14

Visualization of rule behaviour in active databases

Fors, T.
Department of Computer Science
University of Skövde, Box 408, 541 28 Skövde, Sweden
email: tomas@ida.his.se

Abstract

One of the characteristics of database systems is the ability to efficiently handle large amounts of information. This characteristic must apply also when rules are introduced into database systems, resulting in so-called active database systems. Hence, active database systems need to manage both a large amount of data and many rules.

One of the problems with large sets of rules is the complexity of interaction between rules. A rule that is perfectly workable by itself may in the interaction with other rules produce serious malfunctions. To prevent such situations, a rule debugger is needed, preferably a visual one.

In this paper we introduce Adela, a rule visualization tool suitable for the relational data model. We introduce a multilevel rule concept, which is important when, for example, rule visualization tools need to display how rules are related to the data model and the data instances. Three different levels within the multilevel rule concept are provided: model level, rule level, and instance level. An event/rule trace tree is introduced to display: i) the context in which rules are fired, and ii) the execution states of rules. A data trace is introduced in order to display what is happening to data instances when they are manipulated by rules. We also address issues such as scalability of rule visualization tools, and on-line rule visualization tools, which in real time show what is happening in active databases.

Keywords

Rule visualization, active databases

S. Spaccapietra et al. (eds.), *Visual Database Systems 3*

1 INTRODUCTION

The introduction of rules into database systems brings in the complexity of rule interaction and behaviour. Exactly what is happening in an active database at any given moment? The user does not know. In fact, there is seldom any need for the user to know. But what if something malfunctions?

Rules in an active database are executed "behind the scenes", meaning that the users of the system will not see that these rules are executed. Assume that the rule set contains some rules implemented in a fashion that will lead to a highly undesirable state if they are ever triggered and executed. Since the rule execution is done behind the scenes, the failure will be a fact without any preceding warnings. It will also be difficult to see what went wrong afterwards.

What is needed is a type of tool that can visualize rule behaviour, so that development and maintenance of active database rules is made much easier. Otherwise it will be very difficult to see the complex interactions between rules, which can hide the cause of a serious malfunction. A visualization toolkit would, if not solve this problem, then at least make it possible to prevent some failures, and if a failure does occur, make it easier to analyse the situation.

Database systems, whether they are active or not, have to efficiently manage large amounts of information. An active database system thus has to manage both a large amount of both data and many rules; a fact that may cause developers to hesitate in choosing an active database system. This can be exemplified as follows. Assume that a database system is being developed for an environment where there must not be any mistakes made by the system. There are two alternatives: Either you use an active database in which you can easily implement both data and behaviour, or you can use an ordinary passive database, for which you must implement the behaviour in all applications that use the database in question.

The active database alternative is more appealing since it is easy to update both behaviour and data, and applications will be smaller and need not be updated when some constraint changes.

Unfortunately, you will most likely be inclined to choose the passive system with all its disadvantages. Why? Simply because it is easier to understand what the system will do in a given situation. It is all there in the source code. Of course, when implementing an active system, the rules can also be checked, but only one at a time. There is no way of knowing what the system will do with all the rules interacting with each other. Diaz, Jaime, and Paton (1993) say that:

> Experiences using active rules in database systems has shown that, while such rules can be utilized beneficially in a range of applications, it is not a straightforward task to implement, debug or maintain large rule bases.

which is consistent with what Stonebraker (1993) says:

> An increasing expressiveness in rule languages is often associated with increasing complexity. As the number of rules increases, it becomes difficult to foresee the possible interactions among rules.

Some sort of debugging and explanation facility is thus important if the use of active rules is going to spread. Diaz, Jaime, and Paton (1993) mention two reasons for introducing such a facility:

- Inform the user which active rules have been fired during the execution of an operation thereby increasing the user's confidence and understanding of the system.
- To help the designer to refine and analyze interactions among rules at execution time.

The Adela project (Fors, 1994) is located within this problem area. That is, it is aimed at finding a way to visualize rules and rule behaviour. The visualization is aimed at making the context of each rule firing clear, and to make it easier to debug rule sets by making it possible to restrict the tracing to certain rules, events, or data.

The remainder of this paper is organized as follows. Section 2 contains a presentation of work done in the area of rule visualization. The DEAR rule debugger is discussed therein. In section 3, we propose the multilevel rule concept, which is a suggestion of how to define the role of rules in active database systems. We provide three different levels for the multilevel rule concept: model level, rule level, and instance level.

Section 4 describes Adela, a prototype rule visualization system that visualizes rule behaviour in a relational framework. This is followed in section 5 by a brief description of the features of Adela.

