A QCARD Wrapper

Albert Choi

CISC 499 Project Report

April 14, 2006

Supervisor: Dr. James Cordy

Abstract

This paper details the work that I did as part of my CISC 499 undergraduate project at Queen's University. The project consisted of defining and creating a better interface for the QCARD registration system in the form of a wrapper. A reverse proxy server method, written in PHP, was used to wrap the existing system and redisplay the data in the new format. Six of the QCARD pages were chosen for redesign and five of them were successfully implemented in the online wrapper. Several navigational problems with the existing system were identified and all of these were fixed in the new interface. The result was a functional wrapper that provided improved access to the actual QCARD data using only techniques that were publicly available.

Contents

Introduction

The Queen's Computerized Access and Registration Database (QCARD) is the online database system used by all Queen's University students to register and view their courses, timetable, fee account, marks, and other important information. While the system is accessed through a web browser, the interface still resembles that of the old terminal computer system, text-based with a limited amount shown on each screen and function keys to navigate between screens. The function keys have been replaced by form buttons and some layout has been replaced by tables, but otherwise the interface is still the same (Figure 1). Yet today's graphical web browsers are capable of much more, and this has left QCARD feeling dated and cumbersome to use in comparison.

Figure 1. The QCARD interface is still very much like the text-based system, including buttons for Page Up and Page Down. Also, accessing other pages almost always requires returning to the Main Menu.

Goals of the Project

The goal of this project is two-fold: design an improved interface and implement it so that students can use it to access real QCARD information when the project is

complete. We have chosen to implement it as a wrapper, i.e. instead of accessing QCARD's back-end database directly, we will use the existing QCARD web interface and create a second front-end for it, effectively 'wrapping' the site. This can be done with what is already publicly accessible and does not require the QCARD team to provide us with any special privileges. As such, the project will entail an amount of reverse engineering. Additionally, the wrapper technique means that our implementation should not be used as a permanent replacement for the system, even though we would like to see the design part replaced. Thus this project will try to keep the design portion and the wrapper portion as separate and modular as possible.

Tools and Technology

HTML 4.01 and CSS 2.1

The improved QCARD interface was developed as a website conforming to HTML (HyperText Markup Language) 4.01 Transitional and CSS (Cascading Style Sheets) 2.1 specifications. These were chosen with the intention that the interface should render perfectly in Internet Explorer 6.0, Mozilla 1.0 and variants (including Netscape 7.0 and Firefox), Safari (based on Konqueror), and Opera 8.5, but will still look acceptable in older browsers. Specific fixes were included to enable it to also render correctly in Internet Explorer 4.x and 5.x.

Reverse Proxy Server and PHP 4

The wrapper part is created using a reverse proxy server, specifically a Linux web server running PHP 4. Like any proxy server, it sits in between the user and the QCARD server, relaying information between them. The difference between a forward proxy and a reverse proxy is in its usage – a forward proxy hides the identity of the user and allows the user to be protected by a firewall, whereas a reverse proxy hides the identity of the web server/application and allows the server to be behind a firewall (accessible only through the proxy) (Stricek, 2002). In our case, the proxy is placed on a publicly accessible web host that is on neither the user's nor the QCARD server's internal network, and so is only a reverse proxy server in its purpose.

The reverse proxy works as a wrapper by accepting a user's requests, then navigating the existing QCARD website to find the relevant information. Once this is done, the proxy processes the information and reformats it into our web design. It returns the resulting webpage to the user (Figure 2). By implementing the proxy server on a web server (i.e. a web proxy), the job of serving webpages is already handled.

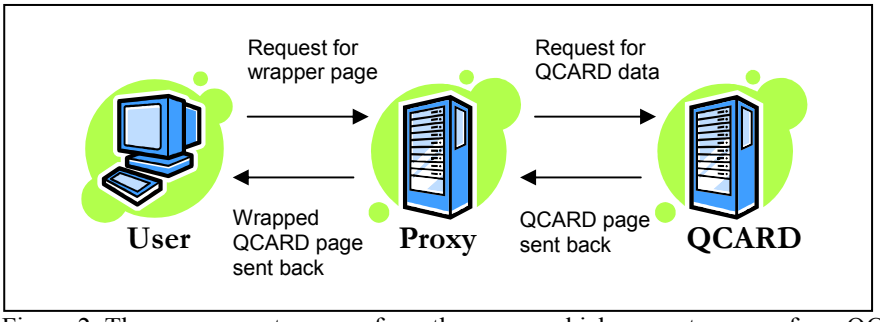

Figure 2. The user requests a page from the proxy, which requests a page from QCARD. QCARD sends data back, which the proxy wraps before sending back to the user.

