddOns\\Applications\\MSP\\Caches
# To call methods on Java objects it is needed to set JavaObjectReferences to be true
JavaObjectReferences=true
```

Figure 27. Configuration on Msp.conf File.

4. Security Implementation

During the login procedure, the student is prompted for identification and a password, the Coordinator receives the answer, and a query to the database is placed only with valid ID. From there the password is retrieved. If the information was incomplete or the user ID does not exist or the answer is wrong, a "not authorized" page is retrieved. If

the user wants to login again, he or she must open a new instantiation of the browser. This measure disables guessing at the password or the use of automated software to attack the site. The correct identification and password will create a session object for the authorized user. This session object will carry information (credentials) necessary to navigate the user-authorized pages. Each page will check the appropriate authorization.

5. Testing the Appropriate Installation

Configuration checking is important when implementing any application. The test of all components becomes a key part that ensures the correct functionality of the complete system. Because of this, analyzing the correct behavior component by component is necessary. As a web server listens to the well-known port 80 in the server, the Tomcat application server works by listening to a configurable port. By default Tomcat uses port 8080. This can be used to check if the application server is running. By opening a browser and typing the URL "http://localhost:8080/", a page with the Tomcat logo is retrieved. This indicates the application server is running and listening to the appropriate port. To check the correct configuration of Java, one could run the examples provided with Tomcat. If a problem appears at this time, an incorrect Java configuration or incorrect installation may be the cause. If this occurs, the environment variables must be checked.

It is possible to test WebMathematica by typing the examples that came with it on the URL "http://localhost:8080/webMathematica/index.html". If an example does not run, the problem is located in the WebMathematica or in the J/link installation.

6. The Knowledge Database Implementation (The Course)

The component that gives coherence to the whole application is the set of courses from where the information is presented to the student and from where the student can answer the test questions. The set of courses was implemented using JSPs. A directory called "nps_dsp" was opened on the "D:\Jakarta-tomcat-3.2.1\webapps\" in order to

support the main application developed. In this directory the Servlets and the set of JSPs that were developed for the DLTOT were placed. JSP technology allows control over each page by only authorizing the students that have been registered in the correspondent course. The next code presents the security functionality implemented in the JSP. Note the Java code in blue, which checks for proper authorization and presents one behavior in each case. For simplicity, the security constrains and the rest of the code have been deleted. We believe this allows a better understanding of the main logic.

```
<%@ page session="true" %>

<%
        String studentName= (String)session.getAttribute("Name");
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Signal Processing 1</TITLE>
<BODY text=#000066 vLink=#666699 aLink=#990099 link=#3333cc bgColor=#ffffff
background="/graphs/sumtextb.jpg">
<FONT face="Verdana, Arial, Helvetica">
<H1 align=center><FONT color=#003366><IMG height=111
src="/graphs/top.ht1.jpg" width=81 align=left border=0> </FONT></H1>


<%      if (studentName != null ){ %>

        </body>
        Show the course
        </html>
<% }
else {
%>

        </body>
        Reject the user
        </html>
<%
}
%>
```

Figure 28. JSP Security Scheme.

Each JSP presents a rich set of graphics text and, on certain occasions, an applet that permits the student to practice. The JSP structure controls access to the information and also controls advancement in the course. Without the use of JSPs, and by using only plain HTML, the courses could be accessed without a selective control.

7. Test Implementation

Having tested all components, we ensured a successful installation. From here we built a set of Servlets, JSPs, MSPs and HTML code that would control the final application. Once all components were properly installed, the Mathematica Kernel could handle the requirements from the coordinator level. In order to do that, two steps were required:

- Evaluating the correctness of a symbolic expression in which the Mathematical Kernel is provided with the correct answer and the student's answer.
- Storing the evaluation so that it can be retrieved by the browser and analyzed by the Coordinator.

This procedure was performed by an MSP that does not retrieve information in a graphical or HTML representation. The MSP receives two values and a function and places the result in a session object where it is captured for future analysis. A JSP triggers the MSP and presents the question. The JSP presents the question, and then once it has the answer, it sends both the student answer and the correct one. It calls the MSP, which sends the question to Mathematica. Once Mathematica has an answer, it retrieves it to the MSP. At this point the MSP places the result in a session object and stops. The JSP only receives the session attribute and checks to determine if it is correct or incorrect by displaying a different HTML for each case and the result is stored in the database. If the answer is correct, a link to the next question is presented and a figure indicates a correct answer to the user. Figure 30 shows the code of the MSP that evaluates and retrieves the

result in a session object with scope request. This allows the result to be collected only when the whole transaction is finished and will be invisible for other transactions. Figure 29 shows the required JSP to trigger the evaluation.

```
<%@ page language="java"
        import="java.lang.*"  %>
<%
// Need to set MSPIncludedCall to make sure that the output is not closed
// by the MSP servlet
          String url =response.encodeRedirectURL("/nps_dsp/jsp/Correct.jsp");
          String equ = request.getParameter("equ");
          if ( equ == null)
          equ = "1";
          String val1 =  java.net.URLEncoder.encode( "0.5^n *u[n]");
          String val2 =  java.net.URLEncoder.encode( equ);
    String msp = "/MSP?$ScriptName=Test/dspFunSim&$IncludedCall=true&fun="
+ val1 + "&fun2=" + val2;
%>
<HTML>
<HEAD>
<TITLE>Test Demo</TITLE>
</HEAD>
                    Question goes here
   your Answer =
   <INPUT TYPE="TEXT" NAME="equ" ALIGN="LEFT" SIZE="24" VALUE ="<%=equ%>" >
  <INPUT TYPE="Submit" NAME="Submit" VALUE="Compute">
<jsp:include page="<%= msp %>" flush="true" />
<%
 String resultFromMSP = (String) request.getAttribute( "ResultFromMSP");
 if (resultFromMSP.equals("True")) {
%>
<p>
Correct
<% }
else {
          result="Incorrect"
%>
<p>
Wrong answer </p>
<% } %>
</BODY>
</HTML>
```

Figure 29. JSP Question That Handles the Evaluation Using MSP.

```
<%Mathlet
        MSPBlock[ {$$fun, $$fun2},
        res1 = ToString[Simplify[N[$$fun] - N[$$fun2] == 0]];
        $ServletRequest@setAttribute[ "ResultFromMSP", MakeJavaObject[res1]]];
%>
```

Figure 30. Evaluation MSP.

In Figure 29, the code implemented was simplified neither showing the security aspects, already presented in Figure 28, nor the grading logic. The implementation of the questions was performed in one JSP per question. This JSP implements all the logic defined during the modeling phase.

8. Grading Implementation

When the grading logic described in the modeling part was implemented we had a major problem in terms of the variables managed at the session level. WebMathematica could only manage the session object within the application directory. Outside this directory it could not share the variables. In other words, the test only worked in the WebMathematica directory, but the rest of the application was on the "nps_dsp" directory. In order to overcome this problem, we decided to move the entire WebMathematica application inside the "nps_dsp" directory. After this change, we had full control on the required variables. No other changes on the configuration were required. The logic for the test was coded in a general JSP that controls all the locations for the questions. This information came from a query to a database. The result of the query is a set of five JSPs in which each question was coded. Each question was implemented on one JSP and all logic for control was implemented on it. The main logic for each question was implemented using a set of internal variables that indicated the first use of the question and allowed to control the number of times the JSP requires the MSP evaluation. Once the evaluation returned a correct result or the maximum number of intents was reached, the JSP presented a link to the main test and the main test presented

the score for that question. Since the number of evaluations was maintained at the session level, obtaining more that four evaluations on one question was impossible.

E. SUMMARY

In this chapter, we have seen the modeling aspects for the application. We also analyzed the components required for the implementation and analyzed how these components performed. The logic for the test was presented and the technology was integrated in the DLTOT system.

This chapter illustrates the main architecture of the DLTOT application by showing the main model and the corresponding implementation. Using JSPs and its functional interaction with MSPs allows the efficient evaluation and control not only of the courses but also of the tests.

V. USING THE DLTOT APPLICATION FOR A DSP COURSE

A. INTRODUCTION

The implementation and use of the system probably best validates the design. In this chapter, we analyze the complete process in which a course is created, students are registered, and a test is prepared and given to the students. The implemented course covers material from the Signal Processing track at NPS. Formally, EC 2400 "Signals and Systems" and EC3400 "Digital Signal Processing" courses are used to implement material in this web-based course. To avoid repetition, only certain parts of the courses were implemented.

The test was selected to demonstrate the validity of the Mathematical Kernel process and grading. The Mathematical Kernel was also used during the course to enrich the student experience.

B. BUILDING A DSP COURSE

The main source of the information was a set of already prepared material from Dr. Roberto Cristi [14][15]. Material was facilitated in digital format and properly authorized by the author.

1. Knowledge Database

The knowledge database came from a Mathematica notebook [15], and a set of PowerPoint presentations and HTML pages previously used in Dr. Cristi's courses [14]. The identification process for the desired material was conducted with the professor and web-designer's participation. After the material was selected, the main problem was to convert it to the desired format.

The course uses JSPs to display information. As we saw in Chapter III, the JSP format accepts HTML code. Because of the HTML availability, the information sources were converted to HTML. Once converted to HTML, the next procedure was to use an

Integrated Development Environment (IDE) to integrate JSP and HTML. In this case we used JCreator Pro, which allows one to edit any source of code in a unique development interface. Even though many formats were used — Servlets, JSP, MSP and HTML— the IDE allowed us to work on a unique main project that contains all the rest of the sub-projects or codes. Another main use for the IDE was to allow copying and pasting from HTML to JSP and to be able to consult different parts of the code from a single interface. The JSPs used to implement this course had the same structure as the one showed in Figure 26. The course content was placed in the authorized area of the JSP (see Figure 28). Figures 31, 32, 33, and 34 illustrate this process:

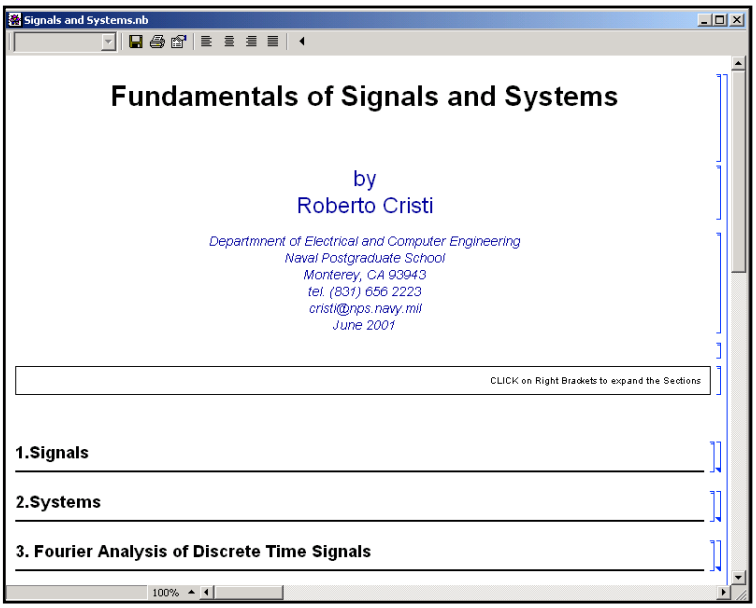

Figure 31. Knowledge Database Source in a Mathematica Notebook.

The process of conversion from the Mathematica notebook was simple because Mathematica allows saving in the HTML format by using the option "Save as Special." Figure 32 shows the result of the Mathematica Notebook converted to HTML. Figure 33 shows the JCreator IDE creating the first page of the course. Figure 34 shows the resulting course in a JSP format. Note that in Figure 34, the student is identified and already authorized.

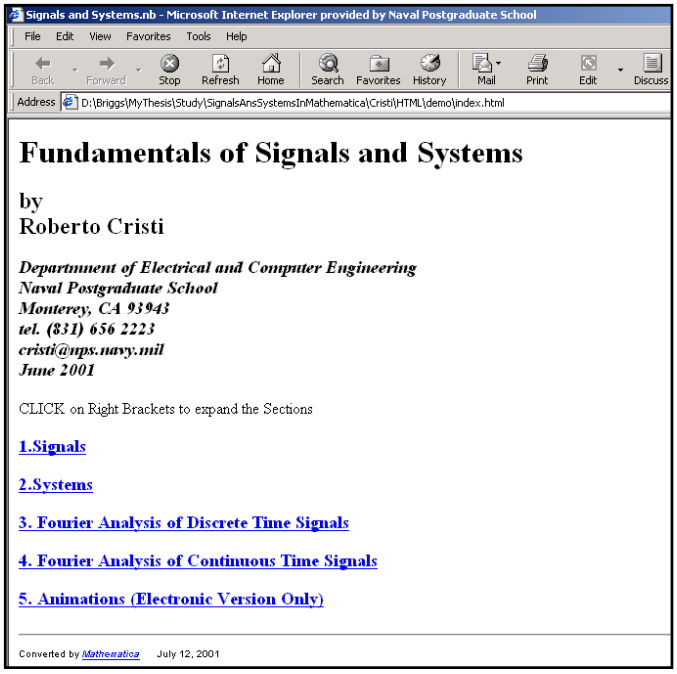

Figure 32. Conversion from Mathematica Notebook to HTML.

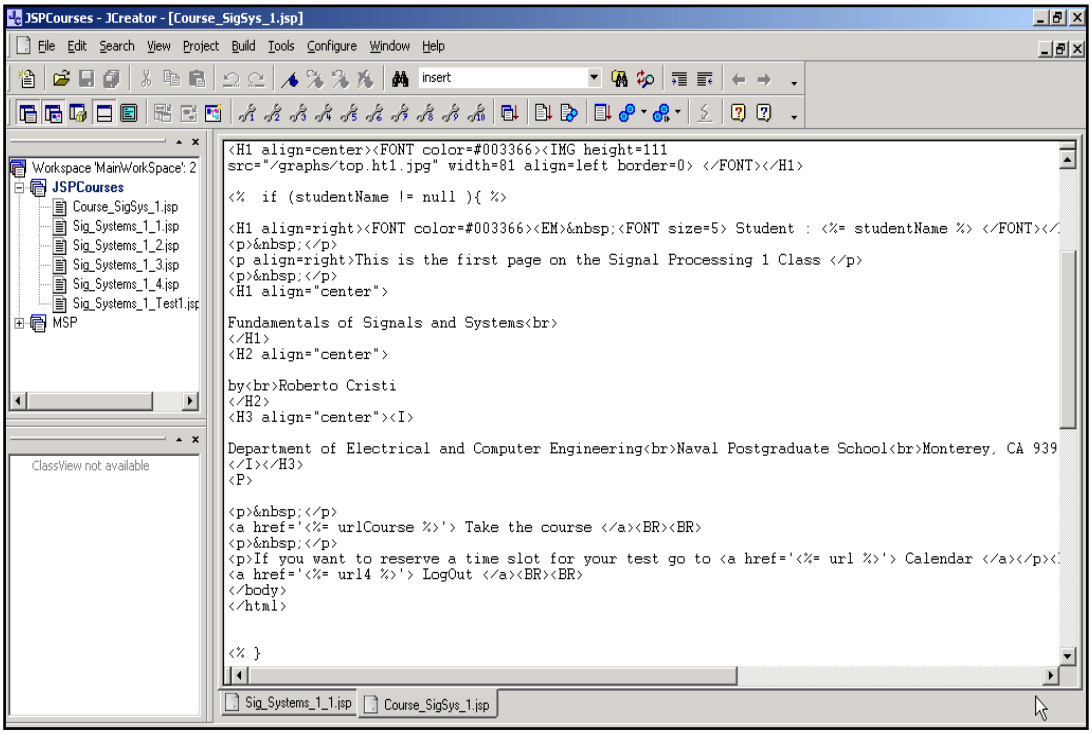

Figure 33. JCreator IDE Converting HTML to JSP Using a Template.

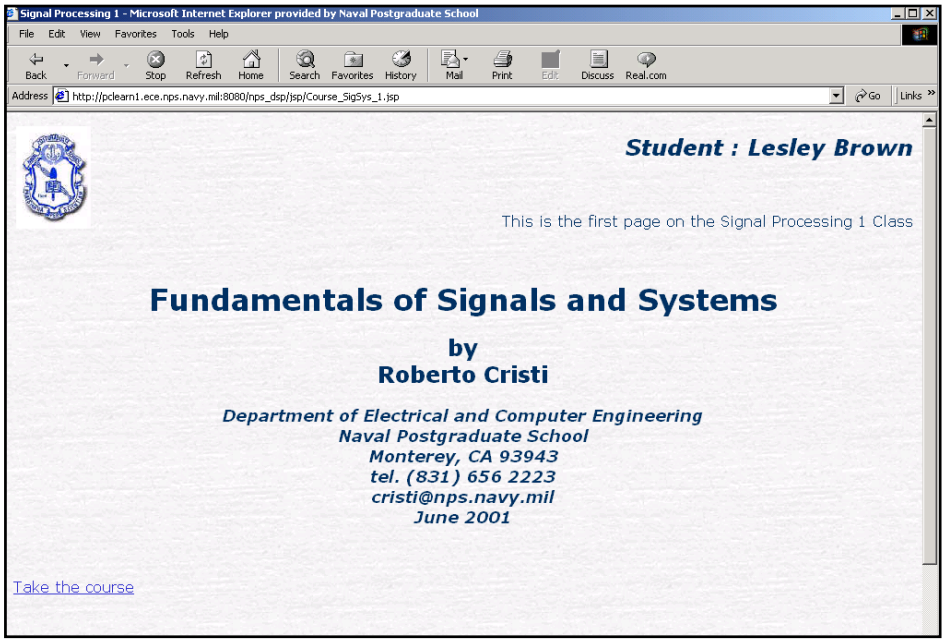

Figure 34. Final Course Using JSP, Coordinator, Interaction and Security.

2. General Database

In the General Database registering a new course and assigning the location of the first page of the course was necessary. Then we enrolled new students in the course. This process required the assignation of a login identification and a password. The two created parameters had to be send to the student via email. The email had to recommend changing the password as the first step in taking the course. A set of questions was prepared and registered in the test table. From here, the system knows where to look for the questions.

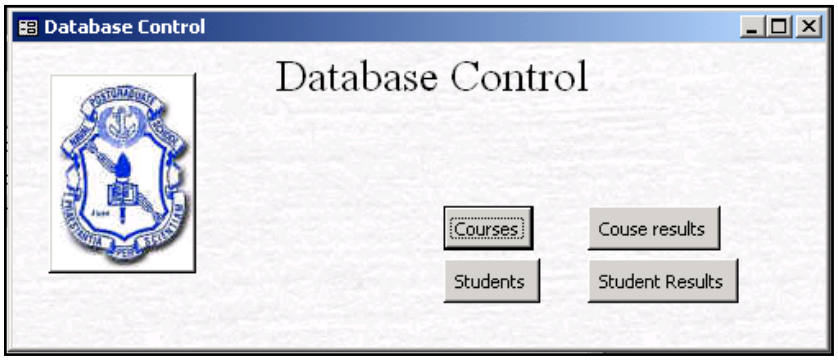

Figure 35. Main Database Interface in MS Access.

