Do Triggers Have Anything To Do With Exceptions?

Elisa Bertino
Dipartimento di Scienze dell'Informazione
Università di Milano - Milano, Italy
bertino@dsi.unimi.it

Giovanna Guerrini *Isabella Merlo*
Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova - Genova, Italy
{guerrini,merloisa}@disi.unige.it

1 Introduction

The object-oriented paradigm played and is still playing a crucial role in getting the programming language and the database fields closer to each other. The Java^{TM} [1] programming language is establishing itself as the most popular object-oriented programming language, and the increasing need of persistently storing Java objects, through minimal modifications to applications, led most object-oriented database management systems (OODBMs) – e.g., ObjectStore, GemStone and POET – to develop a Java version. The ODMG standard also has been enriched with a Java binding.

An important functionality provided by current relational and object-relational DBMSs is represented by *triggers*. This typical database functionality has no direct counterpart in object-oriented programming languages, and though trigger usefulness has been recognized also in the object-oriented context and a considerable amount of research has been carried out in the area of active object-oriented databases (see Section 2), triggers are not supported in current commercial OODBMSs. A preliminary investigation on the issues involved by the introduction of triggers in Java-based databases has been reported in [4].

A relevant feature of the Java language is represented by its *exception* mechanism, which represents a very powerful mechanism for handling error situations. Since one of the most relevant functionalities achieved through triggers in the database context is integrity constraint enforcement, the two notions seem to be potentially related. Aim of this paper is thus to investigate whether and how the two notions can be related, and if the support of exceptions can offer any help or insight towards the introduction of triggers in Java-based OODBMSs. In the following, we provide an overview of triggers, then we compare the two mechanisms, and finally we conclude by discussing some open issues.

2 Triggers: Basic Notions

A trigger is a syntactical construct to define the reaction of the system. Triggers are usually specified by means of the *event-condition-action* (ECA) paradigm. Systems supporting triggers are called *active databases*. Active databases enable important applications, such as alerting users that a given event has occurred, reacting to events by means of suitable actions, and controlling the invocation of operations and procedures. Examples of functions that can be effectively performed by active database systems are integrity constraint enforcement, monitoring, authorization, statistics gathering and view handling.

In the last ten years, there has been a growing interest in extending object-oriented database systems with triggers. One of the earliest object-oriented active database projects is HiPAC [8] which provides an expressive active database rule language with flexible execution semantics. One of the best known active object-oriented database systems is Ode [11], which extends the O++

database programming language with facilities for expressing rules in the form of constraints and triggers. Among other projects we would like to mention Adam [9], NAOS [7], SAMOS [10], TriGS [12] and Chimera [3, 5]. Among them, NAOS and TriGS are extensions of commercial object-oriented database systems with active capabilities. All the proposals support *event-condition-action* (ECA) rules.

In an active rule, the *event* specifies what causes the rule to be triggered. Relevant triggering events can be internal events related to database operations (for example object creations and deletions), temporal events, external (that is, raised by the application), and user-defined events. In addition, in some systems it is possible to specify whether a rule must be triggered before or after its triggering event. Triggering events may also be *composite*, that is, combinations of other events. Useful operators for combining events are logical operators, such as conjunction, disjunction, negation, and sequences. Finally, we recall that in some approaches events are managed as objects.

In an active rule, the *condition* specifies an additional condition to be checked once the rule is triggered and before the action is executed. Conditions are predicates over the database state. In the most common approach, the condition is expressed as a query returning data, that are then passed to the rule action. Moreover, some systems allow triggers to refer to past database states.

In an active rule, the *action* is executed when the rule is triggered and its condition is true. Possible actions include database operations and calls to application procedures. Several active database rule languages allow sequences of actions to be specified in rules.

The cited systems differ not only with respect to the supported rule language, but also in terms of rule execution semantics. First of all, active database rule execution can be either *instance-oriented* or *set-oriented*. With an instance-oriented execution, a rule is executed once for each database "instance" triggering the rule and satisfying the rule condition. By contrast, rule execution is set-oriented if a rule is executed once for all database instances triggering the rule and satisfying the rule condition. Among the considered systems, all but Chimera support an instance-oriented rule execution. Chimera supports set-oriented rule execution, while NAOS rule execution is instance-oriented for immediate rule and set-oriented for deferred ones.

The most straightforward approach to *rule processing granularity* is to evaluate a triggered rule condition and to execute its action within the same transaction in which the triggering event occurs, at the soonest rule processing point. However, for some applications it may be useful to delay the evaluation of a triggered rule condition or the execution of its action until the end of the transaction; or it may be useful to evaluate a triggered rule condition or execute its action in a separate transaction. These possibilities result in the notion of *coupling modes* [13]. One coupling mode can specify the transactional relationship between a rule triggering event and the evaluation of its condition, while another coupling mode can specify the transactional relationship between a rule condition evaluation and the execution of its action. Possible coupling modes are: *immediate* (takes place immediately following, within the same transaction), *deferred* (takes place at the commit point of the current transaction), *decoupled* (takes place in a separate transaction).

