

This paper was presented at CUMREC 98, The College and University Computer Users Association Conference. It is the intellectual property of the author(s). Permission to print out or disseminate all or part of this material is granted provided that the copies are not made or distributed for commercial advantage and that the title and authors of the paper appear. To copy or disseminate otherwise, or to republish in any form, requires written permission from the authors.



A Case Study in Client/Server Design: ASU West Student Prospect Tracking

Niraj Gyawali
Application Systems Analyst
ASU West Information Technology
4701 W. Thunderbird Rd.
PO Box 37100
Phoenix, AZ 85069-7100
Phone: 602 543-8318
FAX: 602 543-3260
Email: Niraj@asu.edu

Katherine J. Ranes
Application Systems Analyst, Principal
Information Technology
Arizona State University
PO Box 870101
Tempe, AZ 85287-0101
Phone: 602 965-2772
FAX: 602 965-0028
Email: kranes@asu.edu

Arizona State University (ASU) is one of three public universities in Arizona and is located in the Phoenix metropolitan area. It is an expanding Research 1 institution. Over 49,000 students enrolled at ASU in the fall of 1997. The Main campus in Tempe serves students from freshman level through doctorate and has 12 colleges. The West campus offers only upper-division and graduate courses. An emerging campus, ASU East, and the Extended Campus serve additional parts of the county and state.

Client/Server computing forces developers to revisit the principles of design. This paper will review the design decisions made by the team as they built an application to track student prospects.

Key questions are:

- Graphical User Interface Design: Should all departments who work with prospects use the same graphical interface?
- Location of Business Rules: How can the rules for the enterprise and rules for a local office be enforced? Which design will give flexibility and provide ease of maintenance?
- Data Base Design: Can a database be designed to meet the needs of all users? Which data should be housed in a central database vs. managed in a local database?
- Reuse: How can components be designed so they can be reused by other applications or extended as the system grows?
- Tool Preference/Choice: What tools should be used for development?
- Customer Acceptance: Will the customers easily relate to the application?
- Performance and Data Use: Can the same database be used for reporting as for transaction processing?
- Security: How do you build a security scheme that is sufficient yet is not burdensome to maintain?

A Case Study in Client/Server Design: ASU West Student Prospect Tracking

Business Problem

ASU West wanted to identify the activities which were most effective in recruiting students. Then once the student applied, it was important to know if these same students were admitted and enrolled and how they progressed toward their degree objective. Information about recruitment efforts was kept in differing systems in each college. These local applications were not easily connected to the enterprise Student Information System (SIS) or the Data Warehouse. Some recruitment efforts were duplicated because there was no single source for information about prospective students at ASU West. The need for a campus-wide student tracking system was recognized.

Similarly prospect files about Main campus prospects were kept in department systems in colleges and in Undergraduate Admissions. There was no direct link from any of the student prospect systems to the Student Information System. Capturing much of the demographic information about a student, name, address, etc. required multiple entry, originally in the prospect system and then again in the SIS when the student applied. Inconsistency in systems made it difficult to match an individual's information.

Other Improvement Initiatives

As this need was recognized, two University-wide projects had an impact on how the problem was approached. The Student Process Reengineering Project established the Student Tracking Project as one of its subprojects. This meant that the solution would be approached from an enterprise level and project team members were drawn from various departments.

A new client/server application, the Affiliate Management System, was in development. The purpose of this system was to build a system to maintain core information about people and organizations that have a relationship with ASU. It would store names, addresses, e-mail addresses, phones, demographic data and the type of affiliation the individual had with ASU. This system would interface with other systems that maintained more specific information about the affiliation, for example the Student Information System (SIS) and the Human Resources Management System (HRMS). The Student Tracking team was directed that the new Student Tracking System should use the Affiliate database as the source for data and rules whenever possible.

