A Business Rules Framework

by Terry L. Wiechmann

Abstract

Fundamental to transforming legacy M applications to new technology is the transformation of business rules. Business rules constitute years of investment and represent the content of any business. This paper has been written within the spirit of interoperability and assumes that organizations are looking for ways of transforming their application systems to new technology. It proposes a framework for extraction, organization and distribution of business rules. It assumes a client server environment where the rules are implemented on the appropriate tiers, independent of any particular technology. Additionally, it assumes the framework foundation to be object oriented. Although the M language does not contain approved object oriented extensions, there are two object oriented M implementations in use today that qualify as targets for business rule distribution.

Organizational Goals

There are certain goals that most organizations want to accomplish once they decide to transform their applications to object technology. They are:

- Reuse as much of the underlying infrastructure investment as possible.
- Save the data definition, its storage structure and the application business rules within an object repository for the purpose of re-evaluation and future extension.
- Move to modern technology such as object technology and client server.
- Extend the application using the new technology.

Historical Overview

Most M applications consist of code that is tightly bound to the database. Business rules are largely embedded in the code. Some rules are embedded in a data dictionary if it exists. This tightly bound approach to creating applications has historical significance in that it offered a low price/performance ratio when computers were much slower and more expensive than they are now. However, tightly bound systems tend to be inflexible whereas loosely bound systems are more flexible. Within the last 30 years, computer hardware has increased tremendously in speed and has become comparatively economical. Technology has allowed application development to evolve from the tightly bound approach to a loosely bound, distributed approach. In addition to this trend, object orientation has added an organizing paradigm that is highly compatible with the distributed trend.

Moving from a tightly bound, inflexible legacy application to a loosely bound, flexible, distributed object oriented application is largely a matter of:

- Building an object repository to hold the legacy data definitions, data structure, business rules and object model, allowing these components to be easily modified and extended to represent the current business model.
- Using runtime generators to distribute the business rules as executable code across a distributed client server environment. This means to other language environments.
- Supporting various database systems.

The following sections describe how distributing the business rules can distribute the processing with the consequence of specializing the M environment as a database, and more importantly, permit the integration of other database technologies in a homogeneous fashion.

Business Rules Organization

One of the most important goals of migrating M applications to a distributed object oriented environment is to distribute the business rules as executable code onto the appropriate tiers, primarily to achieve optimum performance. To accomplish this goal, business rules must first be categorized so that they can be applied in a flexible, scalable fashion.

This process of categorizing business rules involves the following steps:

- Describing the concept of a business rule and its organization
- Describing the Model-View-Controller concept
- Showing how these two concepts can be combined to create a business rules framework

Business Rules Concept

Business rules are derived from a business requirement. Any business requirement can be decomposed into one or more business rules. A business rule is a constraint that is applied to a particular business situation. It can be represented as a conditional statement in the form:

IF Constraint THEN Action

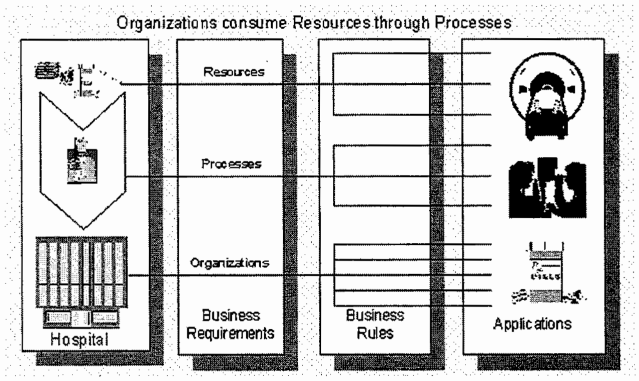
where Constraint is a statement of fact about the business requirement and Action is the action to be taken or not taken based upon the truth value of Constraint.

For example, a business requirement may be:
All prescriptions that are filled must clearly display any warnings on the container.

Many business rules are associated with this one requirement. An example of one business rule produced by this requirement is:
IF the prescription is the drug Claritin™, THEN the following warning labels must be clearly displayed on the container: Take medication on an EMPTY STOMACH 1 hour before or 2 to 3 hours after a meal unless otherwise directed by your doctor. May cause DROWSINESS. ALCOHOL may INTENSIFY this effect. Use care when operating a car or dangerous machinery.

Organization of Business Rules

The diagram below illustrates the basic premise behind any business organization. That is:



Business Rules Logical Distribution

Organizations consume Resources through Processes

Within the healthcare industry for example, hospitals consume resources such as pharmaceuticals, expendables (ultimately money) through, and as a consequence of, processes that occur daily such as surgery, exams, etc. Applications that are built to model and support the activities within any business environment must enforce business requirements. This is accomplished through business rules.

From a business perspective, this approach to organizing business requirements, and consequently business rules, is a common approach.

Model-View-Controller

The Model-View-Controller (MVC) concept evolved out of the Smalltalk language. It is typically explained using Graphical User Interface (GUI) applications. However, the concept is general and can be applied to any application. It is a design pattern.