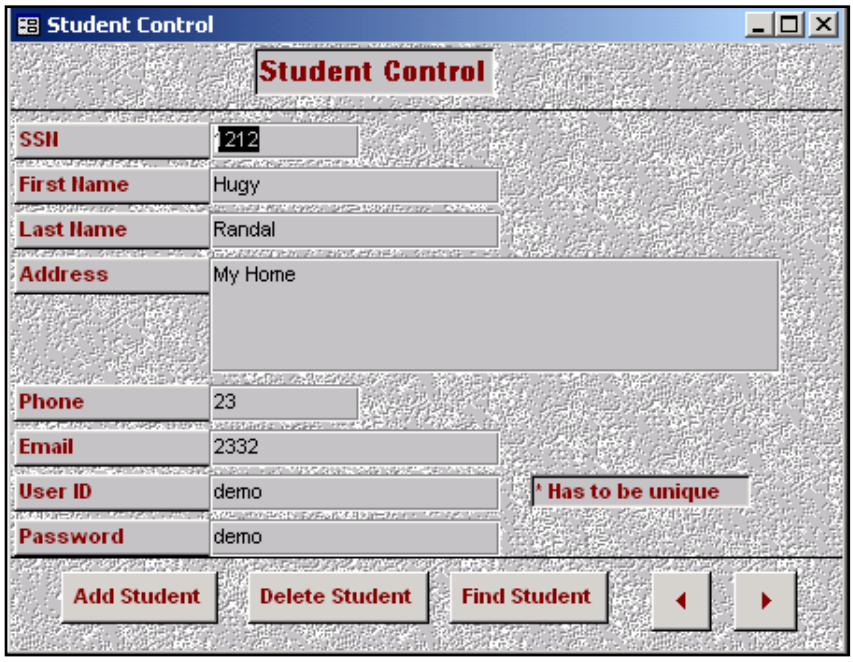

Figure 36. Course Control Interface.

Figures 36 and 37 present the interfaces in the DSPTest database where a student can be enrolled by assigning an identification and a password. These figures also show the interface from where the courses can be managed.

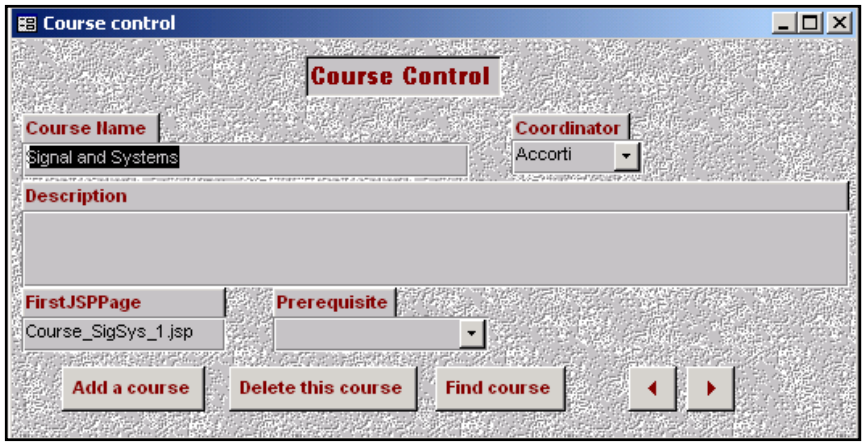

Figure 37. Student Control Interface.

3. Mathematical Kernel

The Mathematical Kernel physically has two different areas. The first is in the Mathematica directory where the MSPs are stored, and the second is in the application server directory where the JSPs are. Each of the JSPs stores a test question and can call an MSP to evaluate the students' answer. We generate a specific directory in which to place all the evaluation MSPs. This directory was called "test" and was under the Mathematica directory as shown in Figure 38.

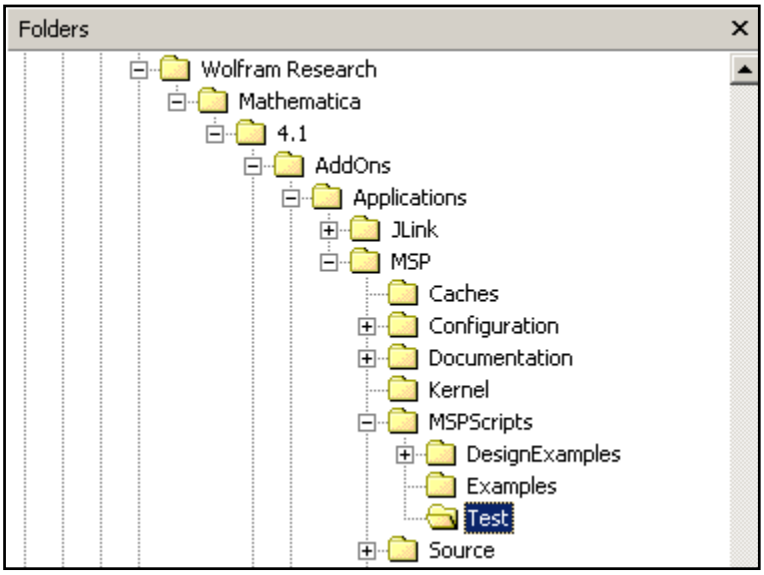

Figure 38. Location of MSPs for the DLTOT.

This directory allows identifying the main components of the application later in maintenance phase. The Mathematical Kernel was also used to present a Signal Processing example to the student. This process also required a JSP and an MSP.

4. Coordinator

Since the Coordinator was modeled and implemented as part of the functionality of the system, it did not intervene during the preparation phase of a course. The main use of the Coordinator becomes necessary when running the application.

C. BUILDING A DSP TEST

The construction of the test questions was based on the proof of concepts criteria. We decided the test questions had to be representative of the Signal Processing area and also able to be answered in a single line of a mathematical expression. Our main interest was to probe the idea of the correct evaluation of the mathematical expression, regardless of how it is written. In the next paragraphs, we will see how these types of questions can be presented to the student in a challenging manner.

1. Questions on the Test

The types of questions that we implemented are ideal for a direct answer. Most of the questions allow:

- Asking for an equivalent transform like Fourier, Z, Laplace
- Asking for expression equivalences
- Asking for the characteristics of a signal, etc
- Describing filter equations

We implemented a set of questions of this type by using MSWord and the equation editor. The principal advantage of this approach was that MSWord could save the question as a web page. In this type of format, the equations are stored as graphical files, and from here converting the information to a JSP is simple. Figure 39 shows a test question.

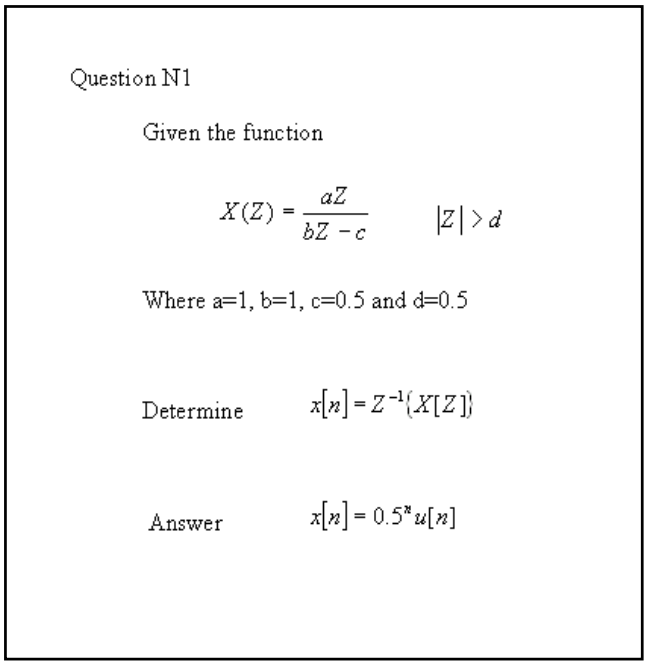

Figure 39. Creating a Test Question.

In this question, the student can easily use pen and paper and make the necessary computations. The answer could take the form $1^n/2^n * u[n]$ or some of the multiple variations like $u[n] * (Sin[\pi/6])^n$. Many answers can be obtained from the students and the Mathematical Kernel must accept all the correct ones. The results showed the kernel's correct functionality. One basic restriction is that the string must be written in Mathematica notation. This means, as an example, the sin(x) must be written Sin[x]. Other than this, the evaluation performed correctly, and we were able to obtain accurate results under any equivalent expressions.

We implemented the question inside a JSP where the grading functionality was controlled. We probed the correct functionality of the test logic and the results were highly satisfactory. If the student fails the first time, the question was presented a second time. An internal set of session variables controls the different parts of the test. This type of internal variables performs the same as physical tickets. If the student begins a test,

this is considered signaling with a session variable, and each time he or she enters a question the in-place control does not allow the student to leave the question without an evaluation. In other words, if a student does not answer a presented question, the student receives a score of zero.

Owing to the logic for grading a test, which was discussed in Chapter IV, a student can now answer and fail and then move on to a second opportunity. If the student fails a second time, an equivalent question is presented, this time with a hint. All of the previously described logic for evaluation was coded in a JSP. One of the observations that we made in Chapter III regarding the complexity of code when using JSPs versus Servlets seems to occur. If the JSP must be so complex why not implement its functionality as a servlet? The answer to this question was the necessary interrelation with MSPs. The way that MSPs accept being triggered did not work with Servlets. WebMathematica does not allow the use of a Servlet when calling an MSP.

Once all desired functionality was implemented and tested, we tried a second variation of the same type of question with a longer answer. Figure 40 shows the question and the answer.

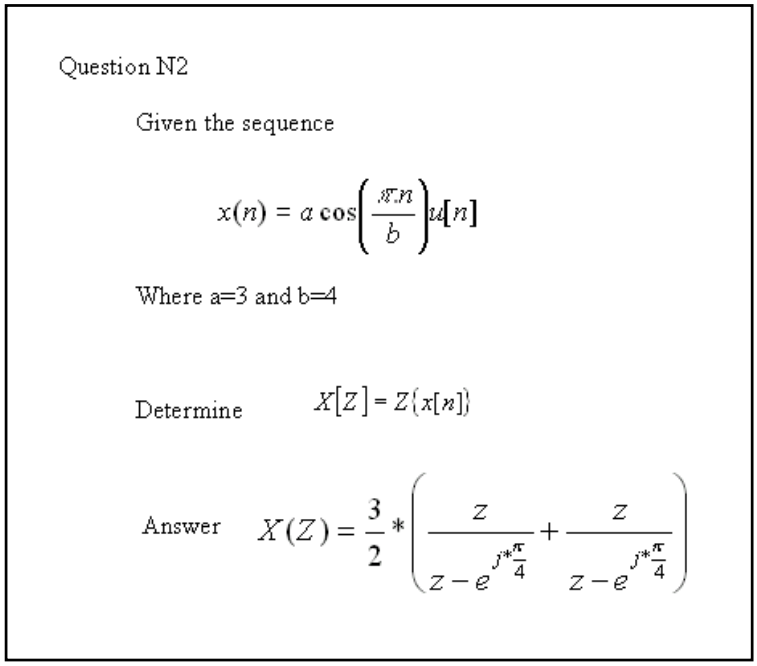

Figure 40. Question with a Longer Answer.

In this case we realized that the HTML input boxes did not provide the most efficient way to input an equation this complex. The main point here was to find an efficient way to allow the student to input his or her answer in a text box, which is the only available input on an HTML or a JSP page. A text string like: 3/2(z/(z - E^(I * Pi/4)) + z/(z - E^(-I * Pi/4))) is understandable by Mathematica but also is prompt to errors when the student is typing it. For this reason, an editor based on WebMathematica was implemented. This editor allows a string as input, and presents the student with a graphical and intuitive mathematical representation of it. From here, the student can visually check the accuracy of his or her answer. After the implementation of this editor the results showed that the input errors were diminished. The graphical equation-representation helps correct the text string before evaluating it. Figure 41 shows the editor written in JSP and MSP.

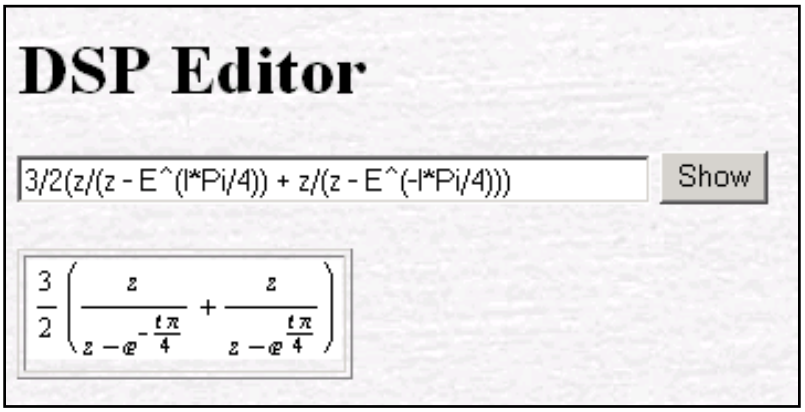

Figure 41. DSP Editor.

We had technical problems with the editor when trying to express summations and integrations. The editor evaluates the expressions before presenting them in a graphical way. When the technical support of Mathematica was consulted, they said that they needed to work on that feature, and that it will be available later. With the editor limited to only algebraic expressions, we initiated a research on possible editor

alternatives. At this time, we are forming a team to develop an equation editor for the web [16], and we are revising the initial code that we wrote. No efficient solutions have yet been found, but we believe a solution could be implemented as a graphical editor that gives a Mathematica string as output. This string could be placed on the question page, and by doing that, we could eliminate changing the actual implementation. A more complex type of solution is to write a Java Script editor and make it run inside of each question page. This technology has not been investigated yet because of the overloading complexity on the question page and the extra maintenance required.

2. Evaluation on the Test

The evaluation process is performed on each test question page. A JSP per question was used. Each JSP has a question number and is not interchangeable. Each test question manages its own set of session variables and all of them are controlled on the main page of the test. This allows the use of a template approach for each test. No matter what the questions are, the page behavior is the same. On a per test basis, the professor must create the questions considering the templates, one per question. Once all created questions are inserted on the corresponding JSP, the test is ready to be assigned to the database.

The template-based approach is one of the interesting parts of the solution presented. Each question has its own logic for evaluation. The JSP could seem complex to understand, but using it only required insert; the two questions as a HTML code and the corresponding answers. The professor has to copy the created questions and their answers in the appropriate places on the JSP. Since JSPs does not require compilation by just saving the JSP file, the test is ready to be assigned to a student.

D. THE FINAL APPLICATION

The DLTOT application was deployed on a web site inside the NPS firewall. This approach allowed us to test all functionality in two months. Every time the system

required a change or an upgrade was implemented, we had to test all functionality again. This was performed in order to ensure non-secondary effects in the main application. Since the appearance of secondary effects is generally related to a non-accurate design, we also verified that our design was correct. Figure 42 shows the main structure of the NPS_DSP test site. All static components as HTML are represented on the upper right and in light yellow. In the lower left and after the login page are all dynamic components as JSP, MSP and Servlets.

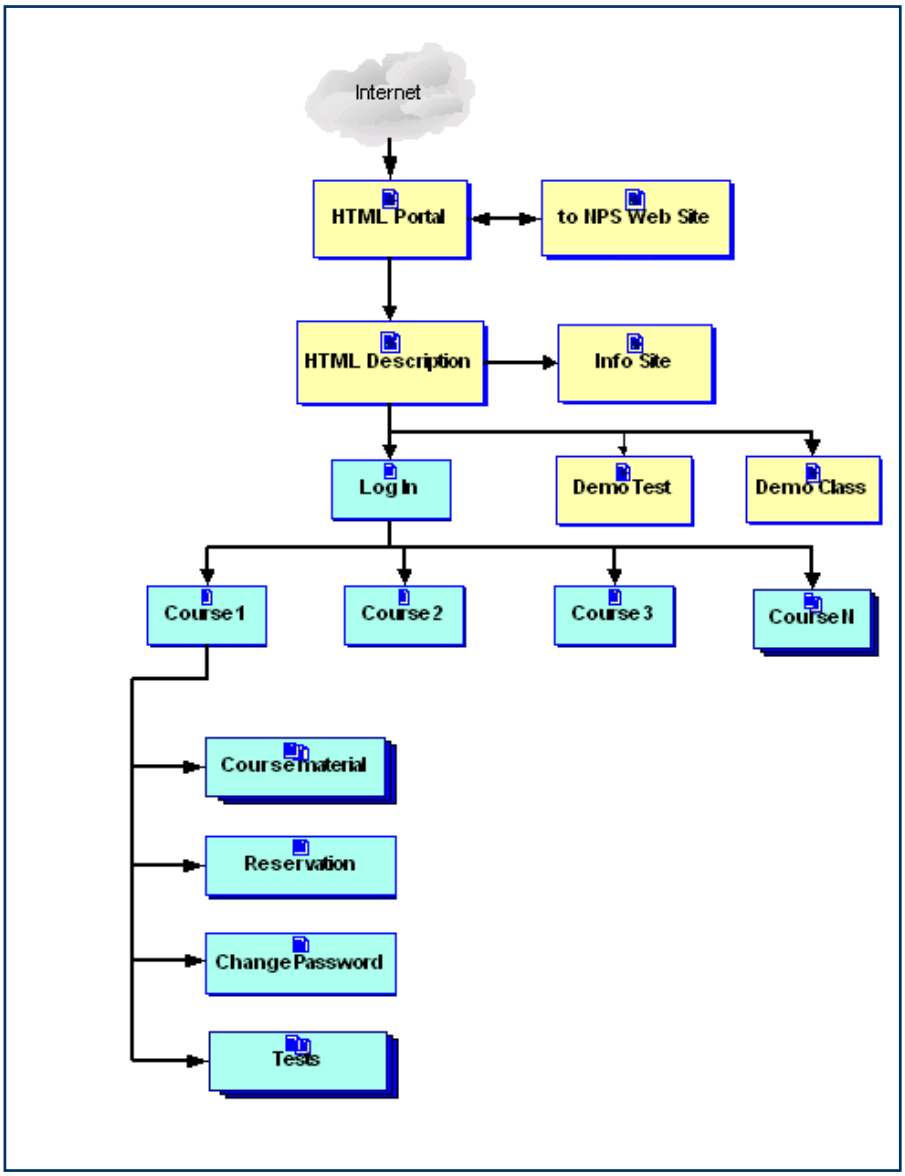

Figure 42. DLTOT Structure in the Web Site.

E. DLTOT RESULTS

The results on the complete site can be represented by a set of multiple figures, all of them showing the functionality of the DLTOT application. Due to the extension of this set of figures, only the test behavior is represented in this chapter. In Appendix C, we cover all figure set.

The testing logic is the main intended behavior to be represented. As a first component, the test is presented to the student. Figure 43 shows the main interface with the student.

