IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4115

Event Modeling in UML

Lars Bækgaard
Department of Computer Science, Aalborg University, Fr. Bajers Vej 7E, DK-9220 Aalborg Øst, Denmark
Tel: +45 9635 8080, lars@larsbaekgaard.com

ABSTRACT

We show how events can be modeled in terms of UML. We view events as change agents that have consequences and as information objects that represent information. We show how to create object-oriented structures that represent events in terms of attributes, associations, operations, state charts, and messages. We outline a run-time environment for the processing of events with multiple participants.

INTRODUCTION

Events are atomic occurrences (Jackson 1981). They are change agents that have consequences and they are information objects that represent information (Bækgaard 2001). When a customer order is received packing, shipping, and billing events may be triggered and information about the order may be recorded and stored in a data warehouse. When a book is borrowed in a library a recall event may be triggered 30 days later and information about the state of library books may be recorded and stored in a database.

Many existing approaches to object-oriented event modeling recommend that events should be objectified. Anderson (2000) uses event objectification as the basis for event patterns. Aarsten (1996) uses event objectification as the basis for reactive systems. Ran (1995) uses event objectification as the basis for event-driven systems.

We outline a method that supports the modeling of real-world events in terms of UML (Rumbaugh et al. 1999). We show how to create object-oriented structures that represent events in terms of attributes, associations, operations, state charts, and messages. Our approach is based on event objectification but unlike other approaches it is based on an event model where events are viewed as both change agents and information objects.

In Section 2 we introduce the event model on which our work is based. In Section 3 we use attributes and associations to model information about events and their participants. In Section 4 we use object operations and state charts to model the consequences of events. In Section 5 we use messages to model the occurrence of events. Section 6 we outline a run-time environment that ensures that no event is accepted unless each participant admits it. In Section 7 we conclude the paper.

EVENTS

Our event model is based on the event-entity-relationship model in which events have participants, descriptive properties, and consequences (Bækgaard 1999, Bækgaard 2001). A borrow event in a library may have a borrower and a book as participants. The event may have descriptive properties like date. And as a consequence of the event the status of the book may be changed to "borrowed".

We use expressions on the form "Name [Participants] (Properties)" to define individual events. The element "Participants" represents the participants of the event. The element "Properties" represents the descriptive properties of the event. For example, the event expression Borrow [$Borrower_1$, $BookItem_4$] (September 29, 2001) may represent the fact that $Borrower_1$ and $BookItem_4$ have participated in a Borrow event on September 29, 2001.

We use expressions on the form "Name [ParticipantTypes] (PropertyTypes) {Consequences}" to define event signatures. An event signature defines a type of conforming events. The element "ParticipantTypes" defines the types of participants that can participate in events of the defined type. The element "PropertyTypes" defines the types of properties of events of the defined type. The element "Consequences" represents the action that is initiated when an event of the defined type occurs. The notation used to specify "Consequences" may be chosen freely to suite the requirements of the operational environment.

Inherent consequences are strongly coupled to an event in the sense that they are defining characteristics of the event. For example, a library book changes status from "available" to "borrowed" when it is borrowed. This change of status is a defining characteristic of what it means to borrow a library book. Triggered consequences are loosely coupled to an event in the sense that they are not defining characteristics of the event. For example, a return-monitor activity may be triggered when a book is borrowed. The monitor activity is not a defining characteristic of borrow events. It defines a book recall policy in the library.

An event expression conforms to a signature expression if it has the same name as the signature and if its participants and properties conform to the participant and property types of the signature. For example, the event expression Borrow [$Borrower_1$, $BookItem_4$] (September 29, 2001) conforms to the signature expression Borrow [Borrower, BookItem] (Date) {…}.

INFORMATION ABOUT EVENTS

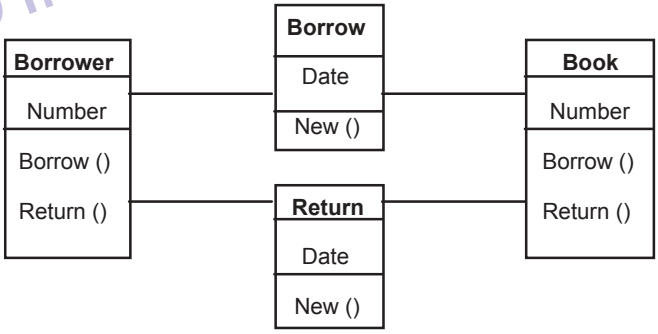
We use attributes and associations to model information about events.