Direction to the Team

Management gave the Student Tracking Development Team clear direction. The charge to the team was to deliver

1. A central database that contained information that most departments needed about student prospects
2. Centrally managed processes to enforce the business rules for the central data
3. A design and programs that could be used with a variety of GUI application development tools, including Web tools
4. Modules that used the Affiliate database and enforced the same rules as the Affiliate Management System
5. Security to protect data at the enterprise level
6. Graphical application(s) developed and maintained by distributed developers.

This direction was based on the idea that applications consist of layers and it is possible to segment these layers. Different layers could be developed by different groups, which did not necessarily have to use the

same tool for development. This architecture seemed more feasible using client/server development where different layers of processing occur on different machines.

PRESENTATION	Processing Constraints
EDITING AND DOMAIN CHECKS	
BUSINESS RULES	
DATA RULES	

The focus of this paper is to discuss how the development team met this directive. The “PRESENTATION” layer was assigned to the department developers, in this case, ASU West Information Technology. The “EDITING AND DOMAIN CHECKS” became a joint responsibility with both client application and server programs editing for valid data types and checking for valid domain values. The “BUSINESS RULES” were enforced by server programs and the DBMS; these were the responsibility of the central IT organization. The DBMS and Stored Procedures enforced “DATA RULES”. Both the client and server development was influenced by the “Processing Constraints” imposed by the computer architecture in place.

It is unlikely that others will necessarily choose the same tools we used for development, but we believe others who design for client/server will ask many of the same questions we did. We believe the success of our efforts relied heavily on our design decisions. We hope others will benefit from our experience.

How do you paint your GUI?

Graphical User Interface Design: Should all departments who work with prospects use the same graphical interface?

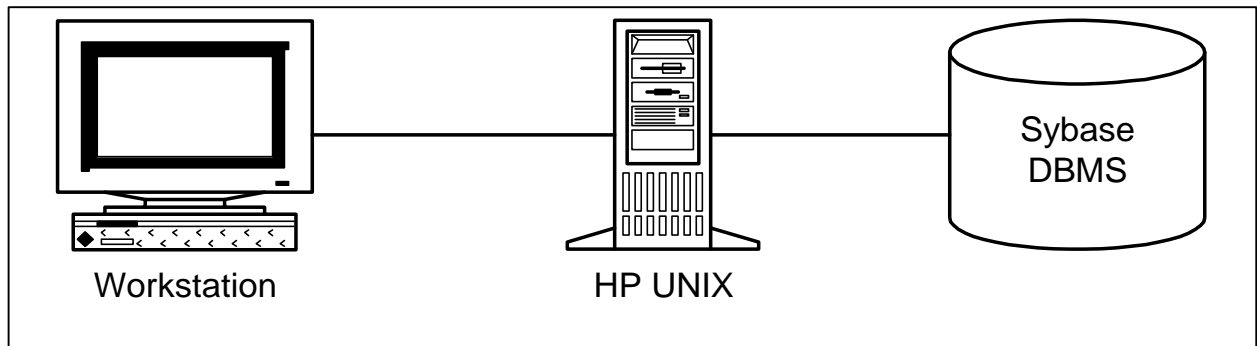
One of the advantages of Graphical User Interface, GUI, is that it is more intuitive and easier to use. The graphical elements echo concrete objects and the user does not need to learn complicated coding and navigation schemes. Some standardization has come to graphical interface because everyone promotes “common look and feel”. However, in graphical design there are many more degrees of freedom than in the standard character-based interface. Application development tools are evolving and some tools offer more development options than others.

A decision was made that all student tracking applications *would not* need to use the same interface. This is because in a University, one size does not fit all. This is particularly true with the different recruiting offices. Main campus Undergraduate Admissions does significant work with those coming to ASU directly from high school. Standardized test scores are a key source for prospect information and they use this data significantly to target recruiting efforts. ASU West has students entering from other colleges as juniors, so standardized test scores have almost no role in their recruitment process. Significant effort could be expended to reach a “consensus” and the result would be an application that did not fit any office’s needs. Instead, we decided to build multiple interfaces (applications) and design flexibility into the remainder of the application so it would work with all of them.

Rules, rules, where do you put the rules?