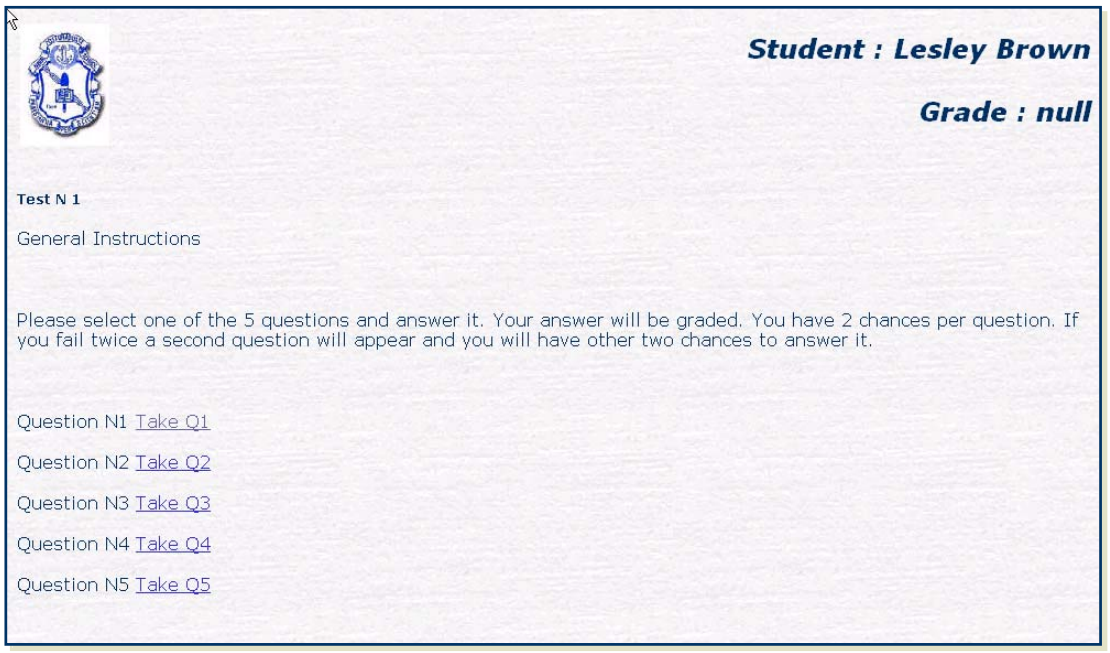

Figure 43. DLTOT Main Test.

Once a question is selected, the correspondent question is presented to the student. The student answers the question and requests its evaluation. In this case we caused the answer to be incorrect. The evaluation is performed at the MSP and WebMathematica responds on a session object. The variable in the session object is captured by the JSP and in this case the same question is repeated for the student. Since the student already

answered the question incorrectly, the five-point penalty was applied. Figure 44 represents this behavior.

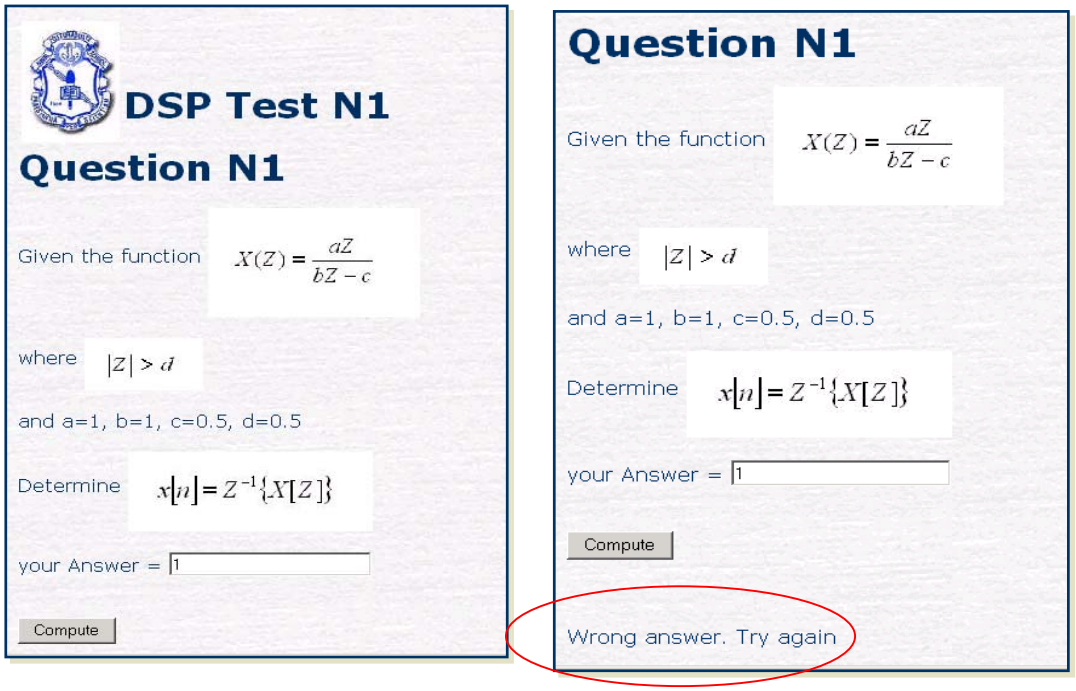

Figure 44. DLTOT Question Answered Incorrectly on the Test.

The student enters the correct answer and the MSP identifies it as a correct one. The result is send to the JSP as a session object. The JSP now presents the link to go back to the rest of the questions and indicates a correct answer. Figure 45 illustrates this.

Figure 45. DLTOT Question Answered Correctly on the Test.

Once the student returns to the main test, a result of the number of points obtained on the question is presented. Also the anwered question no longer appears on the test. Using the same procedure, the student can answer the rest of the questions. Figure 46 shows the main test with the result of the evaluated question and the rest of the unevaluated questions available to the student.

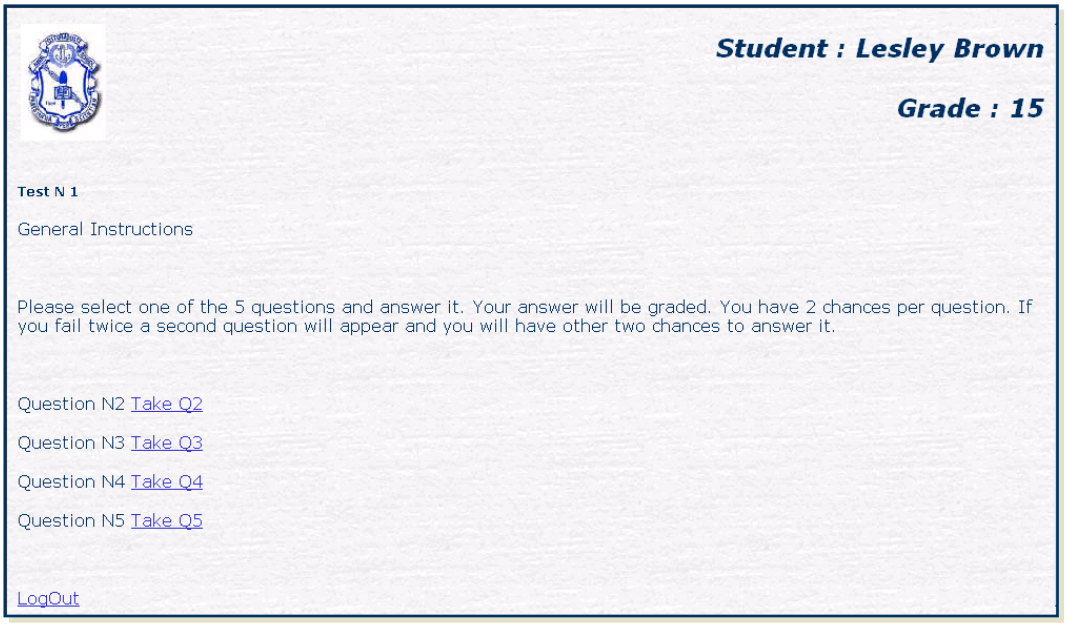

Figure 46. DLTOT Main Test after Evaluation of Question One.

As a main component of the DLTOT application, the results obtained on the test represent most of the application behavior and certainly represents the most important part of the problem presented in this thesis.

After the complete deployment of this application we must express our satisfaction with the obtained results. The DLTOT application will control a complete set of online courses, will present examples, and most importantly will present an alternative to multiple-choice tests, a test composed of challenging questions suitable for electrical engineering courses. The DLTOT will also grade the results of the test in an automatic form leaving the professor only the task of creating the online content for the courses, creating the test questions, and assigning those various tests to the students.

From the reports obtained in the database, it will be possible to assess the material in the course. If a low result is obtained on a particular type of question, the professor can improve the material that explains that concept, creating a more efficient content that allows the student to comprehend the material easily.

The use of this tool can be considered as an automated teaching and testing tool, leaving the professors the time to maximize the teaching material and, the type of questions asked. This gives them more time for research while enriching the quality of the online courses.

F. SUMMARY

In this chapter, we have presented a course created with the DLTOT system. The created course was based on a Mathematica notebook and a set of digitally formatted sources [14]. We also presented the integration of the course with Java Applets as a mean to enrich the student's participation. The test developed in this chapter allows a professor to present a set of challenging questions to the student and to grade the results in an accurate form, which accurately reflects the students' real understanding of the material

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Creating an online set of courses based on Java technology that can handle Electrical Engineering content is now possible and desirable. This online testing technology can maximize the students' acquisition of content and can simultaneously assist professors in evaluating their students. Such an evaluation in turn aids professors in assessing their own teaching materials and techniques.

By integrating databases, mathematical software, and three-tier technology, one can build an online test that can handle not only multiple choice questions but also a more challenging set of questions, suitable for the rigors of science- or math-based courses, such as those found in Electrical Engineering programs.

Also, the logic within the JSPs allows a professor to construct an examination and to grade the students' responses accurately and easily. Such testing in turn accurately assesses the students' knowledge of the subject matter while eliminating the time that a professor must spend on grading the material.

Finally, complexity in the implementation as well as a distributed architecture allows one to deploy an online course that is easy to use, that is platform independent and that is available 24 hours a day, seven days a week. Moreover, in addition to these advantages, the design phase allows one to deploy such an online NPS course in a very secure manner.

B. RECOMMENDATIONS

We strongly recommend that further research be conducted on an equation editor. This editor creates the string from a graphical environment. After the student creates the equation, the editor automatically converts the equation to a text string. The best solution is likely an application running at the client, which manages graphical equation and retrieves a Mathematica string as a result. This string can be used by the actual implementation. As a second alternative, we recommend writing an equation editor in Java and creating an Applet that can be downloaded from the server. This solution, however, has a drawback: one must download a large file, but it also has the advantage of centralized maintenance.

The DLTOT application performs correctly in the designed environment, but in order to enrich the students' learning experience, it would be possible to create a set of virtual laboratories that allow students to interact with the learning material at the same time that the student is tacking an online class. These types of laboratories are suggested in the actual implementation, but we believe that a vast research area still exists for future research.

The DBMS used in the implementation of the DLTOT was selected considering a use of the system for a relative medium to low number of users. In order to support over 100 users we recommend using a different DBMS as Oracle. Since the implementation to control the database was coded in Java programming language, we can ensure the implementation on a different DBMS will work as long as the models are correctly implemented and the name of the tables are maintained from the current implementation.

APPENDIX A. MODELING AND IMPLEMENTATION ASPECTS OF DLTOT APPLICATION

On table A1 we present a list of the different components of the application indicating their type name, a brief description and the relation with the other components of the DLTOT.

Number	Type	Name	Description	Linked
1	Bean	CalendarBean.java	Generates calendar	18
2	Bean	StudentData.java	Keeps information of the student	4
3	Servlet	Serv_DB_Control.java	Manages the access to the database	4,5,6,7,8, 9,17
4	Servlet	Serv_login.java	Controls login and creates the session	3,17
5	Servlet	Serv_SelectCourse.java	Finds all courses where the student is enrolled	3,17,11
6	Servlet	Serv_PlaceReserve.java	Place the reserve once the student has confirmed the option	3, 17
7	Servlet	Serv_Reservation.java	Retrieve information from the database where slots are available or reserved	3,17
8	Servlet	Serv_PlaceDelete.java	Delete the slot reserved once is selected	3, 17
9	Servlet	Serv_DeleteReserve.java	Allows to select the time slot to be deleted	3, 17
10	Servlet	ServletUtilities.java	Set of utilities that allows the better work with Servlets	3, 17
10a	Servlet	Serv_NewPassword	Makes checks to the new password and request to the data base the update of the info	

11	JSP	Courses.jsp	Main Menu for the student	5, 9, 24
12	JSP	Course_SigSys_1.jsp	Course	18, 24
13	JSP	Course_DFiltDesig_1.jsp	Course	18, 24
14	JSP	Course_DFT_1.jsp	Course	18, 24
15	JSP	Course_MultiRate_1.jsp	Course	18, 24
16	JSP	Course_ZTrans_1.jsp	Course	18, 24
17	JSP	AccessDenied.jsp	Shows a page indicating the no access allowed	
18	JSP	Calendar.jsp	Presents the actual month and the next one also creates controls that allow to select a date in this two months.	7
19	JSP	BadDateRequested.jsp	Presented when a student intent to reserve a time slot in the same day or in the past	9
20	JSP	ReservedPlaced.jsp	To indicate the success of the reservation	6,7
21	JSP	DeleteConfirmation.jsp	Request confirmation for a deletion	8,9
22	JSP	DeletedPlaced.jsp	Informs the result of a deletion procedure	8,9
23	JSP	FailOnDB.jsp	Informs a failure in the database transaction	3
24	JSP	LogOut.jsp	Indicates the finish of a session	
25	JSP	ChangePassword	Present a input interface to change the password	10a
26	JSP	ConfSuccessPassword	Indicates success in the password change	10a
27	JSP	ConfFailedPassword	Indicates the failure in the password change	10a

Table A1. DLTOT Component List and Interactions.

A. UML CLASS DIAGRAM FOR DLTOT

The next section shows the class diagrams using UML notation for the DLTOT application.

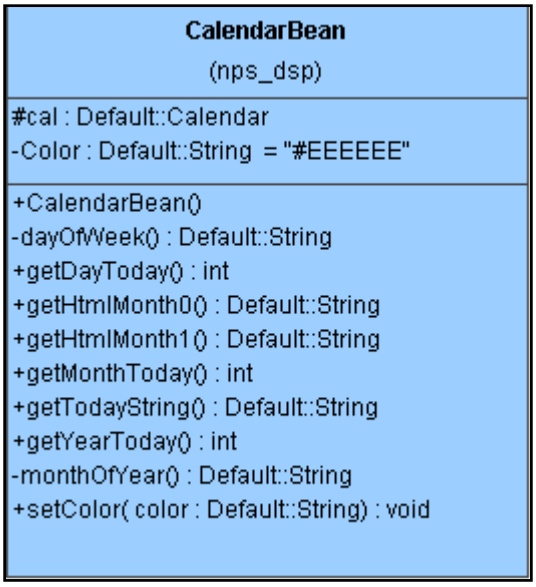

Figure A1. CalendarBean.

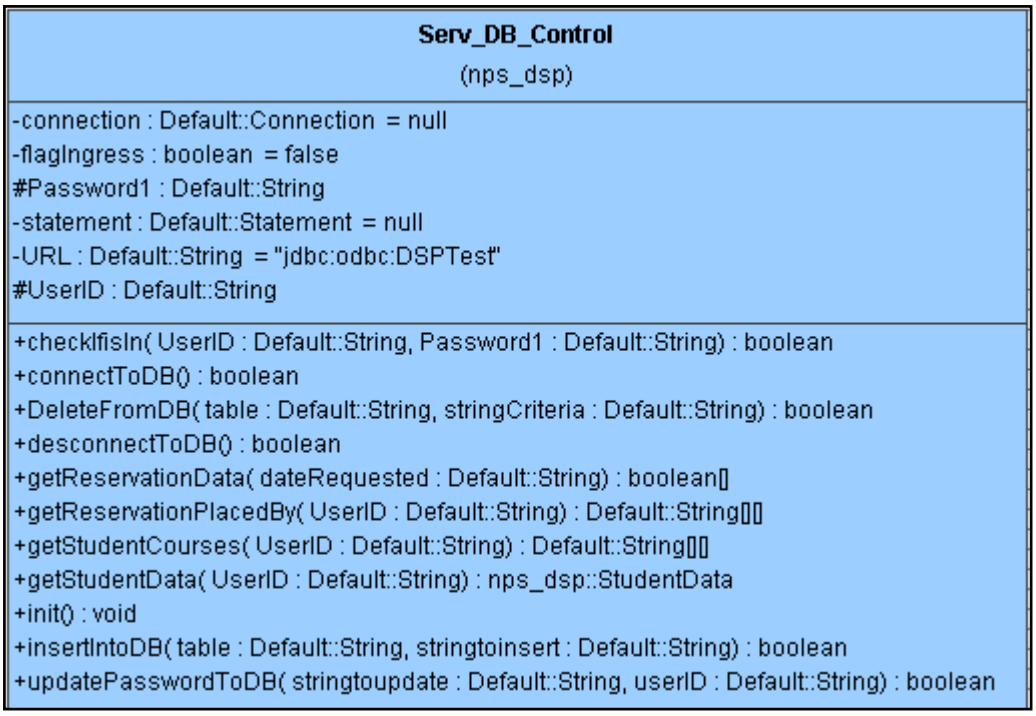

Figure A2. Ser_DB_Control.

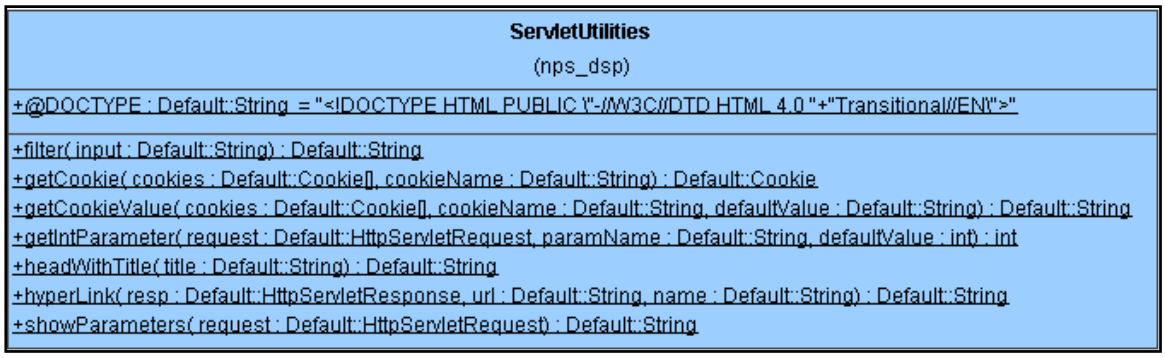

Figure A3. SevletUtilities.

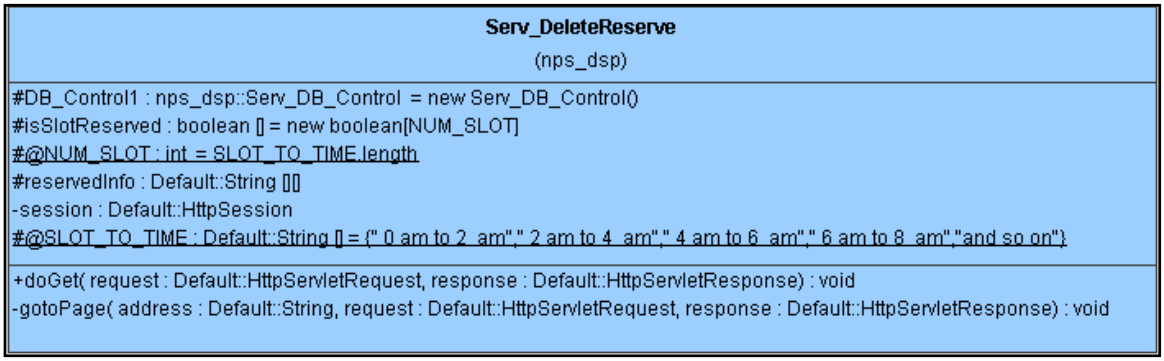

Figure A4. Serv_DeleteReserve.

