INTRODUCTION

As it is well known, business requirements are changing faster than applications can be created and/or modified. Most of these requirements are in the form of or are related to business rules. Business rules are precise statements that describe, constrain and control the structure, operations and strategy of a business. They may be thought of as small pieces of knowledge about a business domain. They offer a way of encapsulating business semantics and making them explicit in the same way that databases enable the separation of data from application programs.

Traditionally, business rules have been scattered, hard-coded and replicated by different applications. The lack of a formal approach to the management of business rules and a standard business rule language has made it virtually impossible to create, modify and manage business rules in a flexible way. As a result, it has been difficult to adapt applications to new requirements quickly.

Isolating the business rules from the application code enables developers to easily find and modify the pertinent rule(s) when a policy change is needed or when application requirements change. This makes it possible to quickly change rules without modifying the rest of the application code, thereby enhancing maintainability.

In the last decade, one of the trends in database technology has focused on extending conventional database systems (DBMSs) to enhance their functionality and to accommodate more advanced applications. One of these enhancements was extending database systems with powerful rule-processing capabilities. These capabilities can be divided into two classes: deductive, in which logic-programming-style rules are used to provide a more powerful user interface than that provided by most database query languages (Ceri, Gottlob, & Tanca, 1990), and active, where production-style rules are used to provide automatic execution of predefined operations in response to the occurrence of certain events (Act-Net Consortium, 1996; Dayal, Buchmann & McCarthy, 1988). The latter is particularly appropriate for enforcing business rules as it has been demonstrated in Ceri and Widom (1996) and Paton (1999). Database systems enhanced with active capabilities are known as active databases systems, or aDBMSs for short.

By means of active database systems, general integrity constraints encoded in applications have been moved into the database system in the form of rules. These rules go beyond key or referential integrity constraints. Active databases support the specification and monitoring of general constraints (rules), they provide flexibility with respect to the time of constraint checking, and they provide execution of compensating actions to rectify a constraint violation without rolling back the involved transaction. Additionally, support for external events and actions were introduced mostly to satisfy the requirements of monitoring applications.

As a consequence, applications sharing the same database system and data model can also share business rules. In this way, the business knowledge that was dispersed in many applications in the form of programming code is now represented in the form of rules and managed in a centralized way. Consequently, when business rules change, only those affected rules must be modified in the aDBMS.

BACKGROUND

Historically, production rules were used first to provide automatic reaction functionality. Production rules are Condition-Action rules that do not break out the triggering event explicitly. Instead, they implement a polling-style evaluation of all conditions. In contrast, Event-Condition-Action (ECA) rules explicitly define triggering events, and conditions are evaluated only if the triggering event was signaled. Active databases adopted this event-driven approach to avoid unnecessary and resource-intensive polling in monitoring database changes and applications.

In active relational databases, events were modeled as changes of the state of the database, i.e., insert, delete and update operations on tables (Hanson, 1989; Stonebraker, Jhingran, Goh, & Potamianos, 1990). This basic functionality is common fare in commercial DBMSs today. In object-oriented systems, more general primitive events were defined: temporal events, both absolute and relative; method invocation events; and user-defined events (Dayal & Blaustein, 1988; Gatzio & Dittrich, 1993; Zimmermann & Buchmann, 1999).

In addition to these primitive events, more complex situations that correlate, aggregate or combine events can be defined. This is done by using an event algebra...
3 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the product's webpage:

www.igi-global.com/chapter/active-database-management-systems/11113?camid=4v1


www.igi-global.com/e-resources/library-recommendation/?id=1

Related Content

Modeling Temporal Dynamics for Business Systems
www.igi-global.com/article/modeling-temporal-dynamics-business-systems/3297?camid=4v1a

Co-creation and Collaboration in a Virtual World: A 3D Visualization Design Project in Second Life
www.igi-global.com/article/creation-collaboration-virtual-world/47417?camid=4v1a

A Web-Based Application to Exchange Electronic Health Records and Medical Images in Ophthalmology
www.igi-global.com/chapter/web-based-application-exchange-electronic/7978?camid=4v1a

Evaluating the Performance of Dynamic Database Applications
Zhen He and Jerome Darmont (2006). Advanced Topics in Database Research, Volume 5 (pp. 294-319).
www.igi-global.com/chapter/evaluating-performance-dynamic-database-applications/4398?camid=4v1a