

This paper was presented at CUMREC '97, The College and University Information Services Conference. It is the intellectual property of the author(s). Permission to print out copies of this paper is granted provided that the copies are not made or distributed for commercial advantage and that the title and authors of the paper appear on the copies. To copy or disseminate otherwise, or to republish in any form, print or electronic, requires written permission from the authors.

# Creating a Web-based Convenience Store: One Stop Shopping

David S. Alderson  
Applications Manager  
Academic Data Systems  
Univ. of Maryland, College Park  
Student Population: 33,000  
3101 Mitchell Building  
College Park, MD 20742  
Phone: (301) 405-1728  
Fax: (301) 314-9568  
E-Mail: [dalderso@deans.umd.edu](mailto:dalderso@deans.umd.edu)

Mary Ann Granger  
Associate Director  
Records & Registration  
Univ. of Maryland, College Park  
Student Population: 33,000  
1130 Mitchell Building  
College Park, MD 20742  
Phone: (301) 314-8218  
Fax: (301) 314-9568  
E-Mail: [mgranger@deans.umd.edu](mailto:mgranger@deans.umd.edu)

The University of Maryland at College Park had been looking for an efficient and convenient way of placing services directly in the hands of students. The Client/Server model was thoroughly researched and given a wait-and-see approach until several issues were resolved. In the meantime, the World Wide Web grew in popularity on campus, and within a few months made itself the obvious choice in creating an electronic one-stop-shop. Its platform independence, freedom from application distribution, ease of programming, and a near ubiquitous presence in the university environment has made development and implementation virtually worry-free. As a result, universally accessible applications such as the Schedule of Classes and a schedule maker (Venus), secure downloaded systems such as Grade Inquiry and Student Account Inquiry and Payment, and real-time applications such as Drop/Add and Schedule Adjustment are all becoming available on the Web and getting delivered into the hands of end-users.

# Creating a Web-based Convenience Store: One Stop Shopping

## In the Beginning

Academic Data Systems (ADS) and Records & Registration at the University of Maryland College Park brought themselves into the World Wide Web like many other university departments - with a little curiosity and creativity. While experimentation with new technology and no deadlines make our jobs fun, anyone who works for a state university knows they would be laughed at if they requisitioned equipment for experimental purposes in the student information area. Before we had even started, we faced our first problem to conquer: was there any "free" equipment available?

It is amazing what one can find rummaging around in a storage closet. A few years back the university decided to change the platform for its e-mail system from VAX's running Ultrix to a Sun Solaris cluster. Over 20 department VAX's had been replaced with four Suns and relegated to storage. Once we found the VAX's, we had found our server. The home of our initial Web site was built from a 1983 Dec VAX Station II, with NCSA's Web server software. With a little text editing and some very basic graphics design, ADS had created a home page. Two student service units at the University of Maryland now had a machine known as the ADS/R&R server that would take them into the world of the Web.

The first application, developed in early 1995, was a survey for the *Maryland Automated Registration System (MARS)*, our voice response and telephone registration system. This was truly an experimental time for us. The Web was new to the campus and everyone was talking about the wasted hours students were spending "surfing the Web." We worried it was only a fad, but within two minutes of bringing up our first application someone had actually taken the survey. The power of the Web hit us full force. We began planning for the future, focusing on what students could be offered in this new and exciting environment.

We decided that the Schedule of Classes would be a true test in determining the functionality of the Web. The application would use downloaded data in read-only format, making it very easy to write.