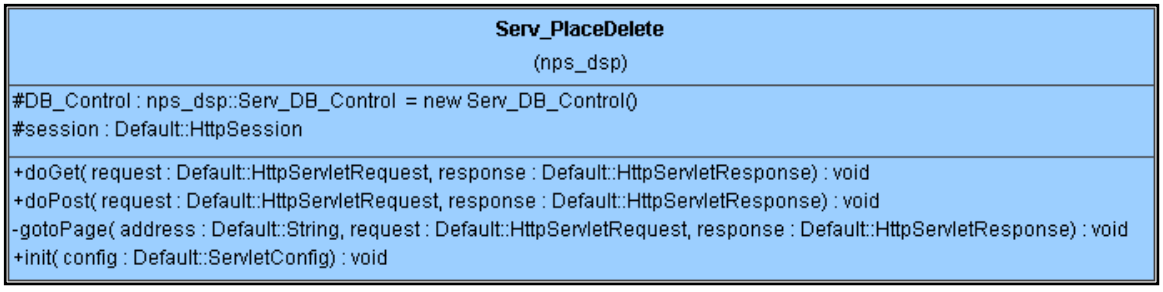

Figure A5. Serv_PlaceDelete.

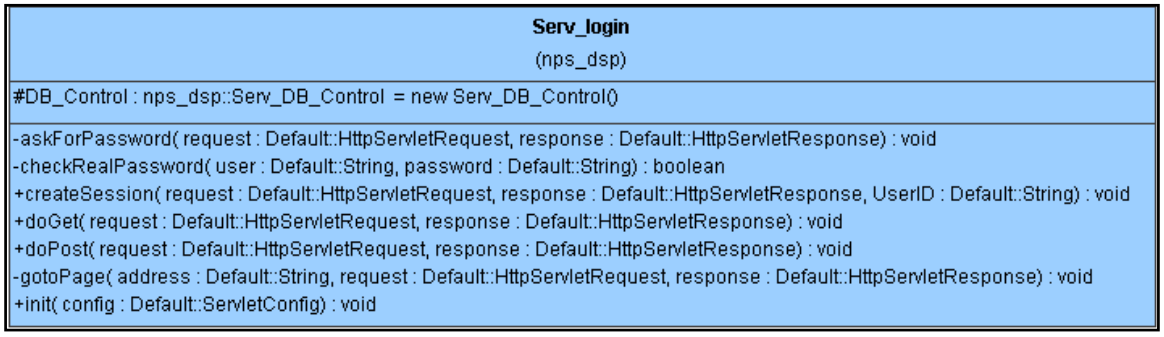

Figure A6. Serv_login.

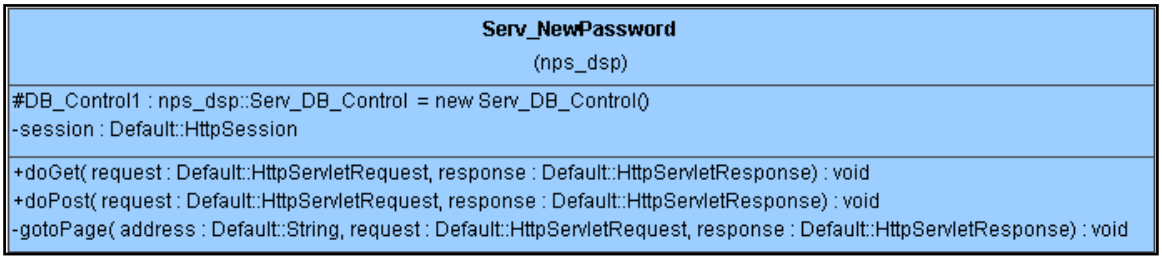

Figure A7. Serv_NewPassword.

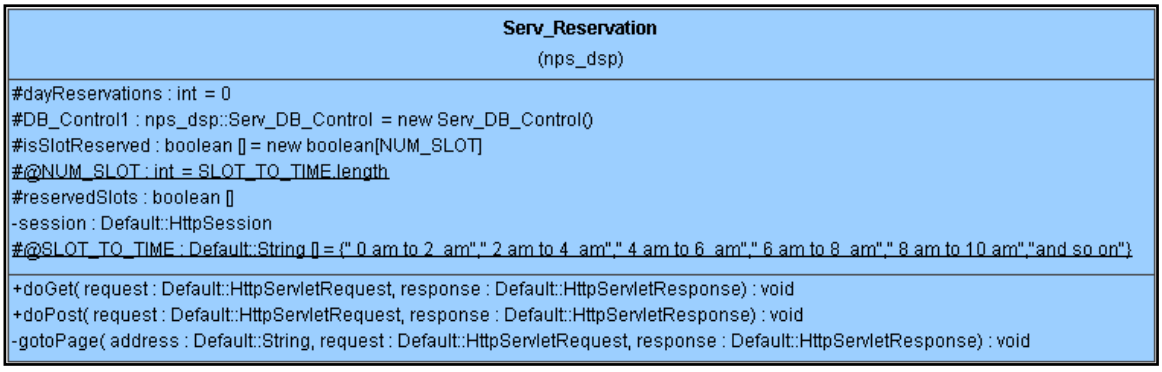

Figure A8. Serv_Reservation.

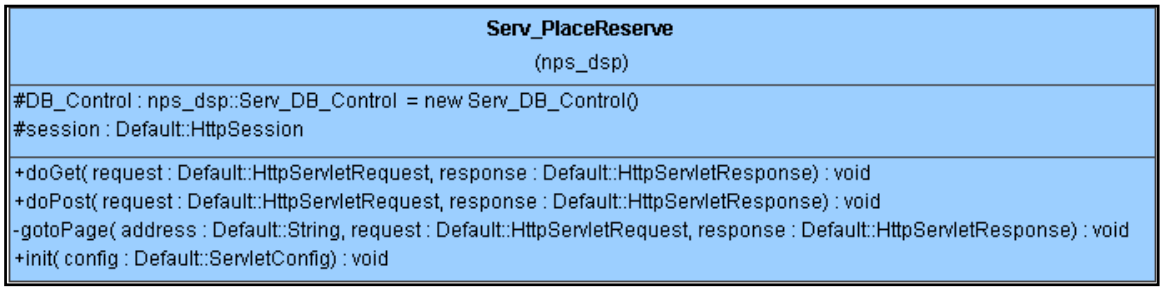

Figure A9. Serv_PlaceReserve.

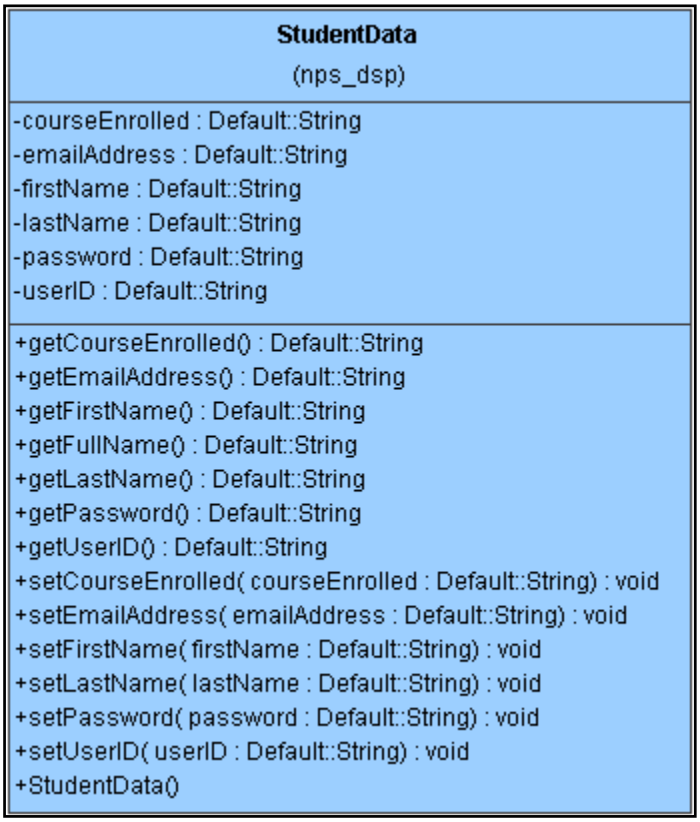

Figure A10. StudentData.

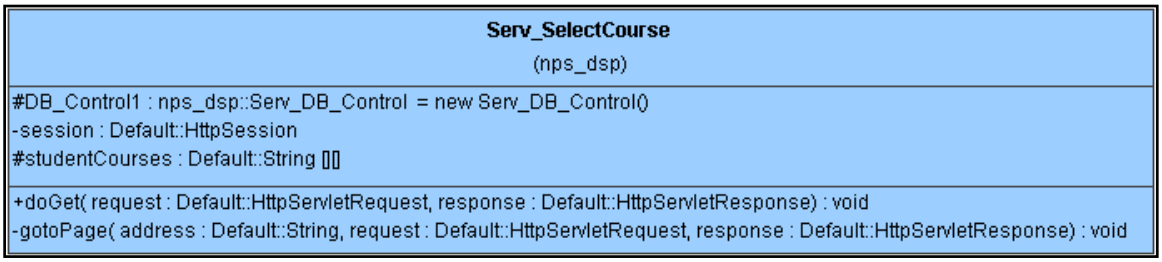

Figure A11. Serv_SelectCourse.

B. MS ACCESS SPECIFICATIONS FOR DSPTEST

The following list was obtained using the utility DOCUMENTER in MS Access DBMS. We intend to give detailed specifications on the table structure of the DSPTest Application.

Table: TabCoordinator

Properties

Date Created:	4/29/2001 7:16:13 PM	GUID:	Long binary data
Last Updated:	6/24/2001 12:05:00 PM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0
RecordCount:	5	Updatable:	True

Columns

Name	Type	Size
ID_Coordinator	Text	15
FirstName	Text	50
LastName	Text	50
Departament	Text	5
Observations	Memo	-
User_ID	Text	50
Password	Text	50

Relationships

TabCoordinatorTabCourse

TabCoordinator		TabCourse
ID_Coordinator	1	Coordinator

Attributes:	Enforced
RelationshipType:	One-To-Many

Table Indexes

Name	Number of Fields
ID_Coordinator	1
Fields:	Ascending
PrimaryKey	1
Fields:	Ascending
User_ID	1
Fields:	Ascending

Table: TabCourse

Properties

Date Created:	4/29/2001 7:13:17 PM	GUID:	Long binary data
Last Updated:	8/15/2001 1:59:05 PM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0
RecordCount:	6	Updatable:	True

Columns

Name	Type	Size
ID_Course	Long Integer	4
Coursenumber	Text	8
Name	Text	50
Description	Memo	-
Duration	Text	50
Prerequisite	Text	50
Coordinator	Text	15
FirstJSPPage	Text	50

Relationships

TabCoordinatorTabCourse

TabCoordinator		TabCourse
ID_Coordinator	1	Coordinator

Attributes:	Enforced
RelationshipType:	One-To-Many

TabCourseTabStudCourse

TabCourse		TabStudCourse
ID_Course	1	Course

Attributes:	Enforced
RelationshipType:	One-To-Many

Table Indexes

Name	Number of Fields
ID_Course	1
Fields:	Ascending
PrimaryKey	1
Fields:	Ascending
TabCoordinatorTabCourse	1
Fields:	Ascending

Table: TabEnrollTest

Properties

Date Created:	4/29/2001 8:32:01 PM	GUID:	Long binary data
Last Updated:	8/20/2001 10:54:58 PM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0
RecordCount:	0	Updatable:	True

Columns

Name	Type	Size
ID_CourseTest	Long Integer	4
EnrollStudCourse	Long Integer	4
Test1	Long Integer	4
Test1Perform	Yes/No	1
Test2	Long Integer	4
Test2Perform	Yes/No	1
T1Q1	Text	50
T1Q2	Text	50
T1Q3	Text	50
T1Q4	Text	50
T1Q5	Text	50
T2Q1	Text	50
T2Q2	Text	50
T2Q3	Text	50
T2Q4	Text	50
T2Q5	Text	50

Relationships

TabStudCourseTabEnrollTest

TabStudCourse		TabEnrollTest
ID_Stud_Course	1	EnrollStudCourse

Attributes:	Enforced
RelationshipType:	One-To-Many

TabTestsTabEnrollTest

TabTests	TabEnrollTest
ID_Test	Test1

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabTestsTabEnrollTest

TabTests	TabEnrollTest
ID_Test	Test2

Attributes:	Not Enforced
RelationshipType:	One-To-Many

Table Indexes

Name	Number of Fields
ID_CourseTest	1
Fields:	Ascending
PrimaryKey	1
Fields:	Ascending
TabStudCourseTabEnrollTest	1
Fields:	Ascending

Table: TabMathReservation

Properties

Date Created:	6/23/2001 4:18:28 PM	GUID:	Long binary data
Last Updated:	6/24/2001 2:09:36 PM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0
RecordCount:	8	Updatable:	True

Columns

Name	Type	Size
Math_ID	Long Integer	4
Student_ID	Text	50
TimeSlot	Long Integer	4
Res_Date	Text	50
Comment	Text	50

Relationships

TabStudentTabMathReservation

TabStudent	TabMathReservation
User_ID	Student_ID

Attributes:	Not Enforced
RelationshipType:	One-To-Many

Table Indexes

Name	Number of Fields
Math_ID	1
Fields:	Ascending
PrimaryKey	1
Fields:	Ascending
Student_ID	1
Fields:	Ascending

Table: TabPracticeQuestions

Properties

Date Created:	4/30/2001 11:21:41 AM	GUID:	Long binary data
Last Updated:	8/20/2001 10:30:20 PM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0
RecordCount:	0	Updatable:	True

Columns

Name	Type	Size
ID_Number	Long Integer	4
Area	Text	15
Difficultie	Integer	2
ProfName	Text	50
JSPPage	Text	50

Relationships

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question1

Attributes: Not Enforced
RelationshipType: One-To-Many

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question2

Attributes: Not Enforced
RelationshipType: One-To-Many

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question3

Attributes: Not Enforced
RelationshipType: One-To-Many

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question4

Attributes: Not Enforced
RelationshipType: One-To-Many

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question5

Attributes: Not Enforced
RelationshipType: One-To-Many

Table Indexes

Name	Number of Fields
ID_Number	1
Fields:	Ascending
PrimaryKey	1
Fields:	Ascending

Table: TabPracticeTests

Properties

Date Created:	4/17/2001 1:05:49 PM	GUID:	Long binary data
Last Updated:	4/29/2001 9:11:40 PM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0
RecordCount:	0	Updatable:	True

Columns

Name	Type	Size
ID_Test	Long Integer	4
Course	Text	10
Question1	Long Integer	4
Question2	Long Integer	4
Question3	Long Integer	4
Question4	Long Integer	4
Question5	Long Integer	4

Relationships

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question1

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question2

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question3

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question4

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabPracticeQuestionsTabPracticeTests

TabPracticeQuestions	TabPracticeTests
ID_Number	Question5

Attributes:	Not Enforced
RelationshipType:	One-To-Many

Table Indexes

Name	Number of Fields
ID_Test	1
Fields:	Ascending
PrimaryKey	1
Fields:	Ascending

Table: TabQuestions

Properties

Date Created:	4/17/2001 12:53:47 PM	GUID:	Long binary data
Last Updated:	9/7/2001 4:05:25 PM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0
RecordCount:	5	Updatable:	True

Columns

Name	Type	Size
ID_Number	Long Integer	4
ID_Course	Long Integer	4
Area	Text	15
Difficultie	Text	10
ProfName	Text	50
JSPPage	Text	50

Relationships

TabQuestionsTabTests

TabQuestions	TabTests
ID_Number	Question1

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabQuestionsTabTests

TabQuestions	**TabTests**
ID_Number	Question2

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabQuestionsTabTests

TabQuestions	**TabTests**
ID_Number	Question3

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabQuestionsTabTests

TabQuestions	**TabTests**
ID_Number	Question4

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabQuestionsTabTests

TabQuestions	**TabTests**
ID_Number	Question5

Attributes:	Not Enforced
RelationshipType:	One-To-Many

Table Indexes

Name	Number of Fields
ID_Course	1
Fields:	Ascending
ID_Number	1
Fields:	Ascending
PrimaryKey	1
Fields:	Ascending

Table: TabStudCourse

Properties

Date Created:	4/29/2001 8:08:02 PM	GUID:	Long binary data
Last Updated:	4/29/2001 9:00:13 PM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0
RecordCount:	4	Updatable:	True

Columns

Name	Type	Size
ID_Stud_Course	Long Integer	4
Student	Text	15
Course	Long Integer	4
PeriodYear	Long Integer	4

Relationships

TabCourseTabStudCourse

TabCourse		TabStudCourse
ID_Course	1	Course

Attributes: Enforced
RelationshipType: One-To-Many

TabStudCourseTabEnrollTest

TabStudCourse		TabEnrollTest
ID_Stud_Course	1	EnrollStudCourse

Attributes: Enforced
RelationshipType: One-To-Many

TabStudentTabStudCourse

TabStudent		TabStudCourse
SSN	1	Student

Attributes: Enforced
RelationshipType: One-To-Many

Table Indexes

Name	Number of Fields
ID_Course	1
Fields:	Ascending
PrimaryKey	1
Fields:	Ascending
TabCourseTabStudCourse	1
Fields:	Ascending
TabStudentTabStudCourse	1
Fields:	Ascending

Table: TabStudent

Properties

Date Created:	4/29/2001 8:16:03 PM	GUID:	Long binary data
Last Updated:	6/6/2001 2:45:14 PM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0

RecordCount:	8	Updatable:	True

Columns

Name	Type	Size
SSN	Text	15
FirstName	Text	50
LastName	Text	50
Address	Memo	-
Phone	Text	15
Email	Text	50
User_ID	Text	50
Password	Text	50

Relationships

TabStudentTabMathReservation

TabStudent		TabMathReservation
User_ID		Student_ID

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabStudentTabStudCourse

TabStudent		TabStudCourse
SSN	1	Student

Attributes:	Enforced
RelationshipType:	One-To-Many

Table Indexes

Name	Number of Fields
PrimaryKey	1
Fields:	Ascending
User_ID	1
Fields:	Ascending

Table: TabTests

Properties

Date Created:	4/17/2001 1:05:13 PM	GUID:	Long binary data
Last Updated:	4/30/2001 11:27:03 AM	NameMap:	Long binary data
OrderByOn:	False	Orientation:	0
RecordCount:	2	Updatable:	True

Columns

Name	Type	Size
ID_Test	Long Integer	4
Course	Long Integer	4
Date	Date/Time	8
Question1	Long Integer	4

Question2	Long Integer	4
Question3	Long Integer	4
Question4	Long Integer	4
Question5	Long Integer	4

Relationships

TabQuestionsTabTests

TabQuestions	**TabTests**
ID_Number	Question1

Attributes: Not Enforced
RelationshipType: One-To-Many

TabQuestionsTabTests

TabQuestions	**TabTests**
ID_Number	Question2

Attributes: Not Enforced
RelationshipType: One-To-Many

TabQuestionsTabTests

TabQuestions	**TabTests**
ID_Number	Question3

Attributes: Not Enforced
RelationshipType: One-To-Many

TabQuestionsTabTests

TabQuestions	**TabTests**
ID_Number	Question4

Attributes: Not Enforced
RelationshipType: One-To-Many

TabQuestionsTabTests

TabQuestions	**TabTests**
ID_Number	Question5

Attributes: Not Enforced
RelationshipType: One-To-Many

TabTestsTabEnrollTest

TabTests	**TabEnrollTest**

ID_Test	Test1

Attributes:	Not Enforced
RelationshipType:	One-To-Many

TabTestsTabEnrollTest

TabTests	**TabEnrollTest**
ID_Test	Test2

Attributes:	Not Enforced
RelationshipType:	One-To-Many

Table Indexes

Name	Number of Fields
ID_Test	1
Fields:	Ascending
PrimaryKey	1
Fields:	Ascending

APPENDIX B. INSTALLATION INSTRUCTIONS FOR DLTOT APPLICATION

A. INSTALLING JAVA

This text was obtained from [17] entirely.

