

Modeling Active Object-Oriented Database Applications using Multi-level Diagrams

Mauricio J. V. Silva*

C. Robert Carlson

Dept. of Computer Science, Illinois Institute of Technology, 10 West 31st Street, Chicago, IL 60616
email: silvmau@charlie.acc.iit.edu, cscarlson@minna.acc.iit.edu

Abstract

Active Object-Oriented Database Systems (AOODBSs) are based on the object-oriented paradigm and provide an event driven behavior for implementing time critical reactions by integrating event-condition-action (ECA) rules to the database. This paper focuses on the modeling aspects of AOODBSs. We propose an integrated approach, called A/OODBMT (Active Object-Oriented Database Modeling Technique), which integrates and extends the Object Modeling Technique (OMT) method for conceptually designing active object-oriented database applications. A/OODBMT models database applications by defining and integrating four new types of models, namely the nested object model (NOM), the behavior model (BM), the nested rule model (NRM), and the nested event model (NEM).

1 Introduction

Active object-oriented database systems (AOODBSs) [4, 5, 6, 8] continually monitor changes to the database state and react by executing an appropriate action without user or application intervention. This paper adds to the research on AOODBSs by developing an integrated method, called A/OODBMT (Active Object-Oriented Database Modeling Technique), which integrates and extends the Object Modeling Technique (OMT) method [10] for conceptually designing active object-oriented database applications.

A/OODBMT models database applications by defining and integrating four new types of models, namely the nested object model (NOM), the behavior model (BM), the nested rule model (NRM), and the nested event model (NEM). Each model describes one aspect of the system but contains references to the other models. The nested object model contains descriptions of the classes and ob-

jects in the system on which the behavioral model operates. The operations in the nested object model are described in the behavior model. The behavior model also uses operation events defined in the nested event model to describe the control aspects of the objects. Rules referenced in the nested object model are defined in the nested rule model, and are executed in the context of the database transactions defined in the behavioral model. Rules specified in the nested rule model are triggered by events defined in the nested event model.

2 Related Work

The Nested Object Model in A/OODBMT provides a multi-level representation approach to the definition of the objects in a system, including their attributes, methods and rules.

In the object-oriented paradigm, the functionality of a system is achieved by the interaction and execution of object operations. Most object-oriented methods have used state diagrams to model object operations [7, 10]. Like [10], we use state diagrams with data-flow diagrams. The difference is that instead of representing the data flow diagram for the whole system, we create a data-flow diagram for each operation and its activities.

The interaction of the objects in a system defines a transaction. Moreover, the active behavior of the systems (i.e. rules) is only executed within the context of a transaction. Although current object oriented methods (e.g. [7, 10]) model the interaction of objects in a system, they do not model database transactions. Our approach is to extend the modeling of object interactions with database transaction commands.

Existing visual approaches to rule specification include [1, 9]. Our approach is to integrate previous approaches and extend them by visually representing a comprehensive set of rules and their interactions in a multi-level model, called the nested rule model. We also

* Supported by the Brazilian Government Agency - CAPES

provide the semantics for overriding rules and support a wealth of coupling modes for rule execution [2].

The modeling of events is crucial to active object-oriented database systems, since they determine when rules will be evaluated. Existing visual approaches (e.g. [1, 9]) do not support a comprehensive set of both simple and composite events. Our approach deals with these problems by supporting a comprehensive set of events and by providing a high-level graphical representation of events in multi-level diagrams.

3 Active Database Conceptual Design using A/OODBMT Models

In this section, we give the steps followed during conceptual design for the modeling of an active database application. We illustrate the application of these steps by modeling a library database system. The requirements of the library database system are described in Figure 1. From these requirements, we build the A/OODBMT models and show their integration. Only the essential points of the A/OODBMT models will be discussed in the forthcoming subsections. The detailed description of the application of A/OODBMT can be found in [11].

In the following subsections we describe the nested object model, the behavioral model, the nested rule model, and the nested event model in turn.

