Tool-Based Approach to Distributed Database Design:

Includes Web-Based Forms Design for Access to Academic Affairs Data

David A. Owens
System Engineer
Lockheed Martin Mission Systems
9970 Federal Drive
Colorado Springs, CO 80921
(719) 277-4896
dave.owens@lmco.com

Frederick T. Sheldon
Department of Computer Science
University of Colorado, Colorado Springs
1420 Austin Bluffs Parkway
Colorado Springs, CO 80918
(719) 262-3327
fsheldon@mail.uccs.edu

ABSTRACT

This paper describes a tool-based approach for designing and prototyping a distributed database application. This approach is demonstrated for an Academic Affairs Information System (AAIS) to assist the Webster University main campus and its 70+ remote sites in managing the information required to admit students, approve programs, schedule courses, assign faculty, register students, and generate the required queries and reports**.**

ORACLE® Relational Database Management (RDBMS) tools and products for Windows NT® were used to support AAIS requirements analysis, design, and prototype implementation. The Designer/2000® Process Modeler tool was used to document the top-level business functions, and the Data Modeler tool was used to develop a third normal form data model. The Developer/2000® Forms tool was used to prototype several user interface forms for main campus staff, remote staff, and students to enter and update student and program data. A Web Server was also installed, along with the Java software and AppletViewer, to test the prototype forms from a Web Browser.

Keywords

Distributed, Database, ORACLE®, Web, and Academic Affairs.

1. INTRODUCTION

This paper proposes an automated Academic Affairs Information System (AAIS) to allow a more efficient exchange of information between the Webster University main campus and its remote sites. Webster University is composed of a main campus in St. Louis, MO., and over 70 remote sites in the U.S., Europe, and Asia. [10] The Webster Academic Affairs group is responsible for student admissions, student registration, course scheduling, and faculty assignments. Webster currently maintains Academic Affairs information on a mini-computer at the main campus.

The remote sites mail or fax local student and faculty information to the main campus, and the information is manually entered into the main campus computer. Some of the remote sites also enter the same information into a Dbase III+ application to support local operations. The upgraded AAIS will automate the current manual process for information distribution. It will be used by the main campus and remote sites to admit students, schedule courses, assign faculty, register students, and generate required queries and reports.

1.1 Known Solutions

One option is to expand the mini-computer at the main campus to a central database server of sufficient power and connectivity to service all of the remote sites. While a central mainframe server is a simple, effective design, there are several shortfalls with this approach. For example, hardware maintenance costs, network traffic, development costs, scalability, availability, and responsiveness to remote site needs are significant disadvantages of a centralized approach.

Another option is to use network computers (NCs) at the remote sites to access high-end servers at the main campus via the Internet. This option is similar to the first option in that application software and data would be maintained at the main campus server. The remote site NCs typically use a compact operating system that can be booted from the server, and downloads Java applications from the server. The operating system hosts a Web browser, which launches the Java Virtual Machine when a Java application is downloaded.

A NC approach may lower the cost of ownership, because all of the software is controlled and implemented by the main campus. However, this approach is less flexible in responding to new requirements, or customization requested by the remote sites. Additionally, Java is an interpreted language, which runs slower than compiled programs. Performance of this approach is a concern, as well as network traffic and availability.

A distributed client/server architecture is a third option. With this approach, a small-to-medium size database server could be used at each remote site to provide data services to multiple local clients, to include local student home computer clients. Servers of this class are currently resident at most of the remote sites, and there are plans to network these servers together. Once the data servers are networked together, the distribution of Academic Affairs data between the main campus and the remote site servers could occur.

Preprint: ACM Proceedings Symposium on Applied Computing San Antonio TX, Feb. 28 – Mar. 2, 1999

This approach should provide good scalability, availability, and performance at the remote sites without upgrading to high-end servers. The amount of long-haul network traffic would also be less than the other two options. This approach is also more flexible in providing standardized software to manage Academic Affairs data, while being responsive to the customization needs of the remote sites. However, a significant design issue with this approach is maintaining data synchronization between the main campus and the remote sites. This approach may also be more expensive because of local system administration and COTS license requirements.

The factors that were considered most significant in recommending a distributed client/server database approach for the AAIS included:

- The desire to place frequently referenced data close to client applications that need to reference it, thus minimizing network or server load and access time.
- A reduction in the impact of a single point of failure such as a central server going down.

1.2 Project Scope

A tool-based software system engineering process consistent with the first cycle in the Boehm Spiral Development Model was used to analyze, design, and implement a prototype distributed database design capable of supporting AAIS functions at the main campus and the 70+ remote operating locations. [8] The prototype design allowed assessment of the risks of using a distributed database, as well as refinement and clarification of Academic Affairs operational requirements. A top-level system design for the prototype is shown at Figure 1.