Once you've downloaded the JDK, you'll need to install it. This is quite easy on the Mac, not too difficult on Unix, and quite a chore on Windows, especially if you're installing over an older version of the JDK. Here are the detailed instructions.

Windows Installation Instructions

First deinstall any previous versions of the JDK you may have installed, especially if you want to put the new JDK in a different directory. You may also have to use *regedit* to delete any keys previous installations have left behind. Be careful, though. *regedit* is a dangerous tool.

The Win32 X86 releases are self extracting archives. You will need about sixty megabytes of free disk space to install the JDK. Execute the file by double-clicking on its icon in the File Manager or by selecting Run... from the Program Manager's File menu and typing the path to the file. This will unpack the archive including all necessary directories and sub-directories. The full path is unimportant, but for simplicity's sake this book assumes you install it in *C:\jdk*. If you unpacked it somewhere else, just replace *C:\jdk* by the full path to the directory where you put it in what follows.

You will need to add *C:\jdk\bin* directory to your PATH environment variable. For example,

```
C:>set PATH="C:\jdk\bin;$PATH"
```

You can do it more permanently by putting this command in your *autoexec.bat* file. In Windows NT, you should set it from the Environment tab of the System Control Panel as shown in Figure 2-1 instead.

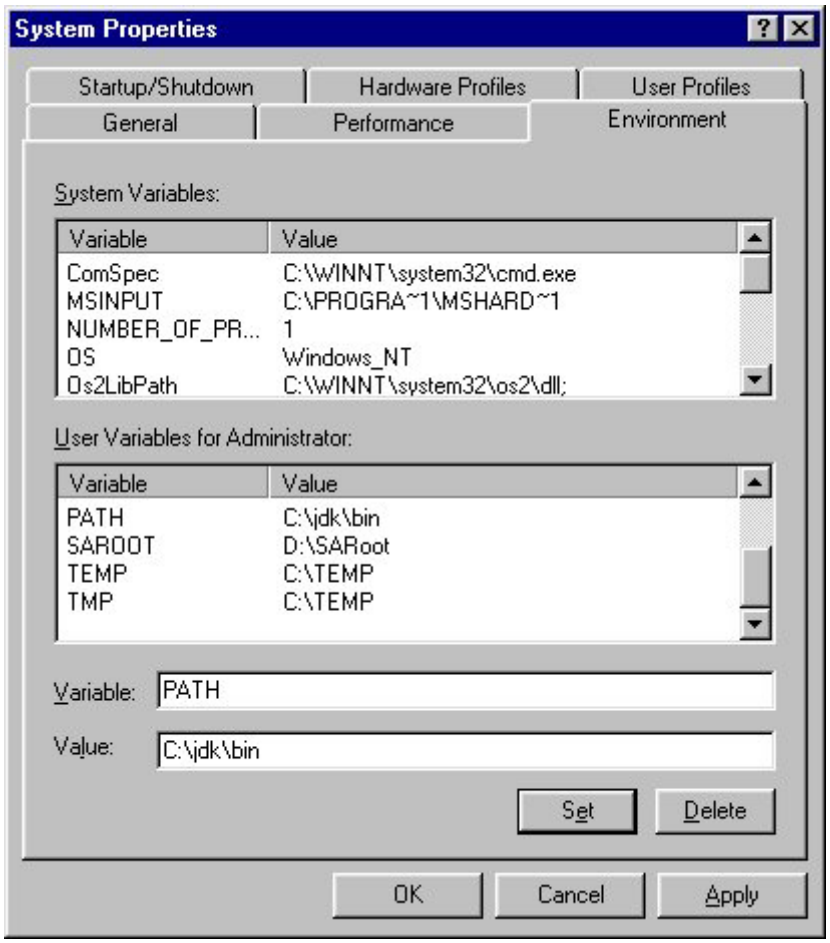

Figure B1. Setting the PATH environment variable under Windows NT.

To make sure your Java environment is correctly configured, bring up a DOS window and type "javac nofile.java". For example,

C:\>javac nofile.java

If your computer responds with "error: Can't read: nofile.java ", you're ready to begin. If, on the other hand, it responds

The name specified is not recognized as an internal or external command, operable program or batch file.

or something similar, then the Java environment has not been properly installed or your PATH is not set correctly. You'll need to fix it before continuing.

Linux/Unix Installation Instructions

If you have an account on a shared system at a university or an Internet Service Provider, there's a good chance Java is already installed. Ask your local support staff how to access it. Otherwise follow these instructions.

The Unix release is a gzipped tar file. You will need about sixty megabytes of free disk space to uncompress and untar the JDK. Double that would be very helpful. You do this with these commands:

```
% gunzip jdk1_2_2-linux-i386.tar.gz
% tar xvf jdk1_2_2-linux-i386.tar
```

The exact filename may be a little different if you're retrieving the release for a different platform such as Irix or if the version is different.

You can untar it in your home directory, or, if you have root privileges, in some convenient place like */usr/local/java* where all users can have access to the files. However root privileges are not necessary to install or run Java.

Untarring the file creates all necessary directories and sub-directories. The exact path is unimportant, but for simplicity's sake this book assumes it's installed it in */usr/local/java*. If a sysop already installed it, this is probably where it lives. (Under Solaris it's also possible the sysop put it into */opt*.) If this is the case the files live in */usr/local/java/jdk1.2.2*. If you're unpacking a different version, then you may find it installed in a slightly different directory such as */usr/local/java/jdk1.3*. This allows multiple versions of the JDK to coexist harmoniously on the same system. If you unpacked it somewhere other than */usr/local/java*, just replace */usr/local/java* by the full

path to the *java* directory in what follows. If you installed it in your home directory, you can use *~/java* instead of a full path.

You now need to add */usr/local/java/jdk1.2.2/bin* directory to your PATH environment variable. You use one of the following commands depending on your shell.

csh, tcsh:

% set PATH=($PATH /usr/local/java/jdk1.2.2/bin)

sh:

% PATH=($PATH /usr/local/java/bin); export $PATH

You should also add these lines to the end of your *.profile* or *.cshrc* files so you won't have to repeat them every time you login.

To verify that your Java environment is correctly configured, type "javac nofile.java" at a shell prompt like this

% javac nofile.java

If your computer responds with

error: Can't read: nofile.java

you're ready to begin. If, on the other hand, it responds

javac: Command not found

or something similar, then you the Java environment is not properly installed or your PATH is incorrect. You'll need to fix it before continuing.

Macintosh Installation Instructions

The file you get will be a self-mounting image called something like MRJ_2.2.smi.bin. If you use Fetch, Anarchie, or a web browser to download it will be automatically converted into the self-mounting image called MRJ_2.2.smi.bin. Double-click it to mount it and then run the installer from the image it mounted. The installer will prompt you for a folder to install it in on your computer. Put it wherever is convenient.

It will probably be helpful to make aliases of the Applet Viewer, the Java Compiler and the Java Runner and put them on your desktop for ease of dragging and dropping later, especially if you have a large monitor.

B. INSTALLING WEBMATHEMATICA

This text was obtained from the WebMathematica documentation [18].

Mathematica Server Pages, the core technology of web*Mathematica*, are based on a standard Java technology called servlets. Servlets are special Java programs that run on a web server machine. Most web servers do not support servlets natively--instead, support is provided by a separate program called a servlet container (or sometimes a "servlet engine") which connects to the web server in some way. Essentially all modern web servers support servlets natively or through a plug-in servlet container. This includes Apache, Microsoft's IIS and PWS, Netscape Enterprise Server, iPlanet, and so-called "application servers" (such as IBM WebSphere). Popular servlet containers that can plug in to a wide variety of web servers include Tomcat, http://jakarta.apache.org, and JRun, http://www.jrun.com. Both Tomcat and JRun include standalone web servers as well, so they can be used as total solutions themselves without requiring an external web server such as Apache. This mode of operation is quite convenient for initial installation and configuration.

These installation instructions focus on setting up the servlet container to run web*Mathematica*, but we provide some information on configuring the servlet container

to plug into an external web server. If your servlet container has its own standalone web server, and you are setting up the servlet container for the first time, we recommend that you get web*Mathematica* running using the container's internal web server. After that is successful, you can configure the servlet container for use with Apache, IIS, or some other external server if you desire. This won't affect your webMathematica installation.

Most modern servlet containers, including current releases of Tomcat and JRun, support the so-called "servlet 2.2 specification". Some older servlet containers, notably Apache JServ, do not. We encourage people to use the most up-to-date products they can find, but web*Mathematica* does not require a servlet 2.2 container. Users with legacy installations of Apache with JServ do not need to upgrade to Tomcat (which is essentially the replacement for JServ as a servlet container for Apache and IIS). The installation instructions are slightly different for users with servlet 2.2-compliant containers and those without.

We have tested web*Mathematica* with both Tomcat and JRun. While it should work with any compliant servlet container we have not carried out any validation tests of other configurations. If you have a particular interest or experience in running web*Mathematica* with other containers please contact Wolfram Research.

Install the MSP *Mathematica* Application

The MSP *Mathematica* application must now be installed into the *Mathematica* layout. This application is found in archive form on the web*Mathematica* Tools CD-ROM, the appropriate archive should be unpacked and its contents placed into the `AddOns/Applications` directory in your *Mathematica* directory. When this is done it should be possible to launch *Mathematica* and select Rebuild Help Index from the Help menu. This will cause the web*Mathematica* documentation to appear in the Help Browser under the Add-ons section. The layout of files in the MSP application is shown at the end of this section. It is also possible to open the documentation directly in *Mathematica*, which might be useful for printing it out.

The Configuration directory holds an important configuration file MSP.conf. This is used to set various site specific parameters and may need modification for your site. The MSP.conf located in this directory is suitable for a typical Unix installation, a copy of this file is available as MSP.conf.unix. In addition the file MSP.conf.nt contains settings suitable for a typical NT installation. The settings that can be placed into MSP.conf are described in the section on site configuration in the Appendix.

Here is a typical content for Unix:

```
MathLinkArguments=-linkname 'math -mathlink' -linkmode launch
 KernelAcquireLimit=200
 MSPDirectory=/usr/local/mathematica/AddOns/Applications/MSP/MSPScripts
 ImageDirectory=/usr/local/mathematica/AddOns/Applications/MSP/Caches
 KernelNumber=1
 KernelTimeLimit=30000
 VerboseLogs=true
 MSPIncludesDirectory=/webMathematica/MSPIncludes
```

Figure B2. MSP Configuration File for Unix.

On NT a typical contents would be as follows (note the double backslashes):

```
MathLinkArguments=-linkname 'c:\\Program Files\\Wolfram
Research\\Mathematica\\4.1\\MathKernel.exe' -linkmode launch
 KernelAcquireLimit=200
 MSPDirectory=c:\\Program Files\\Wolfram
Research\\Mathematica\\4.1\\AddOns\\Applications\\MSP\\MSPScripts
 ImageDirectory=c:\\Program Files\\Wolfram
Research\\Mathematica\\4.1\\AddOns\\Applications\\MSP\\Caches
 KernelNumber=1
 KernelTimeLimit=30000
 VerboseLogs=true
 MSPIncludesDirectory=/webMathematica/MSPIncludes
```

Figure B3. MSP Configuration File for Windows NT.

This describes the command used to launch Mathematica, some parameters relevant to the running of Mathematica, and the location of the MSP pages. It also sets

the directory that will be used to store images. If these parameters are not set it will not be possible to launch Mathematica and the log files will contain a description of the errors. For an explanation of log files see logging. The values in these files are set for a default installation of Mathematica, if you install Mathematica into some location other than that described in MSP.conf you will need to make appropriate changes.

You may store your webMathematica license in its own password file by using the file MSP/Configuration/Licensing/mathpass. It will then be necessary to modify MSP.conf to ensure that Mathematica used this location:

MathLinkArguments=-linkname 'math -mathlink -pwfile /usr/local/mathematica/AddOns/Applications/MSP/Configuration/Licensing/mathpass' -linkmode launch

The details of MSP.conf are given in more detail in the site administration section.

Under Unix it may be necessary to add a FrontEndLaunchCommand parameter so that the front end can run properly. This is described in the section on site configuration in the Appendix.

The Caches directory which is used to store images needs to have suitable permissions so that the user running Tomcat can write, read, and delete files. If this is not done, then graphics will not work correctly.

Install the webMathematica Web Application

This section describes how to install webMathematica components into your servlet container. For most servlet containers this involves deploying the webMathematica web application that is found in archived form on the webMathematica tools CD-ROM. Separate installation instructions are given for some different servlet containers. A web application is a collection of HTML and other web components which are placed in a specific directory structure. Any servlet container that supports web applications will be able to use these files in a standard way. Web applications support a special type of archive called a WAR archive, which is supported by some servlet containers. A WAR archive of the webMathematica archive is provided on the webMathematica tools CD-ROM.

Tomcat

This section describes the deployment of the webMathematica webapp in Tomcat. There are two important steps, unpacking the webMathematica archive and configuring the deployment descriptor file.

First, you should unpack one of the webMathematica archives (choosing an archive suitable for your platform) from the tools CD-ROM and place the webMathematica directory into the Tomcat webapps directory. This is usually found in the root directory of Tomcat. A web application called webMathematica has now been created.

Second, you must configure the deployment descriptor file web.xml, which is found inside the WEB-INF directory. This contains various settings that govern the operation of the web application. One important setting is the location of the directory which holds the MSP.conf file (see MSP.conf). The web.xml included here is suitable for a typical Unix installation, a copy of this file is available as web.xml.unix. In addition web.xml.nt contains settings suitable for a typical NT installation.

A portion of web.xml suitable for Unix is shown below:

```
<web-app>
  <servlet>
        <servlet-name>MSP</servlet-name>
        <servlet-class>MSP</servlet-class>
        <init-param>
              <param-name>wolfram.configuration_directory</param-name>
              <param-
value>/usr/local/mathematica/AddOns/Applications/MSP/Configuration</param-value>
        </init-param>
  </servlet>
</web-app>
```

On an NT system it would need to be something like:

```
<web-app>
  <servlet>
        <servlet-name>MSP</servlet-name>
        <servlet-class>MSP</servlet-class>
        <init-param>
              <param-name>wolfram.configuration_directory</param-name>
              <param-value>C:\Program Files\Wolfram
Research\Mathematica\4.1\AddOns\Applications\MSP\Configuration</param-value>
        </init-param>
  </servlet>
</web-app>
```

APPENDIX C. MAIN SCREEN OUTPUTS FROM DLTOT APPLICATION

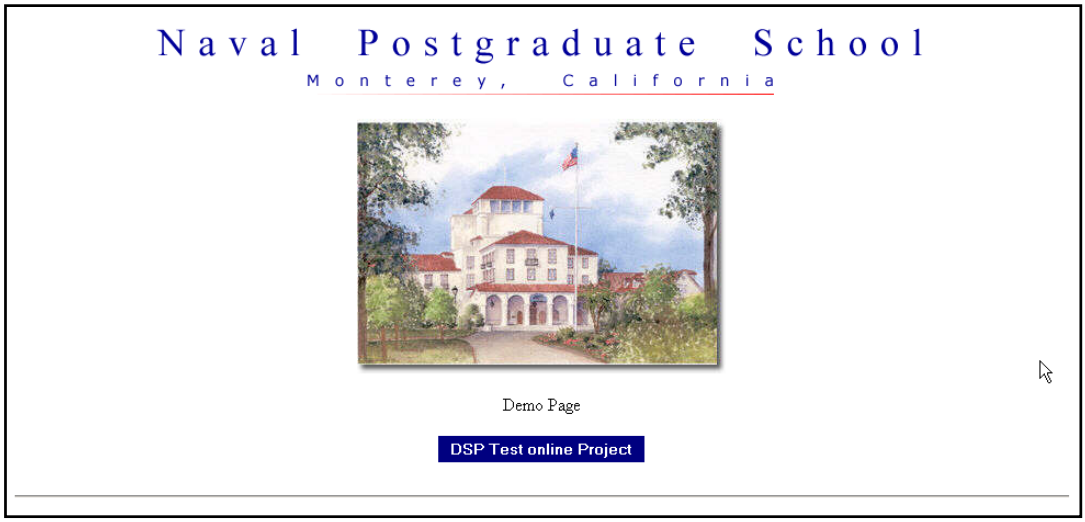

Figure C1. DLTOT First Page.

