

**Web-Based Software Conversationalist**

**Matt Ramsay**

**Faculty Sponsor: Dr. Dick Blandford**

**Faculty Advisor: Dr. Dick Blandford**

**University of Evansville**

**Evansville, IN**

**4/20/02**

## **Acknowledgements**

I would not have been able to complete this project without the help of my faculty advisor, Dr. Dick Blandford. I would like to formally thank Dr. Blandford for the initial suggestion of this project and his help with its implementation.

## **Table of Contents**

Introduction .....	Page 1
Problem Statement .....	Page 2
HTML Design .....	Pages 3-4
CGI Design .....	Pages 5-7
Results .....	Pages 8-10
Project Maintenance & Future Use.....	Pages 11-12
Conclusion .....	Page 13
References .....	Page 14

## List of Figures

Figure 1: Screenshot of Default Screen .....	Page 3
Figure 2: Working Example 1 .....	Page 8
Figure 3: Working Example 2 .....	Page 9

## Introduction

This paper will describe the project that I implemented during the 2001-02 school year to fulfill the senior project requirements outlined in the EE 495 and EE 497 courses at the University of Evansville. This paper will introduce my project, describe and outline the problem that it intends to solve, summarize the design process used to combat this problem, document the resulting software and draw a conclusion based on the entire process.

In the Spring of 2001, as part of the EE 494 course, Dr. Dick Blandford proposed a software conversationalist project during his presentation of potential senior projects. He suggested that this piece of software be able to answer questions about the University of Evansville's Electrical Engineering and Computer Science Department. The tool would be used during the recruitment and orientation of new students for the department, to answer their individual questions and those of their parents. Dr. Blandford proposed that the project allow the user to enter a question into a "question field" then press a "submit" button. The software should then decipher the question, process its meaning and return a reasonably accurate answer.

Initially, Dr. Blandford suggested that I write the project in C or C++, but I decided to implement the software using a combination of HTML and Perl CGI scripts. By coding the software in this way, I was able to make the entire project web-based. The finished software will now be posted on the Electrical Engineering and Computer Science Departmental website. This allows potential students and their parents to request information about the department from home and get a somewhat specific, immediate response.

## **Problem Statement**

As hinted to earlier, the Electrical Engineering and Computer Science Department at the University of Evansville was in search of a way for potential students and their parents to attain specific information about the department without having to unnecessarily contact a faculty member at the university. To solve this problem, Dr. Blandford proposed that a student write a piece of software could “talk” to these parents and students by answering their questions as they submit them. After I agreed to take on the project, he suggested that my program be on a few machines at the university so that it could be used during on-campus departmental recruiting events.