Location of Business Rules: How can the rules for the enterprise and rules for a local office be enforced? Which design will give flexibility and provide ease of maintenance?

In the architecture we chose, there were three locations that business rules could be located. The figure below shows the three components that made up the application structure. The rules could be placed in the client application, in the server program module, or incorporated into the database using database definitions or stored procedures. We placed rules in all three places, depending on the type of rule. Our goal was to have as few rules in the client application as possible.



We assessed the rules to determine which ones were static. For example, birthdate when present would always be a date, gender will probably always have values of “Female” and “Male”, and flags will have values of “Y” or “N”. These static values were placed within the data definition and enforced by the Data Base Management System, DBMS. Additionally, the client application enforced these rules by the features of the presentation, such as checkboxes or field edits. For attributes with less static domains, look-up tables were established on the database. The client application had the choice of allowing freeform entry or reading the look-up tables and presenting a dropdown list. The server program guaranteed that domain checking was done. It did this by comparing input values to the tables before updating the database.

We also looked at rules to determine if they applied to the enterprise or were restricted to the local department. Some enterprise rules were reflected in the database definition and structure by defining relationships. Our application development tool also generated stored procedures to enforce the relationship rules. Some enterprise rules were too complicated to be supported by database definition or to use stored procedures efficiently. “The social security number for a prospect who is also an employee cannot be changed,” is an illustration of this type of rule. These rules were coded in the server programs only. In some cases, the rules for the enterprise were more flexible or more generic than the rules that the department procedures required. When this happened, we judiciously put business rules in the client application.

Yours, mine, and ours, what data do you need anyway?

Data Base Design: Can a database be designed to meet the needs of all users? Which data should be housed in a central database vs. managed in a local database?

The biggest danger to any development effort is that the wrong system would be built. To deal with this risk, a business team was set up. This cross-functional team was comprised of system analysts, business analysts and staff from various departments. Team meetings identified what functionality users needed in the new system. Functionality already present in the disparate systems in numerous departments at the

various campuses was also identified. Furthermore, interviews were scheduled with users and managers in different departments to confirm new system functionality.

Most of the data used in the various prospect systems was common. The differences came in the specifics of whether the data was required and how broad the domains should be. To illustrate, all departments wanted to know when the prospect planned to enroll. However, the Graduate College wanted this information using the University established semester codes and the extended campus wanted this information as a date. In this case, the rule became “for a planned enrollment, semester or beginning date is required.”

The business team came up with a conceptual model of the data and domains that the database needed. Two techniques that were particularly valuable in building the conceptual model were:

1. Activity Diagrams which described the workflow of the business. This technique was helpful in understanding the steps people go through in doing various tasks, and also helped in eliminating unnecessary sequences in business processes.
2. Entity Relationship Diagram that identified key attributes needed to capture data and the relationships between various entities.

The resulting model is very flexible with many optional attributes and some fields are very generic.

Sharing data among various units raised a question among participants. Could they trust each other to update the data correctly and would sharing data lead to competition for recruits? This concern was alleviated by adding tracking fields to identify who had updated the record and by realizing that sharing the workload for collecting and updating data could improve data quality rather than hurt it. Also the idea that everyone would use the same rules and that centrally managed programs would enforce these rules reassured the participants.

The direction for the design did not rule out local database extensions for data that was not common to all offices. As of now, no additional items have been defined that cannot be accommodated in the common structure. However as the application matures and is used by more offices, it is possible that department databases may supplement the central database.

Reduce, Reuse and Recycle, How do you build a reusable resource?

Reuse: How can components be designed so they can be reused by other applications or extended as the system grows?

The goal of the development team was to write as little new code as possible. We wanted to use programs and tools already available. For the programs that we did write, we wanted them to be reusable as other client applications interacted with the server programs or as new features were added.

One way we reduced the number of routines that needed to be written was to use Open DataBase Connectivity, ODBC, calls to retrieve data. This removed the need to write various query and search routines on the server.

