Chaudhri A., Jeckle M., Rahm E., Unland R. (Eds.): Web, Web-Services and Database Systems, NODe 2002, Revised Papers, Springer, LNCS 2593, 2003, pp. 199-205

Enhancing ECA Rules for Distributed Active Database Systems

Thomas Heimrich[1], Günther Specht[2]

[1] TU-Ilmenau, FG Datenbanken und Informationssysteme, 98684 Ilmenau
[2] Universität Ulm, Abteilung Datenbanken und Informationssysteme, 89069 Ulm

Abstract. ECA (event/condition/action) rules have been developed for central active database systems. In distributed active database systems the problem of inaccessibility of partial systems raises and thus the undecidability of ECA conditions referring to remote systems. This work proposes an enhancement of ECA rules for distributed active database systems to react also in the case of inaccessibility and undecidability. Therefore, the ECA evaluation will be enhanced to a strict function with the inaccessibility state Ω and a new alternative action AA enriches the classical ECA rules. The advantages and the usage of this approach are shown by an example of maintaining data consistency in distributed active database systems.

1 Introduction

Today, distributed working becomes more and more important in service enterprises, in field services, and many other areas. In most of these areas work cannot be done completely autonomously: decisions are based on local and remote data, systems have to react on changes at remote hosts or propagate own changes to remote servers. Thus during work online connections are often necessary. However, remote systems may be inaccessible occasionally. In case of active database systems this often leads to undecidable ECA conditions and rules. Up to now, this results in a dissatisfying wait state, if because of timeout people can not go on working since important information is missing.

By enhancing ECA rules with additional actions for the case of undecidability of ECA conditions (e.g. if remote systems are not reachable), it becomes possible for active databases to react alternatively, which makes the entire ECA mechanism more robust. Since abort is also an action, the classical case can be subsumed easily.

The rest of the paper is organized as follows: Section 2 starts with a brief introduction to active databases systems. Then we enhance ECA rules to strict functions and enrich ECA rules to ECA-AA rules by alternative actions. In Section 3 we show the advantage and the use of ECA-AA rules for maintaining data consistency in distributed active database systems. We summarize our work in Section 4.

2 The ECA Mechanism and its Enhancement for Distributed Active Database Systems

Up to now ECA rules have been mainly used in central active database systems [2,5,6]. Simple variants of ECA rules have been integrated into SQL:99 and are available in some object-relational database systems. In these simple rules events are restricted to insert, delete or update operations. In distributed active databases, both event evaluation and condition evaluation can have an indefinite result because of unavailable subsystems. Hitherto this leads to an abort after a timeout even if this is not desirable or necessary at all. The goal of this Section is to develop and present a solution on this problem.

2.1 Active Databases (short repetition)

Active database systems [2,5,8] can react to occurring events using ECA rules. This ability can be used, for example, to control relationships between data objects even beyond system boundaries.

Reactions on events are specified as rules. Rules are triples of the kind (Event, Condition, Action). These ECA rules are also known as triggers or alerters. An event is something that occurs at a specific point in time. Conditions are predicates related to a database. They determine under which constraint an event is important. Conditions are optional. An action specifies what is to be done, if a situation of interest occurs, i.e., event and condition evaluate to true.

Active databases distinguish between different categories of events. The two main categories are simple and composite events. Simple events can be split into database events, time events, and abstract events. A database event is any operation on the database including start, commit, and abort of transactions. Time events are activated at a specific point in time. Using abstract events, a reaction to external events occuring outside of the database become possible. However, the system must be explicitly informed about these events. In practice, this requires the explicit activation of the rule by an application program. Simple events can be combined to composite events using logical operators.

2.2 Requirements for ECA Mechanisms in Distributed Active Database Systems

2.2.1 Decentralized Event Detection

A general architecture for heterogeneous active database systems is introduced in [6]. This architecture enables event detection within distributed active databases. The main components are a central "shared knowledge repository" and a central ECA rule base. The shared knowledge repository contains transformation information and procedures. Based on them, different data models, data manipulation languages, and object representations of diverse database systems can be accessed by an "intelligent agent". System-wide rules are also being held in a central way. Local event detectors

signal occurring events to local components, and pass all events that are of global interest to the central ECA rule base.

The disadvantage of this architecture is its central approach. Sending events to the central rule base makes it necessary to establish a connection to the rule base. For mobile systems, for example, such a connection cannot be guaranteed over a longer period of time. As a consequence a buffering of occurred events is proposed in [6]. The central event detector can react to these events only with delay. But late reactions can lead to undesired effects due to the fact that the state of the database system may have changed in the meantime. Especially, it may happen that attribute values are set to an old or incorrect value due to late update.

Furthermore, a central event detection is not adequate for distributed database systems with high autonomy degree. For this kind of system a decentralized event detection and a decentralized ECA rule base is required.

Up to now, all ECA mechanisms and architectures for distributed heterogeneous active database systems assume a very limited autonomy of the individual subsystems.

