

This paper was presented at the 1995 CAUSE annual conference. It is part of the proceedings of that conference, "Realizing the Potential of Information Resources: Information, Technology, and Services--Proceedings of the 1995 CAUSE Annual Conference," pages 1-5-1 to 1-5-11. Permission to copy or disseminate all or part of this material is granted provided that the copies are not made or distributed for commercial advantage. To copy or disseminate otherwise, or to republish in any form, requires written permission from the author and CAUSE. For further information: CAUSE, 4840 Pearl East Circle, Suite 302E, Boulder, CO 80301; 303-449-4430; e-mail info@cause.colorado.edu.



**You CAN Teach an Old Dog New Tricks:
Extending Legacy Applications to the New Enterprise Architecture**

Nicholas C. Laudato
Dennis J. DeSantis

University of Pittsburgh
Pittsburgh
Pennsylvania

In 1994, a team of faculty and staff at the University of Pittsburgh completed the design of an enterprise-wide information architecture. The architecture provides the blueprint for developing an integrated set of information services, processes, and technologies, enabling significant efficiencies in business and service processes, and facilitating informed decisions concerning information technology expenditures and acquisitions.

Revolutionary in design, the architecture utilizes pattern-based abstraction techniques to enable evolutionary implementation and intelligent use of legacy systems. This presentation addresses the prototype application programs developed to illustrate sample information processing patterns as well as the look and feel of the envisioned architecture. It focuses on the latest and most ambitious of the prototypes, a client/server application that puts a graphical user interface on a character-based legacy application. This "prototype" is currently in use by faculty advisors and staff in five schools, and is being implemented by several others. It allows the legacy system to be integrated with the University's envisioned enterprise-wide information architecture.

You CAN Teach an Old Dog New Tricks: Extending Legacy Applications to the New Enterprise Architecture

Nicholas C. Laudato and Dennis J. DeSantis
University of Pittsburgh

Background

The University of Pittsburgh is an independent, nonsectarian, coeducational, public research institution. Founded in 1787, it is a state-related member of the Commonwealth of Pennsylvania System of Higher Education. In addition to the main campus in Pittsburgh, the University operates four regional campuses in Western Pennsylvania. Among its five campuses, the University offers over 400 degree programs, and, during the Fall Term, 1994, enrolled 32,519 students (headcount).

In the fall of 1992, the newly-appointed Senior Vice Chancellor for Business and Finance conceived of an initiative to transform the University into a modern organization where information is viewed as an asset and used to strategic advantage. He selected a senior faculty member from the Department of Information Sciences to design and direct a special project. The project director initiated the *Information Architecture and Process Innovation Project* in February, 1993, with four individuals selected because of their background, knowledge and experience with varied components of the University. The Team defined the following mission:

- Design an architecture for the University Information System (UIS) that will provide a framework for making decisions about information systems and for improving the UIS in the future;
- Establish a methodology for business process reengineering using the UIS; and,
- Develop a plan for migrating from the current systems to the envisioned UIS.

The project team made their final recommendations to the University community in June, 1994. The architecture project is documented in *Reshaping the Enterprise: Building the Next Generation of Information Systems Through Information Architecture and Process Reengineering*.¹

This paper addresses two important features of the UIS architecture and its implementation: the information processing patterns that are central to defining and evolving the architecture, and the prototype application programs developed to illustrate those patterns. It focuses on the latest and most ambitious of the prototypes, a client/server application that puts a graphical user interface on a character-based legacy application. This "prototype" is currently in use by faculty advisors and staff in five schools, and is being implemented by several others. It allows the legacy system to be integrated with the University's evolving enterprise-wide information architecture.

Addressing Legacy Systems

Part of the original motivation for the Information Architecture and Process Innovation Project stemmed from limitations of the University's existing legacy systems. Like most institutions, the University acquired or developed its core business applications over 20 years ago, and implemented them in a centralized IBM mainframe environment. With the explosion of local computing and networking capabilities, individual schools, departments, and administrative units have built local systems to supplement or complement the older central systems. This proliferation of systems has resulted in islands of automation and redundant data.