- (1) The library system is composed of books and members.
- (2) The library is opened from 10 a.m. - 10 p.m. (Mondays to Fridays) and from 10 a.m. - 6p.m. in the weekends.
- (3) A member is a person and is characterized by a name, a *ss#* and an address. There are two types of members: students, and faculty.
- (4) A book is characterized by a title, an author, a date of publishing and a publisher.
- (5) A person may check out books only if he/she is a member of the library. Only available books can be checked out.
- (6) All members may check out books for 2 weeks.
- (7) When a person checks out a book, he/she must provide the title of the book and his/her *ss#*.
- (8) When a book is checked out, a due date for returning the book is set.
- (9) Books are expected to be returned by the due date. If the book is returned after the due date, it is considered to be overdue.
- (10) When a person returns a book, if it is overdue a fine of 10 cents per day is charged to the member for each book not returned on time. A faculty member will receive only warnings for the first 5 overdue books. After that, the faculty member will start paying fines for overdue books.
- (11) A notice is mailed to a member if he/she has a book that has been overdue for seven consecutive days.

Figure 1. Library Database System Example

3.1 Nested Object Modeling

The nested object model (NOM) is used to represent the static aspects of applications and is based on the object model [10] enhanced with nesting capabilities and rules. The major concepts found in NOM include class, object, relationships, attributes, operations, rules, and complex objects. A detailed description of NOM can be found in [11].

The nested object model derived from the library database example is described in Figure 2 with the following objects: simple classes (*Book*, *Student Member*, and *Faculty Member*), simple associations (*return*, and *checkout*), complex classes (*Person*, *Member*, and *Library*), complex association (*uses*), generalization (*Person* with *Member*, and *Member* with *Student Member* and *Faculty Member*), and aggregation (*Library* with *Book* and *Member*). Note that in Figure 2, we have also included the attribute, method and the rule part of the classes which will be discussed later in this article.

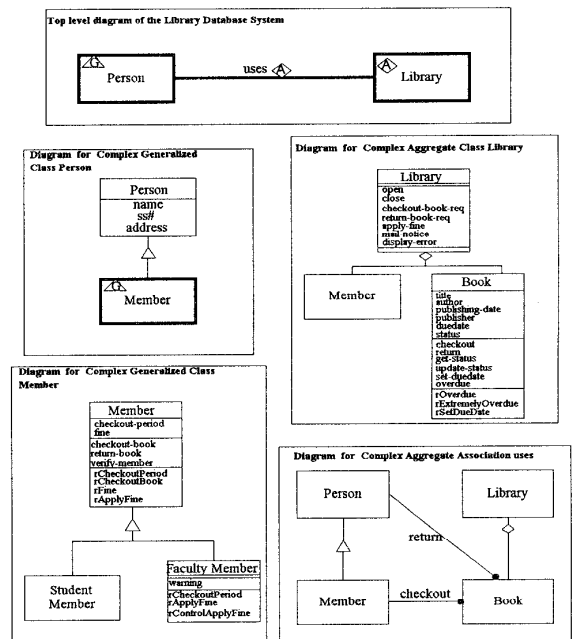


Figure 2. Nested Object Model of the Library System

3.2 Behavioral Modeling

The behavioral model represents the temporal and transformational aspects of a system. It combines and integrates the dynamic (state diagrams) and functional model (data-flow diagrams) originally proposed in [10],

adding database transaction capabilities. Thus, the behavioral model is composed of three diagrams: state diagrams, data-flow diagrams and transaction diagrams.

