Tiger by the Tail

Art Brooks,
Director
Administrative Data Processing

University of Missouri - Rolla

Institution information:

The University of Missouri - Rolla, founded in 1871 as the Missouri School of Mines and Metallurgy, is one of four campuses which constitute the University of Missouri system. While throughout its history, engineering and science has been the focal point of its educational efforts, the campus also grants bachelor level degrees in several liberal arts areas. UMR offers bachelor, master and doctoral degree in twenty-eight disciplines in three colleges and schools; the College of Arts & Sciences, the School of Mines & Metallurgy and the School of Engineering. Although its student population has consistently hovered around 5,000 in population, the campus has a history of ranking among the nation's leaders in bachelor degrees granted in engineering and its graduates can be found employed around the world.

Paper abstract:

The University of Missouri - Rolla is in its eleventh year of maintaining a local data warehouse in a relational database. That foundation put UMR in a position to make the transition from a traditional legacy structure to a distributive environment for presenting information to the campus community. Taking advantage of that foundation, in 1996 UMR embarked on a project of delivering an entire admissions system through the World Wide Web. This paper will provide a background of the data warehouse which made that endeavor possible. Focusing on the development of an admissions system on the Web, the thoughts, arguments and difficulties which an effort of this type encountered will be presented. It will be demonstrated in creating a Web based system, traditional system development approaches and techniques must be replaced with an entirely different set of perspectives and approaches.

Providing information in an electronic form to the campus community is a prime responsibility of today's computing operation. However, it has not always been viewed in that perspective. Legacy systems were built for the sole purpose of storing and manipulating data. When these systems were created, the techniques for distributing data were not existent nor was there an interest from the constituent public. The advent of the PC in 1981 and the change in the perspective of the youth altered all of that. Suddenly, computers were expected to do more than just crunch data and print something out. People wanted computers to be useful tools for their own imaginative ventures without the interference of some over-protective programmer. What was worse, they actually wanted to be able to extract data from systems that had nearly nothing in common in their design.

One must excuse these people as they did not know how unrealistic such expectations were. After all they were expecting the data mongers to open up their systems to data inspection and on-demand reporting. In addition, to them
computers are magic boxes which could be manipulated at will by the computer programmers. In their view, all the magician had to do was perform the magic a little different to get their desired results. Having seen the results, the obstacle in their minds was merely the stubbornness and protectionism of both the data mongers and the programmers.

In reality the barriers in addressing this growing perspective took two forms. First of all the legacy systems database platforms were never designed for such activities. Such traditional systems were generally too fragmented in their design and had firmly built boundaries which forced the data to stay in their silos of safe haven. Secondly, since each system was designed independently, their very structures prevented cross system integration. There did not and does not exist a method by which legacy system platforms can effectively interact in their native environment. This has the same effect as a collection of amateur musicians trying to play music in harmony without the advantage of either sheet music or a conductor to lead them.

Not only were these inherent barriers existent, but the database custodians were extremely reluctant to release their data so freely. There is no denying the fact a portion of their hesitancy is based on the fear of loss of control and a resultant impact on their job security. However, the substantial fear for the managers of data lies with their concern for the accuracy of data reported without their careful scrutiny of the finished product. They have very legitimate reasons to be concerned with the misinterpretation or misuse of the data when reports are produced by individuals not intimately familiar with the data.

Taking the shape of something like a budding tsunami, the ground swell from the user community by the beginning of the nineties required a total change in perspective on the part of the data managers and the computer programmers. Today's college students would no longer tolerate their data being tightly locked in data silos and the demand for reporting from every administrator required a change in direction and perspective. There was no doubt it was only a matter of time before the silos would be opened up like tin cans by an enraged giant. The problem was how to provide the data in an accessible and managed form while providing data security and assuring accuracy.