We define an event class for each event type and we define a participant class for each participant type. The properties of the event type are represented by attributes on the event class. Event-related attributes of the participants are represented by attributes on the participant classes. We associate the event class with each participant class. The associations are used to represent participation in events.

We have used the signature expressions Borrow [Borrower, BookItem] (Date) {…} and Borrow [Borrower, BookItem] (Date) {…} to define the attributes and associations in the class diagram in Figure 1.

Each Book object represents one library book. Each Borrower object represents one library borrower. Each Borrow object represents one borrow event. Each Return object represents one return event. Each Borrow and Return object has an attribute called Date that contains the occurrence date of the corresponding event.

Figure 1: Library class diagram



This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference.

The association between Borrow and Book/Borrower represents the fact that books/borrowers may participate in borrow events. The association between Return and Book/Borrower represents the fact that books/borrowers may participate in return events.

CONSEQUENCES OF EVENTS

We use operations and state charts to model the inherent and triggered consequences of events.

When we model inherent consequences we define an operation called New on the event class. This operation creates a new event object with initialized attribute values (event properties) and sends messages to participation objects and activity objects (see Section 5). Also, we define an event operation on each participant class. Each of these operations handles a part of the inherent consequences that involves a specific participant.

In Figure 1 we have added an operation called New to the event classes Borrow and Return and we have added two operations called Borrow and Return to the participant class Borrower and Book.

Triggered event consequences may not be related directly to a specific participant. In such cases we define a new activity class whose objects are responsible for the activity. In some cases we supplement the operations of the objects with a state chart.

In Figure 2 we have modeled an activity that handles the recall of library books that are returned too late. The activity class Monitor is associated with the event classes Borrow and Return. A new Monitor object is instantiated each time a book is borrowed. The object is terminated if the book is returned within 30 days. Otherwise, a Recall message is activated before the object is terminated. NewDay is a message that is sent automatically on time each day.

EVENT OCCURRENCES

We use messages to represent event occurrences.

For each event signature we define a rule that transforms conforming event expressions into a set of messages that are send to the event object, to the participant objects, and to the activity objects.

Name [x_1, …, x_n] (y_1, …, y_m) {…}:
  Name [x_1, …, x_n] (y_1, …, y_m) ®
    Name.New (… parameters …)
    x_1.Name (… parameters …)
    …
    x_n.Name (… parameters …)
    z_1.Name (… parameters …)
    …
    z_k.Name (… parameters …)

Each transformation rule defines the messages that must be send when an event of the corresponding type occurs. One message is send to the event object (Name). One message is send to each participant object (x_i). One message is send to each activity object (z_j). The parameters is a subset of {x_1, …, x_n, y_1, …, y_m }. In some cases additional parameters may be needed.

The following example is based on Figure 1 and the event signature Borrow [Borrower, BookItem] (Date) {…}.

Borrow [Borrower, BookItem] (Date) {…}:
  Borrow [Borrower, BookItem] (Date) →
    Borrow.New ()
    Borrower.Borrow ()
    Book.Borrow ()
    Monitor.New ()

Each borrow event occurrence is represented by four messages to objects in Borrow, Borrower, Book, and Monitor.

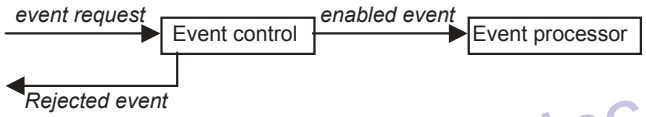
RUN-TIME ENVIRONMENT

State charts can be used to define the dynamics of state-dependent objects that may refuse to respond to certain messages when they are in certain states (Rumbaugh et al. 1999). For example, state charts can be used to model object life cycles (Bækgaard 1997, Bækgaard & Godskesen 1998, Jackson 1983) and they can be used to model activity objects as described in Section 4.

Each occurring event is transformed to a set of messages as described in Section 5. However, one or more state-dependent objects may refuse to respond to one of the generated messages. In such situations the event should be refused as whole.

As indicated by Figure 3 event requests must be controlled before they are admitted.