All of these systems could not be replaced simultaneously, but would have to be addressed in priority order, determined by their importance to the University's mission, their

relative state of disrepair, and the relative degree to which they adhere to or contradict the information architecture. Of particular interest was the University's student information system.

The Integrated Student Information System (ISIS) addresses the admissions, financial aid, course, registration, housing, billing, grading, and academic history functions of the University. It feeds the financial, human resources, ID center, and alumni development systems, and is central to the work of the University. ISIS is the newest and best of the University's legacy applications, and works well for its original purpose. It is a character-based application, written in COBOL, that runs in CICS on the University's IBM 3090 mainframe. ISIS is implemented in SUPRA, a quasi-relational, non-SQL compliant database management system from Cincom, Incorporated.

To access ISIS, a user logs into the mainframe via a 3270 terminal session. ISIS transactions are invoked by entering a four-character transaction code, followed by a transaction key. The transaction key specifies parameters for the transaction. For example, the ISIS Course Section List screen is invoked by the "SQQL" transaction code, followed by five parameters: the term code, subject code, course number, availability code, and campus code.

Sample ISIS Course Section List Screen

| CRN | TYPE | AVLB | GROP | SESS | DAYS | BEGIN | ENDING | BLDG | ROOM | DLIM | ENRL | SREM |
|--------------------------|------------|------|------|------|------|--------|--------|-------|------|------|------|------|
| SQQL 931,MATH,0031,A,P | | | | | | | | | | | | |
| SPEN SUBJECT Mathematics | | | | | | | | | | | | |
| W0AF 07/11/94 04:08 P | | | | | | | | | | | | |
| CAMPUS PGH | | | | | | | | | | | | |
| 0031 | ALGEBRA | | | | | | | | | | | |
| 20048 | LECT | OPEN | D16 | TERM | M W | 07:05P | 07:55P | BENDM | 720 | 0025 | 0003 | 0022 |
| SP INDICATORS: XLST | | | | | | | | | | | | |
| 0031 | RECITATION | | | | | | | | | | | |
| 22567 | REC | OPEN | D16 | TERM | M W | 05:45P | 06:35P | VICTO | 230 | 0025 | 0000 | 0025 |
| CR: 00.0 | | | | | | | | | | | | |
| 0031 | ALGEBRA | | | | | | | | | | | |
| 01441 | LECT | OPEN | E16 | TERM | T H | 07:05P | 07:55P | CL | 232 | 0025 | 0000 | 0025 |
| SP INDICATORS: XLST | | | | | | | | | | | | |
| 0031 | RECITATION | | | | | | | | | | | |
| 15811 | REC | OPEN | E16 | TERM | T H | 05:45P | 06:35P | CL | 236 | 0025 | 0000 | 0025 |
| CR: 00.0 | | | | | | | | | | | | |

ISIS is not compliant with the newly-designed UIS architecture. Its database is not SQL compliant and it is costly and difficult to access in client/server mode. The ISIS user interface is inconsistent with the architecture, and difficult to navigate for the casual user. The transaction keys, in particular, are difficult for the casual user to remember. One course screen may require the user to specify the academic term and course reference number, while a similar screen may specify that the parameters be entered in an entirely different order, or may require additional fields.

ISIS was designed for the specialist who would use a handful of transactions over and over again each day. For this type of user, the character-based interface is effective and efficient. However, the information needs of the University community have expanded since ISIS was originally designed. For example, academic advisors should have access to their advisees' student records, as well as up-to-the-minute knowledge of what course sections are still available for registration. In many of the University's schools, advising is performed by full-time faculty. As an ISIS user, a faculty member does not match the specialist model.