Finally, in section 6, we discuss future work in the area of rule visualization and present conclusions from the Adela project.

2 RELATED WORK

Since research within the area of rule visualization is not as yet well established, related work is very hard to find. This section contains an overview of the DEAR rule debugger (Diaz, Jaime, and Paton, 1993), the only related work found at the time of the Adela project (Fors, 1994).

2.1 DEAR

DEAR is a debugger for active rules in an object-oriented context (Diaz, Jaime, and Paton, 1993). It has been implemented on EXACT (Diaz and Jaime, 1993), which is a rule manager which supports ECA rules in the object-oriented database system ADAM (Diaz, Gray, and Paton, 1991).

A rule debugger should, according to Diaz, Jaime, and Paton (1993), provide at least three kinds of mechanisms which:

- Make explicit the context in which the active rule is fired.
- Focus the search during the debugging process.
- Automatically detect inconsistencies and potentially conflicting interactions among rules.

DEAR keeps track not only of rules, but of both rules and events. Diaz, Jaime, and Paton (1993) identifies that

> ... tracing events gives important hints to the user - the entwined event-rule cycle allows the user to know not only which rules fired, but also what caused these rules to be fired. And vice versa, when an event is awakened, such an entwined cycle permits detection of the context where the event arose.

Rules triggered at the same time, the conflict set, depend on the events raised. Therefore, to show the sequence in which rules are fired would be of little help here. A more meaningful way of keeping track of ECA rules is to show not only the rules but also the context in which these rules are fired (Diaz, Jaime, and Paton, 1993). The context can be understood as the rule's event and the conflict rule set. A drawback with only showing sequence when debugging

rules, is that such a debugger will ignore the fact that a rule has to compete with other rules in order to fire. DEAR attempts to face this problem by:

- Showing the entwined cycle of rules and events. Hence, the user can ascertain which rules are awakened and the cause of the fired rules. In this way, it is also possible to detect the context in which events were triggered.
- Making explicit the conflict set of simultaneously triggered rules in the displayed tree, once the conflicts among those rules have been resolved by the rule manager. An indication of the rules participating in this conflict set is useful for focusing on a restricted number of rules where complex interactions can occur.

The debug tree can easily grow too large to analyse. Therefore, DEAR also provides a number of ways to limit the debug tree produced:

- *Spy_event*. Restrict debugging to certain events.
- *Spy_rule*. Restrict debugging to certain rules.
- *Data_spy*. Restrict debugging to certain data, i.e., a tree will be generated only if an update is detected on the data monitored.

2.2 Discussion of DEAR

The first issue concerns what actually happens to the data as the actions of rules change it. DEAR does not show that clearly. It is our opinion that any rule visualization tool should in addition to visualizing rules, also show what effects rules have on data. That is, which objects were affected, in which way, how many, etc. It should also be possible to examine these objects, i.e. browse them.

DEAR is object-oriented and therefore has some special features that come with the object-oriented paradigm. If these ideas were to be fitted to the relational model, it is likely that we would turn from an instance-oriented view to a set-oriented view. Which data does a certain rule apply to at any given moment? There is no way to be sure. The only thing we know is what data set the rule applies to.

Another issue is that of integrity constraints. It often takes several rules to form one integrity constraint. Our opinion is that a visualization tool for active systems should show the mapping from rules to constraints and from constraints to the data model itself. DEAR does not provide this. However, it is not elementary to provide such a mapping. The fact that one rule may be part of several constraints makes the mapping very complicated. How do we know when one constraint is "active"? This problem is similar to that of composite events.

In summary, the following additional requirements and issues should in general be provided or considered when a rule visualization tool is being developed.

- A rule visualization tool should show how data instances are affected as they are manipulated by rules.
- A visualization tool should provide browsing facilities for both data and rules.
- Rules should be put into data model context, and the concept of integrity constraints should be supported.

3 THE MULTILEVEL RULE CONCEPT

To visualize rule behaviour as an event/rule tree does not solve all problems. There are several open questions such as: i) what roles do rules play in the data model, and ii) how are instances of rules, events, and data related? Questions like these also need to be answered by a rule visualization tool. The *multilevel rule concept*, which was introduced by Fors (1994), is an approach which makes it possible for a rule visualization tool to answer these questions.

A rule visualization tool must be able to show how rules are related to the data model and the data instances. That is what the multilevel rule concept is about. It gives a number of abstractions and relations between these abstractions in order to easily provide the tool with necessary facilities for required visualizations.

The multilevel rule concept was developed in order to attempt to define the role of rules in active databases. This is done by introducing three basic levels, which reflect different aspects of rule behaviour: the model level, the rule level, and the instance level.