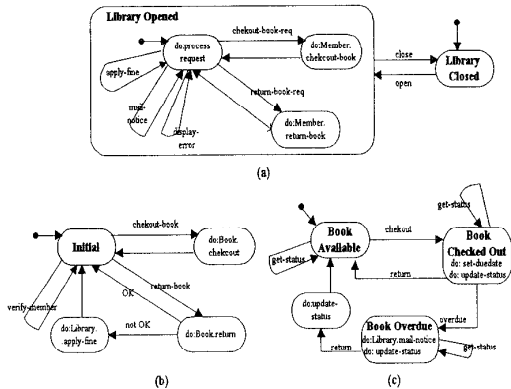


Figure 3. State Diagrams for the Library System

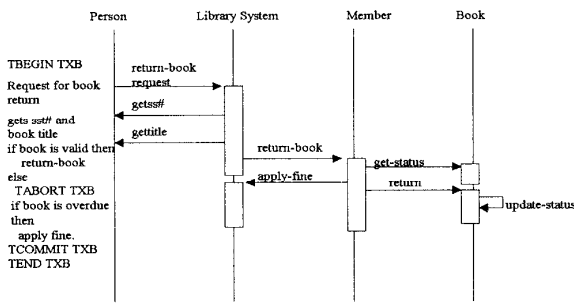


Figure 4. Return Book Transaction

The state diagrams derived from the library system are described in Figure 3. We define state diagrams only for the main classes of the library system, namely *Library*, *Member*, and *Book*.

A data-flow diagram (DFD) is used to describe the operation of an object/class identified in the state diagram. It is represented as a graph whose nodes are processes (transform data), actors (produce and consume data) and datastores (store data), and whose arcs are data-flows (move data) and control-flows (control process evaluation).

Transaction diagrams describe database transactions in the system. Figure 4 shows a transaction diagram for a *return book* transaction.

3.3 Nested Rule Modeling

The nested rule model (NRM) is a high-level graphical approach for conceptually designing rules in active

object-oriented databases. NRM models a comprehensive set of rules, using two types of diagrams - nested rule diagrams (NRDs) and rule interaction diagrams (RIDs).

In NRD, a rule is depicted by an ellipse and has a name and a type (simple or composite), which is represented inside the rule icon separated by a line (see Figure 5). NRDs visually represent both simple and composite rules by using two types of abstraction techniques, embedding and nesting. Simple rules may be used to constrain the structure of objects and may also govern the object's behavior through dynamic rules, such as event-condition action or exception rules. Composite rules enable the designer to express complex object behavior by applying a set of constructors to simple rules and previously defined composite rules. Further, NRDs define the semantics of rule inheritance and overriding, and describe how the coupling mode of rules can be specified.

Static rules define constraints on the structure of a class that must always hold. These constraints are specified in terms of classes, objects, attributes and associations. Since static rules are always true and are defined in the context of an object/class, they are represented by placing an invariant condition within the context object/class all embedded within the rule icon (see Figure 5(a)).

In NRD, static rules can be classified into the following constraint rules: attribute constraints, attribute domain constraints, mandatory/optional attribute constraints, attribute cardinality constraints, population type constraints, association cardinality constraints, existence dependence constraints, relational constraints, and uniqueness constraints. From the library example line (10), we derive an invariant attribute constraint rule *rFine* in class *Member* (see Figure 5(a)).

Dynamic rules monitor the way an object's processes may execute and relate to one another and how objects respond to specific events and exceptions. In NRD, dynamic rules include event-condition-action rules, exception rules, contingency rules, precondition rules, post-condition rules, and production rules. Examples of the dynamic rules not illustrated in this paper can be found in [11].

The Event-Condition-Action (ECA) rule defined by [4] was originally used by [3] to extend relational databases with active capabilities. In NRD, a named event is only referenced in the rule using an event icon (a parallelogram). The actual description of the event is represented using the nested event diagram, which is described in the next section. A condition determines if an action can be executed and is represented by a condition icon (a hexagon). A condition that is always "true", can be omitted. An action is always represented as a process which is applicable to a specific object. An action is implemented

as a method in the object and can actually trigger the execution of other rules. In addition, the action part can be used to control the reasoning of production rules. The complete ECA rule is depicted by an event icon connected with an arrow to the condition icon and which in turn is connected to the action icon with an arrow. Below we show examples of ECA rules for the classes *Member*, and *Book*. ECA rules of class *Member* that are overridden by class *Faculty Member* are described later in this article.

- ECA rules for class *Member*: From line(5), we identify a rule *rCheckoutBook* for checking out a book (see Figure 5(b)).
- ECA rules for class *Book*: From line(9), we identify a rule *rOverdue* for returning an overdue book (see Figure 5(c)). From line (11), we identify a rule *rExtremelyOverdue* for a person who keeps an overdue book more than seven consecutive days (see Figure 5(d)).