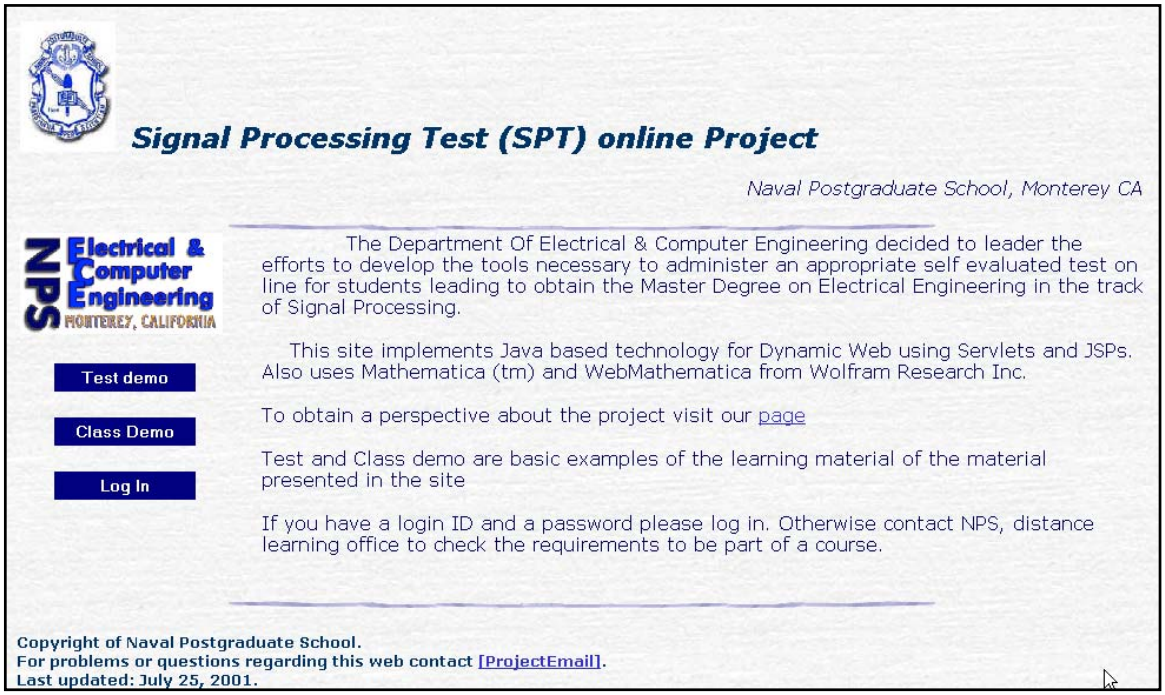

Figure C2. Project Description.

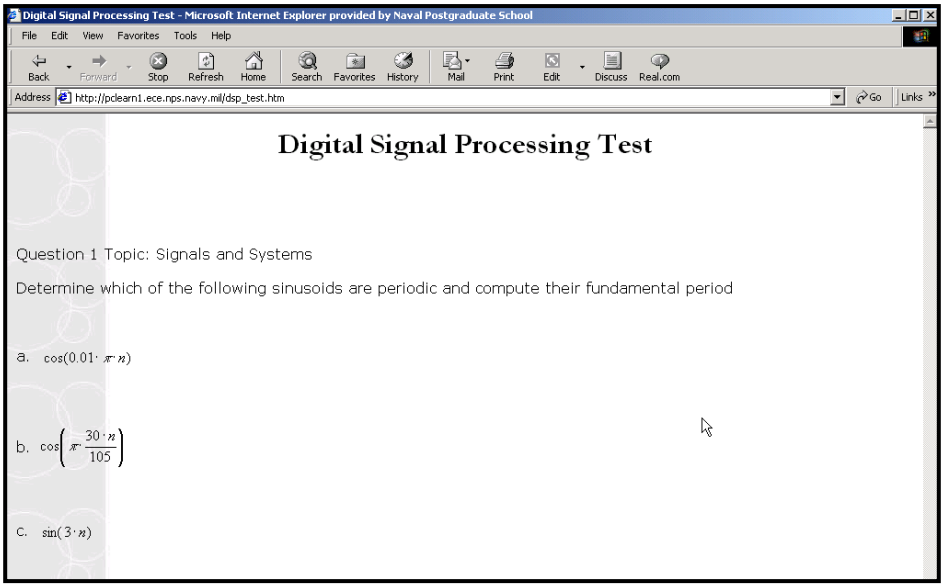

Figure C3. Test Demonstration.

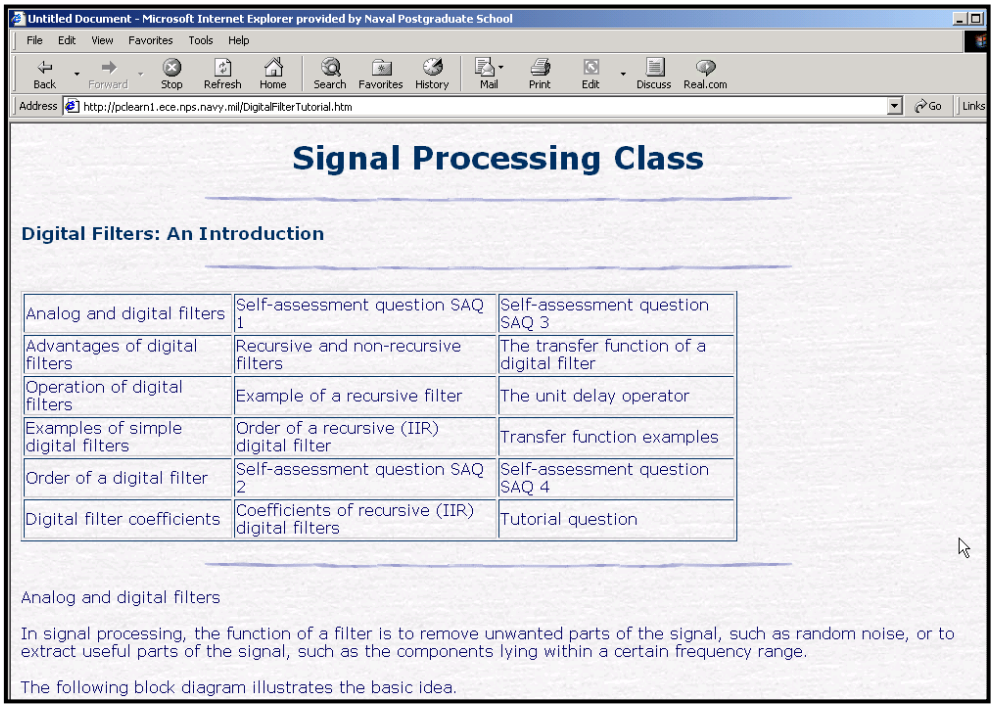

Figure C4. Class Demonstration.

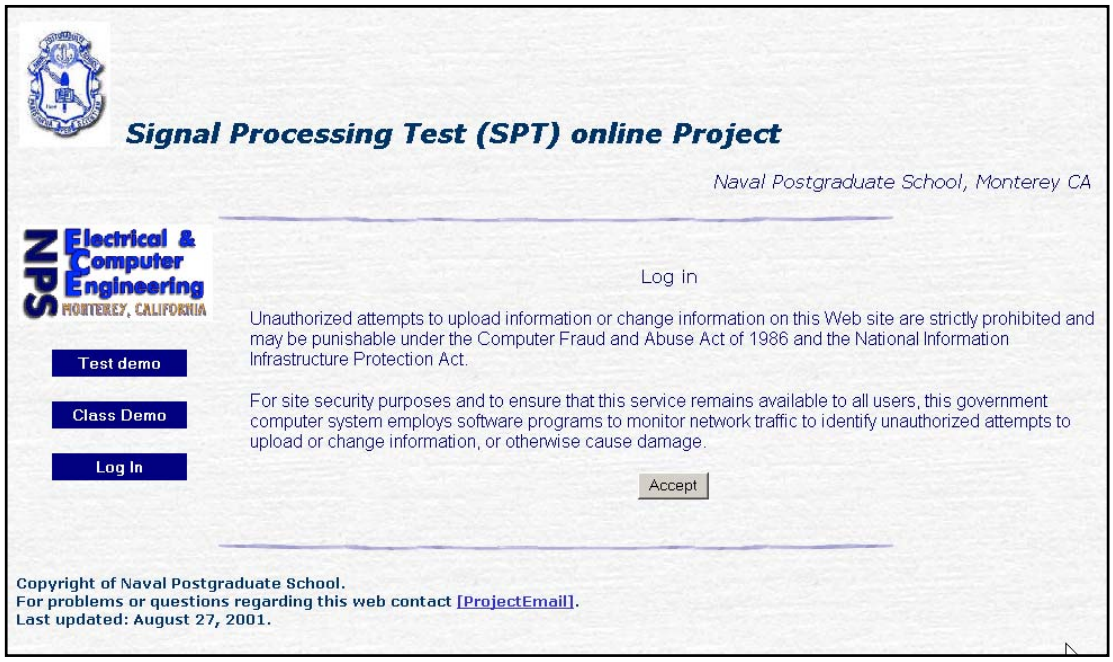

Figure C5. Login Disclaimer.

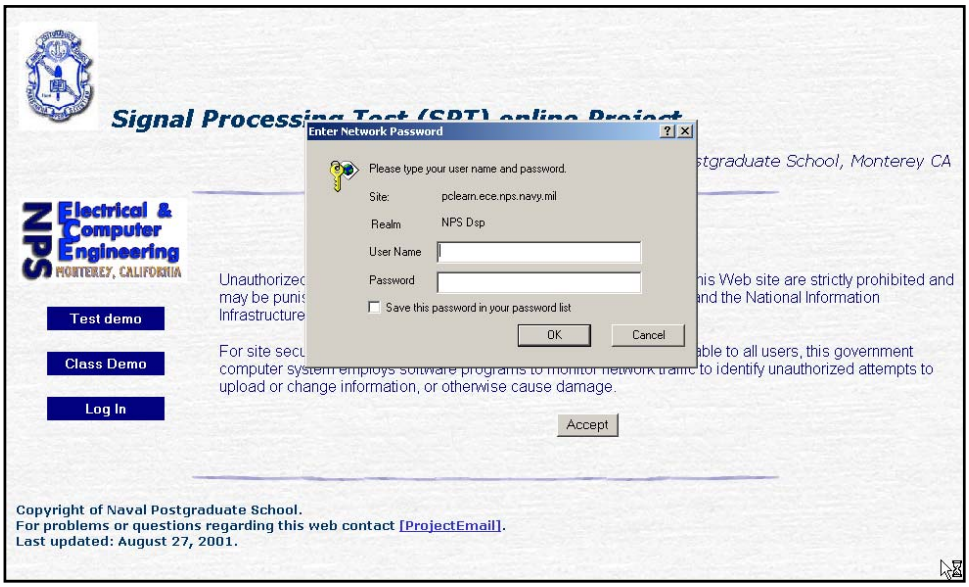

Figure C6. Login.

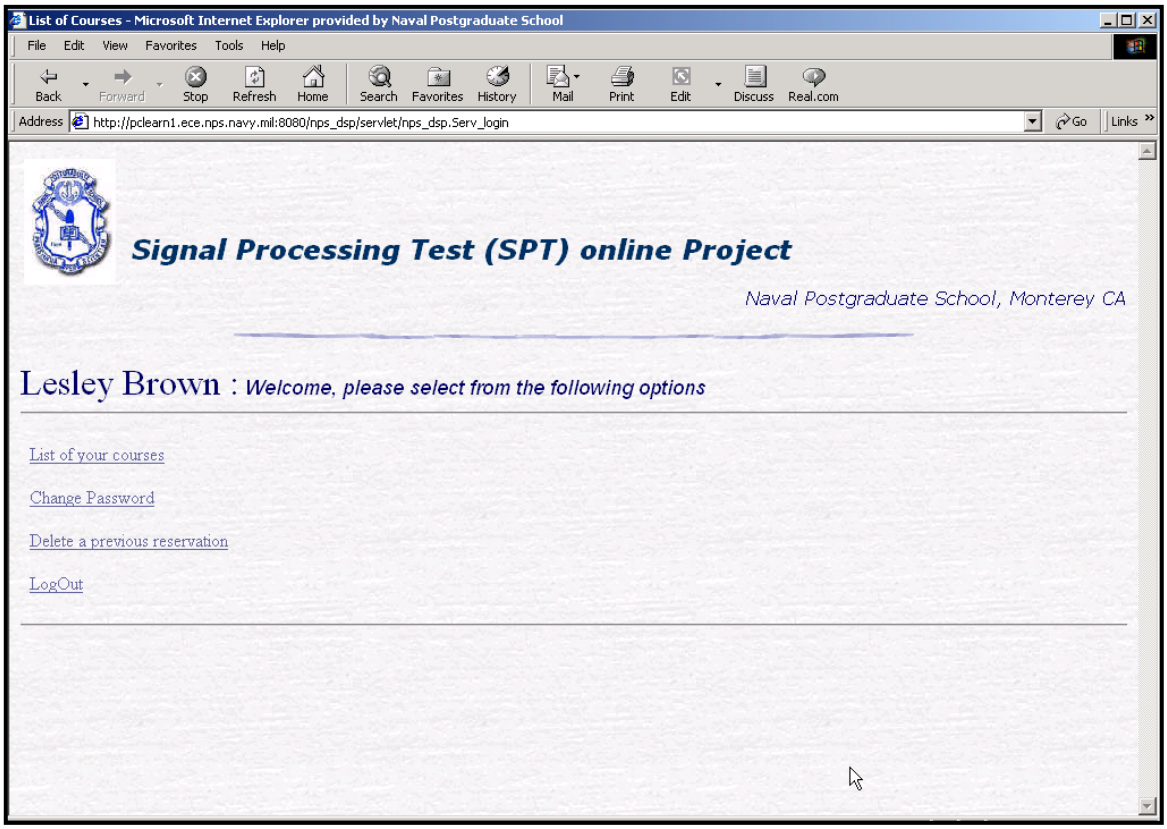

Figure C7. Student Identified by the DLTOT.

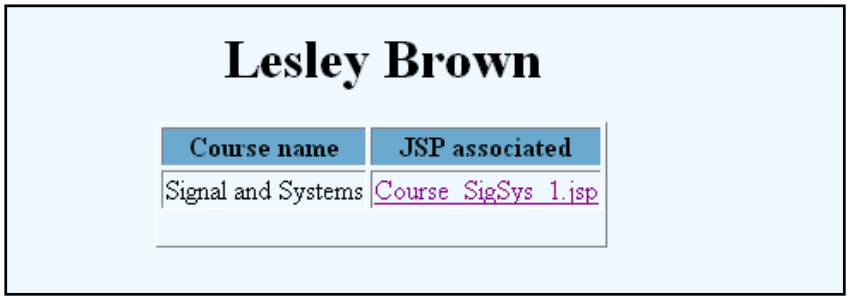

Figure C8. List of Courses where Student is Enrolled.

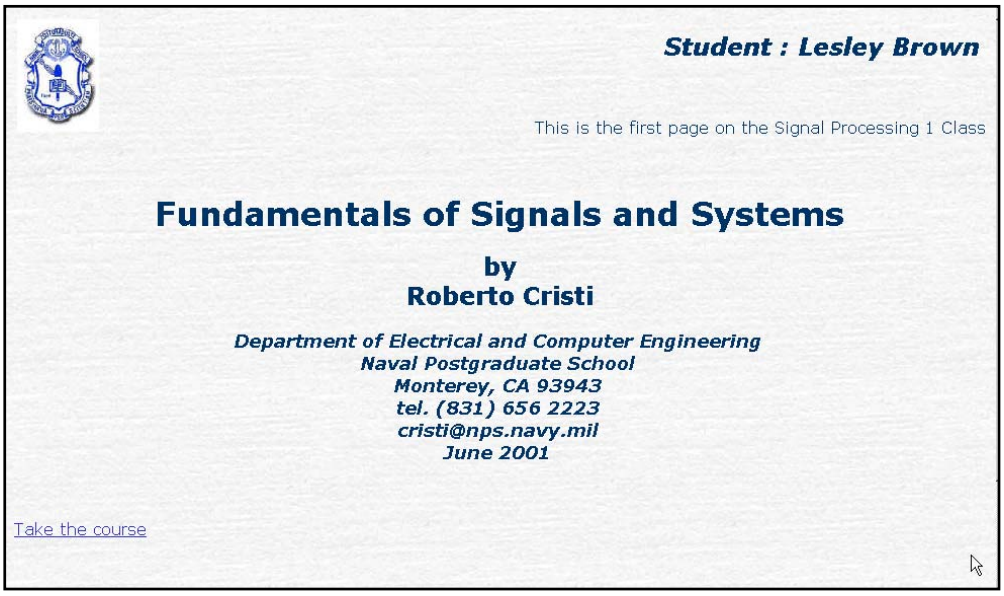

Figure C9. First Page of a Course.

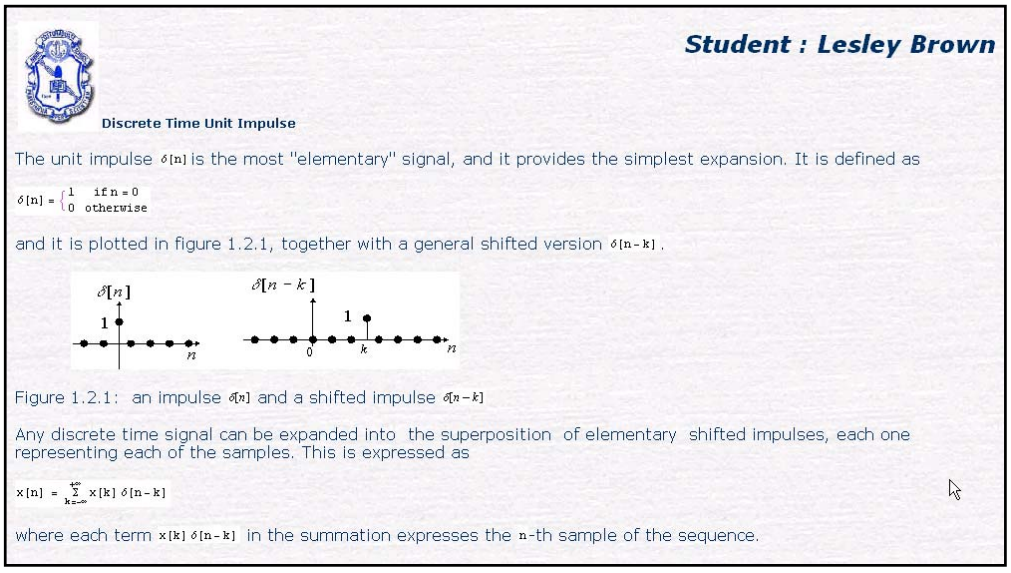

Figure C10. Pages of the Course.

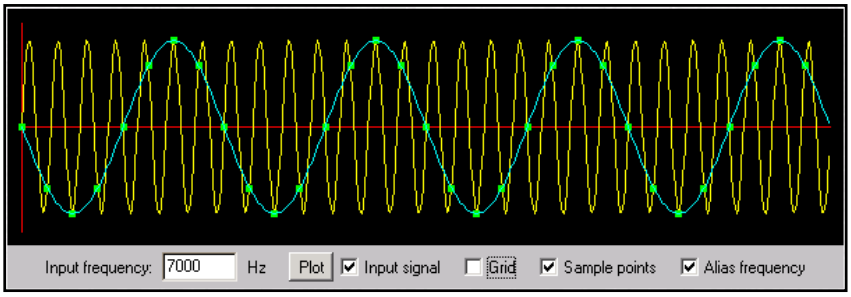

Figure C11. Applet in the Course.