When that new attitude infiltrated the campus, the University of Missouri-Rolla was in a position to address the emerging perspective. In 1986 UMR had ventured into the relational database realm purely out of the need to survive. That need was created by a lack of funds to support the burgeoning demands for reports; funds needed to pay for both computer time and computer programmer salaries. By the beginning of the nineties, UMR had several years of downloading data from legacy core systems to a local relational data warehouse. Several of the database custodians had been trained in report writing from this warehouse and selected campus users had been trained to run or create reports from that data. However, that change in perspective was far from being adequate. The students did not have access to the protected data and basic information was still not generally available.
With the introduction of the Gopher facility from the University of Minnesota, UMR found a way to disseminate information to the general user community from its data warehouse. Introduced in late 1992, STUINFO and UMRINFO opened the doors for access to its data repository. Fundamentally, UMRINFO was more of an electronic bulletin board or resource than it was anything else. It was from this source campus users could have access to the Registrar's reporting schedule, campus schedule of events and campus job opportunities. Additionally, the course catalog and schedule of classes were added to the options. Eventually, the Registrar's course equivalency table for evaluating transfer courses was added to the selections.

In its initial form, STUINFO only provided the students access to their data stored in the Registrar's system. They could see their address, biographic, course schedule and GPA information. Transcript data was not included for fear the availability of the information could make it too easy for devious students to make and falsify their own transcripts. There were certainly good reasons to have some concern for this potential, although falsified transcripts are not a new phenomenon. In this instance, the Registrar did not want to be a party to allowing the individuals to misrepresent themselves.

Since some of the information provided was confidential, the first concern for providing access to data was security. There was no questioning the requirement of a distributive system to prevent some deceitful students from accessing data of other students. In addition, since the predominant form of accessing the data was from student computer labs, there was a concern students could see privileged information by looking over the shoulder of the user. What would happen if a student left the terminal without exiting the routine? All of this had to be considered and resolved prior to presenting the services.

To address the matter of verifying the authenticity of the user, the students were required to enter their student number followed by their pin number. This latter data item had just been added to the student system as a requirement for the utilization of the voice response system. Since that data item was being downloaded for the local data warehouse, the double verification capability was readily available.

The second item of security which was of great concern was the lack of privacy when viewing personal data in a student lab. There was also reason to be concerned a student would walk away from the host session without properly exiting the session, thereby exposing their student number and other personal information. The solution UMR implemented was to refrain from displaying any identifying information after the authentication process had been completed. For that reason the student's name nor student number is ever displayed on any screen at any time. This approach would prevent a curious student from taking unacceptable liberties with the information being provided.

When the bugs were worked out of the system, this provided the service we wanted; allowing the campus to present data stored on an IBM 4380 mainframe to the student population. Of greater importance than finding a method to provide access to the data was the fact an approach to address the security issue had been identified. This approach was successful enough that we still employ this technique today. The only disadvantage we have found is the fact we are presenting downloaded data which is refreshed on a nightly basis. As a result, if a student changes his pin number, the change is not applied in the local access until the next day. This has resulted in some confusion on the part of the students from time to time.

Downloaded data for distributed processing has its advantages for reporting, but for user authentication, it clearly has its drawbacks. This shortcoming was not a factor when in 1997 the University of Missouri initiated access to the live data for the students at all four campuses. As that effort has recently expanded, portions of the local system have been
eliminated. However, those deletions have been judiciously implemented as the draw back to the university wide system is the limited access time. Due to batch processing requirements, the core systems are available for only a portion of each day whereas the local downloaded system is available twenty-four hours a day and seven days a week. Only those informational services deemed of significant interest to the students have been allowed to be duplicated on the campus.

With the basic information available to students, the next task was to provide capabilities which were beyond the student system. Focusing on the idea of converting the semesterly teacher evaluation to an electronic form, the capability of conducting electronic surveys was explored. Very quickly we found we could collect data electronically and store it in a relational table for statistical analysis. Using this technique we proved we could prevent any student from submitting more than one survey response. Utilizing the power of a database we were able to control the questions that were presented to the students. This demonstrated we had the ability to provide an electronic teacher evaluation survey and restrict the responses to only those sections in which the student was enrolled.

For as long as UMR had been conducting teacher evaluations, one of the most frequent complaints from the instructors was the belief some students would use forms intended for other classes to skew the results in a given class. This was potentially possible since the students were given blank forms that required them to enter the reference number for the class they were evaluating. In addition it had been repeatedly shown that some students would inadvertently invert numbers in the reference number. The result was the scanning of a sheet for an invalid reference number, or worse yet, the entry of a valid, but incorrect number.