Compared to the specialist, the average faculty member will have infrequent, sporadic access needs, and may go months between periods of usage. Such users have difficulty retaining the detailed knowledge needed to easily navigate the ISIS screens. If one would compound the idiosyncrasies of the ISIS interface with those of a financial system (faculty often manage research grants), a purchasing system, and a human resources system, the cognitive load required to master several different interfaces quickly becomes a major obstacle to effectively using the information system. In the envisioned information system, the common graphical user interface would resolve this problem.

Pattern-Based Growth versus the Master Plan Approach

Because of the broad scope of the envisioned University Information System, it was clear that its implementation would have to be phased in over several years. The traditional approach to implementation is to specify a detailed *master plan*. However, one of the problems inherent in implementing a traditional master plan over a period of years is the rapidly changing nature of the technology itself. Components of the solution become obsolete before the solution is even partially implemented. The monolithic architecture ultimately is abandoned as irrelevant to the changing times. To avoid these problems, the project team eschewed the traditional master plan approach in favor of a *pattern-based approach* to building the information architecture.

This methodology was inspired by the Oregon Experiment², a unique approach used over the past thirty years in designing the University of Oregon campus. The project team's adaptation of the Oregon Experiment approach recognized many parallels between the architecture of towns and buildings and that of information systems.

In the Oregon approach, a set of six core principles serve to guide the building process through the use of a pattern language. The pattern language describes the detailed patterns for towns and neighborhoods, houses, gardens, and rooms. For example, the Oregon architects noted that "When they have a choice, people will always gravitate to those rooms which have light on two sides, and leave the rooms which are lit only from one side unused and empty³." In a master plan approach, this insight would translate to blueprints showing two particular windows in each room on campus. After waves of renovations, and with subsequent designs, the intent and specific implementations will both be lost. With a pattern-based architecture, the "light on two sides of every room" pattern stipulates that the architect "Locate each room so that it has outdoor space outside it on at least two sides, and then place windows in these outdoor walls so that natural light falls into every room from more than one direction." After twenty years, the guideline specified in the pattern language still works, whereas the blueprints are completely obsolete.

Creation of a Pattern-based Information Architecture

In a pattern-based approach, the information architecture is documented as a set of patterns based on information processing principles. Decisions about developing, modifying, or acquiring components of the architecture are made by evaluating proposals based on their adherence to the specified patterns. The patterns are subject to on-going review and refinement to ensure that they incorporate advancing technology and continue to meet the needs for which they were designed. The information architecture evolves as more and more projects are implemented that comply with its specifications.

The patterns must be communally designed and adopted, and will guide the design of everything in the University Information System. Patterns can be very large and general, as well as very small and specific. Some patterns deal with the behavior of computer interfaces, some with the distribution of data, some with hardware configurations, some with network protocols,

and others with data access methods. More specific patterns deal with report formats, application-specific functions, ordering of data on displays, etc.

In a pattern-based approach, the task of articulating the information architecture becomes one of identifying and documenting the information processing patterns that will underlie all information activities in the envisioned University Information System. This is not a one-time task, but a dynamic, evolutionary activity that requires each pattern to be continuously reviewed and refined. The documentation of the architecture thus becomes a “living document.”

The project team identified and outlined a set of information processing tasks they believed to be common across all business applications. A small sample of these tasks are further documented and illustrated as examples of the patterns that form the basis of the envisioned architecture. The first, a *finder pattern*, is used to identify an object in the database that the user wishes to view. A finder prompts the user for information that could uniquely identify the desired object. If such information is not available, the finder should prompt the user for more general attributes that can be used to search for a collection of objects that meet the specified criteria. In this latter case, execution of the finder would generate a browser. A *browser pattern* provides a list of objects, with enough information to allow the user to specify the exact object to be viewed. The next pattern, a *viewer*, displays the object. The viewer is typically segmented into pages or scrolling sections to allow all attributes associated with the object to be viewed without invoking additional transactions. Viewers also provide “hot button” links to other associated viewers, making it easy to locate related information. Finally, a *view-before-update* pattern specifies that you must view the attributes associated with an object before entering a mode that allows you to modify them.

