Using Business Rules Within a Design Process of Active Databases

Youssef Amghar, Madjid Meziane, André Flory
National Institute of Applied Sciences, France

Modeling behavior is an important task of the information system engineering process. This task is especially important when information systems are centered on active databases, which allow the replacement of parts of application programs with active rules. To relieve programmers from using either traditional or ad hoc techniques to design active databases, it is necessary to develop new techniques to model business rules. For that reason, inclusion of rules during analysis and design stages becomes an actual requirement. In this paper, we propose a uniform approach to modeling business rules (active rules, integrity constraints, etc.). To improve the behavior specification, we extend the state diagrams that are widely used for dynamic modeling. This extension is a transformation of state transitions according to rule semantics. In addition, we outline new functionalities of Computer Aided System Engineering (CASE) to take into consideration the active database specifications. In this way, the designer can be assisted to control, maintain, and reuse a set of rules.

Current design methods for information systems do not consider rules at the design level. In systemic methods such as “Structured Analysis Design Technique” (SADT) (Yourdon, 1979), rules are considered as a part of the design process but they are not modeled explicitly. In Object-Oriented methods such as the Object Modeling Technique (OMT) (Rumbaugh, 1991) or Object-Oriented Analysis (OOA) (Booch, 1994), rules are partially represented in dynamic models, particularly in state diagrams. Moreover, at the development level, rules are often coded in the application programs implying a hard maintenance of business rules. These methods are generally supported by CASE.

To allow designers to exploit the specificities and features of active databases, it is important to build prototyping and monitoring tools to assist the designer during the design and development stage. This kind of tool offers indicators about choice relevancy and writing of rules. An active database management system (active DBMS) is an extension of a passive (relational or object) DBMS by adding trigger mechanisms. The notion of trigger appeared in the seventies, and has been generalized to the notion of active rule that is based on the Event-Condition-Action (ECA) formalism. The semantics of an ECA rule is as follows: when an event E is produced, if the condition C is satisfied, then the action A is executed. Actions are initiated by the DBMS when appropriate events occur, independently of external requests. These rules allow database designers to specify the active behavior of a database application that provides the enforcement of database integrity.

In the literature, several approaches were proposed to integrate active concepts into databases. For most systems, the knowledge model is based on ECA rules and the execution model on the nested transaction model, which authorizes different coupling modes (immediate, separate, deferred). Other systems use a weakened version of ECA rules. Furthermore, a number of research projects on active databases have focused on the rules’ management and their evaluation. Several commercial DBMS include event/trigger mechanism proposed initially by Kotz (1988), such as the PostgreSQL rule system (Stonebraker, 1990), Starburst’s production and alert rules (Lohman, 1991), Ariel’s production rule system (Hanson, 1989), the (ECA) model of HiPAC (Dayal, 88), and the event-action (EA) model of Ode (Gehani, 1992). In addition, there is a general agreement to consider that the new generation of DBMS systems would include active capabilities (Buchman, 1993) to support non-conventional applications such as documentation, geographic systems, workflow, and project management.

The design issue of active database applications is known as one of the most open research problems. Indeed, to design active database applications, programmers use either traditional or ad hoc techniques, which increases the complexity of applications by forcing the user to defer several modeling decisions concerning the active behavior to the development
stage.

To gain benefits of active database capabilities, new approaches require inclusion of rules during both analysis and design stages. Few researchers have addressed the conceptual specification of behavioral aspects of applications independently from any active DBMS. To our knowledge, only IDEA (Ceri, 1993) and SORAC (Peckham, 1995) projects have treated the design of active database. However IDEA methodology is strongly linked to Chimera that is a DBMS developed especially in the framework of IDEA project. In IDEA project, any rules’ specification is proposed. Rules, identified from requirement analysis, are directly expressed to the syntax of Chimera. The SORAC model permits the schema designer to specify enforcement rules that maintain constraints on object and relationships to facilitate the task of the designer. With active databases, it is interesting to replace a part of the application programs by using active rules that are stored, detected and executed by the DBMS. This technique called knowledge independence allows a database to evolve by managing the rules without altering the program code. But a main problem exists regarding the design of such databases. More precisely, the design methods of information systems do not offer satisfying means to describe active rules that represent active behavior of applications. It is necessary to improve the techniques of active behavior description whether at designing or derivation level.

Our work presents a framework for designing applications for which active behavior needs to be correctly modeled, particularly applications focused on active databases. The main contributions of this paper include:

1. Specification of system behavior at a high level of description. This specification improves the modeling of system behavior with the introduction of the ECA rules’ semantics.
2. Derivation process of active behavior specification for relational DBMS’s.
3. Definition of an environment for active database design.

This paper is organized as follows: Section Case Study presents a case study to support our explanations. Section Dynamic Description briefly discusses the techniques currently used to represent the business rules within data models. Section Specification of System Behavior proposes to model the behavior of an active object through a specification based on a description language developed initially for telephone switching systems. Section Derivation Process describes the derivation process of models elaborated in the design stage for a relational schema. Section Rule Modularization presents a strategy for modularizing a set of rules in order to facilitate the rules’ maintenance. This section also presents some features of a prototyping tool. Section Environment for Active Databases describes the original functionality of the architecture of an environment for active databases. The last section concludes this work.

CASE STUDY

We present a case study to illustrate the paper’s propositions. Let us consider an example of a document-loan management system. In this example, reservation, loan and loan prolongation are managed. Some rules are imposed by the application to fix the number of loans and loan prolongations. Two activities are identified: document loaning and document restitution. We delimit the application schema to entities described in Figure 1. The two main entities are user and document. The three relationships indicate the links between a user and a document. Indeed a document can be borrowed or reserved. Let us note that a user can prolong a loan under certain conditions.

Relational Schema

Two main entities constitute the relational schema: DOCUMENT and USER. Each document is described by the following attributes: id-doc: the reference of the document; doc-name: the document name; state: the availability of the document in the document center; nb-copies: the number of copies of the document; and nb-loans: the loan number of the document. A reference (id-user) and his name (username) describe each user. Each loan concerns one user (id-user) and one document (id-doc) and begins at b-l-date and would finish at e-l-date. The document is effectively returned at ret-date. Each reservation concerning one user (id-user) and one document (id-doc), begins at b-reservation-date and finishes at e-reservation-date. The relational schema is comprised of the following relations and their respective attributes:

Document(#id-doc,doc-name,state,nb-copies,nb-borrow),
User(#id-user,name),
Borrowing(#id-doc,#id-user,b-l-date,e-l-date,ret-date),
Reservation(#id-doc,#id-user,b-reservation-date,ret-date).

Note that in this case study, the illustration is simplified by not considering document prolongation. The user requirements provided the basis for the following rules concerning

Figure 1: Entity-Relationship model of a borrowing management system