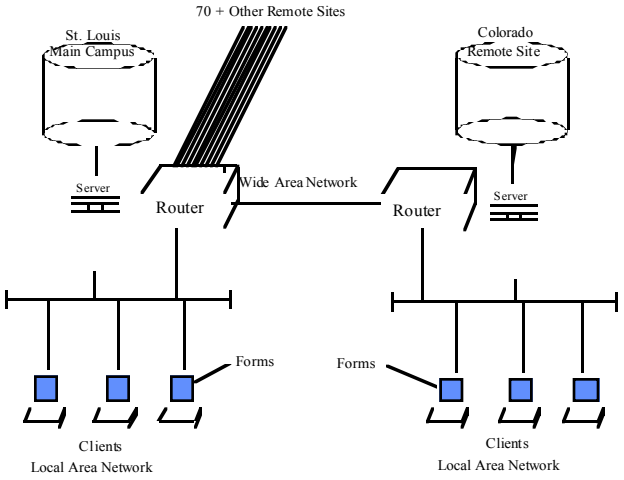

Figure 1. Top-Level System Architecture

2. ANALYSIS PHASE

Designer/2000 tools were used to model the AAIS top-level functions and business processes for the main campus and the remote sites. The following is a list of top-level AAIS functions:

Remote Site Staff Functions

- Advise/Assist Students
- Schedule Classes
- Assign Faculty

Student Functions

- Apply for Admissions
- Register for Courses
- Complete Program Plan
- Pay Admissions/Registration Fees

Main Campus Functions

- Process/Approve Admissions
- Review/Approve Program Plans
- Review/Approve Course Schedules
- Review/Approve Faculty Assignments
- Collect/Process Student Payments
- Pay Faculty

The Designer/2000 Process Modeler supported the analysis and documentation of AAIS functional requirements. It was used to develop a functional model containing the AAIS business functions and activities, to include the inputs, outputs, flow of work, timelines, and organizational units performing the activities, performed at the main campus and remote sites.

Another Designer/2000 tool used during the analysis phase was the Data Modeler. This tool was used to create the initial AAIS data model containing the data entities, attributes, and relationships necessary to perform the functions and activities. The initial data model was a third normal form (3NF) model with the following characteristics:

- All repeating data attributes were removed and placed in a new (related) table.
- All non-key data attributes were dependent only on the primary key, and not some other non-key attribute.

The Data Modeler tool was also used to design and document many of the AAIS business rules using primary and foreign keys, datatypes, check constraints, and indexes. The AAIS data models generated with this tool were not included in the paper due to limited space. [3, 7, 9]

3. DESIGN PHASE

Using the initial data model developed during the analysis phase, a physical data model was produced using the Designer/2000 Server Generator tool for implementation on an ORACLE® 7.3.3 Relational Database Management System (RDBMS) for Windows NT®. The physical data model included the table (or entity) definitions, column (or attribute) definitions, and the relationships between the entities. [1, 3, 7]

The initial 3NF model was iteratively refined, and in some cases the model was denormalized for performance reasons. For example, the Major table was denormalized and major codes were included in the Program table. This was done because a Webster program allows no more than two majors, and generally there is only one major. By denormalizing major codes, queries for information about majors associated with Webster programs were simpler to develop and executed more efficiently on the RDBMS.

3.1 Data Distribution Design

The data distribution design was based on the following questions and answers:

What are the data availability requirements? Should all of the data be available at all locations, or only some of the data at some of the locations?

- Not all of the data in the Academic Affairs database needs to be available at all sites.
- The main campus needs access to the data at all of the remote sites, but individual remote sites do not need to access data from another site.

How often is the data accessed, and is it important to have the current value?

- Since the main campus is located in Saint Louis, MO, and the remote sites are located throughout the United States, Europe, and Asia, Academic Affairs data is being accessed on a daily, sometimes hourly, basis.
- It is important for the data shared between the main campus and remote sites to be current in order to meet student educational needs and Academic Affairs business rules.

Which locations have the authority to change the data? Can a location change all the data, or only some of the data, and how often is the data changed?

- Most of the volatile data, such as student/employee information, program plans, class schedules, and registrations, is owned (and changed continuously) by the remote sites.
- The main campus reviews and approves volatile data entered at the remote sites, and provides frequent (e.g., weekly or monthly) updates to some of the volatile data, such as student admissions status, student billing, and instructor pay.
- The main campus owns and maintains most of the non-volatile data, such as degrees, majors, courses, and curriculums. This data is changed less frequently, e.g., semiannually.