2.2.2 Strictness of ECA Rules

In central databases, the evaluation of events and conditions of ECA rules is always possible. This cannot be ensured in distributed database systems with high degree of autonomy. For them the event or condition evaluation may be indefinite (Ω) due to unaccessability. Thus our second requirement for ECA mechanisms in distributed active database systems is the strictness of ECA rules in order to treat the special case of indefiniteness.

2.3 Enhancing the ECA Mechanism for Distributed Active Database Systems with High Autonomy Degree

The ultimate goal of the enhancement is a more flexible ECA mechanism, which allows us to continue work even if subsystems are not reachable. This is achieved by adding strictness to event and condition evaluation.

We consider the condition evaluation first. The evaluation of a condition c formally corresponds to function $f(c)$ which either evaluates to TRUE or FALSE (see equation (1)). Usually c is recursively composed by $c_1, c_2, \ldots, c_n$ subconditions, which are concatenated by boolean operators. Atomic conditions are all kind of equations, inequations and boolean values. Of course, c_i can also refer data in remote subsystems and subconditions may even be evaluated on remote hosts. Thus we get:

$$f : C \rightarrow \{true, false\} \quad \textbf{(1)}$$

$$f(c) = h(h(\ldots h(f_{@1}(c_1), f_{@2}(c_2)), \ldots), f_{@k}(c_n)) \quad \textbf{(2)}$$

c_i condition i $(0 \le i \le n)$,
evaluated at (remote) subsystem j, denoted by $f_{@j}$ if important
(omitted later on) $(0 \le j \le k \le n)$.
$@j_1$ and $@j_2$ are not necessarily different and may be even the local host

h any boolean operator

In distributed active systems, subsystems may be unreachable. Thus, $f_{@i}(c_i)$ can be indefinite. We denote this by Ω and extend both, the domain and the codomain of *f(c)*, with the Ω element (indefinite). The introduction of Ω formally turns *f(c)* into a total function. We call *f* a strict function, if holds: *f* evaluates to Ω, if any input parameter of *f* is Ω [1]. Of course, *h* has to be total and strict as well.

$$f : C \rightarrow \{true, false, \Omega\} \quad \textbf{(3)}$$

Analogous to condition evaluation, evaluation of events (e) can be defined as a total and strict function *g(e)*. Like *f(c)*, also *g(e)* maps to the codomain {true, false, Ω}.

$$g : E \rightarrow \{true, false, \Omega\} \quad \textbf{(4)}$$

true: event $e \in E$ did occurre
false: event e has not occurred
Ω : it is indeterminable whether the event e occurred or not

The firing of an ECA rule is defined as follows:

$$if\{g(e) \wedge f(c)\}\ then\ \text{execute A}\ [else\ \text{don't execute A}]\ fi \quad \textbf{(5)}$$

If one of the parameters in the *if*-condition is Ω, the firing of the ECA rule leads to the processing of the *else*-case (nothing happens). Up to now, indefiniteness in one of the *if*-conditions is not considered explicitly. That is the reason of enhancing the ECA mechanism with a new, alternative action (AA), which is executed in the Ω-case[1]. Of course, the alternative action can activate further rules and thus further (alternative) actions.

Enhanced ECA rules:
An enhanced ECA rule, called ECA-AA is defined as a 4-tuple (*Event, Condition, Action, Alternative Action*). The *alternative action* is executed (instead of action) when the condition evaluation of C returns Ω. An ECA-AA rule will become a traditional ECA rule if no *alternative action* is defined.

Usually we are only interested in defining alternative actions if E did occur and C is indeterminable (Ω). There are only very limited use cases where also the evaluation of E to Ω is important, like in security systems. For instance in a security control

[1] Deviating from the definition in [1] it is completely sufficient, if the if-then-else statement is only strict concerning the condition and the entered branch (and not globally strict).

system the interruption of operation of an external video camera, acting as event initiator, should cause an additional alternative action, like closing a door and ringing an alarm bell. But usually only positive events cause an ECA rule evaluation, since otherwise the absence of any external event initiator would cause an infinite call of AA, which is in general not intended. We distinguish between both cases. Herewith, the ECA evaluation definition (5) becomes:

Usual mode:

if $\{g(e) \wedge f(c)\}$	*then*	execute action A
else if $\{g(e) \wedge f(c) = \Omega\}$	*then*	execute alternative action AA
	else	do not execute any action

fi

Security mode:

if $\{g(e) \wedge f(c)\}$	*then*	execute action A
else if $\{g(e) \wedge f(c) = \Omega\}$	*then*	execute alternative action AA_C
else if $\{g(e) = \Omega\}$	*then*	execute alternative action AA_E
		/* usually includes suspending this rule in order to avoid infinite calls */
	else	do not execute any action

fi

3 Using Enhanced ECA Rules for Maintaining Data Consistency

In the following we show how ECA-AA rules (in the usual mode) can be used in order to guarantee data consistency in a distributed active database system.

3.1 Specification of Consistence Constraints

Dependences between data objects can generally be described by the tuple <S,D,P,C,A>, also known as D^3 (data dependency descriptor) [7]. S stands for the source objects and D for the destination objects. Source objects and destination objects can be arbitrary database objects (e.g., tables, tuples, attribute values).