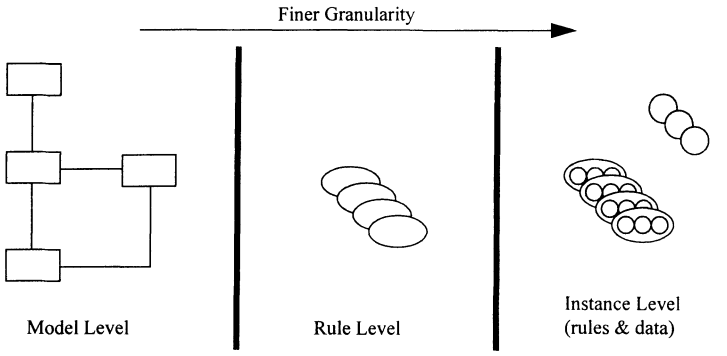


Figure 1 The multilevel rule concept.

3.1 Model level

At the model level, rules are set into a data model context, i.e. this level reflects the rule behaviour on the data model.

A constraint is intimately related to object-types or relations in the data model. For example, a constraint for referential integrity is bound to two relations in the data model. Integrity constraints are thus a part of the data model, and user expectations are governed by them. Not to show this connection would therefore be an oversight.

A constraint itself consists of one or several rules needed in order to define the behaviour needed with respect to the constraint. A constraint is in this respect much like a composite. Similar to composite events, in that several constraints may share one rule. A constraint can be seen as a "composite rule".

It is desirable to know when a constraint has been triggered. Therefore we need to keep track of which rules are triggered, so that when a rule (or rules) representing violation of a constraint is triggered, the constraint itself is triggered.

The rule-constraint problem can be outlined like this. Assume a set of rules *R* and a set of constraints *C*. For every constraint in *C* there are several rules in *R*. For any rule in *R* there may be more than one associated constraint, and for any constraint in *C* there are one or more rules and there may be rules which occur in other constraints. As rules are activated, the system has to check if any constraint is triggered as a result.

It has been mentioned that this problem is comparable to the event/composite event problem. But there is a difference: rules and constraints are concerned with database states, whereas events - in most cases - are operations on the database state. This is a complex and involved problem that will not be further analysed in this paper.

If this concept of constraints is supported within an active dbms, the only problem where the visualization tool is concerned lies in the actual visualization. But if the dbms does not support this, which is the case today, the tool itself must look after this. This is not really what a tool for rule visualization is supposed to do, so it is recommended that active database management systems are enhanced to support the multilevel rule concept.

Rule behaviour and its effects can on this level be visualized by colouring of schema symbols, sounds, text messages, and the usage of graphical symbols as messages.

3.2 Rule level

The rule level (the "middle" level) is the level on which the rule representation in DEAR could be positioned, i.e. it contains a representation of the rules themselves. Rules can be looked at from (at least) three different points of view, or sublevels. These are the constraint level, the atomic rule level, and the ECA level.

The point of having these levels is that it would be difficult to get a feeling for what is happening in the database if we were only to present animations on one single level. The motivation for each sublevel is as follows.

ECA level

As a minimum, a rule debugging environment must show what caused a rule to fire. What triggers a rule is an event, which is specified at the ECA-level.

Atomic rule level

When monitoring the entwined chain of events and rules, it is desirable to regard the whole rule as an abstraction, one single unit. So at this level, the rule level, the event, the condition, and the action parts of the rule are hidden behind the abstracted form of rule.

Constraint level

It is often the case that several rules are needed to express a constraint. It is therefore desired to be able view these rules in a higher form of abstraction, a constraint. It is the constraint that is broken down into individual rules when an active system is designed. Not to be able to see this abstraction afterwards would be an oversight.

3.3 Instance level

While the rule level keeps track of the rules in different forms of abstraction (e.g. constraints, rules as atomic entities, and the ECA parts of the rule), the instance level keeps track of the actual rule firings. To each rule firing comes a number of parameters, such as which event trig-

gered the rule, which data is of interest, the state the data was in when the rule was fired, etc.

Why is this level needed? First, as one rule can be triggered several times in one transaction, it is needed to separate these different rule instances from each other. Second, as instances of one rule can be triggered at different times, different database states will apply to different instances. Hence it is important to reflect rule instances.

The instance level is responsible for rule visualization in respect of the database state in which each rule instance was created.

As events can be seen as objects in a system as well, it follows that they have instances in a way much like rules. The occurrence of an event generates an event instance to which a set of context parameters belong. These parameters are then passed along to the rules triggered by this event occurrence.