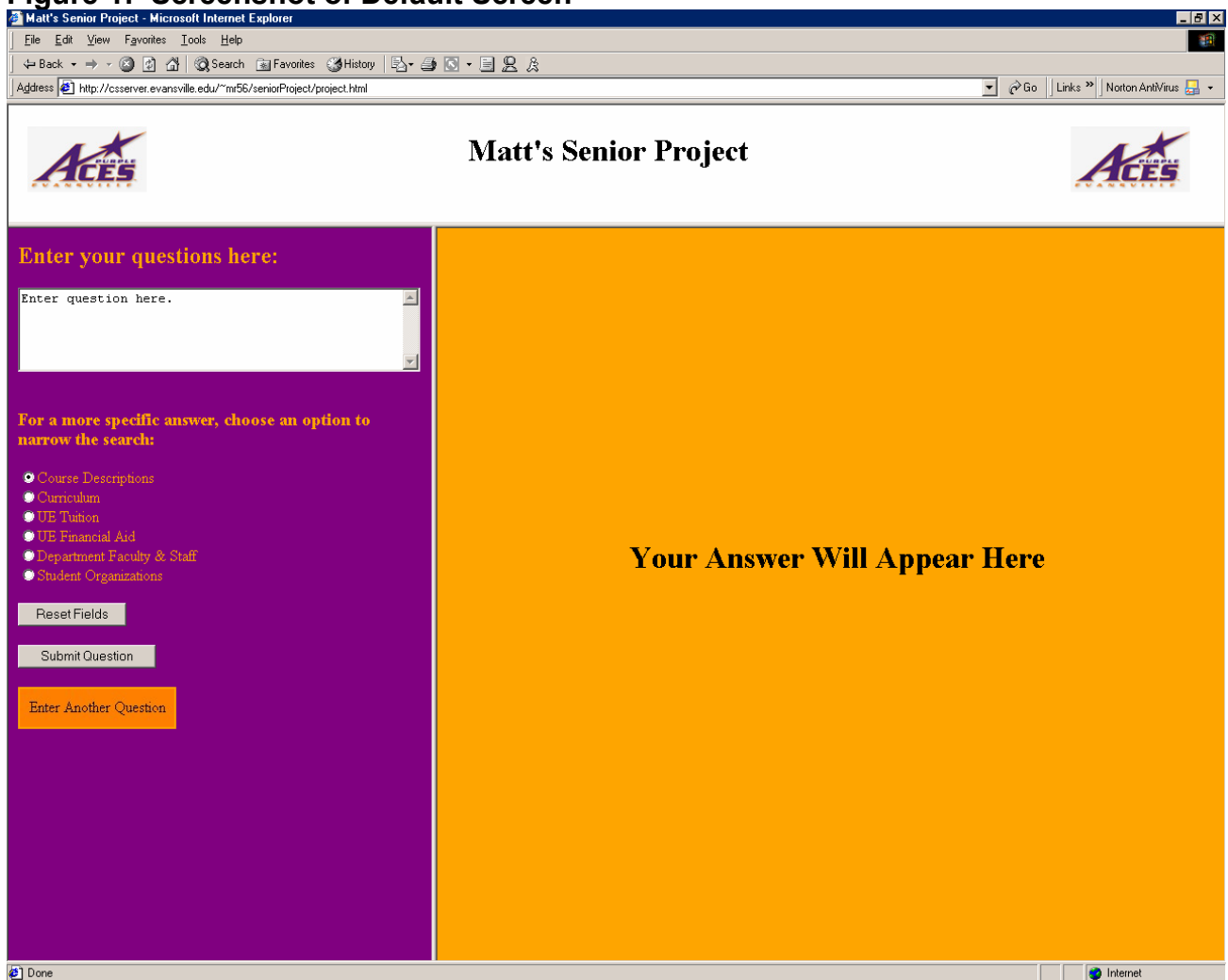
After deciding to take on this project, I examined this problem further. It seemed as though the initial plan for this project did not completely solve the problem. Though my project would provide a working software conversationalist program, the parents and students that would potentially use this program would still have to come to campus to use it. If these users are already on campus, they could just ask a faculty their question, instead of using the program. This initially proposed solution would benefit the faculty by eliminating some of the simpler and more annoying questions from visitors to campus, but it would not benefit the potential users of the program at all.

I suggested that the software conversationalist program be implemented using a combination of HTML and Perl CGI scripts, enabling the tool to be located on the departmental website. This new design adds some programming limitations, but makes the tool much more helpful for the users. Potential departmental students and their parents can now submit questions from anywhere in the world at any time and receive an immediate response.

## HTML Design

To combat the previously presented problem, I designed a three framed website, each frame managed by a different program. The first tier of software is the HTML file, **project.html**, which sets up the webpage and divides it into three frames: 1) the title frame; 2) the question and options frame; and 3) the answer frame. These frames are clearly visible in Figure 1.

**Figure 1: Screenshot of Default Screen**



The title frame is obviously located at the top of the webpage. It contains some unnecessary logos and the title of the webpage, currently "Matt's Senior Project." This

title will be changed on the posted version of the software. The title frame is maintained by the file **header.html**.

The question and options frame provides the user with a textarea for question entry. It also features a set of radio buttons that allow him/her to choose the topic to which his question most closely pertains. These buttons were included to divide the topic search by the software into smaller sections, allowing the answers returned to be more specific and accurate. This search division also made the code much easier to write, cutting its length considerably. Also located in this frame are three buttons:

- **Reset Fields:** This button removes the previously entered question from the textarea, restores the textarea's default text and places the radio button back to the default setting.
- **Submit Question:** This button submits the current question to the CGI program and causes an answer to be displayed in the answer frame.
- **Enter Another Question:** This button returns the answer frame to its initial value, that of **project3.html**.