To accomplish the other tasks, I will write code using the server-side scripting language PHP (PHP Hypertext Preprocessor). As a language, PHP allows the programmer to insert content into a webpage before it is sent to the user. Amongst its

abilities include socket open functions, which the proxy server will use to connect to and read webpages via HTTP. These abilities round out the necessary requirements for the wrapper.

Defining the "Best" Interface

We first decided to establish what students would like to see in an ideal QCARD system. An informal survey of 10-15 students was conducted, asking their opinions on QCARD's functionality, interface, availability, and speed. The last two categories are not part of the goals of the project, but many people have complaints in those areas and these opinions needed to be kept separate. As expected, system availability received the worst score (2.1/5), followed by interface (2.5/5). Speed was the least of students' concerns (3.6/5), and functionality was in the middle (3.1/5). While not completely representative of the student body, these results do confirm that many students share the same annoyances with the interface that we do. Students were also asked for suggestions for improvement, and from these responses we were able to form a list of ideal capabilities for QCARD to have.

<u>The Ideal QCARD Interface</u>

- One-click Navigation
    - a navigation menu that is always available
    - should not have segmented pages with page up/page down buttons
    - should not have "dead ends" (e.g. Account Statement) that require you to logoff and logon

    - eliminate typing “Y” or “FW” in a box to select or drop courses
    - compatibility with browser’s Back button
- Enhanced Display of Information
    - timetable, marks, and fee statement presented in a readable format
    - multiple related pages shown at once, e.g. timetable, course availability, add/drop course
    - highlight parts of the timetable that a course will take up
    - view multiple course information, instead of one at a time
- Allow users to save information for printing or storage
    - printer-friendly version
    - ability to save as image, HTML, Excel file, PDF, etc.
- Additional features
    - show changes in account status, e.g. when marks appear
    - reminders for upcoming deadlines (add/drop, tuition, opt-out, etc.)
    - Q&A, advice, or forum feature
    - offline version when database is down

Defining the Goals for the End Product

The entire list of ideal QCARD capabilities is too large for the scope of this project. We decided that the QCARD wrapper should address the following:

<u>Pages that will be redesigned and connected to QCARD:</u>

- View Timetable

- View Course Offerings
- Add/Drop Courses (consists of Add Courses and View/Change Courses)
- View Marks

Pages that will be accessible from the wrapper, but not necessarily redesigned:

- View Fee Account Information
- View T2202A

Features of the redesign will include:

- a navigation menu
- segmented pages will be consolidated into single pages
- navigation without "dead ends" (e.g. Account Statement) that require you to logoff and logon
- form boxes instead of typing "Y," "FW," etc. (Add/Drop pages)
- function to add course directly from View Course Offerings
- compatibility with browser's Back button
- readable timetable, assuming no course conflicts
- ability to show multiple (2 or 3) pages at the same time
    - Timetable, View Course Offerings, and Add/Drop Courses
- all pages printer-friendly

Implementation Details

Part I: The Design

The first step in the project was to create a static HTML version of the interface, with no PHP code. These design pages would then serve as the targets for the conversion/reformatting step (see Part III: Converting the Pages). Of course I could not anticipate every aspect of the design that would be needed, so at times when working on Part II and Part III, I also updated the design part with additional features.

For the most part, the design adheres to the current web design paradigm of separating style from content. The argument for this separation is well documented on the Internet and I will not go into the details in this report (Wikipedia, 2006). Suffice it to say that by separating the styling of the site into a separate file, a CSS stylesheet, the look of all the pages on the site can be changed simply by modifying one stylesheet. Additionally, because we do not hard-code in table layouts, colours, or font styles into the HTML pages, file sizes are greatly reduced, speeding up the site's load time.