There was some excitement within the ADP staff knowing we could significantly improve this process. It was recognized, however, this approach had one very significant flaw, that being the potential volume of responses. The traditional system had drawn criticism from many of the faculty for the time it took from their already tight teaching plan. This approach pressed the evaluation forms in front of the students with an implied pressure to complete it. By taking the process to a less intrusive and “forceful” method, it was known the response from the students would be significantly lower. As a result, this implementation of the Gopher feature was left as a potential for the future and not attempted until the fall semester 1996. The soundness of the technique, the control of the process and the accuracy of the responses had been proven.

When the technique was implemented for the first time, the teacher evaluation oversight committee and the staff of ADP were excited about the potential of this approach. By eliminating the paper forms, the campus could save money, disruptions in class would no longer be required and control over the student responses could be tightly controlled. The participants in this decision were devastated when the faculty voiced an overwhelming objection to this form of evaluation. While it could be proven the student responses were in the same range as the paper form, there was no denying the fact the number of responses was drastically reduced. Without attempting to find ways to improve the responses, the faculty voted decisively to terminate the electronic form in favor of the traditional paper format.

Having experienced the success with presenting student data and bulletin board form of information, ADP ventured into providing similar capabilities to the faculty and staff. Again relying on downloaded data, a system was designed to present basic personnel information to members of the campus community. Because the Chancellor wanted additional information, more socially directed information than was available from the existing personnel database, an informal committee was formed to discuss the new data needs. Using the downloaded personnel data as a basis, a system was created to record the spouse and salutations type of data which the Chancellor desired. Once in place, it was the Chancellor’s intention this system would provide his office with the ability to use the new information for various campus relations purposes.
By design, the new system would merge core system data with local unique data on a single screen. Because a portion of the information presented was considered sensitive, security both for viewing and updating was a concern. Since the faculty and staff do not have a unique identifying number nor a pin number, the routine was forced to use the individual's social security number and a personally assigned password. Unlike the student data, there was not a concern for the security of the information once it appears on the screen. Because the security was more basic, the development of this aspect of the system was much less labor intensive than the security for the student data.

Placed into production in August, 1993, it was felt this system held promise for disseminating information to the campus faculty and staff. For the first time the members of the campus could change their address in the personnel system on-line. They were also provided the ability to specify the date at which that address would be uploaded. When the date they specified arrived, the routine sent the change in a batch file to update the personnel system; a procedure which worked very well.

Unfortunately, this development effort met with very limited success. Although some members of the campus community appreciated the ability to electronically view and change some of their personal information, most individuals did not avail themselves of the service. Despite the positive reception from most of the people who used it, the security itself created one of the biggest problems in their eyes. Due to the lack of a fully functional security package, the programmer had to develop his own security procedure within the program. This meant recovery from forgotten passwords could only be performed by the programmer himself. Since most of the users utilized the service infrequently, the passwords were often forgotten. In addition, some faculty members who were not particularly computer interested, asked their secretaries to maintain the data for them. This prompted some departments to request the security concept be altered to have the secretary view and change data without having to enter each person's SSN and password. That request was never honored.

As mentioned, a driving force in creating this application was to establish a “Social Registry”; a source for information to extend social invitations and greetings. While an understandable need, no office stepped forward to accept responsibility for the system. Without a focal point for maintenance, many problems immediately arose. The data definitions were never clearly defined, nor was there a succinct explanation of how to complete the information. As a result, the information entered by the users was generally not usable. For example, the joint salutation entered by one member was “Jane Doe and Guest.” Certainly this salutation was appropriate for invitations to special occasions. However, the Chancellor also wanted this information to send Christmas cards to the faculty and staff. Clearly, that salutation looked ridiculous when used for that purpose.

When the Chancellor's office used it for the first time, there were numerous surprises, though none of them unpredictable in using data that had no identifiable office responsible. The electronic service was totally ignored by the husband of one of the most socially prominent wives. As a result, she was not on the list to be invited to the first social event of the school year. In the eyes of the Chancellor's Office, the system should have been intelligent enough to have included this individual.