## The First Major Project

As usual, easy is a relative term, especially when users become involved. It was decided that the Schedule of Classes on the Web had to be more functional than the paper version. Not only should the schedule give students information about when and where a course was being offered, but provide course enrollment status as well. If the course is closed, does it have a waitlist and how long is it? Now a simple downloaded, read-only program to build the Schedule of Classes on the Web required two phases. Phase one involved modifying the Schedule of Classes program so

that it could optionally produce a file with the necessary html tags inside. Phase two consisted of capturing seat counts that would be as up-to-date as possible. There were the normal groans from the programmers; however, as we reviewed programs currently in production, we discussed the feasibility of using a telnet-based application called *Seats*, which had been written in the 1980s. *Seats* allows students to find enrollment information on any published section. The data supporting this application comes from a program that runs nightly on the mainframe. This file contains all course and section information, including course descriptions, instructor, and building/room information. Recently, *Seats* had been enhanced so that hourly, from 8:30am to 10:30pm, another job running on the mainframe downloads only enrollment information. This smaller file is merged with the large nightly download file, providing up-to-the-hour enrollment data. It was a simple matter to merge the enrollment counts file with the Schedule of Classes data and format it so it could be displayed on the Web.

### **Sometimes Students Show Us the Way**

With the first project complete, we knew there was another application just waiting to be written. Over the last several years, a number of students have come in “off the street” with diskettes in hand, asking if we could distribute course schedule-making software that they had developed. While we always thought this would be a valuable application to students, the programs typically developed by ADS and Records and Registrations supported administrators. Additionally, there would be some difficult issues to address -- how could such software be distributed throughout campus and be kept up to date? Not surprisingly, this project never made it to the top of our priority list.

As fortune would have it, ADS had hired a student (Phil Su) who had some of the same ideas as his peers about course scheduling, as well as a love for the Web. He convinced us that the Web was the only way to go if we were going to write applications for student use. Phil was given free reign of the system and in the summer of 1995 commenced writing his scheduler program. By the end of the summer this student scheduler application was launched – Phil’s product, called *VENUS* (Virtual Electronic Network University Scheduler), surpassed our every expectation. This course scheduler uses html forms, allowing users to enter several parameters: semester, required courses, supplementary courses, minimum and maximum credits, waitlist limits, and maximum returns. After entering the desired parameters, every permutation of non-conflicting courses is displayed graphically on a colored grid.

To add more functionality to the system a wildcard option, termed *The Joker*, and a general education requirement wildcard, termed *CoreBoy* were developed. *The Joker* can be used to view offerings at specific levels. For example, BMGT1\* will take you to a Schedule of Classes-like page listing all 100-level BMGT courses. The user can then click on a check box to select the specific course or section to be included in their *VENUS* schedule.

The university’s general education requirements are called CORE and the courses which fulfill this

requirement are often difficult for students to find. Using *CoreBoy*, students can enter the word 'core' in either the required or supplementary course entry fields. *VENUS* will then display a page from which CORE categories such as Humanities and the Arts, Math and Life Sciences, Social Sciences and History can be selected. After the user chooses the desired category, *VENUS* can then display the available courses in a Schedule of Classes-like format. By clicking on the box associated with the desired course and/or section, the specific CORE request is added to the scheduler. Once all *CoreBoy* and *Joker* wildcards are resolved into courses and/or sections, the user presses a button and *VENUS* starts building a schedule.

*VENUS* was designed to be used by both the novice and the advanced user. Advanced users have an additional option to block times during which they do not want a course scheduled (work hours, or even more beneficially, Friday morning sleep-in time). The popularity of *VENUS* was apparent from the start -- students were busy on the Web accessing this new application.

Because Phil Su is a Computer Science major, he used C++ to write *VENUS* (yes, the latter does follow the former; it is a Natural Law here at UMCP). One of the greatest benefits in using C++ is its object orientation. Several objects were used repeatedly within *VENUS* and later in other Web programs written by different programmers. As more programs are written, the benefits of C++ have become increasingly evident. A discussion later in this paper will cover C++ and Perl within the context of which language is best for the general programmer pool (i.e. from beginners to COBOL programmers to C programmers).

## **Enlightenment**

Statistics showed that over 1,500 students accessed the Schedule of Classes and *VENUS* applications per week during each application's first weeks on the Web. As *VENUS* continued to grow in popularity it became obvious to our offices that the Web should be the focus of student outreach efforts.