The design of the pages was fairly simple, each one having a consistent layout – a navigation menu on the left and content on the right. In total, eight HTML pages were created: a login page and one page for each of the six sections that we aimed to implement (Add/Drop, Course Offerings, Fee Account, T2202A Tax Form, Marks, Timetable), except for the Marks page which consisted of two pages. All pages link to three stylesheets: *all.css*, *iefix.css*, and *print.css*. The *all.css* stylesheet is always loaded, whereas the *iefix.css* stylesheet is only loaded for Internet Explorer versions below 6, using the following code:

```
<!--[if lt IE 6]>
<link rel=stylesheet type=text/css href=iefix.css>
<![endif]-->
```

The reason for this IE specific stylesheet is to correct for the non-standard box model used in Internet Explorer 4 and 5. This non-standard model, commonly known as the "IE Box Model Bug," differs in the way element widths (or heights) are calculated when a non-zero size padding (or border) is specified (Çelik, 2003). Internet Explorer 6, when operating in standards compliance mode, does not require this fix (Silver, 2001). In the end, the only CSS rules that needed to be put in this file were a few rules pertaining to the timetable layout.

The *print.css* stylesheet is only enabled when printing, at which point it hides the navigation menu, sets the page width to automatically fit the paper, and otherwise accomplishes the task of providing a printer-friendly version of each page, even without creating any new pages.

Timetable Page

One of the more important goals of the project was to make a readable timetable instead of the text-based one currently offered by QCARD (Figure 3). The classes in the timetable should be positioned based on the timeslot, be space-proportional with respect to duration, and colour-coded (Figure 4). To accomplish all this, I originally considered using HTML tables, but in the end decided on using `<DIV>` tags with CSS. The latter method that I used turned out to be easier to code as well as far more flexible to change.

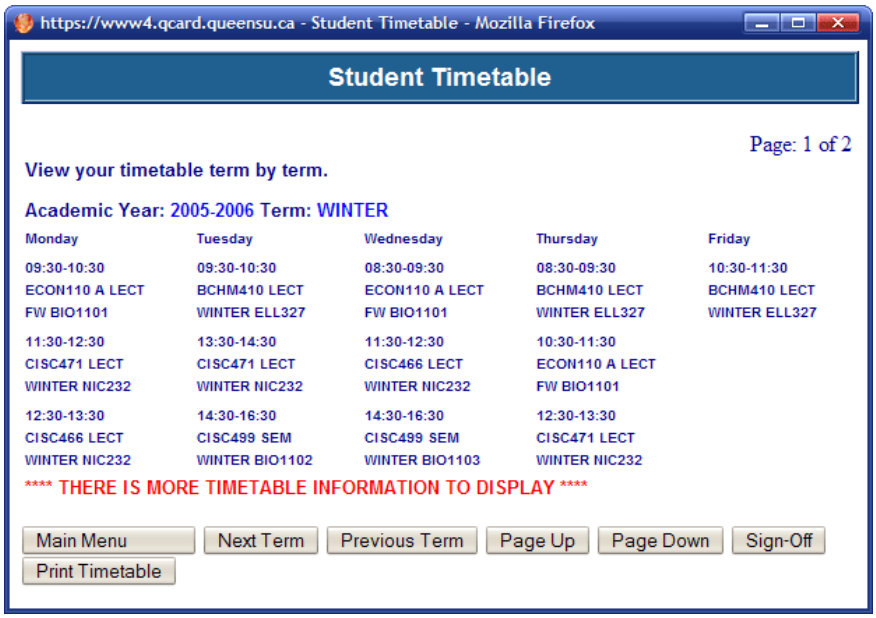

Figure 3. The existing QCARD timetable, text-based and hard to read.