Another difference among existing systems is whether rule definitions are attached to classes (*targeted rules*). Attaching rule definitions to classes enhances modularization and allows an efficient detection of relevant events while there are sometimes useful rules triggered by events spanning sets of objects possibly from different classes (*untargeted rules*).

Another notion that is also extremely relevant for active rule semantics is that of *priority*. The execution semantics for active rules sometimes requires that one rule is selected from a set of eligible rules. For this reason, an active database rule language may include a mechanism for specifying rule priorities.

Another important aspect in introducing triggers in object-oriented databases concerns trigger inheritance and overriding. Since inheritance is one of the most significant features of the object-oriented paradigm this is a fundamental problem to cope with. In spite of that, the influence of inheritance on triggers has not been deeply investigated in existing object-oriented database systems. Under some proposals [2, 7, 14], triggers are always inherited and can never be overridden nor refined. Problems concerning trigger inheritance and overriding have been investigated in [3].

3 Comparison

In this section we identify some differences between the behavior that can be achieved through the specification of a trigger and the specification of a `try` block handling an exception. Of course, the trigger would react to a simple implicit event (explicit -i.e., temporal- events or composite events do not seem to have a counterpart in the exception mechanism). The first obvious difference that emerges is that events are (usually) not explicitly raised, and triggers are not explicitly activated, rather, the occurrence of the monitored event autonomously causes the triggering of the rule (this is the reason to call it *active* rule), whereas exceptions must be explicitly thrown and caught. In addition to that, moreover, we can identify several other differences:

- A first important difference is represented by the fact that a fixed event (independently of where it occurs) always triggers the same rules; by contrast, an exception can be handled in different ways depending on where it is caught. Exceptions have been indeed originally conceived as a mean to notify the caller of the fact that an exceptional situation has occurred (typically error situations), the reaction to the receipt of this information is thus highly dependent on the context.

- The action of the trigger is executed in addition to what the triggering transaction is doing, except than in two cases: instead-of triggers (whose action is executed instead of the triggering operation) and triggers whose action is a `rollback` (by effect of which the whole triggering transaction is undone). By contrast, the body of a `try` statement is executed only until an exception is thrown; when an exception is thrown, the remainder of the block is not executed and we pass to examine the `catch` clauses of the block.

- The occurrence of an event triggers all the rules monitoring that event. This means that an active system must handle problems related to rule priorities, how to choose a rule from the set of triggered rules, recursive and non-terminating behavior, etc. By contrast, exceptions are always caught by a single *catch* clause; if several catch clauses of a try block match with a thrown exception, only the block corresponding to the first catch clause is executed.

- Exceptions are characterized by a very fine "processing granularity", that is, the reaction to the throwing of an exception is super-immediate. By contrast, as we have discussed in Section 2, the processing granularity of triggers can be rather coarse (due to a lot of reasons).

- For what concerns inheritance, triggers and exceptions present deep differences. Usually triggers are not overridden in subclasses and full trigger inheritance is supported. In addition, new triggers can be defined in subclasses. By contrast, when a method is overridden in a subclass, only the exceptions that have been specified in the method in the superclass or a specialization of them can be thrown.

Thus, in some sense, triggers seem more high-level and more declarative constructs. Moreover, with the use of exception seems more inherent the risk of "spreading" the semantics of data, which can be expressed through a single trigger. By contrast, the exception mechanism is more powerful in that it allows to express different reactions to the same situation.

Finally, another important difference we would like to point out is related to static analysis. While each method declares the exception it can throw, and correctness of these declarations is checked statically, the relationships among triggers are not explicitly declared and it is not trivial to deduce them. A lot of research efforts have indeed been devoted to the development of static analysis techniques for sets of triggers [6, 15].

4 Conclusions and Open Issues

One of the main reasons which has led database researchers to introduce triggers in relational DBMSs has been the need to increase the possibility to represent within the database the semantics of data. Triggers are defined as part of the database schema and represent the active behavior of the system when some specified events occur. Thus, not only how data are organized is represented at schema level, but, somehow, data behavior can also be specified.

This aspect of the relational model is completely overcome in the object-oriented model. Since in the object-oriented model interfaces and classes provide both a structural representation (attributes) and a behavioral representation (methods) of data, the model seems to be rich enough to meet the needs of every data domain.