The question and answer frame is maintained by the file **project2.html**.

The answer frame, set initially by the file **project3.html**, is the frame in which the answer to the submitted question will appear. This answer is sent to the answer frame from the Perl CGI script, **senior.cgi**. The answer field displays either text or HTML, depending on what is returned from the CGI program.

## CGI Design

The CGI program that manages the question and answer part of the software is set up in the following way. Initially, the program reads in the entire question from the HTML program into a character buffer. The buffer is then split into individual words, which are saved as the elements of an array called question. This is done with the following line of code:

```
@question = split(' ', $paramList[1]);
```

The elements of the question array are used throughout the entire CGI program as the targets of all pattern matching and answer selection comparisons.

Immediately following the formation of the question array, the status of the category radio button is checked. This status determines which category of answers should be considered for the submitted question. The following section of code makes the category selection:

```
if($paramList[0] == '0'){  
    // user chose "Course Description"  
}  
if($paramList[0] == '1'){  
    // user chose "Curriculum"  
}  
.  
.  
.  
if($paramList[0] == '5'){  
    // user chose "Student Organizations"  
}
```

As alluded to earlier, the ability to break up the search into categories enables the software to more accurately interpret questions and to formulate much more accurate answers. Also, these categories determine what type of data is returned from the CGI program to the answer frame of the webpage. If the user chooses the "Department



analyzed and an answer is returned to the webpage. The following example, which does not actually exist in the CGI code, illustrates this process on a basic level.

```
for($i=0; $i <= $#question; $i++)
{
    if($question[$i] =~ computer & $question[$i+1] =~ science)
    {
        printCS();
        exit (0);
    }
    elsif($question[$i] =~ computer & $question[$i+1] =~ engineering)
    {
        printCoE();
        exit (0);
    }
    elsif($question[$i] =~ electrical & $question[$i+1] =~ engineering)
    {
        printEE();
        exit (0);
    }
}
```

In this piece of code, the question array is searched for three sets of key words. If the consecutive words “computer science” occur anywhere in the question, the printCS() function will be called. Similarly, if the question contains the phrase “computer engineering”, the printCoE() function will be called and if the question contains the phrase “electrical engineering”, the printEE() function will be called. Every pattern matching call in the entire program works in the same way, with the majority of pattern matching sections being more complicated than the sited example.

The remaining section of the CGI program is made up of function calls. Each function called during this program strictly outputs data back to the webpage. I attempted to make the code as modular as possible, making it much easier to maintain.

## Results

The resulting piece of software should be a useful tool for the University of Evansville's Electrical Engineering and Computer Science Department. It allows potential students and their parents to receive specific information about the department at any time, from anywhere. The following example shows the software returning an answer for two different questions. Figure 2 shows the software producing an answer to the question, "Who is Dr. Blandford?" under the category "Department Faculty & Staff." You can see that the software returns all of Dr. Blandford's personal information,