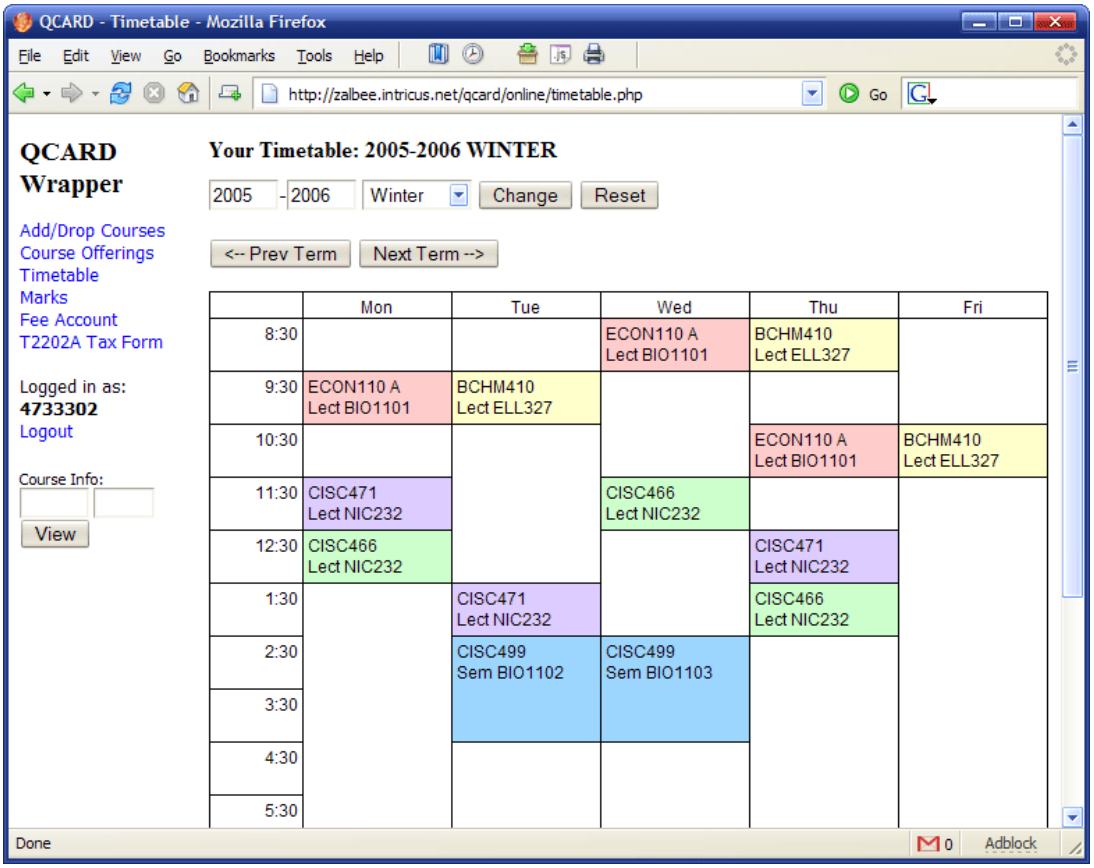

Figure 4. The redesigned timetable with colour-coded, space-proportional lecture slots.

Each element in the timetable that has text is placed in a `<DIV>` tag and is given multiple `CLASS` attributes that describe the column, row, element height, and colour. For example, the code `<div class="col2 row3 ht2 bg4">BCHM410<br>Lect ELL327</div>` will create a box that says "BCHM410 [etc.]" in the 2nd column, 3rd row, with a height of 2 half-hours and the 4th predefined background colour. This positioning represents the Tuesday 9:30-10:30 timeslot. Thus every item in the timetable can be inserted simply with one line of code, independent of the existence of any other items in the timetable. Every element is inserted this way with no exceptions; even the day labels (Monday, Tuesday, etc.) and the time labels (8:30, 9:30, etc.) are done this way.

For this to work, there need to be CSS rules that define what the classes `col2`, `row5`, etc. mean. I devised a simple script that helped me define the pixel measurements of each of these CSS classes, which I wrote in JavaScript (see Appendix B). Originally it was written in this language so that the user's web browser could create the CSS rules dynamically, but I decided it was best to allow the site to work without enabling JavaScript. This script can take a number of parameters (width, height, padding, and border) to generate custom-sized layout rules for both the standard and IE non-standard box model. The output of the script was saved into the CSS stylesheets *all.css* and *iefix.css*.

Add/Drop Courses Page

In the current QCARD system, the "Add or Drop" section actually consists of multiple pages – an initial navigation page and two separate pages for Add Course and View/Change Courses (Figure 5). In the redesign, I wanted to keep all these functions on

the same page, thus merging all of it into a single Add/Drop Courses page. As was desired in the goals of the final product, certain text input boxes where the user needed to type letter codes (term, drop, time-section) were replaced with drop-down menus and radio buttons, bringing the interface more to the level that a web user would expect. A "View" link was added beside each course that would quickly take the user to the Course Offerings page for that course (Figure 6).

