A Framework For Implementing Business Rules In Multi-tier Accounting Information Systems

Randy Weinberg, (E-mail: Randy2+@andrew.cmu.edu), Carnegie Mellon University
Jim Q. Chen, (E-mail: Jchen@St. CloudState.edu), St. Cloud State University
John J. Cheh, (E-mail: cheh@uakron.edu), University of Akron

Abstract

This paper discusses design and implementation of business control rules in a multi-tiered, transaction oriented Accounting Information System (AIS). Business rules describe an organization's policies and business practices. Implementing business rules as a component of an AIS provides many benefits: maintainability, centralization of business rules and controls, and simplification of system design and implementation. At its most basic level, a three tiered application consists of a thin user interface, a business services layer, and a database layer. The business services layer contains business rules and application controls. The database layer implements the database, various field level controls and referential integrity.

Introduction

Developing and maintaining high quality client/server transaction oriented information systems has always been an expensive and difficult activity. Best system development methodologies take into account various non-functional factors including design time, system performance, quality, maintainability, portability, and reuse. Frequently, however, current development practices and processes do not successfully address these issues. This is in part due to the overall design paradigms used to implement today's complex information systems.

Many developers have implemented database/client-server systems in two parts: a user interface residing on a user's workstation and a database residing on a server system. The user interface provides the interaction with the user and enables a user to enter, edit, and retrieve application data. It contains application-specific objects and logic including forms, reports, and queries, database programming logic, error checking, application level controls, and enforcement of business rules. The database component provides the logical and physical implementation of the application's data. It provides a low level data interface, usually implemented via Structured Query Language (SQL) and is capable of enforcing various levels of security and data integrity through the use of validation rules, stored procedures, and triggers.

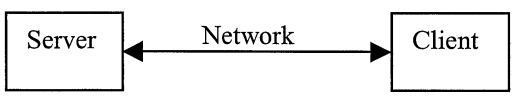
While this implementation separates the underlying database structure and implementation from its applications, the two are, in practice, tightly coupled. Changes to one component can necessitate costly changes to the other. Op-

Readers with comments or questions are encouraged to contact the authors via e-mail.

portunities for reuse of code or design are limited and frequently, the same controls and logic must be implemented in multiple forms, queries, and reports. Further, since all data access is done programmatically, the underlying data model and access methods must be built into the user interface level objects. User interface developers must also be aware of optimizing techniques for SQL performance and data navigation. Further, business policies must be articulated and implemented, with complex logic frequently split between the two layers. This increases the skill level required by all developers. As the underlying data architecture evolves, these embedded techniques may no longer prove adequate. In fact, many existing applications cannot be easily modified to reflect changes in business policies or rules or the underlying database design or technology. Further, any changes to the application controls or business rules must be propagated throughout all objects referencing the database. Modifications are costly and time consuming because the database transport code is spread out throughout the application-specific code. To realize greater efficiencies and potential for component reusability, a multi-tiered approach using a business rules component can be of great benefit to an organization.

In the following sections, a brief review of client-server model is provided and business rules and their implementations in database applications are discussed. A three tiered client-server model for transaction oriented database systems design is then proposed. Finally, the paper discusses some of the benefits of the new model and the future research opportunities.

Figure 1
A Basic Client/Server Computing Model



Client Server Computing And Business Rules

Client/server is a computing model for distribution of functions between two types of independent and autonomous processes: servers and clients. A client is any end user application that requests specific services from server processes. A server is a process that provides requested services for clients (Rob and Coronel, 1997). In a common business scenario, several client computers are connected to one or more servers in a network. A client can request services from servers without regard to the location or the physical characteristics of the computer (see Figure 1).

Client/server architecture has become a popular model for modern business data processing. A typical client/server application consists of a graphical user interface on the client end, an SQL compliant database on the server end, and business rules distributed between the client and the server. Various application development tools have been developed to automate the design processes of both client and server side operations, including as the creation of GUI and SQL queries. However, most of these rapid prototyping tools provide little support in formulating and implementing business rules (Baum, 1995). The business rules and database processing logic, the core of sophisticated applications, often demand the most effort in system development.