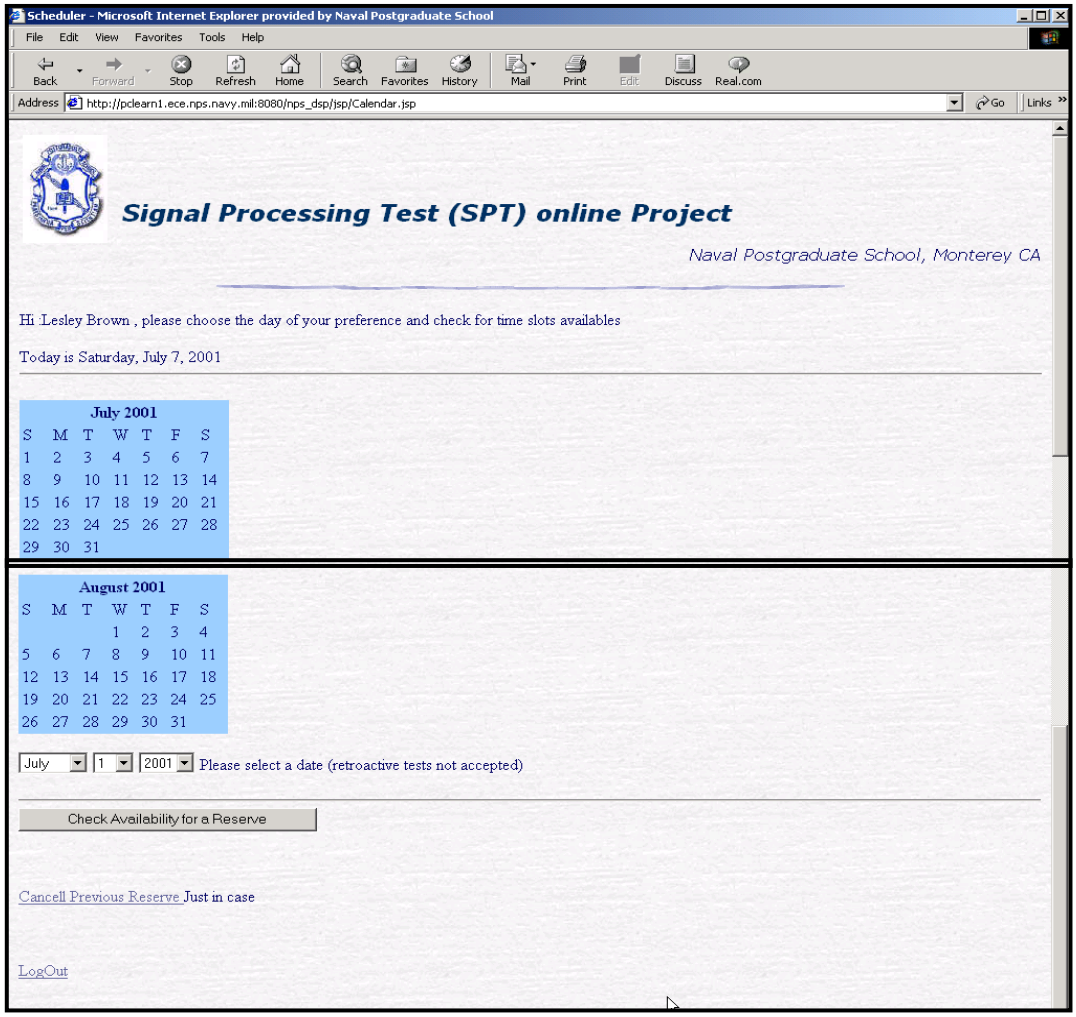

Figure C12. Calendar to Register the Test.

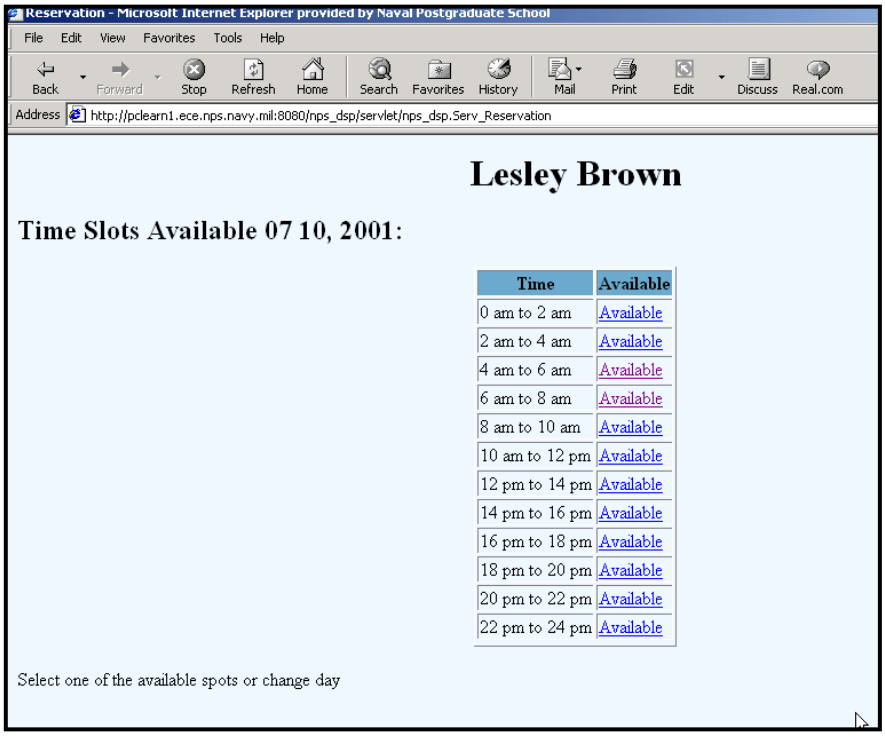

Figure C13. Available Time-Slots for the Test.

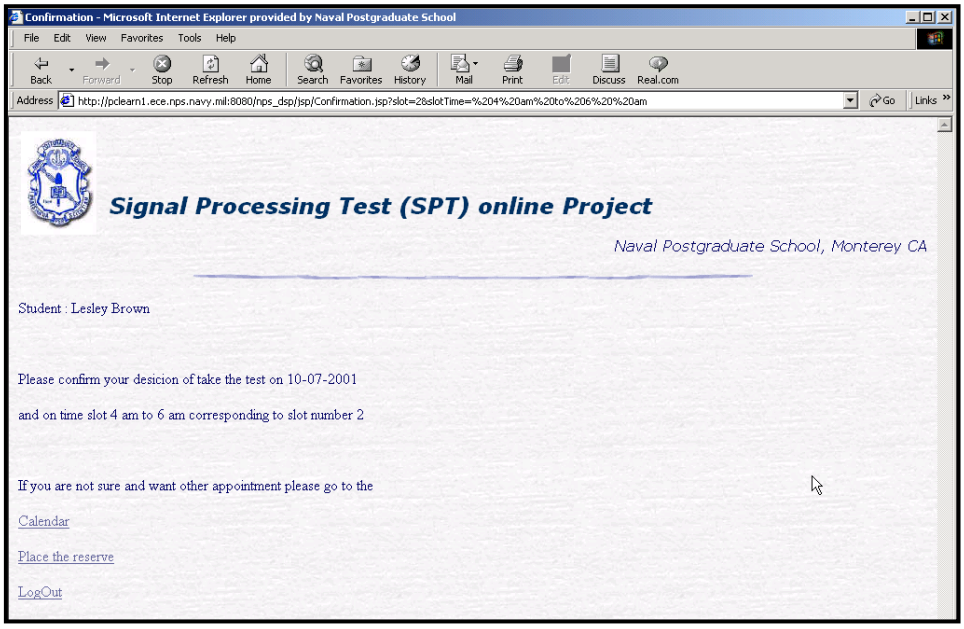

Figure C14. Reserve Time-Slot Request Confirmation.

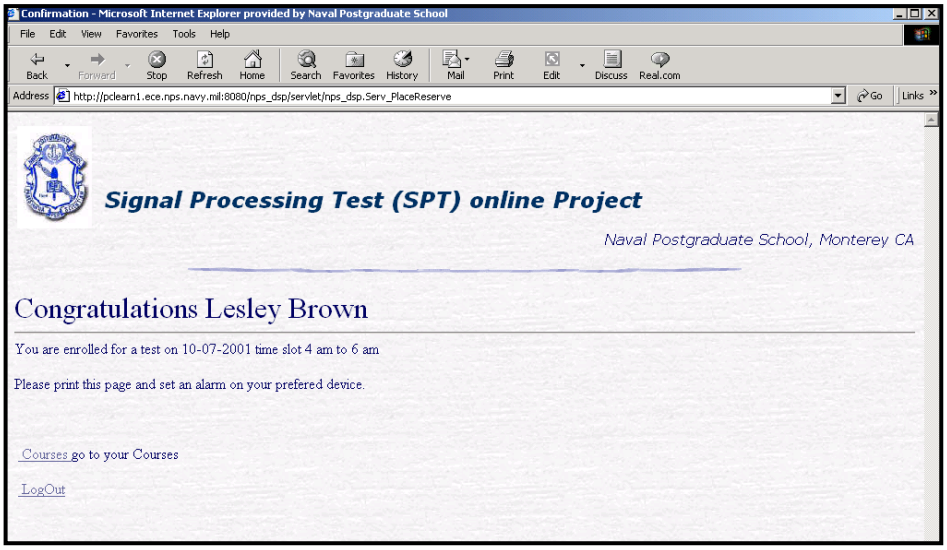

Figure C15. Reserve Time-Slot Confirmation.

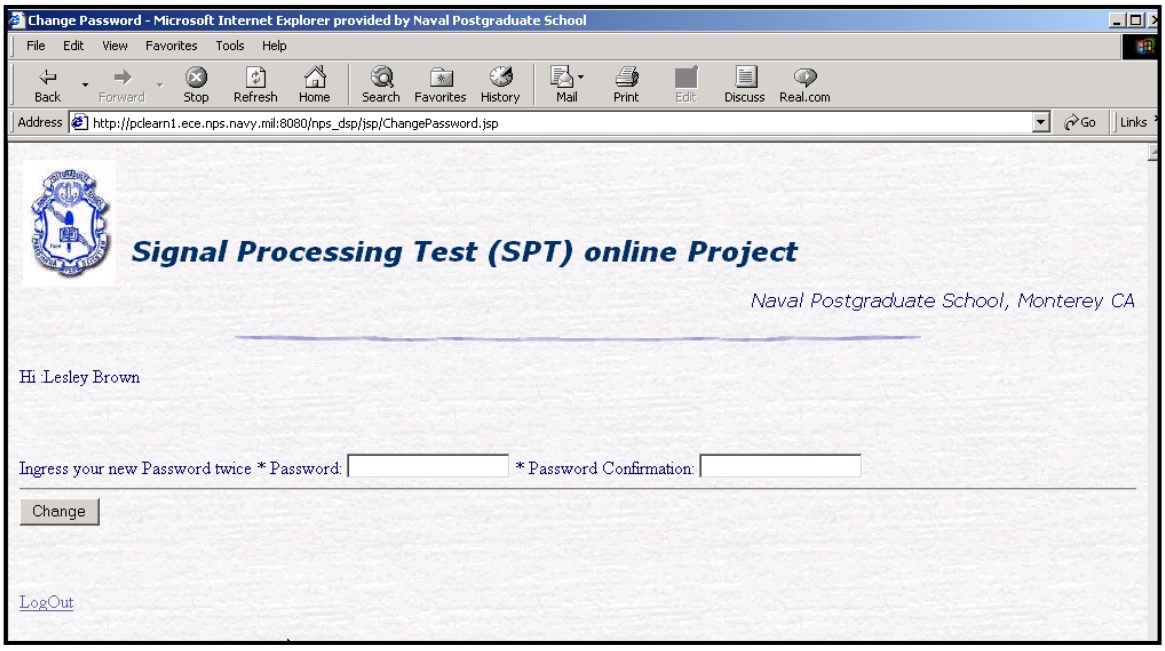

Figure C16. Change Password.

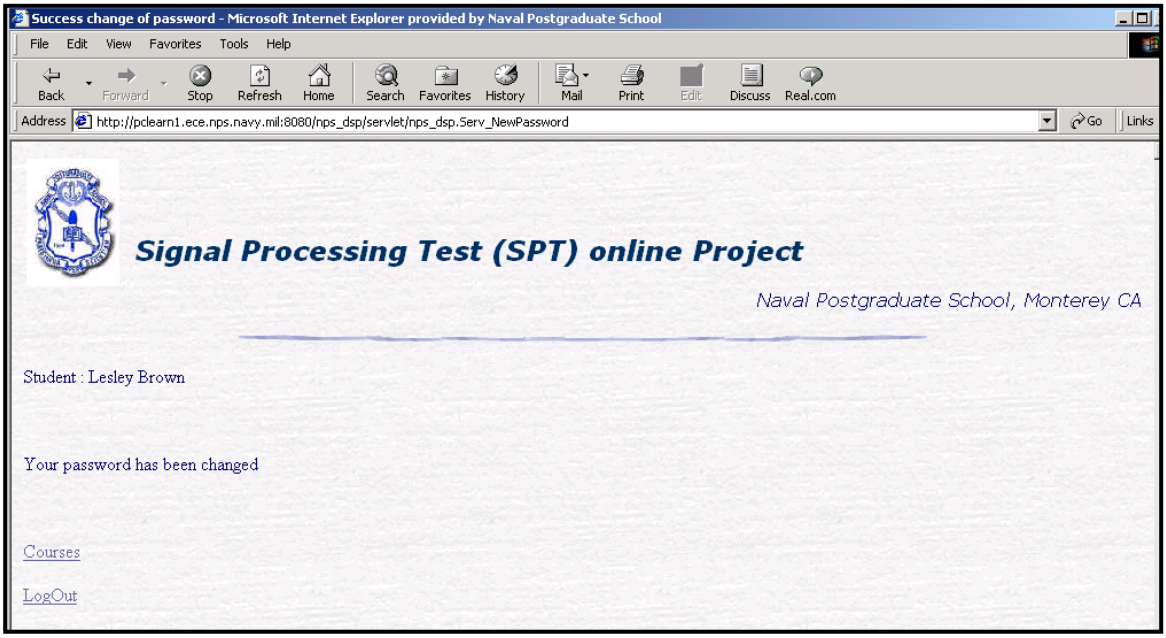

Figure C17. Change Password Confirmation.

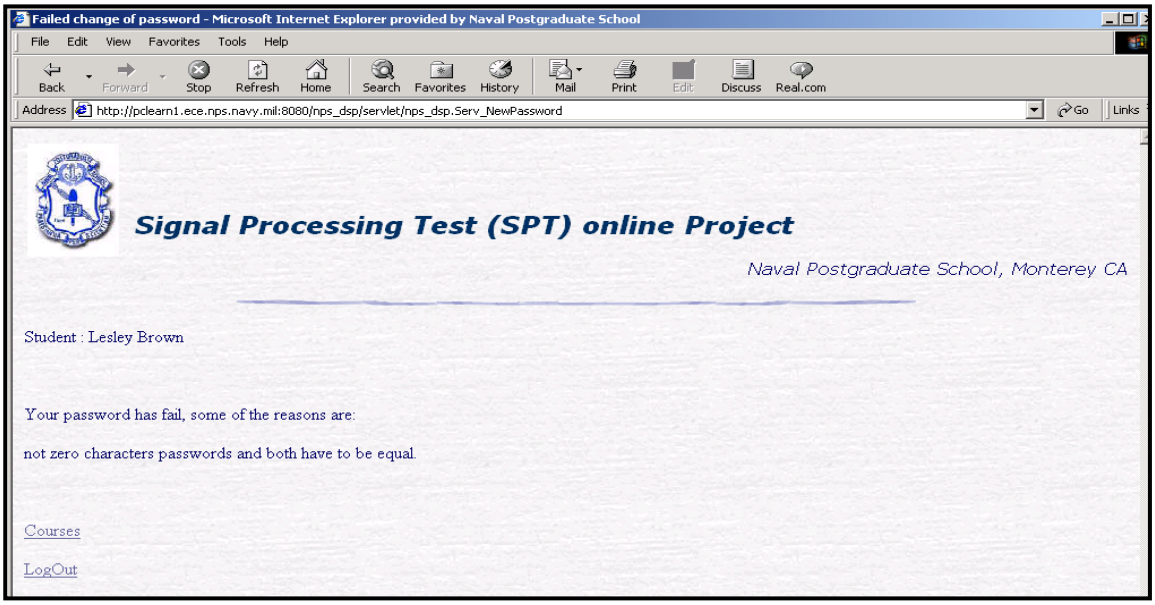

Figure C18. Change Password Failure.

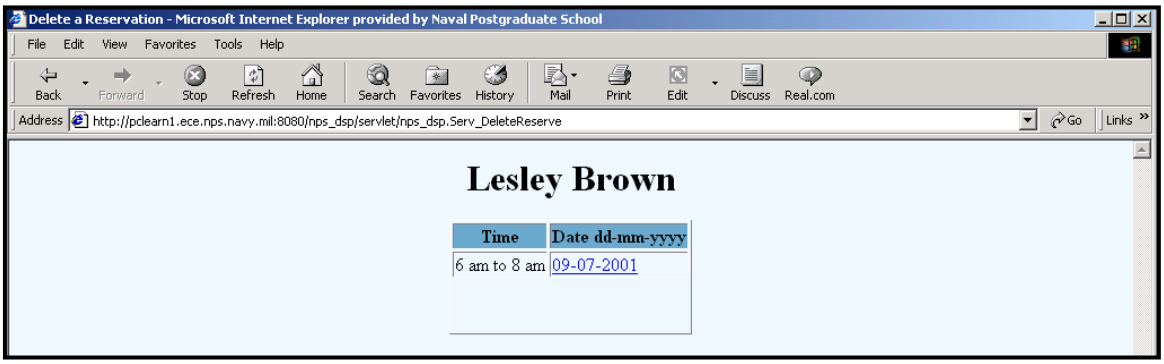

Figure C19. Delete Previous Reservation.

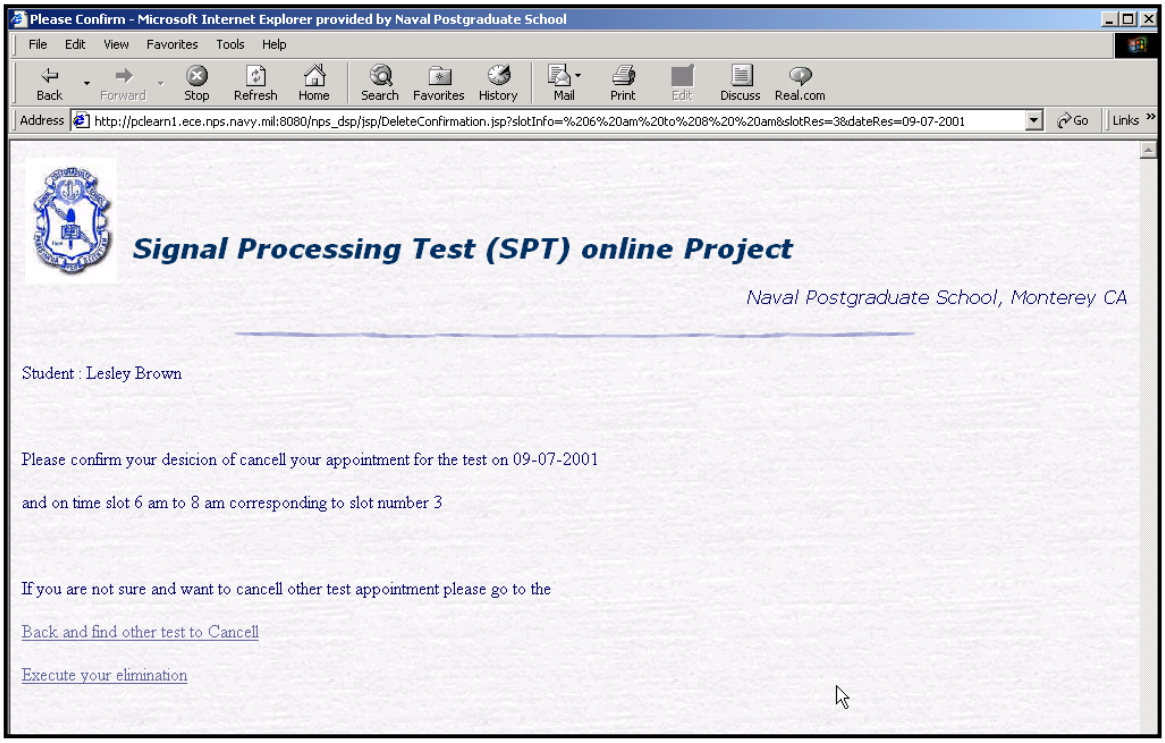

Figure C20. Delete Reservation Request Confirmation.