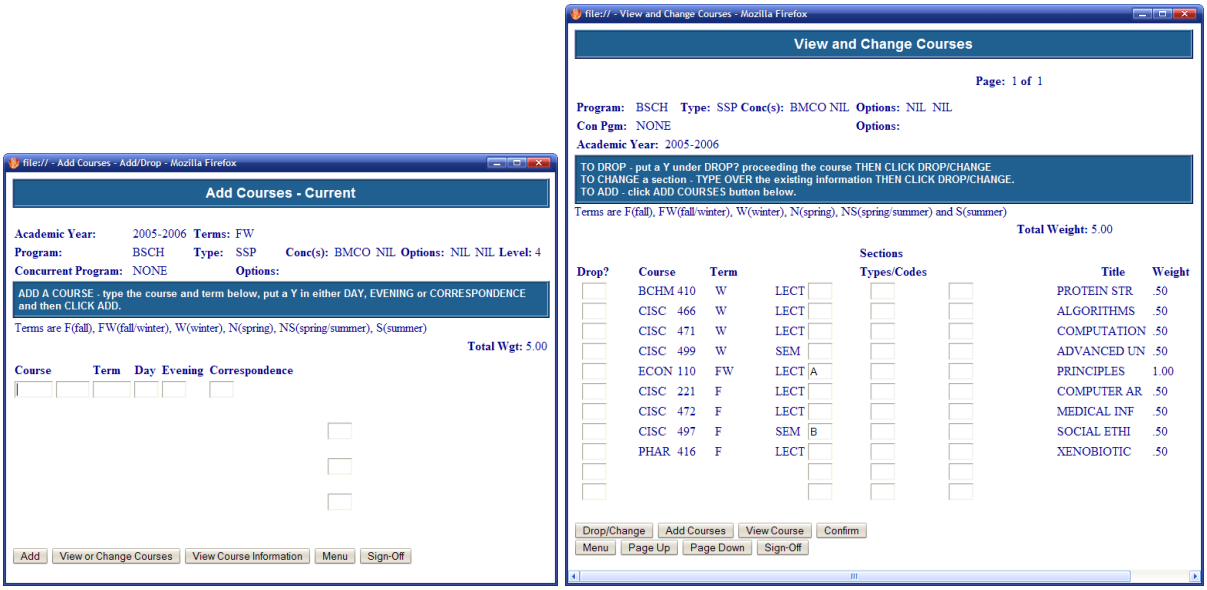

Figure 5. The Add Course (left) and View/Change Course (right) pages are separate in the original QCARD. Almost every function (from adding to dropping) requires the user to type a letter into a text box.

Other Pages

The other pages were not really in dire need of a redesign and so were mostly left intact. The Fee Account and T2202A Tax Form pages are already formatted nicely and so the code was taken directly from the existing pages. Although these pages do not use CSS styling and instead use table layouts and deprecated `<font>` tags that result in inefficient code, it would have taken far too long to recreate the pages in CSS. The redesigned Course Offerings page essentially uses the same table format as the existing page, with

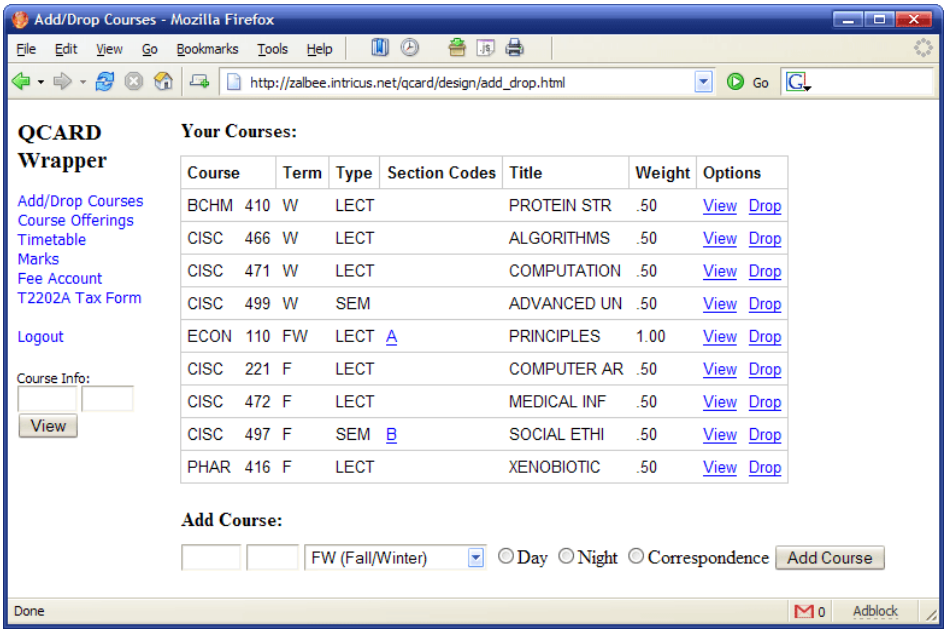

Figure 6. The redesigned Add/Drop page combines the functionality of two pages, and uses dropdown boxes, radio buttons, and links instead of typing letter codes into boxes.

slight changes to fit with the CSS style. An "Add Course" link was also added to the Course Offerings page in accordance with our one-click navigation goal, saving the user from typing in the course code on the Add/Drop page. Finally the Marks section, consisting of two pages (switching between course descriptions and grade distribution), uses the same monospace font for layout as the original QCARD page.