4 ADELA: A PROTOTYPE SYSTEM

The objective of the development of Adela (Fors, 1994) was to design and implement a rule debugger. In general, the implementation would be an animated graphical interface, which would continuously show what rules are firing, and the effects of rule firings in a set-oriented *relational* framework.

Adela is implemented as a prototype in C++, using the animation library Goofy (Ford, 1994). Goofy is built on top of another library, Polka (Stasko, 1992), which is a software system developed by John T. Stasko at the College of Computing, Georgia Institute of Technology, Atlanta. Polka provides functions for smooth animation on top of the X-Window system.

Goofy allows indirect access to Polka through a higher-level interface. With these two packages, it is easy to make sophisticated animations without having to concentrate on the low-level details that lie behind it all.

4.1 Development restrictions

Adela was developed as a prototype with a number of quite severe restrictions. The reason to these restrictions was to make it possible to produce a reasonable result within the time limits of the project.

- Adela is an application, not a database module.
- The database monitor is simulated, not actually built to interact with a real active database system.
- The main objective is to visualize rules and rule firings in a data model context. The project was therefore primarily meant to explain rules, and only secondly to provide a debugging facility.
- Adela should naturally be easy to understand and work with. However, having that as a goal would require that an acceptance test of some kind would have to be carried out. The time limits of the project (Fors, 1994) did not permit such testing to be planned. Therefore, understanding and usability is not a formal project goal.
- The only rules considered are rules that represent behavior concerned with whole objects. All other types of rules, including those with composite events, will be ignored. So the only events considered are those when an object has been updated, deleted, inserted, or retrieved.
- Assume small data model. The problem of showing large amounts of information so that it can easily be comprehended by the user is ignored.

- Deal only with whole data objects on set level. Do not bother with attributes and methods. Regard the data objects as set units, meaning that the visualization of an instance is in fact the visualization of a set of instances.

These restrictions may be seen as very crippling, making the resulting visualization seem trivial. That is true in the sense that for debugging purposes, leaving out attributes, will make the tool quite useless.

However, the Adela project, as far as it has been taken, was primarily meant to explore a way to explain active rule behaviour visually. For that purpose, attributes may be left out - even if it is true that taking them into account would enhance the way that Adela visualizes rule behaviour. Unfortunately, that was not possible to achieve within the time limits of the Adela project.

4.2 Assumptions

Adela is intended to work in a relational context, meaning (among other things) that data is set-oriented. This will impact the visualization of rule actions with respect to changes to data, i.e., all changes are sets of changes.

It is also assumed that the underlying database system will have no problems in supporting the information requirements that Adela has. If Adela is to be developed into a full-scale tool, then there might have to be changes to the database itself. The change that is most obvious is the database monitoring facility, which must take care of all communication between the visualization tool and the active database. Its duties will be of an introspective nature, in which it will monitor triggering and execution of rules, event occurrences, and the buildup of composite events. It must also support browsing of rules, events, and data. Browsing of data that have been changed by rules is also an important feature. This is not a simple task, which due to the restrictions on the objectives was not undertaken.

Due to the lack of work in the area of visualization of rule behaviour, the visualization that is suggested is based more upon reasoning than on any work already done. However, some influences have been taken from DEAR and from existing ideas of how to graphically represent data schemas. The symbols and the approaches to visualization chosen are not explicitly designed to be cognitively correct and easy to understand. They are merely intended to give an illustrative example of how rule behaviour and its effects can be visualized. As mentioned in section 4.1, understanding and usability is not a formal project goal.

4.3 Architecture

Database Monitor

In order to be able to show on screen what is happening in the active database, certain information is needed. This information should be provided by a database monitoring facility, which in the Adela prototype is a simulator of an active database.

Event Information Filter

The event information filter takes input from the database monitor and processes it into something that the graphics manager can handle. The information from the database monitor is mapped into screen objects and animation data.

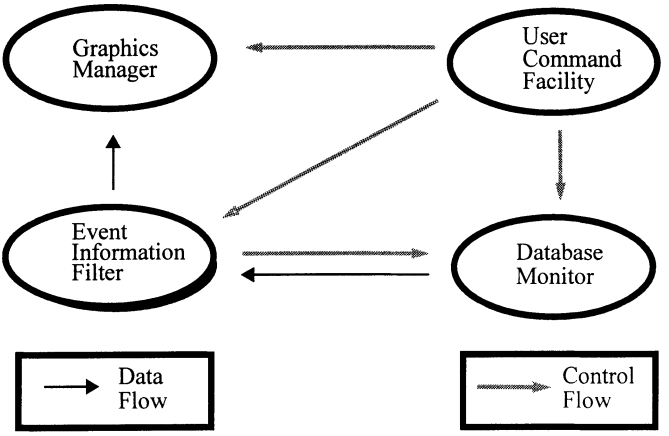


Figure 2 Architecture of the Adela system.