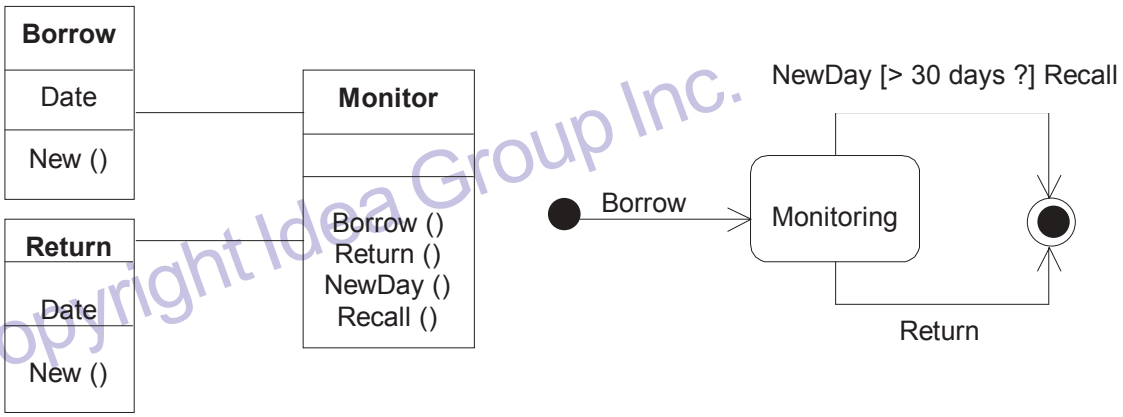
Figure 3: Events and event control



The following algorithm is executed each time an event is requested. We have assumed that the rules embedded in state charts are the only rules that may result in the rejection of event requests.

1. Create object messages as described in Section 5;
2. For each state-dependent participant object:
   2.1. Lock the object;
   2.2. IF the corresponding state machine does not enable the message THEN reject the event request;
3. Send the created messages to the corresponding objects;
4. Release all locked objects.

Figure 2: Book loan monitor



This event control mechanism is similar to integrity checking in database transaction systems if we view the rules embedded in state charts as integrity rules. Like atomic transactions, events are atomic "all-or-nothing" processes.

CONCLUSION

We have outlined a method for event modeling in UML. The method is based on an event model where events have participants, properties, and consequences.

We use attributes and associations to model information about events. We use operations and state charts to model the consequences of events. We use messages to model occurrences of events. A run-time environment ensures that an event is rejected unless all state-dependent participants admit it.

Based on our experiences we conclude that UML supports the modeling of information about events in a coherent and integrated way. All the relevant information about an event type can be modeled as an event class that is associated with its participant classes. We are less satisfied with the way UML supports the modeling of the consequences of events. Such consequences must be defined in a fragmented manner because of the distributed process paradigm that underlies the object-oriented paradigm.

Future work includes extensions to UML that supports the modeling of event consequences in a non-fragmented manner.

REFERENCES

Aarsten, A. et al. (1995). Object-Oriented Design Patterns in Reactive Systems. In (Vlissides et al. 1996).

Anderson, F. (1999). A Collection of History Patterns. In (Harrison et al. 2000).

Bækgaard, L. (1997). Transaction-Based Specification of Database Evolution. 16th International Conference on Conceptual Modeling (ER'97), Los Angeles, California, USA.

Bækgaard, L. & J. C. Godskesen (1998). Real-Time Event Control in Active Databases. Journal of Systems and Software 42(3): 263-271.

Bækgaard, L (1999). Event-Entity-Relationship Modeling in Data Warehouse Environments. International Workshop on Data Warehousing and OLAP (DOLAP'99), Kansas City, USA, November 6, 1999.

Bækgaard, L. (2001). Event Modeling. In (Rossi & Siau 2001).

Coplien, J.O. & D.C. Schmidt, eds. (1995). Pattern Languages of Program Design. Addison-Wesley.

Harrison, N. et al., eds. (2000). Pattern Languages of Program Design 4. Addison-Wesley.

Jackson, M. (1983). System Development. Prentice-Hall.

Ran, A.S. (1994). Patterns of Events. In (Coplien & Schmidt 1995).

Rossi, M. & K. Siau, eds. (2001). Information Modeling in the New Millennium. Idea Group Publishing.

Rumbaugh, J., et al. (1999). The Unified Modeling Reference Manual. Addison-Wesley.

Vlissides, J.M. et al., eds. (1996). Pattern Languages of Program Design 2. Addison-Wesley.