The purpose of the MVC is to separate the application object (model) from the way it is presented to the user (view) from the way the user controls it (controller). User in this case may be any user of the model. The MVC provides a way to de-couple these objects, offering much more flexibility and possible re-use.

The model knows about all the data needed by the views. The model's protocol, its interface, contains all the methods and properties needed to access the data.

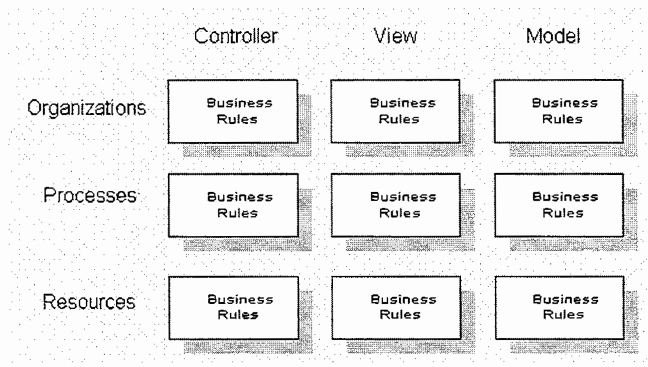
The view accesses the model. It presents the model as a particular organization. Multiple views can exist for any model. They use the methods (or properties) of the model to manipulate the data. The model has no knowledge of the views that are accessing it.

Associated with the model is a controller. The controller acts as an intermediary between the model and the views. Any modification to the model is communicated to the views via controller. The controller object within an application often contains the callback code for event processing. For example, if a model side data element is changed, generally an event will be fired, signaling all processes watching for that event to execute the callback. The callback code executed will synchronize the view with the model.

The MVC provides a common approach to organizing computer system applications. Ignoring this common pattern can lead to tightly bound applications that are hard to extend. Reusability becomes extremely difficult, if not impossible.

Business Rules Framework

As a part of ESI's Transformation Strategy, extracting business rules from legacy M code has forced us to do a great deal of bleeding edge, esoteric work. After understanding the general concepts surrounding business rules, the next challenge is to create an organizational framework that:



Business Rules Framework

1. Organizes the rules within a business context.
2. Provides a paradigm for mapping those rules into a client server environment.

To create a framework for business rules that are implemented within a computerized application, the business organization and MVC concepts can be combined to form a matrix as shown above.

This framework provides a convenient way to map business requirements and rules into a computer application that offers reusability, scalability and extensibility. It partitions the business rules from a business and a systems perspective.

Business Rules Extraction and Exposure

Many of us at ESI have been around long enough to actually witness the evolution of M applications from the tightly bound, no data dictionary days, through the data dictionary and structured programming days, to the present days of loosely bound, object oriented, client server based application systems. The current evolutionary step has proven more difficult for many organizations. There are many reasons for this, which are beyond the scope of this paper. However, one cause has been the increased competition within the business arena. In order to compete within today's business atmosphere, organizations must be able to respond to the needs of their customers. This translates to application systems that must be extensible and highly reusable. Those aspects of a business that are at the heart of the business such as data definitions, business rules, workflow rules, and GUI layouts should be readily available to domain experts. Having them buried in a mountain of application code that requires a team of programmers to decipher is no longer an option.

The framework outlined in the previous section is the basis for the organization and mapping of business rules. This concept is central to ESI's Application Repository Tools (ART). The ART tool set was built using our EsiObjects development and runtime tools. The ART tools create and maintain an Object Repository that holds the data definitions for each data element in an application, the associated business rules as well as all mapping information needed to generate an object oriented runtime environment. The Object Repository resides within the development environment. Runtime environments are generated using the repository as a source of definitional information. The repository is not a runtime component.

The challenge has been to map this logical concept into the object repository such that they can be:

1. Modified and extended by application domain experts.
2. Used by a runtime code generator to generate the rules into executable EsiObjects, Java or C++ code and distribute them to the proper client server tier for maximal performance.

Conclusion

Over the years, M applications have evolved in size and sophistication. Application business rules have become embedded in procedural code structures. The tightly bound, procedural approach to building applications has come in conflict with modern distributed computing realities. Because of the nature of these systems, concepts like reusability and extensibility have been hard to realize. Support and development cost have been rising as a consequence.

The move to distributed computing, facilitated by the move to object orientation have given new meaning to reusability and extensibility. This technology provides an infrastructure for building application repositories that hold the business rules, data and User Interface (UI) definitions. Having these business specific components stored in a development repository and providing runtime generation into any external environment frees the organization from one specific technical solution. Business rules can be mapped across a distributed environment, leaving M to do what it does best, perform as a high performance database system that can co-exist with other database technologies. **M**

NOTE

1. David Taylor, Business Engineering with Object Technology, John Wiley & Sons, Inc.

Terry L. Wiechmann is President of ESI Technology Corporation. He can be reached at twiechmann@esitechnology.com. To learn more about ESI, browse the ESI Web page at www.esitechnology.com.