It was time for the developer who had brought up the ADS/R&R Web server to get his feet wet with a real application. The big decision was what language to use. Expediency won out and C, the language he was most comfortable with, was chosen. This decision had its drawbacks because none of the objects in *Venus* could be used. Additionally, the applications could not be moved, in good conscience, to another platform without converting them to C++. Nevertheless, our experiences have shown us that it is almost always better to compromise on the side of time and just get the applications out. Some of them might need to be revisited, but gaining the experience and the exposure is critical.

Transfer Course Lookup was selected as the next project. As with the Schedule of Classes, data that was already being downloaded for another application was available. The data first had to be transferred from another machine, then sorted in a different manner. A program to manipulate the forms and present them on the screen was written. After only two days, the application was up and running. It became apparent that not only were students heavily using these applications, but

that bringing them up was much easier than anyone expected.

### **Growth Costs**

Up until this point the applications we had developed for the Web were “free.” Free from the standpoint that we had incurred no hardware or server software costs. One student and two full-time staff members had brought up the Schedule of Classes, *VENUS* and Transfer Course Lookup in a short time frame. For the full-timers, these applications were written while they still performed all their other job duties.

Because our Web applications were becoming increasingly popular, there were periods of time when the VAX would become distractingly slow and take minutes to return results. As anyone who uses the Web knows, speed can make or break an application. Slow responses become an issue that needed to be addressed quickly. To add to our response concerns, there was also talk about introducing *VENUS* to the entering class of students during summer orientation. We were faced with the possibility of 300-500 students hitting *VENUS* throughout the day, with 30-60 of these being simultaneous hits. Obviously, our free Web server would need an upgrade soon. Anxiety set in; there weren't any closets storing powerful servers that we knew about.

Once again, creative problem solving came to the rescue. *MARS*, our voice response system, had recently been converted from a 486 Interactive Unix box to a Sun SPARC/20. A load test was run during peak registration times, and it was found that at most 17% of the system was being used. Obviously, there was an abundance of free space to support a Web server! The only question left was whether the Web would overpower *MARS*. Since there was no good way to answer that question, it was decided to put the Web server onto the *MARS* machine, run both, and see what happened.

### **Yet Another Bonus**

Another bonus in moving the Web to the SPARC/20 was the fact that there were several applications on *MARS* using nightly downloaded files. These data files could easily be utilized for Web applications that included Grade Inquiry, Waitlist Check-In, Student Account Inquiry, Graduation Application, Graduate School Application Status, Re-enrollment Status, Financial Aid Inquiry, and View Parking Tickets.

Our Web site would no longer just furnish general information to students. As we increased services, we would now be displaying sensitive data. Students would be required to enter their Student Identification Number (SID) and PIN to access their records. Since security was clearly becoming an issue we decided our server needed more protection. What we required was encrypted transmission over the network between the client and server. It was this reason, combined with the fact that Netscape gives its servers to educational institutions at no cost, that we chose to use Netscape's Commerce Server.

The issue of software security could be more difficult to overcome. In order to develop the applications quickly, and because these applications were only running one transaction, from retrieving the SID and PIN to showing the data, a simplistic approach was taken. Several security points apply for each application:

1. No identifiable data is shown on the screen. Information like SSN, Student ID, and Name are not shown on any resulting page.
2. The page on which the user enters their SID and PIN are not cached locally. As far as we know, this can only be controlled with *Netscape Navigator*. Other browsers ignore this "no cache" directive, although Microsoft's *Internet Explorer* seems to clear input fields when the back-arrow is used. The public access labs at the university also use Netscape, thus we feel somewhat secure. The assumption that individuals accessing our Web site from off campus would be using their own, non-public access equipment was also made.
3. The SID and PIN page, as well as all sensitive resulting pages contain the "refresh" directive. With this directive a timeout interval and a destination URL are specified. After the timeout interval, the browser will automatically jump to the destination URL. This eliminates the possibility of sensitive information remaining if the student leaves the workstation without clearing the screen.
4. "Security through Obscurity." This concept is truly feasible only for applications using just one transaction to display their data (i.e. after the SID and PIN is entered, the data is shown on the next page; there are no subsequent pages the user can access in order to view more information -- therefore only one transaction is processed per SID/PIN verification). This is accomplished by having one entry point for all applications: the SID/PIN entry page. The static page that calls the SID/PIN entry page passes information along that uniquely identifies the application. For example, the first page a student accesses to view their grades has a hidden input field with the value "grades." This field is passed to the SID/PIN entry page. Once the SID/PIN pair is verified, the SID/PIN program translates the "grades" field into an obscure program name which gets the data and produces the resulting page. The security in these applications is achieved by obscuring the name of the application which obtains the information on the student.