Graphics Manager

The graphics manager is nothing more than a slave to the event information filter. It takes input from the event information filter and takes care of the animations required.

User Command Facility

The user command facility provides the user of Adela with a number of commands to control the visualization. Due to some features of the Goofy animation library, which made it moderately difficult to add other windows than those predefined, the user command facility was not implemented. Since Adela is a prototype system intended only to show a way to visualize rules in a relational active database, the absence of this facility was seen as unimportant.

4.4 Rules

4.4.1 Events in Adela

In many systems, events can be composite. Events can therefore be put together with the use of certain operators, like conjunction and disjunction. In Adela it is unimportant how events are detected, so composite events could easily be incorporated. Whether the structure of composite events should be visualized, and more particularly whether partially detected composites should be, are open questions.

```
Event Definition
    Event identifier
    Event signal
    Object type identifier
```

Figure 3 Adela event definition.

An event consists of the following parts: an *identifier*, an *event signal*, and a binding of the *object type* involved. The event signal is a database operation, i.e., one of insert, delete, update, or retrieve. This means that the ECA style that Adela adopts conforms to the relational paradigm. The object type refers to be base relation on which the database operation operates. E.g. for the event E1, INSERT to PERSON, INSERT is the event signal and PERSON is the object type.

4.4.2 Actions in Adela

The definition of actions in Adela is similar to the event definition, except that actions are not objects themselves, i.e., do not have an identifier. Another exception is that the signal is one of insert, delete, update, retrieve, *refuse*, and *instead-of*. For the purposes of Adela, it is not necessary to express rule actions in any more detail than this.

The refuse operation is a rollback of the whole transaction (immediate-immediate coupling assumed), and the instead-of operation means that the action triggering the event is replaced by the action of the rule with the instead-of-action. This has some consequences for the model (see section 4.4.3).

Complex actions are supported in that it is possible to, for example, delete an employee and insert an unemployed as part of the same rule action. Note that for refuse, it is not possible to express a complex action. Whenever refuse is chosen as an action, it must be the only action in the action part of the rule. A similar thing goes for instead-of actions. It is not possible to express an action to be both instead-of and something else. Note that an action that is specified instead-of can naturally be complex in itself.

4.4.3 Instead-of semantics

Suppose an event initiated by a rule, user, or an application has triggered several rules, one of which has an instead-of-action. Then the action triggering the event is replaced by the action of the rule with the INSTEAD-OF in its action.

Other rules triggered by the same event will be dropped. This is because the action triggering the event is undone by the rule with the instead-of-action. If there exist rules which have been triggered by the same event and whose action parts have been executed prior to the rule with instead-of, then the changes made by those are also undone.

If one or several other rules triggered by the same event have caused cascades of rule firing, and if the rule with the instead-of-action is executed to completion, then the rule cascades made by the other rules have to be undone.

4.4.4 ECA rules in Adela

There is a problem of how to visualize the rule concept since a limitation to the objectives for Adela was that attributes would not be modeled; the rule styles of "normal" ECA rules are too complex.

This problem is solved by redefining events, conditions, and actions with respect to Adela. (See Figure 4) Rule priority is represented by a number, where low numbers are high-priority, and high numbers are low-priority.

Because the prototype implementation of Adela will not deal with attributes, the condition is highly compromised. In the Adela ECA rule, the condition is either always false, always true, or randomly true. One might ask why the condition is represented at all. There are two answers to this. The first answer is that the condition (i.e., state check) is a significant part of rule behaviour and as such must be present in a visualization of rule behaviour. The natural way to reflect the fact that a condition can be evaluated as either true or false, taking the restrictions into account, is to let the condition be randomly true.

```
Rule  identifier

Priority

ADELA  event:      event  identifier

ADELA  condition:  <  FALSE  |  TRUE  |  RANDOM  >

ADELA  action:     <list  of  action  definitions>
```

Figure 4 The simplified ECA rule form of ADELA.

The second answer is that in debugging rules it is very cumbersome to only rely on pure chance. One might wish to control the way that rules are triggered and executed. Therefore, with respect to the condition, there has to be a possibility to overrule chance, i.e., to set the condition to always be true or always be false.

5 VISUALIZATION IN ADELA

Due to the time limits of the Adela project, a number of restrictions (section 4.1) were introduced. This means that when regarding Adela from the multilevel rule perspective, only part of the rule level is represented. If Adela is taken any further, the whole multilevel rule concept should be supported.