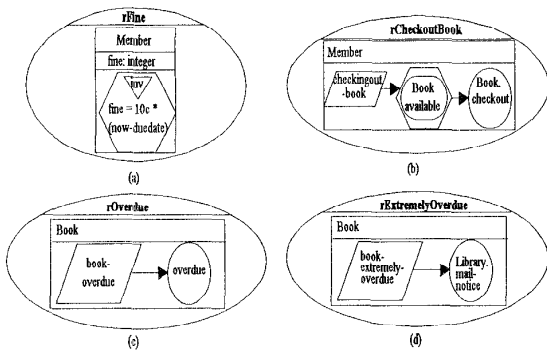


Figure 5. Rules for the Library System

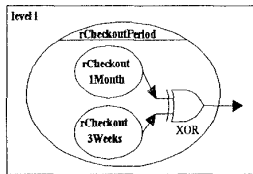


Figure 6. Composite Exclusive Rule *rCheckoutPeriod*

Composite rules enable the database designer to structurally abstract rules. They are particularly useful when the number of rules becomes very large and very difficult to comprehend. In this case, the rules are divided and grouped into related rules (components) represented by a composite rule.

For example, let us consider that we change the requirements of the library system to allow a faculty member to checkout books for a period of one month if he/she

has no books overdue, but reduce the checkout period to three weeks after the fifth overdue book.

To model this example, we assume that rules *rCheckout1Month* (representing a checkout period of one month), and *rCheckout3Weeks* (representing a checkout period of three weeks) have already been defined. Then, we define a more general rule *rCheckoutPeriod* in class *Faculty Member* as a composite exclusive rule of *rCheckout1Month* and *rCheckout3Weeks* (see Figure 6).

In AOODBSs, rules are triggered within a database transaction (i.e., triggering transaction) and are executed according to their coupling modes to the transaction. The coupling modes of a rule determine at what point in the triggering transaction the rule will be evaluated and determine whether the rule will be executed as a separate top level transaction or will be executed as a subtransaction of the triggering transaction.

Figure 7 shows a rule *rExtremelyOverdue* in class *Book*, where the rule is evaluated *immediately*, but the action of *mailing a notice* is executed as a *separate* transaction.

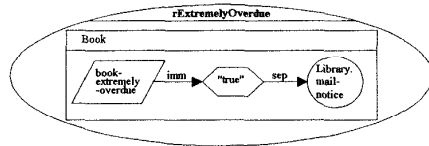


Figure 7. Immediate/Detached Coupling Mode Rule

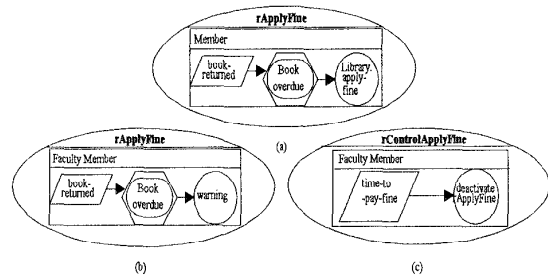


Figure 8. Rule Inheritance and Overriding

Rules attached to a class are automatically inherited by each subclass. Like methods, rules can be overridden. Overriding a rule means that the subclass has attached to it a modification or a refinement of the rule. There are two main reasons the designer may want to override a rule: to specify a rule that is the same as the inherited rule, except it adds some behavior usually affecting new attributes of the subclass, or to tighten the specification of a rule by tightening the type of the arguments used in the expressions.

- Example of Rule Overriding: From the library example line(10), we define a rule *rApplyFine*, which is first defined in class *Member*, and then redefined in class *Faculty Member*. In class *Member*, *rApplyFine* is defined with an action which is the operation *apply-fine* in class *Library* (see Figure 8(a)). In class *Faculty Member*, *rApplyFine* overrides the rule in class *Member* with a different action, which is an operation *warning* (see Figure 8(b)). Also, in order to deactivate the overriding of rule *rApplyFine*, we define another rule *rControlApplyFine*, which is executed after the *Faculty Member* has received five warnings (see Figure 8(c)).