Tight security is a necessity even in simple, single transaction applications. Indisputably, a more sophisticated approach to security will be needed as we make plans to provide more complex applications. ADS is currently in the process of beta-testing a stronger security application now in the production system.

This program, called a daemon, will run continuously on the Sun SPARC/20 and communicate through a network port. Because of its properties, it can communicate with any machine that's

on the internet. It can also refuse a connection from any machine by looking at that machine's IP address.

The program is basically a state security daemon. Because Web browsers are "stateless," and keep no information from one page (or transaction) to the next, an application such as a Drop/Add requires another program to tell it where it is in its processing and what data it was last using. This is the "state" part of the daemon. The secure part of the daemon is in an application's ability to access this state information; in other words, an application's ability to authenticate itself to the daemon.

There is one entry point in the application system for authentication: the SID/PIN entry screen. It is the only application (by virtue of the fact that it is the only one with the correct password) that can initialize state information for a given client. When that client correctly enters the SID and PIN, the SID/PIN entry screen sends a request to the state daemon to set up a state file on the server. If the password transferred by the SID/PIN entry screen is correct, the state daemon sends back a key that will be used by every subsequent application page. This key, along with unique information sent by the browser, will identify itself as the authenticated client. The state daemon also puts an expiration time stamp on the state file (this is currently set at five minutes). If a request to set state information or to retrieve prior state information is passed to the daemon after the state file has expired, the daemon will send back a return value to indicate a need to re-authenticate. The SID/PIN entry screen is called again, this time with a parameter that indicates a need to "re-expire" the current state file (i.e. use the same state file so that information is not lost, but give the file five more minutes before it again expires), and another parameter to indicate the program that called the SID/PIN entry screen. Once the client is re-authenticated, the SID/PIN entry screen will call the program and the application can proceed.

### **Languages and Standards - A Little Philosophy**

Once the security issue was settled, there were several applications that could be written by accessing data already downloaded from the mainframe. Again, there was the question of what programming language should be adopted as the standard. The programmer's preference was to chose C again since C++ was still a foreign language. However, Phil Su had written several objects for Venus in C++ that could be used in these applications. Also, it became obvious that using C++ was going to make the job easier because all the new applications were very similar in nature. The necessity of learning C++ was clear. Once learned, our programmer considered it "the best thing since apple pie", and praises of the language were sung as applications were pumped out at the rate of about one every two or three days.

As additional applications were being added to our site, we began to discuss the development of Web standards. The original "standards" that were constructed for the Schedule of Classes and Transfer Course Lookup were almost of an *ad hoc* nature, as well as very simplistic. *VENUS* had its own standard, what we refer to as the "student standard." For example, if you try accessing the old URL for *VENUS* (<http://www.ads-rr.umd.edu/V/V.html>), you get this reply:

## **Could it be?**

Yes, it's **true!** Venus has found a new, faster home, at:

<http://www.venus.umd.edu/V/V.html>

It's faster. It's better. It's low-calorie, zero-fat.

Though this standard is obviously very loose, we found that the students using *VENUS* appreciated the “student standard.” Additionally, Phil Su asked the student community for suggestions and feedback on *VENUS*. Students can be truly creative and innovative, and those students’ whose ideas were implemented were acknowledged on the next release of *VENUS*. Not only was the product improved by using others’ input, but it was well accepted by the student population because they had ownership in the application.