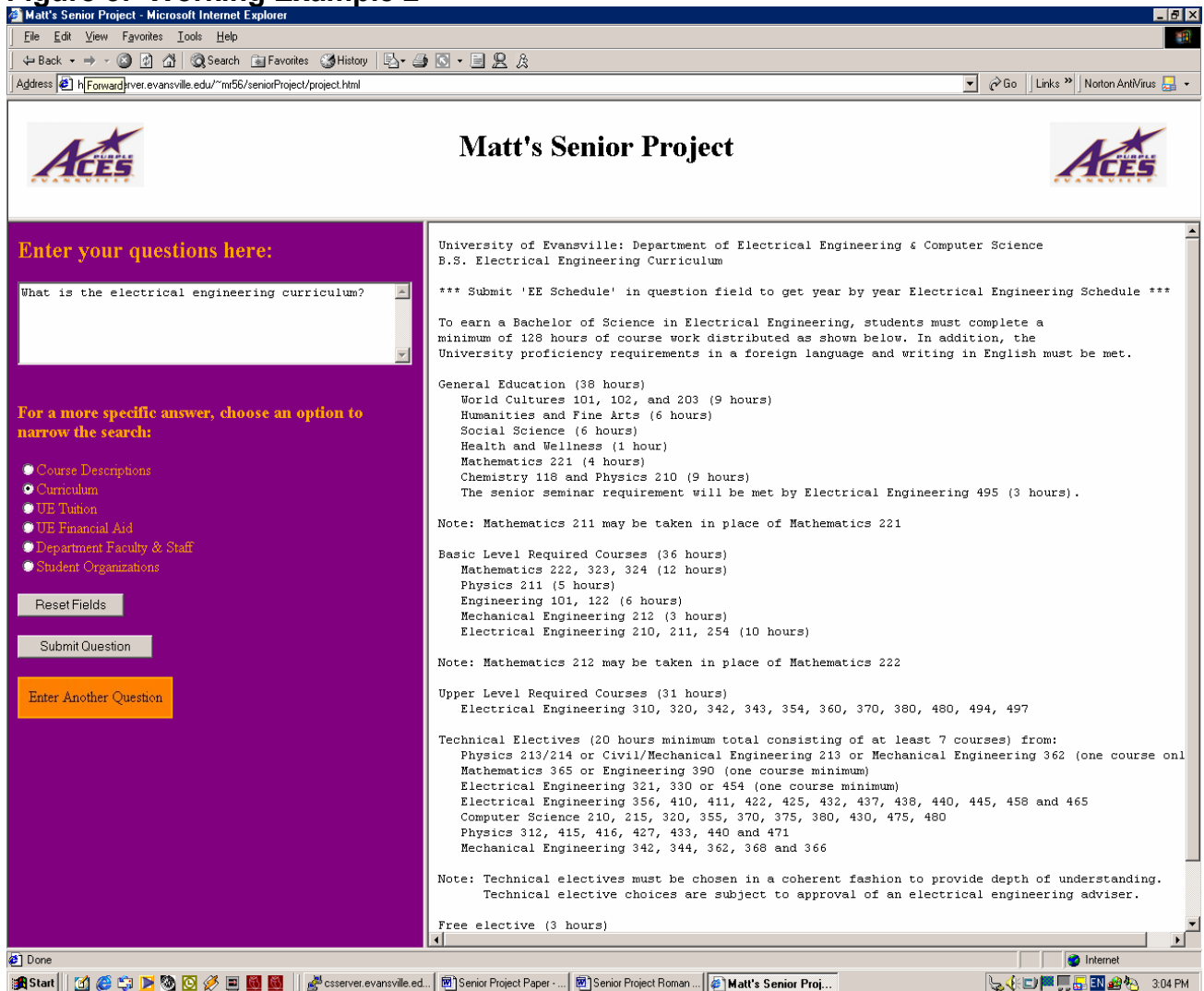
**Figure 2: Working Example 1**

The screenshot shows a Microsoft Internet Explorer browser window displaying a web page titled "Matt's Senior Project". The browser's address bar shows the URL: <http://csserver.evansville.edu/~ml56/seniorProject/project.html>. The page features the "ACES" logo on both sides and the title "Matt's Senior Project" in the center. On the left side, there is a purple sidebar with the heading "Enter your questions here:" and a text input field containing the question "Who is Dr. Blandford?". Below the input field, there is a list of search categories: "Course Descriptions", "Curriculum", "UE Tuition", "UE Financial Aid", "Department Faculty & Staff", and "Student Organizations". The "Department Faculty & Staff" category is selected. Below the list are buttons for "Reset Fields", "Submit Question", and "Enter Another Question". On the right side of the page, the search result for "Who is Dr. Blandford?" is displayed, showing his name and contact information: "Dr. Dick Blandford", "Department Chair, Associate Professor", "Ph.D. University of Illinois", "Phone: (812) 479-2291", "Email: [blandfor@evansville.edu](mailto:blandfor@evansville.edu)", and "Web Site: [csserver.evansville.edu/~blandfor](http://csserver.evansville.edu/~blandfor)". The browser's taskbar at the bottom shows several open windows, including "csserver.evansville.ed...", "The Masters Golf Tou...", "Senior Project Paper...", "Senior Project Roman...", and "Matt's Senior Proj...". The system clock in the bottom right corner indicates the time is 2:44 PM.

including his education background, his office phone and email address, and a live link to his educational website. The live links returned as part of the answer will display the indicated web page in the answer frame of the project, except for those sites not able to be displayed in this fashion. This example is an instance of the software returning HTML instead of just plain text.