Thus, at first sight, triggers seems to be useless in object-oriented databases. We do not believe this is true. Indeed, triggers offer a centralized control of the semantics of data compared with the dissemination of such semantics in methods. Consider for instance integrity constraints: since a single constraint can be potentially violated by the execution of different methods, the code for enforcing the constraint should be replicated in all those methods. Triggers are high level constructs whose activation and execution can be managed by an ad-hoc part of the system. On the other side exceptions represent the active part of an object-oriented database from the programming side.[1] They represent a reaction to a problem as well. We believe that two research directions can be devised in between the two.

The first research direction aims at investigating if the way in which exceptions are implemented in object-oriented programming languages can offer useful insights to efficiently support triggers in OODBMSs. For instance, in the case of Java it could be useful to analyze the management of exceptions at the JVM level. Studying the *bytecode* generated for exceptions, which can be regarded as an active feature of the programming language, could help in generating the byte code for triggers which are active DBMS features.

The second research direction is less technical and more "philosophical". The idea is to investigate whether or not exceptions can offer an alternative to triggers to manage, for instance, integrity constraints. According to this approach the semantics of data which is expressed through triggers is embedded in the application itself which is represented by methods throwing exceptions. Probably advantages and disadvantages exist in both the approaches.

We believe that a "meeting point" between triggers and exceptions exists, we cannot say more about where it is.

References

[1] K. Arnold and J. Gosling. *The Java Programming Language Second Edition.* Addison-Wesley, 1998.

[2] C. Beeri and T. Milo. A Model for Active Object Oriented Database. In G. M. Lohman, A. Sernadas, and R. Camps, editors, *Proc. Seventeenth Int'l Conf. on Very Large Data Bases*, pages 337–349, 1991.

[3] E. Bertino, G. Guerrini, and I. Merlo. Trigger Inheritance and Overriding in an Active Object Database System. In F. Bry et al., editors, *Proc. Fifth International Conf. on Deductive and Object-Oriented*

[1] Note that in OODBMSs methods are usually written in an object-oriented programming language and no ad-hoc manipulation language is provided.

Databases, number 1341 in Lecture Notes in Computer Science, pages 193–210, 1997. Extended version to appear on IEEE Transactions on Knowledge and Data Engineering.

[4] E. Bertino, G. Guerrini, and I. Merlo. Triggers in Java-based Databases. In *Proc. OOPSLA'99 Workshop on Java^{TM} and Databases: Persistence Options*, 1999. Denver (Colorado).

[5] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Active Rule Management in Chimera. In S. Ceri and J. Widom, editors, *Active Database Systems*. Morgan-Kaufmann, 1996.

[6] S. Ceri and J. Widom. *Active Database Systems - Triggers and Rules for Advanced Database Processing*. Morgan-Kaufmann, 1996.

[7] C. Collet, T. Coupaye, and T. Svensen. NAOS: Efficient and Modular Reactive Capabilities in an Object-Oriented Database System. In *Proc. Twentieth Int'l Conf. on Very Large Data Bases*, pages 132–143, 1994.

[8] U. Dayal, A. Buchmann, and S. Chakravarthy. The HiPAC project. In S. Ceri and J. Widom, editors, *Active Database Systems*. Morgan-Kaufmann, 1996.

[9] O. Diaz, N. Paton, and P. Gray. Rule Management in Object Oriented Databases: A Uniform Approach. In G. M. Lohman, A. Sernadas, and R. Camps, editors, *Proc. Seventeenth Int'l Conf. on Very Large Data Bases*, pages 317–326, 1991.

[10] S. Gatziu and K. Dittrich. SAMOS: an Active Object-Oriented Database System. *IEEE Data Engineering Bulletin, Special Issue on Active Databases*, 15(4):23–26, December 1992.

[11] N. Gehani and H. Jagadish. Active Database Facilities in Ode. In S. Ceri and J. Widom, editors, *Active Database Systems*. Morgan-Kaufmann, 1996.

[12] G. Kappel, S. Rausch-Schott, and W. Retschitzegger. Beyond Coupling Modes: Implementing Active Concepts on Top of a Commercial ooDBMS. In E. Bertino and S. Urban, editors, *Proc. Int'l Symp. on Object-Oriented Methodologies and Systems*, number 858 in Lecture Notes in Computer Science, 1994.

[13] D. McCarthy and U. Dayal. The Architecture of an Active Data Base Management System. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 215–223, 1989.

[14] C. Medeiros and P. Pfeffer. Object Integrity Using Rules. In P. America, editor, *Proc. Fifth European Conference on Object-Oriented Programming*, number 512 in Lecture Notes in Computer Science, pages 219–230, 1991.

[15] N. W. Paton. *Active Rules in Database Systems*. Springer New York, 1999.