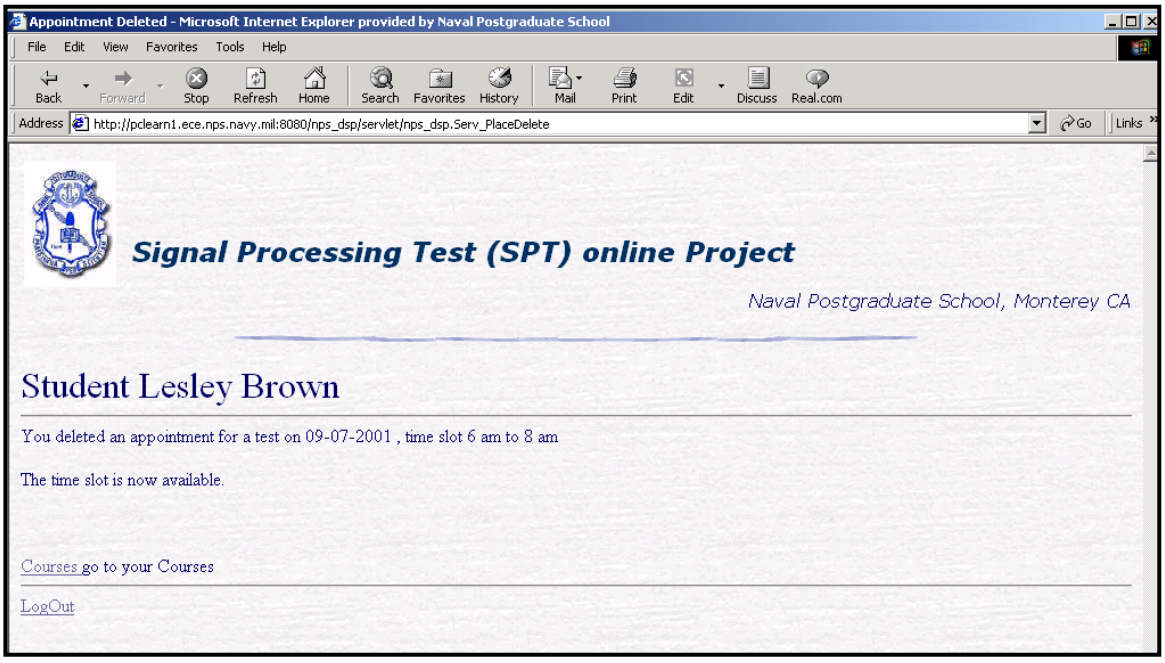

Figure C21. Delete Reservation Confirmation.

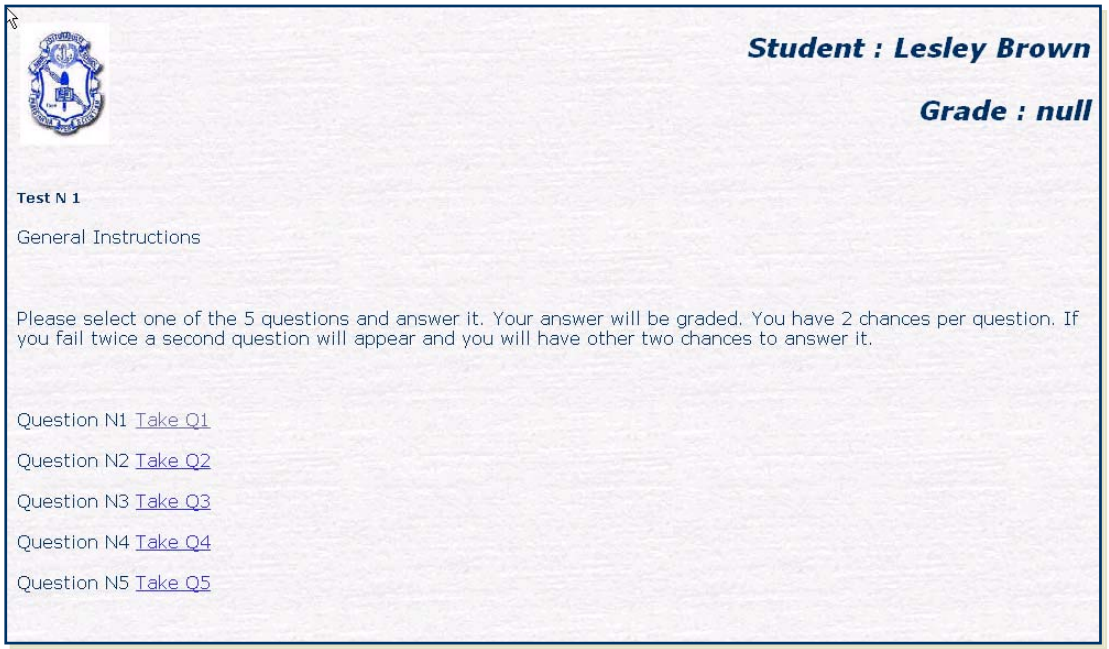

Figure C22. The Test.

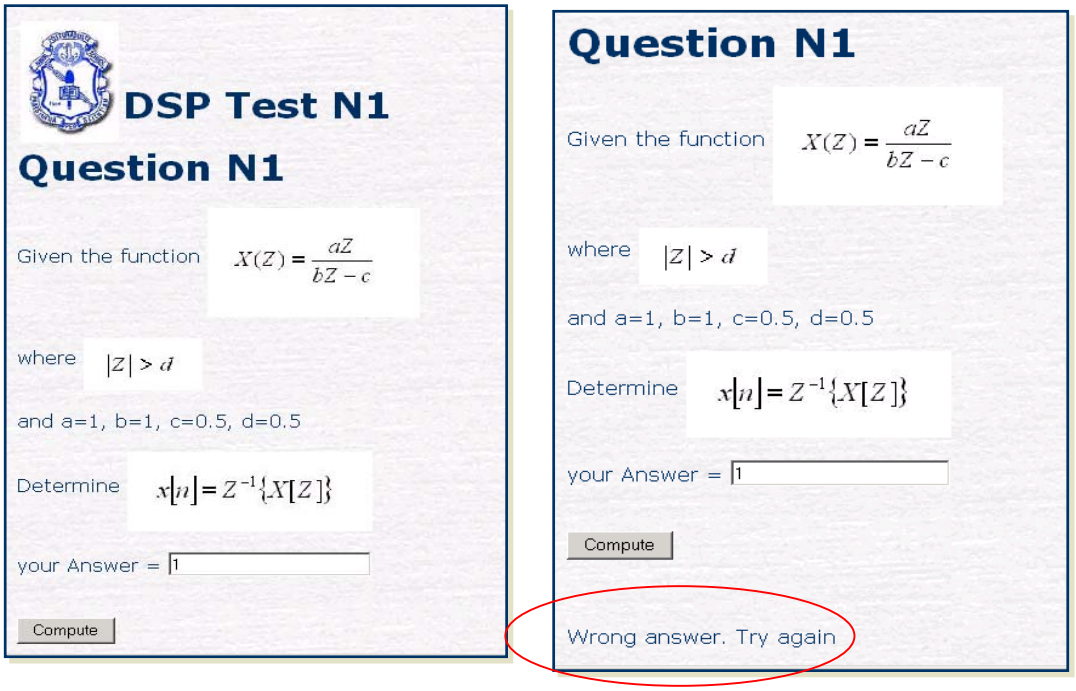

Figure C23. Question and Wrong Answer.

Figure C24. Correct Answer Confirmation.

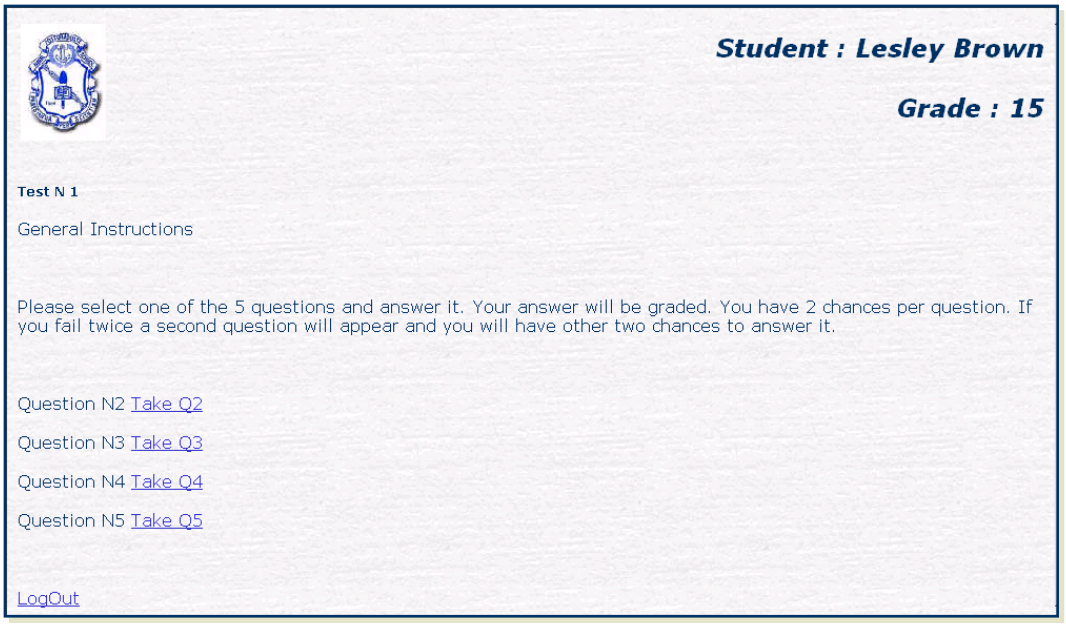

Figure C25. Test Grade.

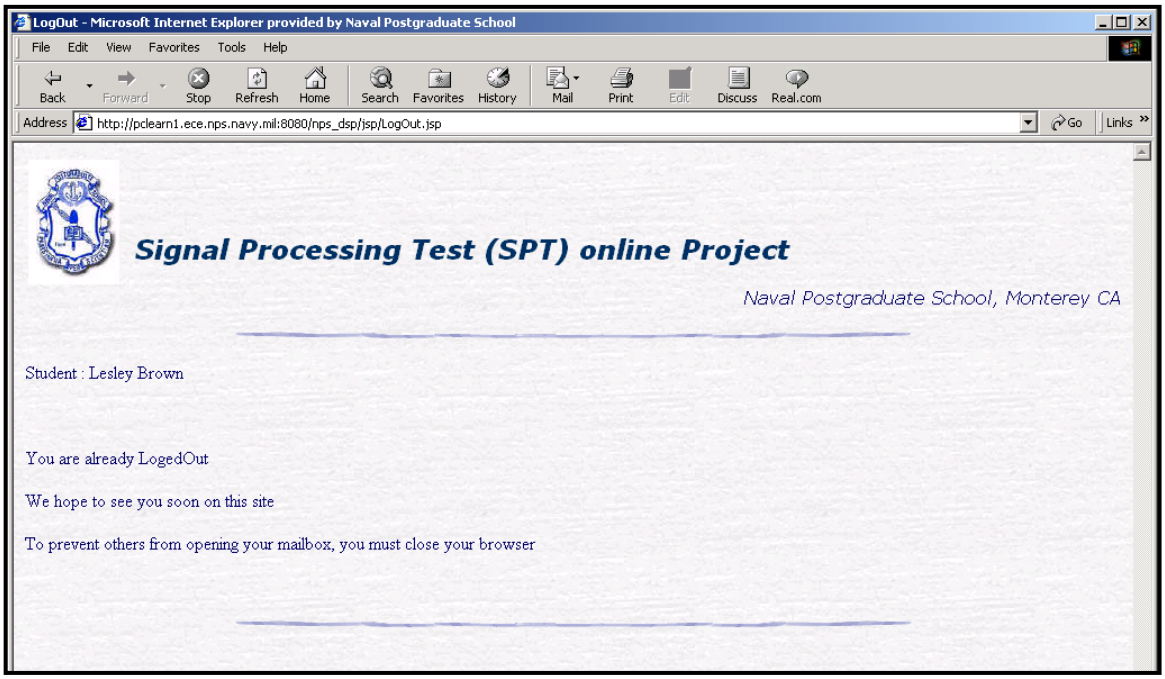

Figure C26. Logout Confirmation.

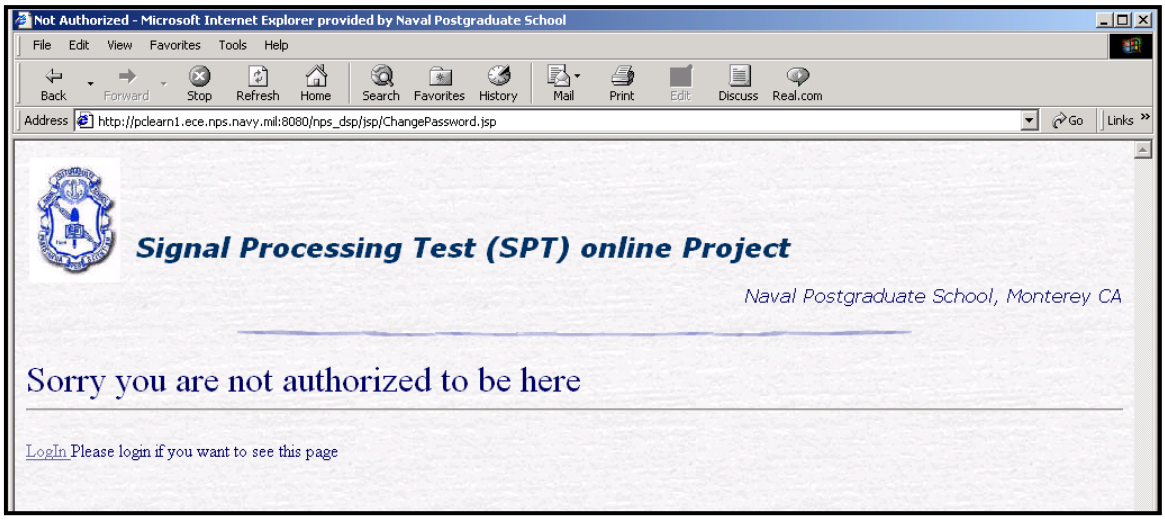

Figure C27. Security Control.

LIST OF REFERENCES

[1] CDi Simulated Quizzer for CCNA™ Certification Training, http://www.netwind.com/html/cisco_exam_questions.html, September 2001.

[2] Etter, D.M.; Orsak, G.C. "*Expanding Team Experiences in DSP Education, Acoustics, Speech, and Signal Processing." Proceedings from ICASSP-97, 1997 IEEE International Conference on Signal Processing*, Volume: 1, 1997, Page(s): 11 -14 vol.1.

[3] Brown, D.J.; Covington, M.; Swafford, M.L. "*Mallard TM: An Educational Tool for Digital Signal Processing," 1996. Proceedings of the Thirtieth Asilomar Conference on Signals, Systems and Computers*, Volume: 1, 1997, Page(s): 231 -235 vol.1.

[4] Web page for: Mallard, an ansynchronous instructional environment. http://www.ews.uiuc.edu/Mallard/, September 2001.

[5] Web page for J/Link is a toolkit that integrates Mathematica and Java. It lets you call Java from Mathematica in a completely transparent way, and it lets you use and control the Mathematica kernel from a Java program. http://www.wolfram.com/solutions/mathlink/jlink/, September 2001.

[6] Web page for WebMathematica program in beta version from Wolfram Research Inc. http://www.wolfram.com/products/webmathematica/, September 2001.

[7] Graham Hamilton, Mark Hapner, Maydene Fisher, "*JDBC™ API Tutorial and Reference: Universal Data Access for the Java 2™ Platform: The Java™ Series*", 2/e Addison-Wesley 1999

[8] Thomas Wu, Database Systems, class notes and projects from CS 3320, Naval Postgraduate School, Monterey, CA, Fall 2000.

[9] Rob Beck and Dave Aitel, NPS CISR Invited Lecture Series "*SHAREFUZZ – Security Related Stress Testing Tool*," @stake, August 1 2001

[10] Jakarta is a project that creates the Tomcat Application Server, details can be read from their web site at http://jakarta.apache.org/index.html , September 2001.

[11] Java 2 Plataform Standard edition, product family http://java.sun.com/j2se/, September 2001.

[12] Pierre-Alain Muller, "*Instant UML*", Wronx Press Ltd.(r) 1997, reprinted 2000.

[13] Presnetation at NPS by Nishikant Sonwalkar MIT Director of the Hypermedia Teaching Facility, July 2001. Mr. Sonwalkar research can be found at http://caes.mit.edu/people/nish.html, September 2001.

[14] Roberto Cristi, course notes from EC2400 Signal and Systems and EC3400 Digital Signal Processing, Naval Postgraduate School, Monterey, CA, Fall 2000 and Winter 2000.

[15] Roberto Cristi, Signal and Systems, a book written in digital format unpublished yet, Naval Postgraduate School, Monterey, CA, Summer 2001.

[16] Vincent Filby, is writing an equation editor for the web in the Java programming language, his work can be found at http://www.stasis.org/~vfilby/projects.shtml. Mr. Filby can be reached at vfilby@acm.org

[17] The Java Developer's Resource Chapter Two: Installing Java. Material can be found at: http://www.ibiblio.org/javafaq/books/jdr/chapters/02.html, September 2001.

[18] Tom Wickham-Jones, "*webMathematica: A User Guide*", Preview Version 0.93 May 27, 2001.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Prof. Jeffrey Knorr
   Naval Postgraduate School
   Monterey, CA 93943
   knorr@nps.navy.mil

4. Prof. Roberto Cristi
   Naval Postgraduate School
   Monterey, CA 93943
   cristi@nps.navy.mil

5. Prof. Thomas Wu
   Naval Postgraduate School
   Monterey, CA 93943
   ctwu@nps.navy.mil

6. Prof. Jon Butler
   Naval Postgraduate School
   Monterey, CA 93943
   jmbutler@nps.navy.mil

7. Prof. Arnold Buss
   Naval Postgraduate School
   Monterey, CA 93943
   abuss@nps.navy.mil

8. Prof. Anthony Ciavarelli
   Naval Postgraduate School
   Monterey, CA 93943
   aCiavarelli @nps.navy.mil