Checking Inconsistency of Rule Sets in Active Real-time Databases*

Jian Zhang
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing 100190, China
Email: zj@ios.ac.cn

Abstract

Using active rules in database systems provides a kind of abstraction and flexibility. But it may also be difficult to use the rules properly, especially when they involve timing constraints. In this paper, we define a type of inconsistency between Event-Condition-Action (ECA) rules, and propose an approach for checking the inconsistency automatically.

Keywords: *active databases, real-time databases, ECA rules, consistency checking, constraint solving*

1 Introduction

Active database systems provide a powerful mechanism for specifying and processing rules [9, 6]. They can take actions on their own, to maintain certain kind of consistency. Such a rule usually consists of three parts: *event*, *condition*, and *action*. Intuitively, it means that, when the *event* happens, if the current database state satisfies the *condition*, then the *action* should be taken. Currently the active rule feature has been adopted in some commercial database management systems.

A set of active rules should have certain properties, such as termination (that is, it should not be able to trigger infinite computation). Another property is that two rules should not tell the system to perform different actions. This kind of consistency will be studied in the current paper. It is not always easy to design a rule set having all these properties.

It is even more tricky when we use active real-time databases [6, 3]. Such an application usually involves some timing constraints. For active real-time databases, past research works mainly focus on the run-time monitoring of the constraints (see e.g., [8]). However, we think that it is quite important to check the rules before using them. In this paper, we focus on a particular property, i.e., inconsistency between the rules. We define a concept called *EC-conflict*, and propose a constraint-based method for checking this kind of inconsistency.

*Partially supported by France Telecom under grant No. 46135653 and the National Science Foundation of China (NSFC) under grant No. 60673044.

2 Rules and Events

In this section, we recall some basic concepts like active rules and events. In active database systems, we usually use Event-Condition-Action (ECA) rules. Such a rule typically takes the following form:

$$ON\ Event\ \ IF\ Condition\ \ THEN\ Action$$

It specifies *when* to fire the rule (by the Event), *what* to check (by the Condition) and *how* to react (by the Action).

The following is a simple example of ECA rules:

$$\begin{array}{rl} ON & patient\_coughing \\ IF & patient\_temperature > 39 \\ THEN & notify\_nurse \end{array}$$

Here is another example [4]: When one withdraws from an account (Event), if the current balance minus the amount being withdrawn is below the minimum amount required (Condition), then indicate the maximum amount that can be withdrawn (Action).

An event can also be associated with timing constraints. For example, "in the evening" (i.e., the current time is later than 18:00). The condition in a rule is usually a relationship between attribute values.

During the execution of the system, when an event is detected, some rule may be triggered and the corresponding action performed. An action may in turn generate some new event (e.g., "update T"), and trigger other rules.

When there are many rules and some rules are complex, the triggering of rules may become unpredictable, and the system's behaviour will be hard to understand. Given a set of ECA rules, we should check if it has some properties like termination and consistency.

1530-2059/08 $25.00 © 2008 IEEE
DOI 10.1109/HASE.2008.58

Event Specification

In [4], an expressive event specification language called *snoop* is presented. It supports temporal, explicit, and composite events in addition to the traditional database events.

In *snoop*, events are broadly classified into primitive events and composite events. Primitive events are further classified into database events, temporal events and explicit events. A temporal event can be an absolute event (which happens at some specific point on the time line) or a relative event (specified using a reference time point and an offset). Explicit events are detected by application programs. They should be registered with the system before they can be used.

A composite event is usually formed from several primitive events using some operators such as disjunction, conjunction and sequence. Disjunction of two events E_1 and E_2, denoted $E_1 \vee E_2$, occurs when E_1 occurs or E_2 occurs. For more details about such operators and their semantics, see [4].

One may also specify periodic events using the periodic event operators. An example of such events is, sample IBM stock every 30 minutes from 8am to 5pm. Let us use $occ(ev, n)$ to denote that the event ev occurs at time point n. So, for a periodic event a that occurs every 5 seconds, we have, $occ(a, 5)$, $occ(a, 10)$, and so on.