One of the premises of the architecture is that these patterns, among many others, would repeat over and over again in different applications, with only the specific data elements changing from application to application. For example, a student finder would prompt for an ID number, but also allow a search on name; a purchase order finder would prompt for PO number, but allow a search by account number, user, vendor, and commodity; and a course section finder would prompt for term and course reference number, but allow a search by subject, number, and campus. If all of the University’s business applications were constructed from such recurring patterns, it would be easy for users to master the interface and seamlessly move from one application to another.

Architecture Prototyping

The information architecture project staff preferred to recommend guidelines that could be implemented using state-of-the-practice technology and reasonably cost efficient methods. For this reason, several of the principles espoused in the architecture statement were illustrated through a set of prototype applications that would serve as “proof of concept.” Some of these principles are:

- The client/server model should serve as the basic paradigm for applications in the University Information System;
- All business applications should share a common graphical user interface (GUI), initially based on the Windows GUI, but ultimately embracing multiple platforms (Macintosh, X-Windows);
- The common GUI should provide a consistent look and feel across all applications;
- The common GUI should be easy to learn and use; it should be intuitive and consistent with the standards relative to its particular platform; For example, if an application is running on a Macintosh machine, it should behave as a Mac application, not a Windows one.
- The common GUI should enable easy transferability of skill from one application to the next, and facilitate substitutability of personnel across applications; and
- Applications should easily integrate with desktop personal productivity tools.

The architecture staff completed four major prototypes during the first year of the project.

Course Inventory Prototype

The first prototype, the *Course Inventory Prototype*, was a Microsoft Windows application created using Visual Basic. The prototype was designed to provide query access to course inventory data that had been extracted from ISIS and converted into an SQL database (Microsoft Access). Because of the relatively static nature of course inventory data, this approach offered an acceptable means of enabling access to the ISIS legacy data. The Course Inventory Prototype illustrated several of the information processing tasks (patterns) that had been articulated in the architecture statement, including the finder/browser/viewer paradigm.

To maximize the reusability of application program code, the first prototype was designed to be as data independent as feasible. For example, instead of hardcoding the association between a database field and its display field in a window, the application used metadata, stored in a relational table, to link all display fields to the database. This allowed developers to quickly create a new window by first creating a new table in the meta-database delineating the required data elements and then cloning a similar existing window.

Application Builder Prototype

The finder/browser/viewer paradigm developed in the first prototype was replicated in a series of six smaller prototypes developed by Information Science graduate students. These student prototypes involved a wide variety of topics, including a classroom scheduling package, a car dealership program, a real estate program, and a purchasing system. This set of prototypes help verify the Team's assertion that the patterns being developed were flexible and generalizable.

Based on the experience of advising the graduate students in using the patterns to create additional prototypes consistent with the architecture, the Team developed a second major prototype, the *Application Builder Prototype*. This prototype carried the data independence discussed above even further. Twelve of the fifteen code modules created for the first prototype were generalized so they could be completely driven by metadata. The remaining three modules could then be tailored to create a unique application with a finder, browser, and viewer. This allows a programmer to generate a new application simply by creating the metadata and laying out fields on the viewer window. The Application Builder Prototype thus illustrates the possibility of creating an application software library containing reusable software components that embodied the identified patterns.

Class Roster Prototype

Because ISIS is the newest and best of the legacy systems, it is difficult to imagine a set of circumstances that would result in it being given a high priority for replacement. Therefore, it would be highly desirable if it could be cost-effectively modified to better fit the architecture. This means, among other things, that ISIS should have a graphical user interface (GUI) instead of the existing character-based interface. In response to this dilemma, the Information Architecture staff began work on a client/server prototype application to access ISIS data via a GUI front-end.