Standards used in the “secure applications” requiring an SID/PIN are more formal in nature and were written by a developer with over ten years at the university. It was decided that the formal standard was appropriate for the presentation of the more serious, sensitive data.

These two different standards are now used throughout our Web site. If you travel to <http://www.testudo.umd.edu>, you will see that the secure applications all look similar. This was done to establish a comfort level with the users. When users become familiar with an application, they find it easy to use. As you travel through our Web site, you encounter many different styles, some good and some with room for improvement. We feel experimentation is the key to finding the best way to present a particular Web application. We found that it is crucial to make changes to pages so that the clientele know that you are paying attention to them and trying to always improve your product.

## **Surfing the Campus**

As we continued to add applications to our site, we went “surfing” and found that other administrative service units were also quickly developing Web applications. Our Undergraduate Admissions office, working with another analyst in ADS, had developed an Undergraduate Application form for the Web. At their site, students can apply to UMCP and pay the fee with a credit card. The data is inputted by the applicant and then sent to a staging area on the mainframe, reviewed by a clerk in the morning, and then added to the database.

The Administrative Computer Center (ACC) houses the IBM 3090 mainframe and is responsible for financial, as well as other student and administrative, applications. One of ACC’s first Web applications was the “Off-Campus Housing Search.” An individual can fill out a profile with information such as student status (graduate/undergraduate), gender, price range, etc. A housing search based on this criteria is processed, and the user is presented with the results. Some of the

valuable features in this application include showing transportation options from a specific location to campus, contact information for renting the property, and the complaint status on the owners.

It was apparent that, in some way, almost everyone on campus was experimenting with the power of the Web. However, the campus wasn't working in a unified way. Though there were many applications, none could be accessed from one coherent menu. UMCP seemed to be developing its own bureaucracy on the Web!

### **One-Stop Shopping?**

During 1995, the campus was involved in several Continuous Quality Improvement (CQI) efforts. As a result of one of these initiatives, student service offices were asked to plan for the implementation of a "one stop shop" to allow students to take care of most of their administrative tasks in one area.

In response to this initiative, a Student Affairs Reengineering Committee was formed. The group began to look at how different schools throughout the country were addressing this concept. A team from the committee visited the University of Delaware in the Spring of 1995 to view their programs. At Delaware, all administrative services are grouped in the same building. When the Maryland team returned to campus, there was both excitement and trepidation within the group. Delaware had successfully located all administrative student services under one roof by sending staff from their main offices to an auxiliary area. As the Delaware approach was discussed, there was apprehension on the part of the Maryland team. UMCP had experienced recent budget cuts and layoffs. Losing more personnel to an auxiliary area could severely hamper existing operations. The team continued to struggle with meeting the one-stop shopping directive.

In the meantime, Maryland's Web development activities were picking up speed, and the next trip to Delaware was fueled by the exciting things they were also doing on the Web. During that trip an idea began to take shape. With appropriate coordination, why couldn't the "one stop shop" idea be incorporated into a Web site? While this might appear to be an easy task, many of the units which support student services report to different vice presidents. Working together to develop a unified product could become a real challenge.

### **Testudo**

The Records and Registrations office called a meeting of all the technical staff that support student service offices. Surprisingly, after just one meeting, our virtual one-stop-shopping area became a reality, named "Testudo" after UMCP's mascot. Students could now find all administrative services listed under one address. No longer would they have to bookmark a number of addresses on their Web browser. With a click of a button, an extensive menu of applications was at the fingertips of the student. Students no longer have to go from one end of the campus to the other for services, and they don't even have to step on campus except to go to

class.

As presentations about Testudo were given around campus, the idea of this virtual one-stop-shopping area was enthusiastically accepted. There was such interest that the Student Affairs reengineering group put together a proposal consisting of dozens of applications to be written in a very short time frame. While the group could not meet such aggressive expectations, the planning was a good example of how the Web was being accepted as an outreach tool for students and alumni.