3 Static Checking of Inconsistency

3.1 The Problem

Suppose there are two ECA-rules (R_1 and R_2):

ON ev_1 IF $cond_1$ THEN act_1
ON ev_2 IF $cond_2$ THEN act_2

If there exists an event satisfying both ev_1 and ev_2, and there are attribute values satisfying both $cond_1$ and $cond_2$, we say that there is an *EC-conflict* between the rules R_1 and R_2. Such a conflict means that it is possible to fire two rules at the same time. The actions in the two rules are usually different, which means that there is some kind of inconsistency. Even if the actions are the same, that indicates some kind of redundancy. In any case, it is desirable to send a warning message to the designer of the rule set.

To check for potential inconsistency between the two rules R_1 and R_2, we need to find a time point t by which the events ev_1 and ev_2 are considered to have occurred, and the conditions $cond_1$ and $cond_2$ are both satisfied. We turn to constraint solving technology to perform the checking.

3.2 Constraint Solving

Constraint satisfaction problems (CSPs) have been studied for many years. For an introduction to this area, see for example, a recent book by Dechter [5]. Here we give an informal description of CSPs. In such a problem, we have a set of variables (x_i), each of which can take values from some domain (D_i); we also have a set of constraints over the variables, e.g., $x_1 + 5x_2 = 11$. The goal is to find a value for each variable, such that all the constraints hold. For example, if we assign the values 1 and 2 to x_1 and x_2 respectively, the above equation holds. If there are more equations, they may not be satisfiable as a set of constraints.

CSP is a general framework. Constraints may take various forms. As a special case, if all the variables are Boolean variables, and each constraint is a clause (a kind of Boolean formula), the problem is known as SAT, a famous NP-hard problem. Despite the complexity of the problem, people have developed many efficient SAT solvers.

In the most general case, CSPs can be undecidable. For example, when there are integer variables and non-linear operators like multiplication, there is no algorithm which can find a solution for arbitrary constraints or conclude that there is no solution. However, usually we may put some restrictions on the constraints, and the CSP can be solved automatically. Currently there are many tools for solving CSPs. They are generally called constraint solvers or constraint programming systems.

3.3 Condition Checking

As mentioned earlier, to check for EC-conflict between two rules, we should decide the satisfiability of the conditions in the rules. In other words, we need to find values for the variables (or attribute values) in the conditions, such that both conditions hold. Depending on the rule language (esp. what kind of conditions are allowed), we can choose different constraint solvers, and use them either directly or transform the conditions to a form that is acceptable to the solvers.

Several years ago, we developed a tool called BoNuS [10, 11]. A motivation for that work is checking the consistency and completeness of a kind of formal specifications. The tool integrates a Boolean constraint solving algorithm with the linear programming package `lp_solve`.

From the user's point of view, the tool accepts constraints written in a natural form as input, and the variables can be in various data types. The constraints can have both Boolean operators (like AND, OR, NOT) and linear arithmetic operators (like addition). But non-linear constraints (e.g., $x^2 + yz = 3$) are not allowed.

With a constraint solver, we can conclude that, for example, the following two conditions may both be satisfied:

$$(1.5 * salary < 2000)$$
$$(salary > 1000) \wedge (salary < 6000)$$

Using BoNuS, we got a solution, i.e., $salary = 1001$. The

tool can also be used to check the satisfiability of Boolean formulas, but it is not so efficient as modern SAT solvers.

If the conjunction of the two conditions is not satisfiable, we conclude that there is no EC-conflict between the two rules. Otherwise, we still have to check the two event expressions in the rules.

3.4 Event Checking

In a distributed/multiprocessor environment, it is quite possible that several events occur at the same time. To check for the possibility of simultaneous occurrence of the two events, we translate the event expressions ev_1 and ev_2 (together with any timing constraints) into a set of constraints over the time domain. Note that the time domain is the set of non-negative integers, which is infinite. In case that the event expressions can be expressed in propositional linear-time temporal logic (LTL), we may use a decision procedure such as a tableau-like procedure to test the satisfiability of temporal logic formulas. But in general, we need to consider timing constraints, and we do not use that kind of procedure. Instead, we use a constraint-based approach.

Let us illustrate the basic idea using a simple example.

Example 1. Suppose we have two rules. The event expression in the first rule is a periodic event a that occurs every 5 seconds, and the event expression in the second rule is another periodic event b that occurs every 4 seconds. That is, $occ(a, 5)$, $occ(a, 10)$, ..., $occ(b, 4)$, $occ(b, 8)$, and so on. At the time point 20, we shall find that both a and b occur.