While this exclusion was totally unacceptable, other anomalies were found, making the data totally unusable. For “personal salutation,” where the person's first name was expected, some individuals entered such data as “Dr. and Mrs. John Doe.” There was another individual who entered his wife's name where his name was required. Even with these problems, no office came forward to accept the responsibility for maintenance and the system fell by the wayside. It is interesting to note a year after the feature was dropped, there were still some people asking for the service to be renewed so they could change their data.

Despite this interesting experience and having gained considerable experience with this form of disseminating
information, UMR abandoned the Gopher format and embraced the “The Web,” when the World Wide Web proved to be a viable platform. All development effort which could be spared was focused on converting to the newer, more impressive platform. This move was stimulated by the announcement of the removal of the out-dated campus IBM 4381 mainframes, the source for all of the information and procedures that made the Gopher access possible for the campus.

With the announcement of the removal of the IBM mainframes, UMR placed all of its locally written systems in jeopardy, including its admissions system which resided on an IBM 4381 in an SQL/DS database. Although this announcement shook the ADP staff to the core, it was also known the IBM mainframes were so antiquated a hardware failure could abruptly terminate their availability for a lengthy time and possibly even terminate them permanently. Recognizing that frailty, the systems staff had been developing expertise with the Unix operating system several years before. The plan, therefore, was to make an operating systems shift from IBM VM to Unix.

From the application development perspective the groundwork for the platform shift had been set by an EPA requirement to better monitor the chemical inventory of the campus. After an extensive review of available software, the campus EPA committee determined the campus would have to develop its own chemical tracking system. In order to develop that system, in the fall of 1993 UMR purchased an HP 9000/755 server and an Informix database license. The results of that mandate was to provide the UMR campus administrative programming staff with an opportunity to develop the expertise to launch itself into applications development in the Unix environment.

The chemical tracking system was successfully placed into production in 1994, giving the ADP staff reason to be confident with the newly acquired skills. Buoyed by that success, it was decided to rewrite the admissions system in a Unix platform using Informix's 4GL as the screen driver; the language used to present the chemical tracking system to the users. The work on the rewrite of the admissions system began in 1995 with the idea to utilize the same techniques learned from the tracking system. Fundamentally, it was a traditional approach with 24 by 80 screen mode presentation. Unfortunately, after a year of effort with countless interruptions and the loss of the experienced programmer assigned to the project, the system had progressed only to the extent of defining the new tables.

With the arrival of the spring, 1996, the issues which faced the office of Administrative Data Processing were:

1. The existing mainframe admissions system documentation was grossly inadequate and very much out of date.
2. The individual who had written the tracking system, the existing admissions system and had exerted all of the work on the new admissions system had resigned months earlier.
3. There was no member of the ADP staff who had even a vague knowledge of the current system nor the planned new system.
4. No one in the Admissions Office had an overall knowledge of the current system.
5. The Admissions Office staff could only identify what “buttons” they pushed and not what went on behind the scenes.
6. No ADP staff member had any familiarity with the Informix 4GL language; the language the Chemical Tracking system used which was in production. The new admissions system programming approach had been intended to mimic that system.
7. The mainframe on which the current admissions system was running was scheduled for removal in four months.

Considering all of the options, the following decisions were made:

1. The Informix 4GL language would be abandoned as there are more efficient languages available to create interactive screens.
2. The new system had to be developed on a single platform since there was not sufficient staff nor time to
develop a version for all desktop platforms.
3. Due to the time constraints, there was not time for a programmer to learn a new language. Therefore, the system had to be developed with a tool one of the current staff had at least some established skills.

After giving the matter considerable thought, the Web was established as the application base for the following reasons:
1. It provided a platform which could readily be distributed across the campus.
2. A single copy of the system could be developed, eliminating the need to write a PC version, a Mac version and a Unix workstation version.
3. Development appeared to be quicker than any other tools available at the time. It was certainly faster than using 4GL.
4. Nearly all of the ADP programming staff had at least a vague familiarity with HTML which meant there was depth for support and advice.
5. The Web had proven to be a basis for the future from which to distribute information.