When visualizing rule behaviour, especially when showing the effects on data, an important question arises: just how much should be animated of the effects on data? Suggestions are:

- **Rules and object types**. Rules are connected to certain object types. This connection can be derived from i) the event definition, and ii) the action definition. If rule R1 is connected to object type 'Person', the connection should be apparent in the visualization.

- **Database operations.** When an event occurs, the animation visualizes the object type in question so that it is clear what is happening, e.g., a DELETE operation on the object type 'Person' would cause the visualization of 'Person' to change so that it is clear that somebody or something has executed a delete command on the object type in question.
- **Rule firings.** When a rule fires, it should be noticed on the object type it is connected to. If the rule itself is viewed, then it should be noticed that it has been activated. The latter is of course obvious in the event/rule trace, but that is not the point. The point is to clearly visualize the connection between the rule and the object type.

5.1 Example

The visualization as implemented in Adela (Fors, 1994) provides two animation windows, the event/rule trace view and the data trace view. The event/rule trace view visualizes the cascade of events and rules and the execution state of each rule as it is fired. The data trace view visualizes the operations of the rule actions.

Since attributes and instances are ignored in the prototype (section 4.1), the data trace view only shows operations by rules on object types. The changes to data are nevertheless animated, but the "instances" are not really instances of object types: they are to be regarded as sets of instances.

5.1.1 Graphical symbols

Event/rule trace

Events are squares which are coloured black when no rules in the branch originating from the event in question is being executed, and gray when there are rules in the branch originating from the event in question that are being executed.

Rules are ellipses that contain three circles. The circles are used to show in which stage of execution the rules are. If all circles are black, then the rule is not being executed. If the leftmost circle is gray and the other two circles are black, then the rule has just been selected for execution. If the rightmost circle is black and the two other circles are gray, then the condition has been evaluated as true. If all circles are gray, then the action of the rule is being processed.

Each node is labelled with a node identifier, which is constructed out of the name of the event/rule, tree depth level, and a count of how many times the event/rule has occurred previously in the tree. This node identifier is used to uniquely identify each event/rule instance.

The arcs between events and rules can be labelled with an 'x'. This means that the tree branch in question has been processed.

Data trace view

Rules are represented in a manner that differs from the representation of rules in the event/rule trace. Here, the execution states are not seen: rules are at all times represented by solid black ellipses. The reason for this is that the data trace view is not dealing with anything other than effects of rule action processing. Therefore to show the execution states of the rules would be to provide irrelevant and redundant information.

Object types are visualized as solid black rectangles. The connection between rules and object types as described in section 5.1 is not implemented completely. The only way rules and object types are visually connected is when the action of a rule performs an operation on an object type. The arcs that connect rules and object types are labelled with the operation.

Changes to instances of object types are visualized as small circles that are labelled with the name of the object type and the node id of the rule. The fill colour of such a circle is white, gray, or black. White circles denote unchanged data, i.e., inserted or retrieved. Gray circles denote changed data, i.e., updated, and black circles denote deleted data.

5.1.2 Visualization description

The steps described below outline the way that rule behaviour is visualized in Adela. (See also Figure 5 and Figure 6).