Of all the pages in this section, probably the Marks page was the only one that could have benefited from an additional redesign, perhaps with a tabular version. However time constraints and the fact that the marks are still quite readable meant that I did not do so. Lastly, all the pages that were previously segmented (Course Offerings, Marks, Timetable) were designed as single scrollable pages.

Part II: The Server Connection

After the design portion was complete, the next step was to get the reverse proxy server to extract data from the existing QCARD website successfully. This part entailed

reverse engineering the way QCARD pages are connected and writing PHP code to navigate these areas.

How QCARD Currently Works

When a connection is made to the QCARD login page at https://www4.qcard.queensu.ca/QCD1/CICS/CSMI/DFHWBTTA/CW01, certain key values are given to the web browser, by way of HTML hidden form fields. As a user navigates the pages of QCARD, they are always submitting an HTML form via a POST operation, which means that these hidden variables are always being sent back to the server. The most important of these is a field called `DFH_STATE_TOKEN` which is a random 8-character long string that identifies a unique user (from here on, referred to as the token). This token is found on every page of QCARD and it is necessary to pass back the same variable on subsequent page requests. Similarly on every page there are several other, more obscurely named, hidden fields that I have not been able to identify (see Figure 7). However, they appear to let QCARD identify which page the user is coming from, and they do not change from user to user. These fields are also absolutely necessary.

```
<!-- The following variables are the
names of the fields that could contain the next
CICS transaction id -->
<input type="hidden" name="DFH_NEXTTRANSID.1"
value="F020800008_F010003">
<input type="hidden" name="DFH_NEXTTRANSID.2"
value="F030110001_F010004">
<input type="hidden" name="DFH_NEXTTRANSID.3"
value="F120030007_F010005">
<input type="hidden" name="DFH_NEXTTRANSID.4"
value="F170020008_F010006">
<input type="hidden" name="DFH_NEXTTRANSID.5"
value="F170110001_F010007">
```

Figure 7. Hidden form fields in the HTML source of the QCARD login page.

A third type of form field used is the fields that are only required to access further information from the page, but not necessary for navigation away from the page. These "sometimes required fields" include both user entered input as well as some hidden fields. For example, when pressing a button on the View Course Offerings page that accesses more information (e.g. View Course or Page Down) these fields are required, but when clicking the Main Menu button, these fields are not required for navigation to be successful. This distinction is important because the "always required fields" are constant, whereas the "sometimes required fields" are not. In my PHP code I can statically store these constant values without reparsing the HTML code to find the fields every time.

To summarize, the navigation across pages is always done via a POST request, and each page has a set of required variables that are sent in this request. One can not simply access any page they wish because there are no separate URLs; the only accessible pages are the ones listed on the page as form buttons, and the page that is accessed depends on the value of one of the form fields. To discover the page hierarchy, I manually went through the QCARD pages I was interested in and copied down the "always required fields" for each page, as well as noting which other pages were accessible from each page[1].

Using PHP to Navigate

The essential idea is to use the PHP function `fsockopen()` to connect to the QCARD server. A request, complete with HTTP headers and POST data, can then be sent to the server by writing text to the open socket stream. Reading from the same stream will

[1] I have not been able find a pattern in the form fields that would allow the access of arbitrary pages, but perhaps an interested reader would like to try.

give us the output page. The function I use to do this is named `go()` and has the ability to send both GET and POST requests; it is a modified version of a function on the PHP documentation pages.

In my PHP code, I created a `Menu` class, representing a page (so named because each page behaves exactly like a menu). Each `Menu` object has a PHP array `$defaultInputs` that maps the "always required" form field names to their (constant) values. Each `Menu` object also has a list of child `Menu` objects that represent which pages can be accessed from there, essentially organizing all the pages into a directed graph. To navigate from a page to another, one invokes the member function `submit()` with one or two arguments. The first declares which page to navigate to. The second is an optional array map that contains the full list of form fields and values, including the "sometimes required" ones. To get this full list, a function `getFormInputs()` is used, which goes through the HTML source of the QCARD page and parses out the form fields. It is at this point that the PHP script can change the values of user-entered input fields, by modifying the second argument. Other global functions, such as `login()`, `goMainMenu()` and `menuSignOff()` are also available. The full source code is provided in Appendix C.