Based on the answers to the questions on data availability, access, and volatility, the following 7.3.3 RDBMS products and features were investigated to meet the AAIS needs: symmetric replication, snapshots, remote copy, and custom triggers/stored procedures. [1]

The symmetric replication feature is a relatively new capability. This feature is primarily applicable where there is a need to maintain updatable copies of volatile data at multiple sites, and both data availability and integrity are of primary importance. With symmetric replication, a change to any of the replicated data tables at any of the sites is propagated to the tables at all of the other sites. The time it takes for a change to propagate depends on various parameters set by the Database Administrator (DBA), and also on the availability of the databases to which the propagation is taking place. When there is a large number of sites/tables, the impact on both network load and server load is likely to be severe for any significant update volume. Since the AAIS does not need to replicate all volatile data to all remote sites, and the number of sites that need to share data is large, symmetric replication was not recommended.

Snapshots can be used to copy a table (or tables) at a point in time from a remote database into a local database. Snapshots are defined using a query on a single table (simple snapshot) or a set of tables (complex snapshot), and a time interval in which the query is to be refreshed. At first glance, snapshots appear useful for distributing non-volatile data (e.g., degrees, majors, courses, and curriculums) from the main campus to all of the remote sites for read-only queries. The main problems with snapshots revolve around their set-up, administration, and efficiency. If the main campus server can sustain the load to record all the changes made to the snapshot tables for the 70+ remote sites, then simple snapshots could be used to distribute the read-only data.

However, the main campus non-volatile data tends to remain stable for long periods of time. Therefore, it is probably better to wait until all the degree, major, and curriculum changes are made at the main campus, and then manually update the remote site tables (using a remote copy). It was beyond the scope of this project to implement either snapshots or a remote copy in the prototype implementation. However, based on an analysis of their capabilities compared to AAIS needs, a remote copy is recommended for distributing non-volatile, read-only data.

The AAIS data distribution requirements for volatile data are based on special rules (or conditions) that are not directly supported by either symmetric replication or snapshots. Therefore, custom PL/SQL triggers/stored procedures were developed. The prototype implementation included a set of custom triggers to distribute student, student admissions, and program plan data between a remote site and the main campus. [1, 2, 7]

3.2 Data Integrity

A simple design was implemented to maintain data integrity if data distribution fails due to a problem in wide area network communications between a remote site and the main campus. This design used exception-handling triggers to write the table name, primary keys, and type of transaction (i.e., insert, update, and delete) to a distribution log table. This design would allow a remote site or the main campus to continue to operate even if the wide area network connecting them was down. Once the network was restored, a site could review the distribution log for unprocessed transactions, and procedurally synchronize the databases. It was beyond the scope of this project to develop code to automate the database synchronization process. This is one area of the prototype AAIS that needs additional refinement prior to fielding. [7].

3.3 Security Design

There are two areas of security that were considered important for the AAIS design:

- Access security to control who can access AAIS applications (e.g., Forms), and how much of its functionality is available once accessed.
- Data security to control which tables a user can reference, and what permissions/ privileges they have to data in the tables (e.g., read, read/write).

The 7.3.3 RDBMS has a variety of mechanisms for identifying and verifying users. The simplest is to require a user to enter a valid name and password prior to accessing a database application. Additionally, object permission features, such as create role, create user, and grants are available to control access to database objects, such as tables (and columns), views, functions, sequences, etc. Also, form controls and custom triggers can be developed to restrict access to specific data in the tables and columns. [1, 2]

The use of object permissions, user name/password, form controls, and custom triggers were included in the AAIS

database security design. During the analysis phase, five separate user roles were identified: developer, main campus staff, remote site staff, students, and faculty. As part of the security design, each role is granted database object and system privileges based on the AAIS business rules for database access. Specific user accounts and passwords are then created, and granted the roles (and privileges) needed to access the forms, tables, views, and other database objects. When logging into AAIS, all users are required to enter a valid user name and password in order to access the database object.

Additionally, form access controls and custom triggers were designed and prototyped to provide another layer of security for student access. This additional level of security is used to prevent a student from accessing personal data, such as the social security number of another student.

3.4 Forms Design

The primary purpose of the AAIS prototype forms was to limit the risk in the following areas:

- Understanding the AAIS requirements through early feedback from users on the functionality, data distribution, and security design.
- Determining the performance, feasibility, and utility of the prototype tools for the final implementation of the AAIS requirements.