None of the earlier prototypes were examples of true client/server applications, nor did they wrestle with the problem of accessing the data associated with legacy applications. A new prototype was needed to address these two important issues. The Information Architecture staff polled faculty in the Senate Computer Usage Committee for suggested applications that would be potentially useful to a large number of faculty. Instructors had long expressed the desire to gain easy electronic access to up-to-date class roster information. A class roster is a list of

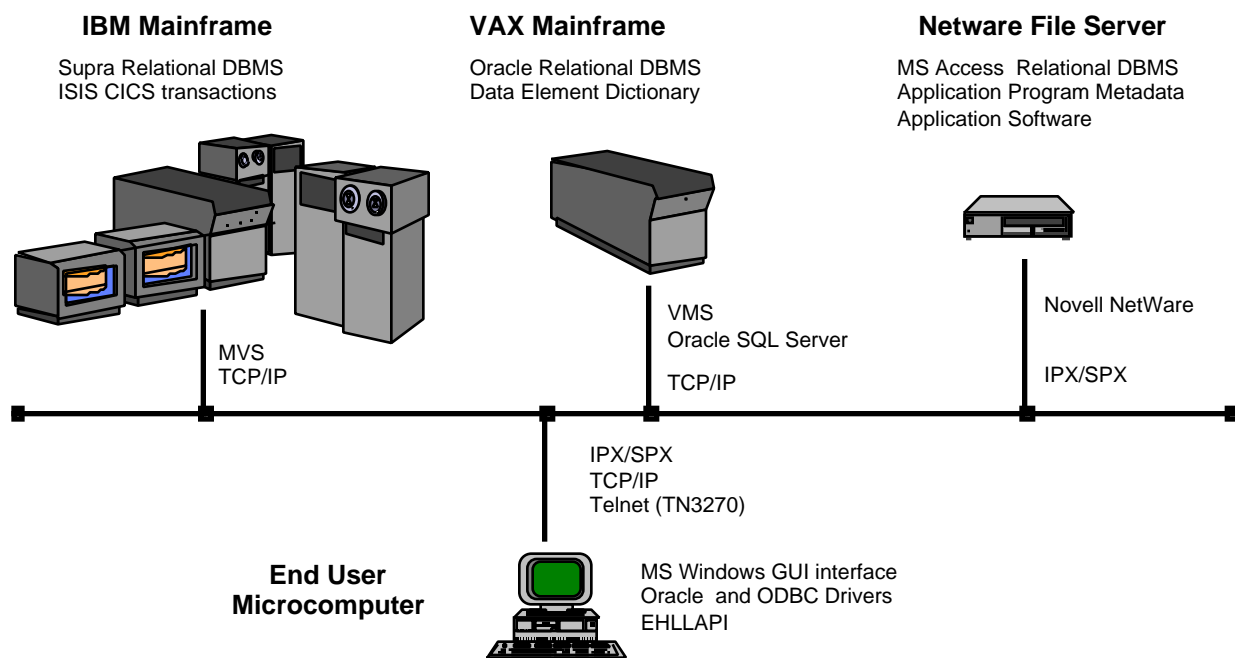
students who are registered for a particular course section. This seemed a simple, but important, topic, so a third prototype application, the *Class Roster Prototype*, was initiated with the goals of: (1) illustrating principles of the information architecture, (2) demonstrating the feasibility of client/server technology, and (3) testing the feasibility of accessing ISIS data through existing CICS transactions.

The project development team first sought to access the ISIS data in its SUPRA database. Because no middleware could be found to directly access ISIS data in client/server mode, the project developers decided to employ “screen scraper” technology to access the course and student data through the existing front end CICS transactions. This approach involves building a windows-based client application that establishes a terminal emulation session in the background. The terminal session is then accessed by the windows application via EHLLAPI (Emulated High Level Language Application Program Interface) middleware. The EHLLAPI middleware allows strings of information to be sent to the host terminal session and retrieved (“scraped”) from the resultant screens. To the host machine, the terminal session appears to be just another interactive user. At the client workstation, the user may not even be aware that data is being retrieved from a terminal session rather than a database.