While these functions allow the navigation of QCARD from a single linear script, one must remember that a PHP script is run whenever a user loads a PHP page from our proxy/web server. Under normal usage, a user will be loading multiple pages, and so we will need to remember certain variables across multiple runs of the scripts. While QCARD's solution is to pass the variables back through POST, I opted to use PHP's built-in sessions, where a user is remembered through a cookie or session ID. Thus, I can very easily set session variables for the token, current page, and others that would need to

be preserved across multiple page views. Additionally, the "Back" button will work perfectly with this setup, fulfilling another user requested feature. An extra benefit is that a user can open the QCARD Wrapper in multiple windows, and they will all allow access to the pages that are requested.

Upon completion of this part, I was able to have a website running on the proxy/web server that could take a student's student number and password, and return certain pages from the QCARD site, without modification. This was accomplished with all pages except the Fee Account Page; I will explain this in more detail in Part III.

Part III: Converting the Pages

The final stage to complete the wrapper was to take the pages acquired from the navigation process in Part II and convert them to the page designs created in Part I. The code for this part is located in the PHP functions named `getMarks()`, `getTax()`, `getTimetable()`, and `getCourseInfo()`, each of which returns a string of the reformatted HTML code. For the six pages that I set out to implement, this process ranged from trivial to long coding, with one page requiring a workaround, and one page being unfeasible.

Converting T2202A Tax Form

Let us start with the easiest pages. In the design stage, I decided that the Tax Form page and Fee Account page were designed well enough, that both could be used directly.

Converting the Tax Form page was as simple as taking all code between the first <TABLE> and last </TABLE> tags and printing them on screen.

Converting Fee Account Statement

Interestingly, I discovered that the Fee Account Statement is not located on the same server address as QCARD. Although visually, the navigation system with form buttons looks similar, the ways the two servers are accessed are completely different. Instead of keeping a token that is sent via POST, the Fee Account pages actually remember a user with a cookie, much like the PHP session implementation. However, in my experiments, the Fee server did not act as expected. There is a Logout button, but it did not appear to do anything useful. Clearing my cache and cookies did not appear to log me out either. For this reason, I decided to forgo the proxying for this section, as I feared that the proxy server would be permanently logged in to the user fee accounts as well.

As a workaround, I implemented the Fee Account page the same way that the current QCARD implements it – by forwarding the user to the Fee Account address. In the final wrapper implementation, this is done in an IFRAME (internal frame) so that it appears to be on the same site as the wrapper. Unfortunately, clicking any link from the Fee Account page will break out of the frames and lose the nice navigational layout that our wrapper provides. On the bright side, it does not appear to have any problems with the Back button and so a user can simply press Back to return to the wrapper page. This also solves the complaint that many users have, where the Fee Account page is a dead end that requires one to log off and log back on to QCARD in order to access any other page of QCARD again.

Converting Marks, Course Offerings, and Timetable

In general, the data on the QCARD pages can be found in two places: hidden form fields and HTML tables. To parse the data from these pages, I used a publicly available, open-source HTML Parser library for PHP. This HTML Parser provides an iterator-style interface to access each element in an HTML page (Solorzano, 2003). Using the library as a base, I wrote functions `getFormInputs()` which returns a map of all the form fields on a page, and `getTables()` which returns a list of all the tables, each as a two-dimensional array of strings. Once these functions were written, it was fairly easy to get the data I needed. The Marks page had data in the form fields. The Timetable and Course Offerings pages each use multiple tables for layout purposes (instead of CSS), and so one must find the relevant tables by size. It was only a matter of coding from there.

Converting Add/Drop Page

Unfortunately, this page was never implemented in the wrapper for technical reasons. The functionality of the add/drop page could not be implemented (as will be explained in the Challenges section below) and so I decided that I would not convert the look of the Add/Drop section to the design version created in Part I. Although I could have done the conversion to create a non-functional version, I felt that this was not worth the time.