The Developer/2000 Forms Designer was used to prototype several forms for main campus staff, remote staff, and students to access the Webster Academic Affairs database. The current Webster University Application for Admissions and Program Plan paper-based forms were the basis for the design. Additional forms, pop-up dialog boxes, and windows were also designed to maintain and control access to student and program data. Examples are shown in Figures 2 and 3. The forms prototyped for student use were also converted to Developer/2000 Java applets so they could be run from a Web browser. [4, 5, 6, 7]

The prototype forms were designed with a standard look-and-feel for displaying and manipulating data by main and remote campus staff and students. The user-interface was kept basic and unobtrusive, and much of the form prototyping effort was spent building controls and trigger code to enforce Webster data security, integrity, and distribution rules.

4. CONCLUSIONS

The prototype implementation is a reasonable model for the final AAIS. The Student Admissions, Program Plan, and Program maintenance forms are sufficiently mature to elicit feedback from the main campus and remote site staff on how a distributed application would work in practice. The prototype forms could be used to validate and refine their requirements for transferring student and program information among students, main campus, and remote site staff.

The ORACLE® tools and products used for the AAIS project provided excellent support. For the most part, these tools appear to be robust enough to support design, development, and implementation of a final system.

Designer/2000 contains a powerful set of tools that can be used at different stages of a database system's life cycle. All of the tools available in Designer/2000 were not used for the prototype application, because of tool complexity and the time available to learn to use the tools. However, the Process Modeler was very useful in modeling the AAIS top-level business functions. The Data Modeler was also satisfactory for designing and documenting the AAIS data models and business rules.

On the other hand, the Server Generator tool used to translate the data model into database objects was somewhat cumbersome to use. In some instances, the DDL generated had to be manually augmented to build the 7.3.3 RDBMS server objects needed by the prototype system.

Developer/2000 includes a powerful, fairly intuitive tool for developing forms applications. By adding custom triggers to the prototype forms, the AAIS data distribution rules were implemented in response to various user interface events (e.g., record-inserted, record-updated, button-pressed, etc.). A few problems were encountered in installing and configuring the ORACLE® Web Server, Java software and AppletViewer. Once properly configured, the prototype forms would launch and run from either the AppletViewer or Microsoft Internet Explorer®. However, these problems prevented extensive experimentation with the performance and security characteristics of the Web-based forms.

5. FUTURE WORK

Some additional analysis, design, and testing is required to evaluate the prototype's performance over a wide area network, as well as handle exception conditions, such as a failure of the wide area network between the main campus and the remote sites. The prototype design and implementation in this area is still relatively immature.

An upgrade to Developer/2000 Forms and a later version of the Web Server (v3) is needed to complete the functionality and performance evaluation of the prototype Java applets over the Web. This is the highest priority future prototyping effort, because forms that can be launched as applets from a Web Browser are applicable to either a NC or a distributed, client-server architecture.

6. REFERENCES

[1] Ensor, Dave and Stevenson, Jan, Oracle Design, O'Reilly & Associates, Inc., 1997.

[2] Koch, George and Loney, Kevin, ORACLE: The Complete Reference, Osborne McGraw-Hill, 1997.

[3] Lulushi, Albert, Inside ORACLE DESIGNER/2000, Prentice Hall PTR., 1998.

[4] Lulushi, Albert, Developing ORACLE FORMS Applications, Prentice Hall PTR., 1996.

[5] Muller, Robert J., ORACLE Developer/2000 Handbook, 2nd Edition, Osborne McGraw-Hill, 1997.

[6] Papaj, Robert and Burleson, Donald, Oracle Databases on the Web, Coriolis Group, Inc., 1997.

[7] Owens, David A., Distributed Database Application Project Report, University of Colorado, Colorado Springs, Master of Engineering Software System Engineering, 1998.

[8] Sommerville, Ian, Software Engineering, 5th Edition, Osborne McGraw-Hill, 1996.

[9] Watson, Richard T., Data Management, John Wiley & Sons, Inc., 1996.

[10] Webster University 1997-1998 Graduate Studies Catalog, Webster University.