Figure 3 shows the software producing the answer to the question “What is the electrical engineering curriculum?” under the “Curriculum” category. To answer this question, the software returns the electrical engineering curriculum as it is shown in the

**Figure 3: Working Example 2**



University of Evansville's Academic Catalog. This is an instance of the software returning text back to the webpage. It is able to do this because there are no active webpage links returned as part of the answer.

In addition to answering the example questions, the software is capable of returning the same personal information for each faculty member in the department, a detailed description of each course offered by the department, a detailed curriculum and year-by-year schedule for each degree program offered by the department, information on tuition and financial aid offered by the university and information on each student organization supported by the department.

## Project Maintenance & Future Use

Considering that my software will be used by the department after I graduate, an important piece of the software documentation should discuss the maintenance of the code. As the department's faculty, classes, tuition and organizations change, the software will have to be updated so that it continues to return accurate answers.

To change the answer returned for a specific question or set of questions, the new programmer must only search through code, locate the answer that he wants to change, and edit the output. No change to the pattern matching code is required to change the output for a specific question. This makes the maintenance of the code much simpler, considering that editing pattern matching code is more difficult and can be intimidating.

The addition of support for a newly anticipated question is a slightly more complex. The programmer must first locate the category that the question will be supported under and then add an **elsif** statement to match the pattern in the added question. The output should be added inside this **elsif** statement with print statements. The following example shows the code that should be added to support questions about a fictitious class called "EE 500."

```
if(($question[$j] =~ EE | $question[$j] =~ ee) & $question[$j+1] =~ 500)
{
    // output about EE 500
}
```

This piece of code will generate an answer for any question containing the consecutive words "EE 500" or "ee 500."

Two main difficulties arise when adding new questions to the software. The first of these difficulties is caused by the sequential nature of the CGI code. The software is

set up to respond to the first pattern that it matches. The new pattern matching statement must be placed in the proper order so that it is not preempted by broader pattern matching statements that are also in the sequence. During the implementation of my software, I checked for this order every time that I added a new pattern match. The new programmer will have to do this check during code maintenance as well, usually by trial and error.

The second obstacle that is encountered when adding a new pattern matching section is the output type restriction. Currently, only two search categories support HTML output, the others are limited to text. This is due to the fact that HTML output was never required in the answers of these sections. The only problem that could be caused by this limitation is if a new pattern matching section requires HTML output and is not located in a section that currently supports HTML. The programmer would then have to change all of the outputs in that section of code to HTML and reset the content type of that entire section.

Other than the specifically mentioned problems, code maintenance for this piece of software should be fairly simple. Each possible answer is generated by an individual pattern matching section of code. Changes to that section of code affect that specific answer only, allowing for small changes to be made easily.

## **Conclusion**

Working on this project has taught me the power of a pattern matching programming language. With a relatively simple, but lengthy, piece of code I was able to produce a useful piece of software. I also learned to appreciate the difficulty of artificial intelligence software. Hopefully my piece of software will be used by the Electrical Engineering and Computer Science Department at the University of Evansville for a long time to come.

## References

1. Christiansen, Tom, Randal L. Schwartz and Larry Wall. Programming Perl. Paris: O'Reilly, 1996.
2. Matthew, Neil and Richard Stones. Beginning Linux Programming. 2<sup>nd</sup> Ed. Birmingham, UK: Wrox Press, 1999.
3. Powell, Thomas A. HTML, The Complete Reference. 3<sup>rd</sup> Ed. New York: McGraw-Hill, 2001.
4. University of Evansville, Department of Electrical Engineering and Computer Science. <<http://cecspf.evansville.edu>>
5. University of Evansville. <<http://www.evansville.edu/academics/>>