Using this approach, the project staff designed an application that would allow a faculty member to interact with a GUI application to access ISIS. The application attaches to the host in background mode, and accesses the existing ISIS CICS transactions to generate a class roster. The class roster window displays general information about the selected course section and displays a scrollable grid containing data about each student enrolled in the course section.

To launch the Class Roster Prototype, the user first logs into an application server, which downloads the required executable images, libraries, and VBX’s (Visual Basic Extensions) to the client workstation. The application accesses a meta-database (Microsoft Access) on the server. This database contains information about the location, length, and type of student and course data on the ISIS screens that would be scraped.

Architecture of Class Roster Prototype



In the background, the application logs into an Oracle database server, which provides a repository for the University's data element dictionary. The application connects to Oracle SQL Server, and accesses tables containing valid values for ISIS data elements. This component of the system represents "true" client/server technology, where the client application transmits an SQL statement to the server, the server's SQL engine processes the statement, and only the results of the query are sent back to the client.

The actual student and course data reside on the University's administrative data processing machine, an IBM 3090. Behind the scenes and transparent to the user, a telnet terminal session logs into the host production database, using a user-supplied name and password, and subject to standard mainframe ACF2 transaction security. The telnet session then sends transaction strings to the host, which processes the transactions and returns screens of information to the client application. Access to this host data is accomplished through a Visual Basic code module that interfaces with EHLLAPI middleware. The middleware is implemented in the McGill University dynamic link library (DLL) and is accessed via function calls defined in a McGill-supplied Visual Basic code module.

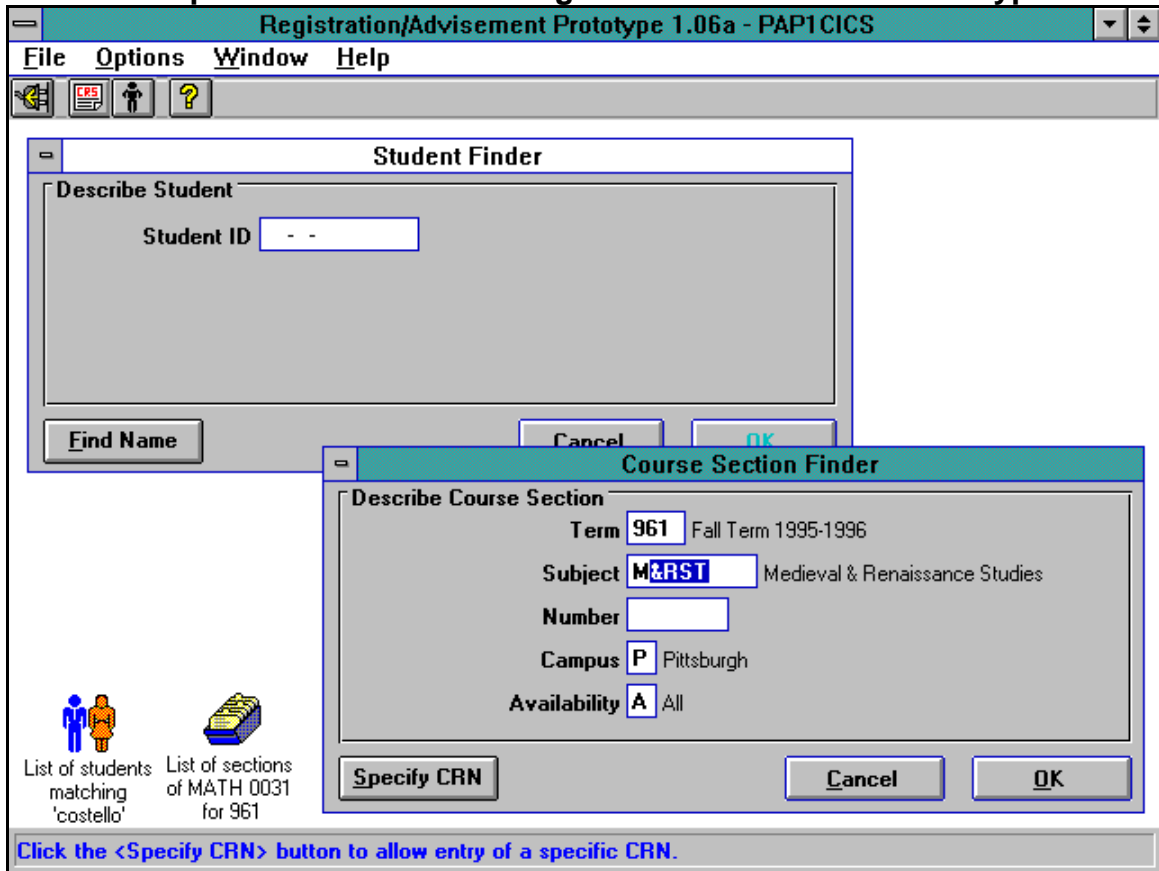
The Class Roster Prototype was a proof-of-concept prototype. It demonstrated a true client/server application that simultaneously accessed a file server, database server, and mainframe. It also illustrated the ability to access legacy system data through the front end CICS transactions.