Convinced of the wisdom of this approach, UMR made the announcement that all future program development would be made using the Web as its base and Perl as the programming tool. This seemed like a reasonable direction to take, although it would have been preferred to have initiated a new direction of this magnitude with an application that was much less complex than the admissions system. However, with the announced removal of the mainframe looming larger by the day, one programmer was assigned to the task on a full time basis. This individual had been on the staff for barely a month, but there was no one else available. She had some limited experience with HTML, but no experience with Perl. Given those conditions, the assignment was initiated with the enthusiasm of a special agent ready to initiate the next assignment into uncharted territory.

Never having developed an entire system with this approach, there were certainly many things to learn and relearn. Expecting the Web would be slower in response, considerable thought was given to screen design and efficiency. In creating the first Web data screen it was realized several of the former 24x80 screens could be combined in a single Web screen. The anticipation was the new format could help the response time by reducing the number of times the database had to be accessed. At least in theory it seemed logical.

When the first prototype screen was developed, a meeting was set up with the two key persons of the Admissions Office staff responsible for the data maintenance of the current system. Anticipating a positive response from the users with the appearance of the better-looking screens and the inclusion of more data, the reaction of the users was a sizeable blow to the programming staff. Immediately, the users rejected the idea of employing a mouse, stating the technique was much too slow for volume data entry.
Shocked, but undaunted, the programming staff retreated and reconsidered the logic in the screen presentation. There was no refuting the fact the prototype screen was cumbersome for volume data entry, although appearing to be more efficient for the casual or new user. Focusing on data entry efficiency, the screen was substantially redesigned. The basic philosophy was to categorize the screen layout to segregate techniques into two major sections. That information which required input from the keyboard was collected in a single area as much as was possible. Information which could easily be skipped or prompted as needed, radio button type entry was assembled in a single section with optional data entry. Confident in the wisdom of this approach, the users were approached for a second review.

Using caution learned from the first meeting, the second presentation was prefaced by highlighting the advantages of the new approach:

1. Development of their system would be faster with the new tools.
2. Consistency of screen presentation would be possible across the campus regardless of the type of operating system used by the user.
3. They would not have to worry what type of desktop equipment their campus users had with this common format.
4. Learning the system would be easier for new users.
5. Data entry would be more accurate by the use of drop down boxes where table items were required. No longer would the user have to know the codes required in many fields.
6. In the mouse intensive section, the data entry was actually faster because the items needed for individual entry could be easily pointed to by the mouse rather than having to tab through all of the empty fields.
7. There would be fewer screens required, thereby potentially increasing the response time.

With unrealistic confidence in the soundness of the new approach, the ADP staff was stunned when the second demonstration was received with no better reception than the first. The reason? It was plain and simple. The two key personnel from the user office did not like to use the mouse. Most reluctantly, the ADP programmer altered the initial screen to give the user the ability to select the method preferred. The default remained a mouse approach, but the second option eliminated the mouse and require them to navigate through the screens with the use of the TAB key.

Having reluctantly created a second option, the ADP programmer addressed other issues with the functioning of the system. Those addressed were:
1. The system had an edit which prevented the user from exiting the present screen if key data fields were not complete with valid information. However, it was noted those key fields were not easily seen on the screen. The solution: present all heading text in blue for key fields while leaving the heading text for noncritical fields in black.
2. In the mouse version of the routine, all date fields had the potential for data entry, thereby slowing data entry. The solution: add a radio button that when clicked inserted the current date into the date field. It wouldn’t eliminate all data entry in these fields, but manual entry would be required only if the current date was not desired.
3. Such large lookup tables as high school slowed response and made the data entry screen very cumbersome. The solution:
   a. Terminate the first data entry screen at the point high school would be entered.
   b. Advance to a new screen, which prompted the user for either the high school code or a portion of
the high school name.
c. Present the matches found to the user.
d. When the user selected the school desired, another screen appeared prompting for the high school data.

4. In inquiring for the existence of a given person on the database, the user felt there were many times where only limited information was really needed. The solution: displaying critical information would eliminate the need to present an extensive detail screen and speed up the process. Therefore, the user wanted the resultant screen from a name search to present more than just prospect names. Address and phone number were added to this screen at the user’s request. If more detail information is needed, the user can select that entry and be presented with additional data on that prospect.