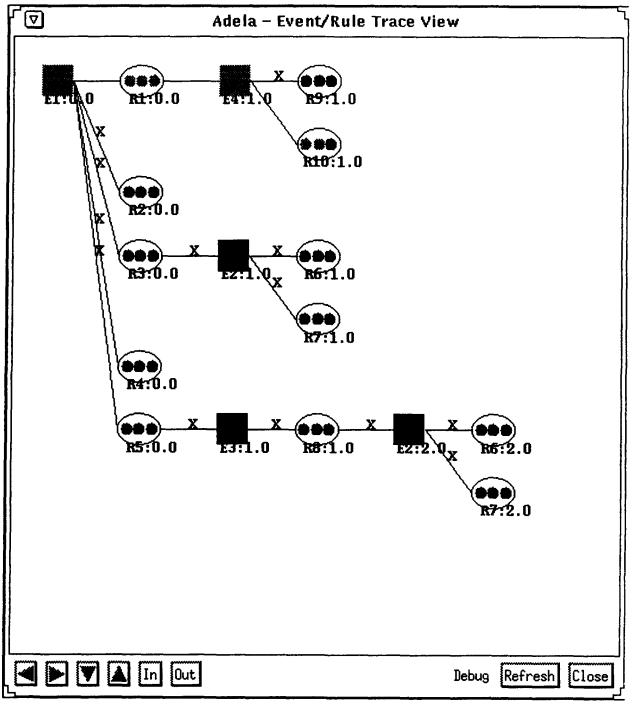
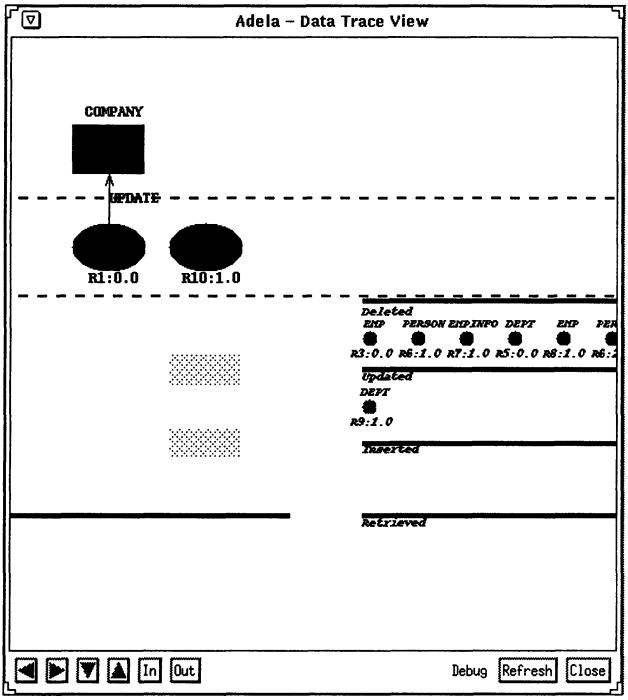


Figure 5 Event/rule trace view example.

1. The detection of an event causes an event node to appear in the event/rule trace view (Figure 5).
2. Rules triggered by an event form a conflict set, which in the event/rule trace view (Figure 5) is visualized in the form of a vertical row of rules (ellipses with circles within them). Each rule in the row is connected with an arc to the triggering event.
3. When a rule has been selected from its conflict set, it appears in the top part of the data trace view (Figure 6).

4. When the action of a rule is being processed, the object types that the action operates on are shown above the rules in the data trace view (Figure 6). The object types are connected by an arc to the rule when the operation is being performed.
5. The operations of rule actions are animated as follows: A white-colored instance symbol appears on the left/mid part of the data trace view (Figure 6) and moves to the right. It stops briefly when it reaches the space between the two grayish rectangles, where it changes color to black if it is to be deleted, and to gray if it is to be updated. This signifies the execution of the operation. After this, the instance symbol continues its movement to the right and settles in the appropriate slot (the slots are labelled deleted, updated, inserted, and retrieved).

When the processing of the action has finished, the rule object types operated on by the rule disappear. Shortly after that, the processing of the rule itself is finished, whereafter it also disappears from the data trace view (Figure 6).



ıre 6 Data trace view example.

6 FUTURE WORK

The ideas around visualization of rule behaviour that have been presented in this paper are not enough if one tries to construct a fully-fledged rule visualization tool. The ideas of Adela have therefore to be extended.

If the Adela system is further developed, it is very important to have usability as a formal project goal. This is because Adela is meant to be a tool for understanding the rules within an active database, and to do actual debugging. As a consequence, feedback from potential users is vital.

The usability goal implies that none of the restrictions mentioned in section 4.1 must be present in the Adela system if it is further developed. In addition, issues such as these below will have to be addressed.

The multilevel rule concept

In its current state, Adela has not a very good representation of the multilevel rule concept. The model level, the rule level, and the instance level should in all their aspects be visually represented.

Database monitor

For a fully-fledged rule visualization tool, the presence of database monitoring is crucial if the tool is to be used for debugging of rule sets.

Protocol for semantics

Even though ECA rules are accepted as a basis for active databases, the semantics of these rules differ from implementation to implementation. Having a separate rule visualization tool for every type of active database is obviously a very cumbersome approach.

The alternative is to build a generic rule visualization tool which can handle all types of rule semantics. What is needed in this case is a protocol for semantics, so that it does not matter which database system is currently used; its semantics will always be reflected correctly by the rule visualization tool.

Composite events

Composite events are a natural part of an active database system and provide powerful possibilities to express when rules shall be triggered. Naturally, a fully-fledged tool for showing rule behaviour must support this feature. Since the responsibility for knowing when composite events occur lies with the dbms, the only issue here is how to visualize events versus composite events in an intuitive way.