The group is now concentrating its efforts on developing applications for the Fall 1997 freshmen class. These programs will take freshmen from the point of application all the way through acceptance. Students should soon be able to RSVP to special open houses hosted by the university, confirm enrollment and pay the deposit, request disability support services, change their major, change their degree intent, change orientation date, and initiate housing and dining services contracts.

### **The More the Merrier**

As exposure continues and the popularity of *Testudo* grows, different offices have become eager to add their applications. Typically, these requests would be reviewed by a committee representing all aspects of the university. So far we have successfully avoided having to use this bureaucratic management style. When a department wants to add an application to *Testudo* they write directly to the Webmaster. This request is then discussed among a few individuals who have experience in the technical and service areas. The primary focus of *Testudo* is to be as functional as possible for students and alumni, but at the same time not overwhelm users with information about the university. If the requested application is a service to an individual it probably belongs on *Testudo*. If the application is informational in nature it belongs on the University's Web pages, called *InforM*. To date this has worked out well, and no one has challenged the decisions that have been made. However, we know that eventually a larger committee will have to be formed to take over this role.

*Testudo's* home is an NT server housed in the Administrative Computer Center. Applications created by all the service offices are displayed on a unified menu, but the applications themselves reside on the office's respective servers. This was a drastic change to normal operating procedures. Since 1988, IBM has been the one constant throughout administrative computing. Even with the advent of distributed systems, loyalty to "Big Blue" has continued. Only recently is the idea of open platforms and open systems beginning to be accepted on campus. *Testudo* proves this well, gathering applications on such varied systems as AIX, NT, and Solaris.

It is not only platforms that have broken the mold, but standards and languages as well. It would be impossible, as well as disadvantageous, to require a strict group of standards on the Web. Because of *Testudo's* distributed nature, offices need flexibility so that various types of applications can be accepted. On the other hand, there is a very basic set of standards that

everyone has agreed to: make it obvious that the application came from UMCP, and keep it “G” rated.

An area in which *Testudo* diverges from normal procedures is the multitude of languages used in writing applications. While 98% of all applications written on our mainframe are in IDEAL, there are many languages in use on the Web. On Unix boxes, the languages of choice are C, C++, and Perl. On NT boxes, the language of choice is Visual Basic. Obviously, this is due in large part to the *ad hoc* nature of programming on the Web; the idea of “get it out there now” overrides “one for all and all for one.” This will present a problem when it comes to maintenance, making a small set of individuals valuable because of their knowledge of several languages. It also tends to exclude those who are not willing to learn complex languages like C and C++. Thus there are large groups of programmers in the IT departments that are being left behind. The question becomes, as on the mainframe, what language is best to use? Is it Perl, the basics of which can be easily learned by beginners and yet complex enough to satisfy experts? But then should Perl be used on NT, which readily supports such languages as Visual Basic (or Visual anything...some of the languages coming out of Borland and Symantec are also very powerful for novice and expert alike)? We are in the middle of this dilemma at UMCP, but it is obvious that a decision needs to be made soon.

### **Troublespots, or *Challenges***

We have been happy with *MARS*, our voice response system, and the student’s acceptance of it. However, we have been frustrated at times because working in a non-visual arena is limited and slow. We have always wanted a remote-site system for the students where they would be able to see the screens and the messages. In particular, our main interest was to bring up Drop/Add and Schedule Adjustment applications. Then, as a part of Drop/Add, students would be offered several other features such as Schedule Printout (a grid showing a student’s course schedule), Open Section Lookup, Registration Blocks Lookup, etc. As *VENUS* gained popularity this desire was becoming more of a priority. Students and advisors were looking forward to the time when they could actually process the registration schedule built using *VENUS*.

During this same time period, the *Mandarin* product was being presented throughout the country. While we were interested in the opportunities for client/server systems that the product offered, we found that we were either already using the Web to provide those services or had them in development. Therefore, we decided to continue our development of client/server systems through this route.