After nearly six months of development, the ADP staff thought they were ready to set an implementation date. However, the campus systems group pointed out there was a significant security breach possible with each URL presented to the users. Having a mainframe mentality, the potential for this intrusion had not at all been considered. The problem fundamentally focused on the fact in a traditional mainframe approach, a person’s access is checked and verified at the entry point of the system. Once access has been granted the security system does not have to be concerned with the individual’s authentication. There is only one entry point with the traditional approach.

What wasn’t anticipated with the Web was the entry point into the system potentially is at each URL. This meant an intruder could enter the system at any point for which they knew a URL. Consequently, security had to be check with EACH URL and not merely at the planned system entry point. This oversight was a direct result of never having developed a full-blown application on the Web and merely not thinking through the dynamics of the new environment. Coincidentally, the systems staff had just completed a program which addressed this very concern. Called the “Privilege System,” this new routine addressed the dynamics of levels of authority as well as user access maintenance. In principal the Privilege System is integrated into the application as a subroutine library. This system attaches the user's authentication (privilege code) with each URL presented, thereby sealing a security hole. The system had been in use with several rather simplistic applications and appeared to work acceptably well. This was, however, the first time it had been used with an application of this magnitude. In order to incorporate this security feature, every section of code had to be modified where a new URL was being called.

In reality the security system implemented is a two-step process. The following is an overview of the security system implemented in 1996:

Security fundamentals:
1. All secured application code is kept in an auth-cgi-bin directory.
2. Every userid must pass authorization to be allowed access to the requested script.
3. Every secured URL requires userid/password authentication.
4. Every secured application script requires a privilege code from the Privilege System table.

Security steps:
1. Authentication -- the set of code which tests the userid's access to a secured system. Unlike traditional systems which authenticates only at system entry point, this system tests authentication at every screen access. (every URL)
2. Access -- determines the authenticated user's level of access to secured systems; a function performed by the Privilege System.

Security procedures:
1. User enters URL for the ADMS system.
2. Userid identified by authentication routine on the Web server.
3. Password requested by authentication routine.
4. When user is authenticated, request passed to Privilege System.
5. Access procedure scans the Privilege System table to determine user's access.
6. If access is authorized, Privilege System passes the request and the proper privilege code to the application program.
7. Application program submits query results to the user's screen.
8. With next URL, the browser submits the userid and password, thereby eliminating the prompt of the user for authentication. Access procedure repeats beginning with step four noted above.

New and basically untested, there was much to be learned with the new security approach. As can be seen from the overview, the design of the Privilege System requires a call from the database server to the security server before the new screen could be presented. The resultant was as could be predicted; absolutely horrible response time. With users accustomed to subsecond response, the new system absolutely crawled. A substantial delay was required to address this unacceptable response time.

Although it delayed the implementation by many months and created incredible frustration on the part of all participants, significant changes were implemented.

1. The programmer, unfamiliar with the hardly documented Privilege System, had imbedded unnecessary database calls in the program. With the assistance of the Systems group, those calls were reduced to a minimum. Additionally, the database was kept open rather than opening and closing it with each access.
2. The Privilege System was modified as follows:
   a. Accessed the database server only once.
   b. Reduced the calls to the database.
   c. Allowed the application to cache the userid.
3. A newer, faster database server with substantially greater memory was installed.

Finally, after fourteen months in development and two months prior to the actual removal of the local mainframe, the new system was rolled out. As many difficulties as this development effort had experienced, the final implementation went shockingly smooth. With only very minor changes requested, the system was used by the campus community during the entire summer of 1997. At last it appeared the new technique had shaken the problems which had arisen in working in an environment which was so new.
Just as the programming staff felt confident in their accomplishment, with the return of the students at the beginning of the fall semester the new system received significant complaints from the Admissions Office. The response time drastically fell off to a totally unacceptable performance level. Concerned with not being able to get their work done, the Admissions Office voiced their concern that if the response time was not improved they would be forced to either require their staff to work overtime or they would have to hire additional data entry personnel. There was no refuting all of their concerns were well founded. The response time had indeed become unacceptable again.