Another way we reduced programming was to share a common design of all server programs. All server programs used the same routines for version checking, security checking, edit routines using the lookup tables and similar input and output structures. Using common routines cut development and testing time. Similar input and output structures made the interface between client application and server easier to adapt from one call to the next.

The Affiliate Management System had server programs already written to update the Affiliate database and to interface as necessary with outside systems. The Student Tracking Team referenced these servers, reused many routines and restructured the server programs so they conformed to the open architecture design.

Writing small single-function routines made the reuse strategy work. We structured the routines' inputs and outputs so they used the smallest amount of data possible. Then the routine could be reused whenever the function was needed and whenever the required inputs were available. Several small routines were assembled for more complex operations.

One of the first design decisions made during the application interface creation was the extent to which Object-Oriented Programming, OOP, capabilities would be used in Visual FoxPro. Rather than use the more traditional, procedural methodology, an application object was used. The Visual FoxPro base class library was created as an abstract class for forms and controls. Functional classes were created from subclasses. These functional classes have data-bound controls to display and input data, composite controls to handle related tasks, and action controls to handle navigation.

Design meets the real world, Can we make it work?

The design supported a vision of an application that would work like this.

Once the user enters the userid and password through a login screen, the application prompts the user to search for a student prospect. If a record already exists for the person who was searched, then the affiliation of the student is queried to see if the person is a prospect. The search function was supported by a server routine. The user can bring up the person's record to determine by viewing more details, if a matching individual is present. This used ODBC search capabilities. Then following the search, the user could add prospect information. If the person did not exist in the system, then a new record is added to the affiliate database and a new prospect record is added to the Prospect database. Any create, update or delete of records first calls an OLE (Object Linking and Embedding) server routine to validate the data before updating the information on the Sybase database. Later displays of the prospect's record and associated information used ODBC calls.

Up to this point, the paper has discussed design concepts and how we approached that process. However, client/server technology is evolving and the project team members were learning how to use it. The constraints of the computing environment altered some of our approaches and raised concerns as we built the system.

What's your favorite tool?

Tool Preference/Choice: What tools should be used for development?

What technology do we use to build a multi-tier client/server application? Similarly, can we build an architecture that actually delivers the functions that users need through a Web browser? Can we use the Object-Oriented Programming methodology for design work? These were the questions before the development team, comprised of developers and database administrators. Initially, the technical team researched and identified what the major components would be and how they would be built and put together. Some important issues that were noted were:

- What will happen if a piece of technology doesn't work?
- What if we can't connect the pieces of the puzzle?
- What is the likelihood of something going wrong? How would we cope if that happens?

We chose tools based on what was available, the team members' experience, existing applications, performance and the ability to expand. Since tools are evolving, you may want to consider a proof of architecture phase for your project. A baseline architecture was designed. One key project deliverable was to show that the tools that we picked would work together. We proved the architecture on a small part of the system before starting full-fledged design.

Our database was Sybase, since the Affiliate database already resided on Sybase. Server programs were written with Composer, from Texas Instruments, since the Affiliate Management System already was written using this tool. This allowed us to reuse affiliate rules and interfaces. Composer also generated stored procedures based on database rules. This meant we did not write any stored procedures. Another advantage of Composer based code was that it is portable to other platforms should we move the database to another DBMS.

Visual FoxPro was chosen for the client application. It was chosen because of project team member's experience with FoxPro. It also was a Microsoft product which improves interfacing with other Microsoft desktop tools.

Another Texas Instruments tool, Arranger, generated OLE objects so that the Visual FoxPro application could interact with the Composer middleware and invoke the Composer routines on the UNIX server. The OLE calls and the version of Visual FoxPro required that the client machines be running Windows '95. Additionally ODBC calls directly to Sybase used Sybase Open Client.

Third time's the charm!

Customer Acceptance: Will the customers easily relate to the application?

What the customer sees shapes the impression of the whole application. The design started with a prototype. A design team comprised of developers, business analysts and users at the West Campus worked to evolve the prototype to a working application. A RAD tool, such as Visual FoxPro was used to build the initial prototype. Once the team approved of the prototype, construction of the functional application was started.