Challenges

One problem that arose is that many of the QCARD pages in the Add/Drop section have extra confirmation pages whenever a change in course is submitted. When I

was developing this stage of the wrapper, I was already in the middle of the Winter 2006 term and could not use the Add/Drop feature of QCARD at that time. As such I did not know what output to expect from QCARD when such a procedure was done. Since my directed graph of `Menu` (page) objects was completely dependent on my manual discovery of links, this lack of information was a major barrier. As it stands, my PHP code has the partial information needed to access the Add Course and View/Change Course pages, but attempting to submit course changes would have an unknown result.

The availability of QCARD also posed a problem. The system often goes offline for maintenance after 8:00pm on weekdays and is not always available on weekends. The frequent downtime put a damper on my progress at times, and was a source of frustration both for myself as well as many of the students that submitted feedback about QCARD.

I would have to say the strangest challenge I encountered was trying to figure out the Fee Account server, which I described in Part III. Its persistent login behaviour with a non-functional logout button was so perplexing that I simply gave up and provided a direct link to the page. Thankfully, the design of those pages was already acceptable and I had not planned to redesign them anyway.

Future Work

The timetable redesign only works assuming there are no conflicts in the user's timetable. If there is a conflict, currently only one of the courses is shown, with the other hidden. In the future this could perhaps be improved by creating a tooltip or layer that would display all conflicting courses when the mouse cursor moves over it. Also, there is currently no CSS rule for unscheduled classes (common with correspondence courses).

Additionally the Marks page, while redesigned here, could be redesigned further into a tabular format, with rows and columns.

Overall, the QCARD wrapper that was developed only managed to implement five of the six intended pages. The most obvious addition to this project would be to work on the wrapper during a time when the Add/Drop section is available and enable that functionality. Also, there are about 10 more pages on QCARD that were not targeted in this project. For sheer completeness, these could probably be implemented into the wrapper. Finally, the list of features of the ideal QCARD interface that were not already addressed is a very good list of goals to aim for.

As mentioned before, the wrapper technique was a good way of quickly implementing an improved and working interface for QCARD, however it is not the best way to replace it. Indeed the current wrapper is necessarily slower because it is working on top of the existing system. The best way would be to directly rewrite the current front-end of QCARD as well as change the peculiarities of the navigation system, which would also involve much rewriting of the back-end code. If QCARD is eventually updated, we hope that they will incorporate some of the designs (especially the timetable) and ideals that we have set out.

Conclusion

This project ended up being much larger than I had expected, but it was very rewarding to have a final product that worked. Fortunately, I was able to use programming languages that I was already familiar with in PHP, HTML, CSS, and

JavaScript. I did however learn many new things about the intricacies of HTTP headers, proxies, and user sessions.

While only five of the six intended sections were implemented, the implemented pages are all fully functional. The final version does in fact allow any student with a QCARD account to login and obtain readable timetables and tax information from any year, plus current course information, marks, and fee information. It fixes all the navigation anomalies that we decided to address. Printer-friendly pages were also created with a simple CSS stylesheet. The only goals that were not accomplished were the ones relating to the Add/Drop page. In the end, I was very happy with the finished product and it also appeared to satisfy many of the students that originally gave their input on improvements to QCARD.

A working version has been uploaded to the webserver space graciously provided by my supervisor, at http://txl.cs.queensu.ca/~choi. It is also mirrored on my personal webspace at http://zalbee.intricus.net/qcard.

References

Çelik, T. (2003). *Box Model Hack*. Retrieved Apr. 13, 2006 from: http://tantek.com/CSS/Examples/boxmodelhack.html

Queen's University (2006, March 31). *QCARD*. Retrieved Apr. 13, 2006 from: http://qcard.queensu.ca/

Silver, L. (2001 March). *CSS Enhancements in Internet Explorer 6*. MSDN. Retrieved Apr. 13, 2006 from: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnie60/html/cssenhancements.asp

Solorzano, J. (2003). *HTML Parser for PHP-4*. Retrieved Apr. 13, 2006 from: http://php-html.sourceforge.net/

Stricek, A. (2002, January 10). *A Reverse Proxy Is A Proxy By Any Other Name*. Retrieved Apr. 13, 2006 from: http://www.sans.org/rr/whitepapers/webservers/302.php

Wikipedia (2006, January 9). *Separation of Style and Content*. Retrieved Apr. 13, 2006 from: http://en.wikipedia.org/wiki/Separation_of_style_and_content

Appendix A

HTML and CSS code for

Part I: The Design

Appendix B

JavaScript code to Generate the Timetable CSS Rules

Appendix C

PHP Code for the Final

QCARD Wrapper