We may represent the occurrence time of a by $5x_1$, and that of b by $4x_2$. Here x_1 and x_2 represent two positive integers. To "prove" that the two events may happen at the same time, we just solve the constraint $5x_1 = 4x_2$.

More generally, assume that we have m independent integer variables x_i ($1 \leq i \leq m$), which can be used to represent the timings of event occurrences. Then we transform the timings of ev_1 and ev_2 to two expressions in terms of x_i: T_1 and T_2. We check if the constraint $T_1 = T_2$ is satisfiable. If yes, the events in the two rules may occur at the same time.

Example 2. Suppose that the first rule is the same as the first rule in Example 1. But in the second rule, the event expression is a disjunction of two events: $E_1 \vee E_2$. Here E_1 is an absolute event which occurs at time point 11, and E_2 is an independent event. Then we have, $T_1 = 5x_1$, $T_2 = 11 \vee T_2 = x_2$. Obviously the constraint $T_1 = T_2$ has integer solutions (e.g., $x_1 = 2$, $x_2 = 10$). So it is possible for the events in the two rules to occur simultaneously.

When it is possible for the two events to occur at the same time, and the two conditions can be satisfied (by giving appropriate values to the variables), we conclude that there is a conflict.

4 Concluding Remarks

As we mentioned in Section 1, most research works in active real-time databases focus on the run-time detection of events and monitoring of timing constraints. In contrast, there was little work on the static checking of various properties of rule sets. There are some works on the static analysis of active rules, e.g., [1], but they do not consider timing issues.

In this paper, we define a kind of inconsistency called EC-conflict between ECA-rules. We also propose an approach for checking that kind of inconsistency. Currently we use existing constraint solvers like BoNuS [11] for experimenting with the approach. Although the examples presented above are quite simple, we can use automated tools to deal with much more complicated specifications. We believe that automated tool support is both necessary and possible to help the designers of active rules. One of our future works is to develop a new tool that translates the event specification into a set of constraints.

Another issue is how to deal with conflicting rules. One way is to put priority on the rules. But it is still not so easy to prevent various undesirable properties. Kim and Chakravarthy [7] developed a rule execution model and a priority scheme to solve the problem.

In addition, it is quite interesting to analyze other properties like termination and confluence, which is quite challenging for real-time systems (esp. when there are also data constraints). In the future, we will try some verification tools like UPPAAL [2] to see if they can be applied for this purpose.

Acknowledgments

The author is grateful to Guihong Xu for giving pointers to some related papers and to Lisa Qiao for her comments on an earlier draft of this paper.

References

[1] A. Aiken, J.M. Hellerstein and J. Widom. Static analysis techniques for predicting the behavior of active database rules. *ACM Trans. on Database Systems (TODS)*, 20(1): 3–41, 1995.

[2] T. Amnell *et al.* UPPAAL – Now, next, and future. In: *Modeling and Verification of Parallel Processes (MOVEP 2000)*, LNCS 2067, 99–124, 2001.

[3] S.F. Andler *et al.* DeeDS Towards a distributed and active real-time database system. *SIGMOD Record*, 25(1): 38–40, 1996.

[4] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. Tech. Rep. UF-CIS-TR-93-007, Univ. Florida, 1993.

[5] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[6] J. Eriksson. Real-time and active databases: a survey. In: *Proc. ARTDB'97*, LNCS 1553, 1–23, 1998. Springer-Verlag.

[7] S.-K. Kim and S. Chakravarthy. A practical approach to static analysis and execution of rules in active databases. *Proc. of the 6th International Conf. on Information and Knowledge Management (CIKM'97)*, 161–168, 1997.

[8] A.K. Mok and G. Liu. Efficient run-time monitoring of timing constraints. In: *Proc. 3rd IEEE Real-Time Tech. and Applications Symp. (RTAS)*, 252–262, 1997.

[9] J. Widom. The Starburst active database rule system. *IEEE Trans. Knowl. Data Eng.* 8(4): 583–595, 1996.

[10] J. Zhang. Specification analysis and test data generation by solving Boolean combinations of numeric constraints. *Proc. 1st Asia-Pacific Conf. on Quality Software (APAQS'00)*, 267–274, 2000.

[11] J. Zhang and X. Wang. A constraint solver and its application to path feasibility analysis. *Int'l J. of Softw. Eng. and Knowl. Eng.*, 11(2): 139–156, 2001.