P is a predicate which describes the data dependencies between source and destination objects. According to the ECA rules, the point in time at which P evaluates to true can be considered as an event.

C specifies a condition that, if fulfilled, leads to the execution of action A. C also can specify a point in time at which P must be true. It is worth mentioning that C specifies no consistency conditions about the dependencies between source and destination objects (see example below). A is an action which can call further actions and which must be executed to achieve the consistency of the overall system. This action makes sure that P is fulfilled.

The use of the tuple <S,D,P,C,A> is illustrated by the following example. The source objects are the attributes s_1 to s_n, which are distributed over different databases on different computers. These computers can be mobile computers, like laptops, which are not permanently reachable. The destination is supposed to be the attribute *d*. Destination objects and source objects are in a consistent state if $s_1 + ... + s_n \leq d$ is satisfied (e.g., planned amount of money for the adjustment of an insured loss must be greater or equal than the sum of all partial damages). This consistency condition is only valid if attribute *c* is greater than 100. Attribute *c* can also reside on a remote database.

The notation using the tuple <S,D,P,C,A> looks as follows:

S:	$s_1 , ... , s_n$	source objects
D:	*d*	destination object
P:	$s_1 + ... + s_n \leq d$	consistency relationship between source objects and destination object
C:	*c>100*	consistency condition
A:	$d := s_1 + ... + s_n$	action

Inaccessibility of systems can always occur in distributed databases. P or C can be indefinite in the above example. As a consequence it is also indefinite whether action A is to be executed or not.

We enhance the tuple <S,D,P,C,A> with an entry for alternative action (AA). Then it is possible to execute a defined action even in case of indefiniteness of P or C. In the above example an alternative action may set the attribute *d* to a maximal value. The notation of the example with the new tuple <S,D,P,C,A,AA> looks as follows:

S:	$s_1 , ... , s_n$	source objects
D:	*d*	destination object
P:	$s_1 + ... + s_n \leq d$	consistency relationship between source objects and destination object
C:	*c>100*	consistency condition
A:	$d := s_1 + ... + s_n$	action
AA:	*d := maximal value*	alternative action

3.2 Transformation into Enhanced ECA Rules

Enhanced ECA rules may directly evaluate Data Dependency Descriptors of the form <S,D,P,C,A,AA>. The following rule shows the general mapping of the tuple <S,D,P,C,A,AA> to an enhanced ECA rule and, in addition, an instantiation on the base of the above example.

Event:	not P	Point in time on which $d >= s_1 + ... + s_n$ is not true for the first time.
Condition:	C	*c>100*
Action:	A	$d := s_1 + ... + s_n$
Alternative Action:	AA	*d := maximal value*

3.3 Advantages of Enhanced ECA Rules

Using enhanced ECA rules, a system architecture without central event detection and central rule base can be built. Therefore every subsystem must consist of an active database system, and it must be able to detect events across systems.

Every subsystem can specify its consistency conditions in the form of enhanced ECA rules. Thus a decentralized rule base is build up. The event detection is decentralized, too, because every subsystem can also detect events in remote subsystems.

With the proposed ECA-AA rules every subsystem can react in case of indefinite event or condition evaluation. Thereby a high degree of autonomy of the subsystems is provided. Data consistency in distributed database systems can only be achieved if the indefinite event and condition evaluation is taken into account.

4 Conclusions

This paper proposes an enhancement of the well-known ECA rules. Traditional ECA rules are enhanced by an element for *alternative actions*. The alternative action is executed if the event or condition evaluation is indefinite. In contrast to traditional ECA rules, the new ECA-AA rules always provide a defined reaction. An example about maintenance of data consistency in distributed active database systems has shown the practical applicability of the approach.

References

[1] Bauer F.L., Wössner H.: *Algorithmische Sprachen und Programmentwicklung*, Springer-Verlag 1981

[2] Dittrich K. R., Gatziu S.: *Aktive Datenbanksysteme - Konzepte und Mechanismen,* dpunkt.verlag 2000

[3] Helal A. A., Heddaya A. A., Bhargave B. B.: *Replication Techniques in Distributed Systems*, Kluwer Academic Publishers 1996.

[4] Oezsu M. T., Valduriez P.: *Principles of distributed database systems,* 2nd Ed. Prentice-Hall 1999.

[5] Paton N. W.: *Active Rules in Database Systems,* Springer-Verlag 1998

[6] Pissinou N., Vanapipat K.: *Active Database Rules in Distributed Database Systems*. Intl. Journal of Computer Systems, 11(1), January 1996, pp. 35-44

[7] Rusinkiewicz M., Sheth A., and Karabatis G.: *Specifing interdatabase dependencies in a multidatabase environment*. IEEE Computer, 24(12), December 1991. Special Issue on Heterogeneous Distributed Databases, pp. 46-53.

[8] Zimmermann J.: *Konzeption und Realisierung eines aktiven Datenbanksystems: Architektur, Schnittstellen und Werkzeuge,* Logos-Verl, 2001