Rule interaction diagrams (RIDs) visually show the interdependence between rules. It is a very useful diagram to show the database designer, the cascading of rules, and how they relate to each other. A detailed description of RIDs can be found in [11]

3.4 Nested Event Modeling

The Nested Event Model (NEM) visually models the events referenced in rules using a multi-level diagram, called the Nested Event Diagram (NED). NED models a comprehensive set of events, integrating the event types present in existing active object-oriented database systems. NED is composed of primitive (simple) and composite (complex) events. Primitive events correspond to elementary occurrences, and composite events correspond to events that are formed by applying a set of constructors to primitive and composite events. Below we use the nested event model to describe the events used in the rules defined for classes *Library*, *Member*, *Faculty Member* and *Book*. Event types not illustrated in this article can be found in [11].

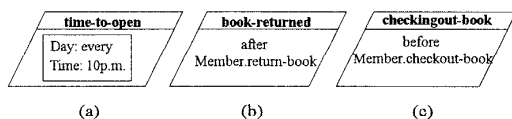


Figure 9. Primitive Events

Primitive events describe a point in time specified by either database events (i.e., method execution events, and transaction events), temporal events, or explicit events.

- Events for class Library: The periodic temporal event *time-to-open* derived from line (2), occurs every day at 10 a.m. (see Figure 9(a)).
- Events for class Member: The method execution event *book-returned*, referenced in rule *rApplyFine* (see Figure 8(a)), occurs after a book is returned (see Figure 9(b)). The event method execution event *checkingout-book*

referenced in rule *rCheckoutBook* (see Figure 5(b)), occurs before a book is checked out (see Figure 9(c)).

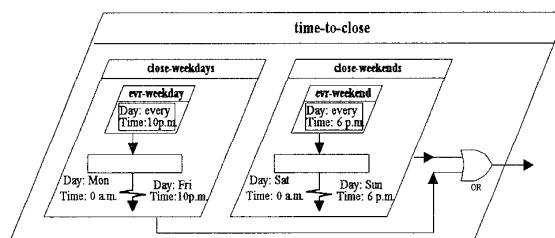


Figure 10. Representation of Event "time-to-close"

Composite events are defined by applying event constructors to previously defined events, called component events, and occurs at the point of occurrence of the last event that was needed to make it happen [6]. Figures 12(a) and (b) illustrate that a designer can use either nesting or embedding abstraction techniques to represent composite events.

In NED, composite events are classified into the following events: conjunction event, disjunction event, monitoring interval event, relative temporal event, closure event, history event, every-nth event, negative event, and sequence event.

The disjunction of two events E1 and E2, occurs when E1 occurs or E2 occurs [5]. Figure 10 represents the disjunction of two monitoring events *close-weekdays* and *close-weekends*.

A monitoring interval event occurs when an event E happens anytime in an interval I and some condition C holds during the interval [5].

An interval I is specified by a starting and ending point in time and is depicted by a bar. The starting point and ending point of an interval can be defined by the occurrences of two events. A condition C is always associated with an interval. It is depicted within the bar interval. If there are no conditions related to the interval we do not represent the condition icon. An arrow with a flash below the time interval denotes the point in time of the occurrence of the monitoring interval event.

- Events for class Library: The disjunction event *time-to-close* derived from line(2), occurs every weekday at 10 p.m., or Saturdays and Sundays at 6 p.m. (see Figure 10).
- Events for class Book: The relative temporal event *book-overdue* referenced in rule *rOverdue* (see Figure 5(c)), occurs at the book's due date after it has been checked out (see Figure 11(a)). The event *book-extremely-overdue* referenced in rule *rExtremelyOverdue* (see Figure 5(d)), occurs when a book has been overdue for seven days (see Figure 11(b)).

• Events for class Faculty Member: The sequence event *time-to-pay-fine* (see Figure 12) referenced in rule *rControlApplyFine* (see Figure 8(c)), occurs after a faculty member has received 5 warnings for overdue books.