Registration/Advisement Prototype

The success of the Class Roster Prototype prompted the Director of Student Registration and Financial Services to request that the Information Architecture Team initiate a larger project to develop an application that could be deployed in production mode across the University. The *Registration/Advisement Prototype* was intended to create an interface to ISIS that was consistent with the envisioned information architecture, and that would allow faculty advisors to access course and student information and to actually register their advisees into course sections in real time. As did the other prototypes, the Registration/Advisement Prototype was designed to illustrate the principles articulated in the architecture statement. However, instead of proof of concept, it would need to be a working application that would read from and write to the production student database.

The Registration/Advisement Prototype Project was initiated in May of 1994, as a joint venture between the School of Library and Information Science, who volunteered to test the new application, Student Registration and Financial Services, and Computing and Information Services. The Registration/Advisement Prototype used the same technology as the earlier Class Roster Prototype.

Sample Window from the Registration/Advisement Prototype



The sample window from the Registration/Advisement Prototype illustrates two instances of a finder pattern. At the top of the window is a student finder, prompting the user for a student ID. At the bottom, is a course section finder, which is in the mode of prompting for attributes to generate a browser. This mode is relevant when the user does not have the precise information needed to uniquely identify the desired object, in this case a term designator and a unique course reference number. In addition to the two finders, the prototype uses the following windows:

- Browsers: Course Inventory, Course Section, Open Course Section, Student, Advising Records
- Viewers: Course Section, Class Roster, Student (demographic, registration history, service restrictions), Advising Record, Registration, Academic History
- Update Windows: Student (demographic, registration history, service restrictions), Advising Record, Registration
- Utility Programs: Login/Logout, Change-Password, Download (download to file, send to printer, or copy to clipboard)

This set of windows, through their associated ISIS transactions, can provide access to all of the information that is relevant to an advising and registration session. The advisor can check the student's address information, ensure that the student is enrolled in the appropriate academic program, and ensure that there are no service restrictions (holds) associated with the student's record. This data can be corrected and updated as needed. In addition, the advisor can examine the student's class schedule for any term, as well as the academic record (transcript data) sorted by term or by subject. The advising component allows the advisor to view and create electronic notes and associate them with the student's records. These are stored in a local Microsoft Access database (at a school- or department-level). The advisor can access up-to-the-minute information about course sections, and, if a desired section is closed, even search for alternative

sections that do not conflict with the student's schedule. Finally, the advisor can register the student, and give the student a printed confirmation of registration.