Even though there was reason to believe the response time phenomenon was load based, the Admissions system code was thoroughly reviewed to see if something in the code could be causing the problem or whether a change in the code could improve the response time. The results of the review provided no indication of a code related cause, nor did it reveal a performance gain from code modifications. At that point, a meeting with the programming staff, the Systems staff and the Networking staff was convened. The programming staff had added a log to its data entry screen which showed a subsecond response time between the database server and the Web server. While there did seem to be a correlation between time of day and the response time, nothing conclusive could be derived. Unfortunately, by the time the meeting was held, the response time had improved to an acceptable level.

Identifiable factors in the response time dilemma:
1. The response time had been at an acceptable level until the students returned.
2. There seemed to be a predictable degradation of response time in the afternoons.
3. The same response time dilemma was being observed in the new Web based chemical tracking system which was in production test; a system using the same techniques and written by the same programmer.
4. There was a repeatable difference in response time between versions of the browser.
5. The response time improved after the students were settled in for the semester.

The potential causes of the problem evolving from the meeting were:
1. The use of tables in the Web script was inefficient and could cause the desktop to thrash.
2. The current version of Perl cannot be compiled, therefore the code is potentially vulnerable and cannot operate with maximum efficiency.
3. The ability to archive the database had not been completed, therefore the size of the database was growing and causing the server to be appropriately slowed.
4. The version of the Web browser was shown to definitely have an impact. (A query submitted on a Pentium 200 with plenty of memory using a version 3.0 of Netscape actually ran several seconds slower than the same query run on a Pentium 150 notebook running version 4.0)
5. There could be a cache problem on the desktop.
6. The users could have other operations active on their desktop which were interfering with the response time.
7. The Web server could have a load problem.
8. The campus backbone or the routers could have a load problem. (A possibility which was considered by all to be most unlikely.)

The effort to locate the source of the problem was seriously hampered by:
1. Lack of a definitive statement from the user. (They could only say the “response was slow” or “the response was unacceptable.”)
2. Lack of a definitive statement of what the users were doing at the time of the slow response.
3. Lack of specific statement of the time the slow response occurred.
4. The sudden improvement in the response time.

While many of the potential causes of the problem were sound, the fact remains the system had performed reliably with each screen prior to the start of the fall semester. The use of tables, Perl and the version of the Web browser could not be substantiated as sources of the problem since the system response was not consistent nor predictable. The insertion of a monitor between the Web server and the database server showed prevalent second or subsecond responses, although this was inserted after the response time had improved. There was still a logical basis on which to believe the phenomenon was load based and not program code based.

Lacking a conclusive solution, the Admissions office was asked to keep a definitive log of their response times as soon as the system seemed to slow. They were also asked to immediately call the ADP programmer when a response time problem is seen. The server monitor is still in place, but insertion of a network monitor did not seem justifiable since the response time had improved and there was little reason to believe that is at fault. At the present time, this remains an open problem.

Regardless of the experiences with this approach and the resultant frustrations, the ADP staff still is convinced of the soundness of this form of application development. The Web based version of the chemical tracking system which went into production in early October, 1997 uses the same techniques as does an equipment inventory system which was placed into production in July, 1997. All ancillary operations which support the admissions and chemical tracking systems are also written with the Web as the access method for the users. While written by a different programmer, the curricular transcript system introduced in 1996 is also a Web based system. In addition, UMR has provided such abilities as student elections, application for admissions, student organization membership maintenance, electronic class roles and teacher of the year balloting through the options from the Web.

The World Wide Web in many respects is a tiger still to be tamed. There are complexities which perplex and challenge the programmers who wish to make it jump through the hoops. Unlike traditional systems which dealt with intelligent mainframes and dumb terminals, this new medium requires the developer to implement multilevel strategies; strategies which include not only placement of data on the screen, but fonts, colors, graphics and complex links. Due to its complexities the application designers must more thoroughly design and test their product, then be prepared for addressing unpredictable failures reported from every corner of the globe, caused by variations in browsers or versions of browsers as well as desktop configurations and service providers. Failure to establish a sound application strategy, including security, can maul an operation as badly as a Bengal tiger can maul a careless trainer. The developers can truly feel they have a tiger by the tail. Despite this challenge and the new techniques which must be learned, today there exists no better platform for delivering a computer product to the customers. When properly performing, the Web puts on as spectacular and awesome a display as the circus tigers in the arena.