Construction of the applications interface was done in a series of iterations. For each iteration there was analysis, design, coding, testing, and integration. Thus, each iteration built production-quality software that was tested and integrated and which satisfied a subset of the requirements of the project

Faster than a speeding bullet? How do I get to my information fast?

Performance and Data Use: Can the same database be used for reporting as for transaction processing?

The database was designed to meet the business rules. Little denormalization was done to improve performance. Some relationships were removed from the data model and retained by "hard coding" foreign keys to related tables. This was done so that related records could be stored in different databases.

Every server routine does version checking, security checking and edits the input data. This adds additional input/output. The design required this and we are currently monitoring application performance to see if this causes poor performance or database locks.

Another concern about performance is raised when in addition to transaction processing, reporting is added to the database overhead. There is concern that large queries against the relational database will degrade on-line performance. One way to solve this contention is to have reporting queries run against a replicated data source. At this time, resources are not available to replicate the data on another database on another machine.

Performance can be improved by using stored procedures. We consulted with the database vendor to be sure we were using stored procedures when they are efficient. Their advice was that stored procedures are advisable sometimes.

The tools we used have a TP monitor feature. This means multiple threads can be managed on the server and programs can be preloaded. Tuning of this environment can be done to improve performance. Many environments do not include this feature and rely solely on the database to manage application threads.

Do you have the keys? Now that we've built it, do we need a security system?

Security: How do you build a security scheme that is sufficient yet is not burdensome to maintain?

It would be ideal if a single signon and password could authenticate a system user and authorize access to appropriate resources on the network. ASU does have a single id scheme with all system users having the option to obtain an "ASURite ID". At present there is no coordinated method to secure administrative applications using this ID and the associated password. The security structure at ASU will evolve as we build more client /server applications.

The Student Tracking application uses a combination of security mechanisms to secure the application and associated data. The client application itself is run from the ASU West LAN, so all users of the system must have access to the ASU West LAN and be authenticated with its passwords. This network security controls whether the user can run the application. Also, the authorities within the application are recorded in the Application Security database. A user is authenticated and given his/her authorities by running the ASU Login application. Thirdly, to query the data directly to use it for reports, without using the application, the user must be given database access. This means to use all functions of the application, a user id (hopefully the same one) must be recognized by three separate security systems, maintained by three separate units. It's not pretty but it is available.

You may be faced with some of the hurdles for securing your application. Include some time in your project plans to investigate the security capabilities and options at your institution.

Summary

We used a four-phase methodology.

Inception: The business rational for the project was established. An initial analysis was done to get a sense of the size of the project. Then the sponsors gave a commitment to go further with the project.

Elaboration: At this stage the go-ahead to start the project was given. We had basic direction but we needed a better understanding of the problem. This understanding was gained by gathering business requirements and proving the architecture for the technical solutions.

Construction: Two teams were assigned to construct the application: the Main campus team which built the server routines and the West campus team which built the graphical application. The understanding we gained during the elaboration stage was the key to success during construction.

Transition: During transition, there is no development to add functionality. Most of the development done was to fix bugs. At this stage, customer feedback was used to add small or absolutely needed functionality. It was also the point between the beta release and the final release of the application.

Coming Attractions

With the server routines in place there are many avenues we can pursue to extend the initial application. Several items on the drawing board are building a Web interface, extending the application to other offices on other campuses, adding batch interfaces to support updates related to mass mailings, and enhancing reporting capabilities. The design we developed did meet the initial charge to the team and it has potential for the future growth. Stay tuned.

Student Tracking System Components

Key:

OLE – Object Linking and Embedding standard

ODBC – Open Data Base Connectivity standard

Arranger – middleware from Texas Instruments (now Sterling Software)

Composer – application development tool & environment from Texas Instruments (now Sterling Software)

HP – Hewlett Packard

SIS – Student Information System

SybaKse Open Client – Middleware from Sybase to communicate with Sybase DBMS