A suggestion for how to visualize composite events is to visualize the different composite events as they are "built". The first problem lies within the database: can the event monitor in the dbms report these things? With respect to the event/rule trace tree, a composite event need not be any different from an atomic event: an event is an event. The second problem relates to scalability; vast numbers of "incomplete" composites may be generated which are ultimately irrelevant.

To enhance the debugging facilities it might be useful to provide means to visually reprioritize and suppress rules as well as rule groups, and to manipulate the rules themselves with respect to the different ECA parts.

Exactly how this is done will depend on whether Adela is meant to monitor the database, or

to be a simulator. If Adela is to be a monitor, then database functionality can be used to do the actual manipulation, while Adela provides the visual interface of that manipulation. On the other hand, if Adela is only to be a simulator of rule behaviour, then a subset of the manipulation functionality in an active database must be implemented in Adela.

7 CONCLUSIONS

Visualizing rules in database systems is a new area of research, in which very little work has been carried out. In this paper we have briefly described a rule visualization tool for the relational model: Adela. This is in contrast to DEAR, which is a similar tool for the object-oriented model.

We have also introduced the multilevel rule concept, in which different aspects of rule behaviour can be captured. The multilevel rule concept results in three different levels: model level, rule level, and instance level. The purpose of the model level is to set the rules in a data model context. The purpose of the rule level is to represent the different aspects of the rules themselves. The purpose of the instance level is to represent event-, rule-, and data instances. The multilevel rule concept is important when rule visualization tools need to display how rules are related to the data model and/or the data instances.

Adela provides the user with an event/rule trace tree which makes it possible to display the context in which rules are fired. The event/rule trace tree can also display the execution states of rules, i.e. when the rule has been selected for execution, when the condition of the rule has been evaluated, and when the action of the rule is being processed. We also believe that it is important that rule visualization tools can provide a data trace. The objective of the data trace feature is to display what is happening to data instances when they are manipulated by rules.

Since visualization of rules in database systems is a fairly new research area several open questions remain. For example, it is not clear if it is feasible to use rule visualization tools for on-line monitoring of events in an active database.

The Adela prototype system with all its restrictions does not allow for large rule sets or data models. A full scale system must be able to cope with very big rule sets and data models. One way to approach this problem may be to develop a visualization which visualizes everything in such a way that patterns in the information are made clear. One possible way to do this might be to enhance the event/rule trace tree with distinctly different colours for different types of nodes and make the nodes so small that even a very large tree could easily fit within the animation window.

So it may be possible to have a visualization tool which is *scalable*. But what about *speed* and *massive information*? One of the demands on databases is that they do their job *fast* and *efficiently*. Can an on-line visualization tool be developed to match this? As the Adela prototype is implemented, there is absolutely no way it can be made to visualize in real-time. Even if it could be done - would any of the visualizations make any sense to the user? Everything would probably just be a big blur if nothing radical is done to the visualization itself in order to present sensible animations at high speed. Obviously much work lies in this.

But is it necessary to have an on-line visualization tool at all? Might it not be enough to visualize slowly, on the basis of some sort of database log? For most applications this may very well be enough. But as always, there is an exception to every rule, meaning that there will probably be applications that absolutely have to have on-line visualization of rule behaviour.

8 REFERENCES

Diaz, O. and Jaime A. (1993) EXACT: an extensible approach to active object-oriented databases. *Submitted for publication*, 1993.

Diaz, O., Jaime A. and Paton, N.W. (1993) DEAR: a debugger for active rules in an object-oriented context, *in Rules in Database Systems*, Springer-Verlag.

Diaz, O.,Gray P. and Paton, N.W. (1991) Rule management in object-oriented databases: A uniform approach, *in Proceedings of the 17th International Conference on VLDB.*

Ford, L. (1994) Goofy animation system, *Technical Report 266*, University of Exeter, UK.

Fors, T. (1994) ADELA: animated debugging and explanation of active database rules, *M.Sc. dissertation*, department of computer science, University of Skövde, Sweden.

Stasko, J.T. (1992) POLKA animation designer's package, College of computing, Georgia Insitute of Technology, Atlanta.

Stonebraker, M. (1993) The integration of rule systems and database systems, *Memorandum No. UCB/ERL M93/25*, Electronic Research Laboratory, College of Engineering, University of California, Berkely.

9 ACKNOWLEDGEMENTS

The author wishes to thank Dr Brian Lings of the University of Exeter for giving the inspiration that lead to the Adela project, and Mikael Berndtsson of the University of Skövde, who has given much help in making this paper to what it is.

10 BIOGRAPHY

Tomas Fors got his M.Sc. degree in computer science in the fall of 1994 from the University of Skövde, Sweden. He is presently working as a database consultant in Upplands Väsby, Sweden.