In some applications an event may repeatedly occur. In such a case, a history event designates a specific occurrence as the triggering event [5]. The history event is denoted by placing an occurrence identifier to the "out arrow" of the event (e.g. see event *warning* in Figure 12).

A closure event signals only the first occurrence of an event E, even if the event E continues to occur [5]. The closure event is denoted by placing an '*' to the "out arrow" of the event (e.g. see event *checked-out* in Figure 12).

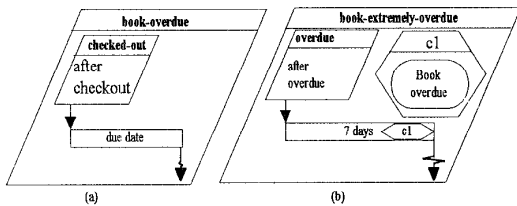


Figure 11. Relative Events for Class Book

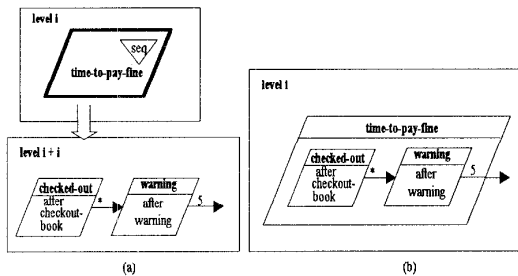


Figure 12. Sequence Event "time-to-pay-fine"

4 Conclusion

In this paper we proposed an integrated approach to active object-oriented database conceptual design, called A/OODBMT. A/OODBMT models database applications by defining and integrating four new models, namely the nested object model (NOM), the behavior model (BM), the nested rule model (NRM), and the nested event model (NEM).

The presented approach is well suited for the design of active object-oriented database application, because of its semantic richness and the ability to deal with the complexity of many object/classes, rules and their interaction, and events. We are currently developing a CASE tool that

will, not only support the graphical notation proposed in this paper, but also support the automatic code generation of the active object-oriented database schema.

References

- [1] Bichler, P., and Schrefl, M., "Active Object-Oriented Database Design Using Active Object/Behavior Diagrams," In *Proceedings of the Fourth International Workshop on Research Issues in Data Engineering*, IEEE Comp. Soc. Press, Los Alamitos, CA, USA, 1994.
- [2] Branding, H., Buchmann, A. P., Kudrass, T., and Zimmermann, J., "Rules in an Open System: The REACH rule system," In Paton, N., and Williams, M. (eds.), *Rules in Database Systems*, Workshops in Computing, Springer-Verlag, 1993, pp. 111-126.
- [3] Carlson, C. R. and Arora, A. K., "UPM: A Formal Tool for Expressing Database Update Semantics", In *Proceedings of the Third International Conference on Entity-Relationship*, North Holland, NY, 1983, 517-526.
- [4] Dayal, U., "Active Database Management Systems," In *Proceedings 3rd International Conference on Data Knowledge Bases*, Jerusalem, Israel, June 1988.
- [5] Gatzju, S., and Dittrich, K., "SAMOS: An active object-oriented database system," In *IEEE Bulletin of the Technical Committee on Data Engineering*, Vol. 15, No. (1-4), December 1992.
- [6] Gehani, N. H., Jagadish, H. V., and Shmueli, O., "Event Specification in an Active Object-Oriented Database," In *Proc. of the 1992 ACM SIGMOD Int'l Conf. on Management of Data*, CA, June 1992, pp. 81-90.
- [7] Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.
- [8] Kappel, G., Rausch-Schott, S., Retschitzegger, W., and Vieweg, S., "TriGS: Making a passive object-oriented database system active," *Journal of Object-Oriented Programming*, June/July 1994, pp. 40-51.
- [9] Martin, J., and Odell, J., *Object-Oriented Methods: a foundation*, Prentice Hall, Englewood Cliffs, NJ, 1995.
- [10] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., *Object-oriented modeling and design*, Prentice Hall, Englewood Cliffs, 1991.
- [11] Silva, M. J. V., *A/OODBMT, an Active Object-Oriented Database Modeling Technique*, Ph.D. Thesis, Illinois Institute of Technology, 1995.