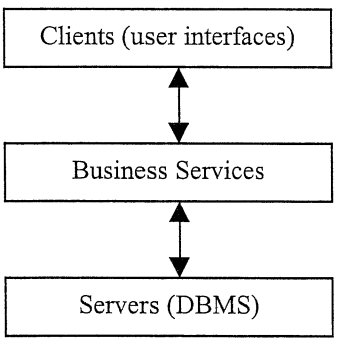
Business rules are statements that define or constrain some aspect of the business; they provide substantial value to an application. Business rules may reflect accounting controls in an e-commerce application, for instance, or they might reflect certain business policies. For example, a business rule in a truck rental agency might be: "A rental truck with accumulated mileage greater than 6000 since its last service must be scheduled for service." Some of the rules are clearly stated, while others are implicit.

Business rules are derived from organization's policies, procedures, and standards. As such, business rules may require frequent updating to reflect dynamic business environments. Unfortunately, existing tools and techniques make this difficult since the rules are buried deep in procedural code or as complex database commands in SQL.

A Three-Tier Architecture

Realizing a three-tiered application model requires a "thin" client application, an intermediate "business rules" layer, and a database layer implementing the actual database (see Figure 2). The client application provides end user interface services and allows for entering, editing, and displaying of data. The intermediate layer is a collection of services for data access control, data validation and manipulation, and enforcement of business rules. The database server provides "data services." Logic is divided among these various logical layers. If properly designed, this type of design can simplify application development and reduce the duplication of data access specific code, controls, and validation rules.

Figure 2
A Three Tiered Client/Server Model



The User Interface

The user interface layer responds to user interaction. It is responsible for enabling the user to enter, edit, retrieve, save, and display data. Reports, queries, forms and other user-visible application specific objects are defined and maintained in the user interface. In a three tiered design, the user interface contains no database specific code or references.

Objects in the user interface respond to user activities via event handlers. Through event handlers, various application specific controls and general data validation can be done prior to database updates. Specifically, user interface controls typically check for data typing, reasonableness of data, limits, validity of data, invalid combinations of items, and self checking digit algorithms (Rittenberg and Schwieger, 1997; Summers, 1991).

The user interface layer makes requests to the business services layer to connect to the database and to access records in the database. The various functions, objects, and services defined in the business services layer are visible to the user interface through predefined function calls or object methods. Since the user interface level contains largely application specific objects, it is almost entirely non-reusable.

Business Services Layer

The business services layer acts as an interface between the application front end and the actual database server. Conceptually, the primary objective of this layer is to eliminate dependence of client-side applications on underlying database implementation. This layer is responsible for all communications between an application and its external data sources. It translates all requests for database services into the specific operations required by the underlying database. It is responsible for loading, sorting, validating, and formatting data, as well as enforcing business rules. The business services layer is called on to populate the controls in the user interface forms.

Various objects or modules are imple-

mented in a business services layer to maintain access to the database and to enforce common application controls and business rules. Public methods are provided so that interface developers can activate the procedures necessary to invoke these services. Specifically, the business services layer performs two main functions:

First, it acts as a transport utility, moving data between the front-end application and the data server. It centralizes data access and common code for fetching and saving data. Specifics of data access objects and methods are encapsulated in the transport services. Thus, database objects like recordsets, snapshots, SQL commands, and ODBC handles are all encapsulated in the business services layer. Examples of functions implemented in this layer, yet available to the interface designers, could include operations such as register customer, delete customer, and various database queries. This is largely application specific code, mostly non-reusable outside the application's boundaries, yet reusable by all forms or user interface objects that must perform these functions.