The program developers employed an iterative development methodology based on principles associated with rapid application development. This methodology involved the successive repetition of the analysis, design, prototyping, testing, and review steps in close conjunction with the targeted end users. The application underwent seven upgrades using this methodology during its first year of operation. During that time, it was deployed to faculty and staff in the School of Library and Information Science, the School of Education, the School of Pharmacy, the School of Nursing, to the Director of the College of Arts and Sciences Advising Center, and to staff in the Student Registration and Financial Services area, for a total of 90 users. It is also being deployed at the Bradford, Greensburg, and Titusville regional campuses.

By all measures, the Registration/Advisement Prototype has been a resounding success. With the significantly improved access to course and student information, faculty feel much more efficient and effective in fulfilling their advising role. Students have been vocal in praising the responsiveness of the new process. They can do "one stop shopping" in their advisor's office, avoiding a trip to the central Office of the Registrar, and the inevitable wait in line. More importantly, they never have to return to their advisor's office because the course they were advised to take is closed.

The Registration Project

The Information Architecture and Process Innovation team recommended that the architecture be implemented through a project approach. Projects are proposed by design teams that are formed within the administrative and educational units of the University. These teams may be reengineering teams, or they may be smaller incremental improvement teams. The teams propose projects in accordance with detailed guidelines that ensure they will be aligned with the information architecture.

Based on the success of the Class Roster and Registration/Advisement Prototypes, and on the University's long-standing interest in implementing a telephone registration system, the **Registration Project** was initiated to investigate ways to improve the registration process.

This Registration Project followed the University's business process reengineering methodology, except for one fundamental point - it did not start from scratch, but rather assumed that the legacy student system (ISIS) would not be significantly changed. Lynch and Werner⁴ observed that the most profound element of reengineering methodology is reconceptualizing the fundamental nature of work. The Registration Design Team attempted to meet this goal by including a mix of personnel from several academic, administrative, and technical units. This was intended to foster the notion that such partnerships will produce a better design in a more cost effective manner than if any one of the team components attempted to implement the project alone. The design team concept took advantage of the expertise available across the University and permitted multiple views of the information system project.

The Design Team's recommendations call for further deployment of the Registration/Advisement Prototype, as well as the creation of new capabilities for the related processes of course scheduling and academic advisement. Implementation will be phased in over a three year period, culminating in student self-registration via client/server technology and telephone.

This Registration Project is one of six projects that were proposed as implementations of the UIS architecture. To date, four of the projects have been accepted for implementation: (1) the registration project, (2) implementation of the procurement process (reengineered during the

Information Architecture project), (3) reengineering of the human resources process, and (4) acquisition of key components of the infrastructure, such as the site license for the Oracle DBMS and development tools.

Summary

The Registration/Advisement Prototype had its roots as a proof-of-concept application to demonstrate the feasibility of client/server technology and the principles articulated in the University Information System architecture statement. It has evolved into a powerful application that promises to fill an important information need for faculty, staff and students.

Experience during its first year of development and implementation shows that legacy systems can be enhanced with relatively little effort to have the same look and feel as more user-friendly client/server GUI applications. Such enhancements enable legacy data to be available to the occasional user, and not just to the trained specialist. Additional efforts are currently underway to address the other serious problem with legacy systems, the ability to easily access legacy data on the back-end. Consequently, developers believe that legacy systems can be modified to become a viable component of the envisioned University Information System, at least until priorities allow them to be more completely addressed.

¹ Nicholas C. Laudato and Dennis J. DeSantis, "Reshaping the Enterprise: Building the Next Generation of Information Systems Through Information Architecture and Process Reengineering" *CAUSE/EFFECT*, Winter 1995

² Christopher Alexander, *The Oregon Experiment* (New York: Oxford University Press, 1975)

³ Christopher Alexander, *A Pattern Language* (New York: Oxford University Press, 1977)

⁴ Robert F. Lynch and Thomas J. Werner, *Reengineering Business Processes and People Systems*. (QualTeam, Inc., Littleton, CO. 1994)