REMEMBER ...
Appendix

The admissions system in its final form: (Admissions Data Management System or ADMS)

1. Basic demographic information (all basic information maintained on prospective or accepted students. Because not all students corresponding with the campus provide a social security number, the key to this and many of the other ARMS tables is the date/time stamp when the student’s record is first created.)

2. High school information (all basic high school information including such items as: a CEEB code, class rank and expected date of graduation.)

3. High school subject matter (as the name implies, this table contains all subject matter the student completed in high school as required by the state.)

4. Web application for admissions (since the ARMS system receives information ranging from postcards to formal applications, it was felt too “casual” to include the Web application for admission data with other ARMS data. Written and introduced in April 1995, this feature has remained a totally independent table.)

5. Official test scores (again to segregate self reported test scores from official test scores, a separate table was created just for this data. In the previous system this process required an ADP programmer to execute a cumbersome routine to load the data from a diskette to an SQL table. The Web routine introduced in the late spring/early summer, 1997 provided the Admissions Office with the ability to do this themselves. The only intervention from ADP is to make once a year table definition adjustments according to the changes instituted by the testing agents.)

6. Systems tables (a combination of tables required by the ADMS system and existing tables downloaded from the SIS.)

7. Contact table (the Admissions Office in its original specifications years ago wanted the ability to identify all contacts the campus offices have with prospective students. This feature allows the Admissions Office and the various campus offices to record campus visits, phone calls, mailings or other contacts with the student.)

8. Tracking table (the Admissions Office in its original specifications years ago stated the need to track a prospective student's file around the campus. This table allows Admissions, departmental, dean and international student offices the ability to denote their reception and action on a prospect's file. By this manner the Admissions office can inform the student of the status of their application should they call to inquire.)

9. Orientation program (in the new system it was determined there was a need for the Admissions Office and the office coordinating the new student orientations to work more closely together. There was a table and screen included in the new system to maintain information on a given student's orientation
10. Promise program (a new program introduced in 1996 required the ability to maintain information on students interested in this option. Support for this activity was included in the new ADMS system.)
11. ACT demographic data (added October, 1997, this information is pulled from the ACT provided diskettes and contains high school subject and GPA data supplied by the students with their ACT test application. The intent of the additional information is to determine admissibility of students reporting ACT scores to UMR)

Ancillary operations:
1. Test scores reported by SAT (This Web routine provides the Admissions Office with the ability to upload data from the SAT diskettes to the ADMS system via a Web routine.)
2. Test scores reported by ACT (a modified version of the SAT routine, this feature provides the user office with the ability to upload ACT diskettes to the ADMS system.)
3. Archiving (This routine placed into production in November, 1997 provides the Admissions Office with the ability to specify the parameters to be used to move students from the active tables to the history tables of the ADMS system. The user office initiates a Web script which when complete sets up a batch job that night to archive the data according to the script's specifications. This eliminates the need for ADP to perform the archiving for them.)
4. Ad hoc reporting is handled by PowerSoft's InfoMaker software. It is a GUI program that replaces the SQL/QMF software utilized in the previous ADMS system.
5. Delete records. There was a concern for restricting who could delete student records from the database. At the user's request, the system was rolled out with the delete capability restricted to only two people designated by the Admissions Office.
6. User access. Through the use of the Privilege system, the "system administrator" in the Admissions Office has been given the ability to maintain the user access profile, eliminating the need for ADP to perform this database function for them.
7. Batch program run to create WordPerfect merge file to produce letters to students. Programs set up to run automatically on varying days according to specifications of the user.

Decisions made relevant to the Web development approach:
1. The program language would be Perl
2. Where possible, all maintenance and reporting functions would be performed by the user office, thereby reducing the load for ADP and leaving the scheduling at the discretion of the user office.
3. The code would be written to eliminate the database vendor or browser vendor as factors in the functioning of the system. Code would be generic enough that a change in database vendor would not require a significant modification to the system. Since ADP could not control the browsers being used, no vendor specific features could be allowed.
4. Any system designed would have access to the local data warehouse.
5. Any system designed would use data from other systems where available rather than duplicating the data.
6. Mouse functions and keyboard data entry had to be segregated where possible.
7. Data security had to be preserved throughout.