Second it enforces business rules and provides validation services, data manipulation. These routines are tied to the database but may be common to applications that use it. Business rules describe policies, procedures, or principles within a specific business environment. They can be enforced by procedures or triggers. For example, the rule, "a training session cannot be scheduled for less than ten people or for more than thirty people," can be enforced by a procedure that checks the enrollment figure before the scheduling of the session is allowed. Business rules are in addition to rules for maintaining the domain and referential integrity of tables in a database. Typical business rules implemented could include operations such as validation of customer accounts and credit limits, checks on legitimacy of transactions, and review of past transactions.

Reuse is enabled when common routines are centralized in a business services layer. Needed services can be invoked from objects in the user interface as needed. This approach offers the following potential benefits:

Database abstraction: Changes to the underlying database access methods, database design and implementation can be made without the need to change user objects like forms, reports, data entry screens, and queries.

Increased modularization: Different personnel can work on different components of the overall application.

Reuse: Common routines are centralized and reusable.

The Database Layer

The database layer provides data access services to the business layer. It also implements built-in field level validation and rules, and user access controls. The database layer retrieves data, saves data, and modifies data as requested by the business services layer. All returned data is passed back to the business layer for further processing and presentation to the user interface layer. Any errors in validation or data access are returned to the intermediate layer for handling.

These controls are implemented through the use of stored procedures, triggers, and validation rules that are stored with the database.

Triggers: A trigger is a procedure executed by the database management system whenever the business services layer requests to modify the data in a specific table. Triggers usually occur prior to the execution of Insert, Delete, or Update statements so that the effect of the statement on referential integrity can be examined. In a customer order system, for instance, when an order is placed, a trigger could be executed to see if the customer exists and has an available line of credit.

Rules: A rule controls what an application can or can not enter into the column of a table. This allows the database administrator to control the data in the database. For example, a rule could be written to prevent duplicate keys or invalid information. Rules for maintaining relationships between tables and enforcing referential integrity can be defined.

Stored Procedures: Stored procedures are programs using SQL statements, control statements, and variables to manipulate data on the server. They are executed on the data server with the resultant data items, messages and errors being returned to the business services layer.

Conclusion

In a multi-tiered approach to client/server application development, various tiers, or layers, are implemented. Typically there are layers for the user interface, an intermediate layer to provide enforcement of business rules and data access services, and a database layer. Implementing business rules in an intermediate layer can help to centralize and segregate the business logic from the user interface and the database layers. A key advantage to this approach is that developers and business analysts do not have to encode complex business processing and rule enforcement in the user interface layer or create and maintain complex database triggers and stored procedures.

The user interface layer interacts with the end user. It allows for data entry, edit, and display as well as report generation and running queries. The user interface layer is highly application specific; it contains code specifically written for forms, reports, and other user objects.

The intermediate layers encapsulate all data access methods and provide services to the user interface layer. These services act as intermediary between the user interface layer and the network or database. These layers implement database specific procedures for linking to a database, manipulating the database, and implementing business rules. Services in this layer are available to the user interface via public function, or procedure calls. From the point of view of the user interface, all access specific logic is embedded, or hidden in this layer.

The database layer implements the logical and physical view of the data and provides standardized data access mechanisms to the data access layer. Various stored procedures, trigger, and validation rules are implemented in the database to provide low level field and intra-record validation.

Suggestions For Future Research

Future research should investigate how to make specification of business rules a part of conceptual modeling. Various software engineering and reengineering tools are appearing in the marketplace to assist developers in definition of their business rules. Evaluation of these tools in live practice is a natural next step for researchers interested in improving the quality of client/server application design and implementation. Dynamic business rules derived from data mining operations and artificial intelligence techniques may be on the horizon.

References

1. Baum, David, "The Right Tools for Coding Business Rules," *Datamation*, Vol. 41, No. 4, pp. 36, 1995.
2. Rob, Peter, and Carlos Coronel, *Database Systems: Design, Implementation, and Management*, 3rd Edition, Course Technology, 1997.
3. Rittenberg, Larry, and Bradley J. Schwieger, *Auditing Concepts for a Changing Environment*, 2nd Edition, The Dryden Press, 1997.
4. Summers, Edward L., *Accounting Information Systems*, 2nd Edition, Houghton Mifflin Company, 1991.