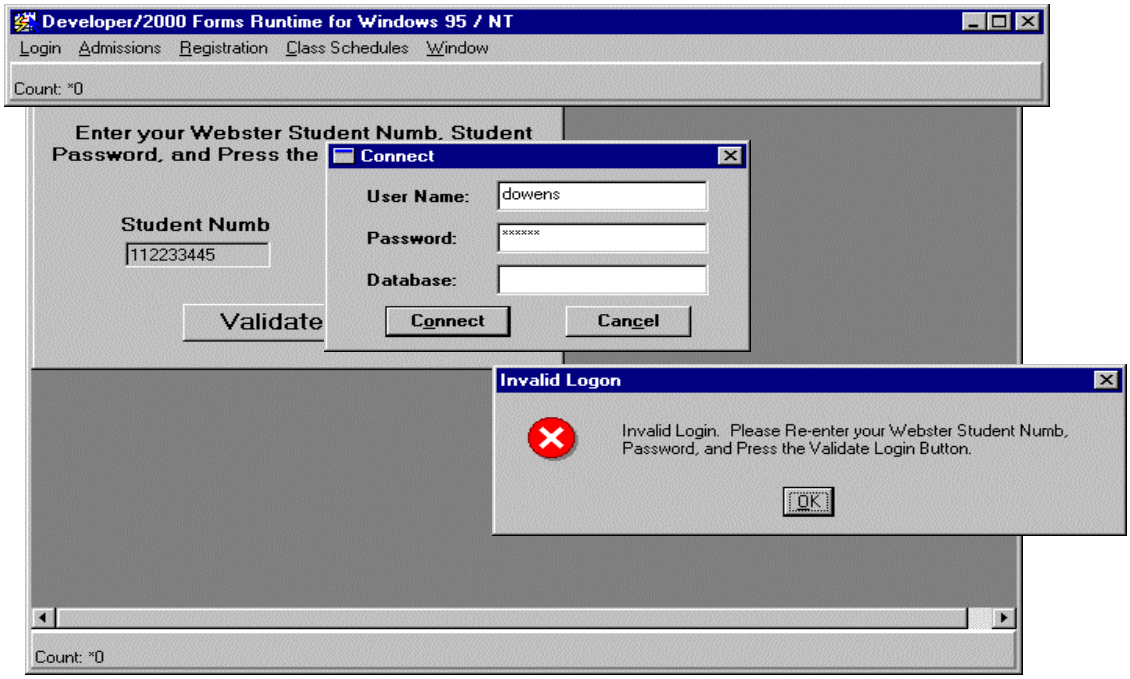

Figure 2. Prototype Login Forms

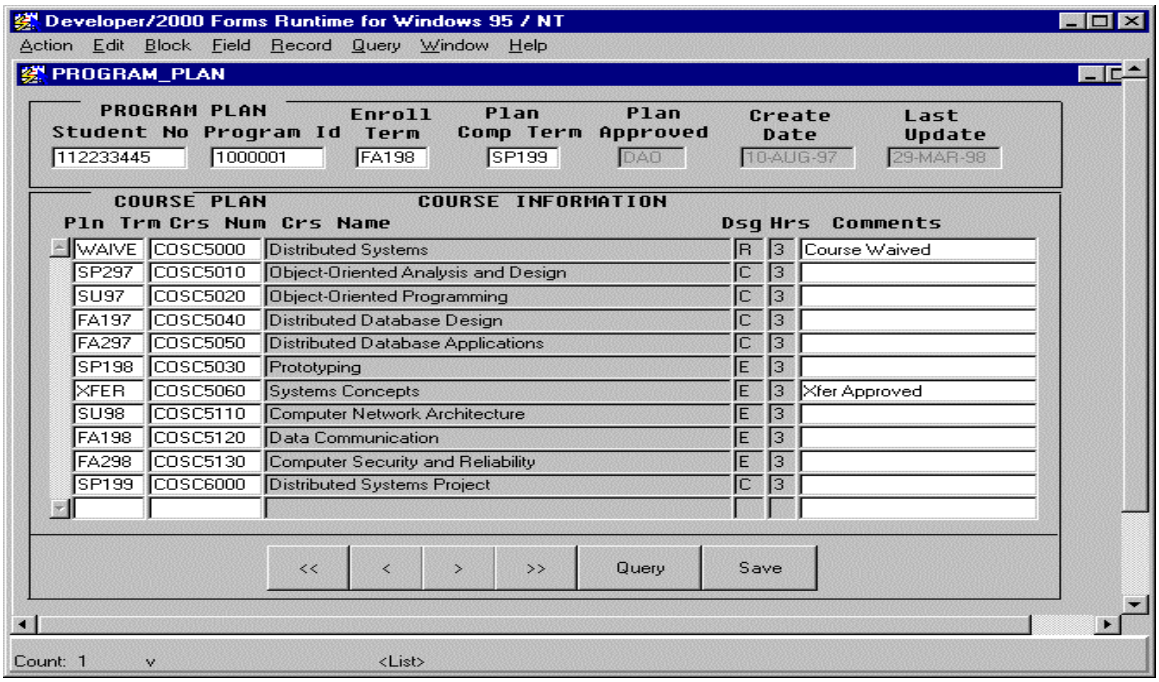

Figure 3. Prototype Program Plan Form