Even with all the benefits of the Web, we still faced some problems. No matter what platform we chose, UMCP had somewhat of a unique obstacle in establishing a live connection to the legacy systems. When we converted from an HP/3000 to an IBM 3090, the fourth generation language, Computer Associate’s IDEAL, was chosen. It was a perfect solution at the time, allowing for a robust integrated Student Information System and a short conversion period. However, with the current technology and distributed method of computing, we are finding that IDEAL comes up a

little short.

One major reason is that IDEAL does not work with outside systems. While it can call languages such as COBOL and C, no other language can call IDEAL. It is an environment unto itself. Therefore, all the legacy programs written for our Student Information System are inaccessible from the outside.

One of the solutions we investigated was to build or buy a *screen scraper*. This is a commonly used method in which a distributed system only deals with the screens of legacy systems, exactly as an end-user does, in order to input and retrieve information. Our voice response system, MARS, uses screen scraping in its real-time applications. Limits are set by restricting the maximum number of simultaneous users to 96. However, screen scraping uses a significant amount of resources. These resources and the cost for new resources must not be ignored when building new systems.

The number of incoming requests to the mainframe from MARS is easily limited by the number of telephone lines. Because telephones are nearly ubiquitous, a busy signal, however frustrating, is easily understood. Busy means “try again.” On the other hand, the Web is comparatively new. While we have all experienced forms on the Web that never come back after they are submitted, it is quite a bit different than a busy signal on a telephone. Even a “system is busy, please wait” on the Web creates quite a bit more frustration than a busy signal. PC users expect a response in a reasonable amount of time.

The point of this comparison is to emphasize the importance of response when implementing a Web system. During heavy periods, the Web could easily be used by several hundred people simultaneously. There needs to be a way to handle the load without failure. If a student experiences a failed attempt time and again, distrust of the system could develop, and a “customer” would be lost. A student’s courses are, to them, quite a bit like a paycheck to us. If we won’t risk passing our money along a flaky Web application (or even a well-behaved one!), then we must not expect students to do the same. A solid foundation must be built for the Web system.

Using this rationale, it was decided to convert applications to be used on the Web to COBOL. Calling these COBOL programs from the Web would avoid using CICS, which is one of the main drains on CPUs. Since IDEAL can call COBOL, the existing programs could also use the converted subprograms. IDEAL would be concerned with screen presentation on the mainframe and COBOL would do all the data manipulation. The Web can use the COBOL programs the same way that IDEAL will.

To accomplish this, a method for calling stored procedures on the mainframe was needed. Fortunately, UMCP had just finished obtaining a site license for Oracle. The Oracle Gateway for MVS should work well for this purpose. Once it is installed, we will be running real-time applications on the Web.

Part of the excitement in writing applications on the Web is being on the leading edge. But being there presents its own challenges. Studies of dorm and off-campus use show that 40% of connections to campus are gopher-based, which means that a large part of the population that can only reach the Web in text mode. Writing applications that keep this in mind can be a hindrance to creativity, or at the very least almost double the effort to provide both text-based and graphical-based pages and applications. It doesn't stop there. To create applications that have the client/server look require writing in Java or ActiveX. At this point in time, doing so would severely limit the audience. However, looking back on how quickly the Web in all its various forms and versions came out, the idea of writing with the future in mind seems to be the only way to go. After all, the distant future is at most six to eight months away.

## **Conclusion**

Very few people could have realistically expected the Web to become as powerful a tool as it is today. *VENUS* showed us that the Web provided the answer to the troubling questions of client/server. Experimentation was the key to our success. Writing applications on a free VAX required no management decisions, but proved to management that Web applications were well-received and well-used by the student population. Never in the past had we relied so heavily on student programmers. Their new way of thinking and creativity taught a few old dogs some new tricks.

As with any new project, especially with those that use new technology, there are always hurdles and we are still struggling with some of these. Overcoming them is part of what makes any project challenging and interesting. We have already met one of these challenges with much success. No one ever dreamed that cross-sectional units could set their priorities in a coherent fashion to create a product that provided a multitude of services for students. The excitement from this has made *Testudo*, the one-stop shop for students, one of the most satisfying projects we have yet encountered at the University of Maryland College Park.