Active Rules in Deductive Databases

Bertram Ludäscher Georg Lausen

Institut für Informatik, Universität Freiburg, Germany

Overview

I. Basic Concepts of Active Rules (ECA Rules)
ECA rules, execution models, coupling modes, problems

II. Production Rules ($\approx$ CA Rules $\approx$ Procedural Datalog)
syntax & semantics, queries vs updates, expressive power, termination

III. Deductive Rules ($\approx$ Declarative Datalog)
declarative semantics for queries

IV. Pure Statelog (Integration of Production Rules & Deductive Rules)

V. A Statelog Framework for Active Rules

A Spectrum of Rule Languages: (adapted from [Wid93])

Deductive Rules *"higher" level*		*Production Rules*				*Active Rules* *"lower" level*
stratified, well-founded Datalog$^{\neg}$	. RDL procedural Datalog$^{\neg}$	. A-RDL	. Ariel	. Starburst	. Postgres	

Contents:

- Motivation: active vs. passive databases
- ECA rules:
  - primitive & composite events, event algebra
  - conditions
  - actions
- Evaluation of ECA rules: coupling modes
- Theoretical aspects: termination, confluence, semantics

- **Passive Databases**
  do not support (re)active behavior, transactions are submitted explicitely by the user or the application program
- **Active Databases**
  add (re)active capabilities to traditional (passive) systems

Application Areas:

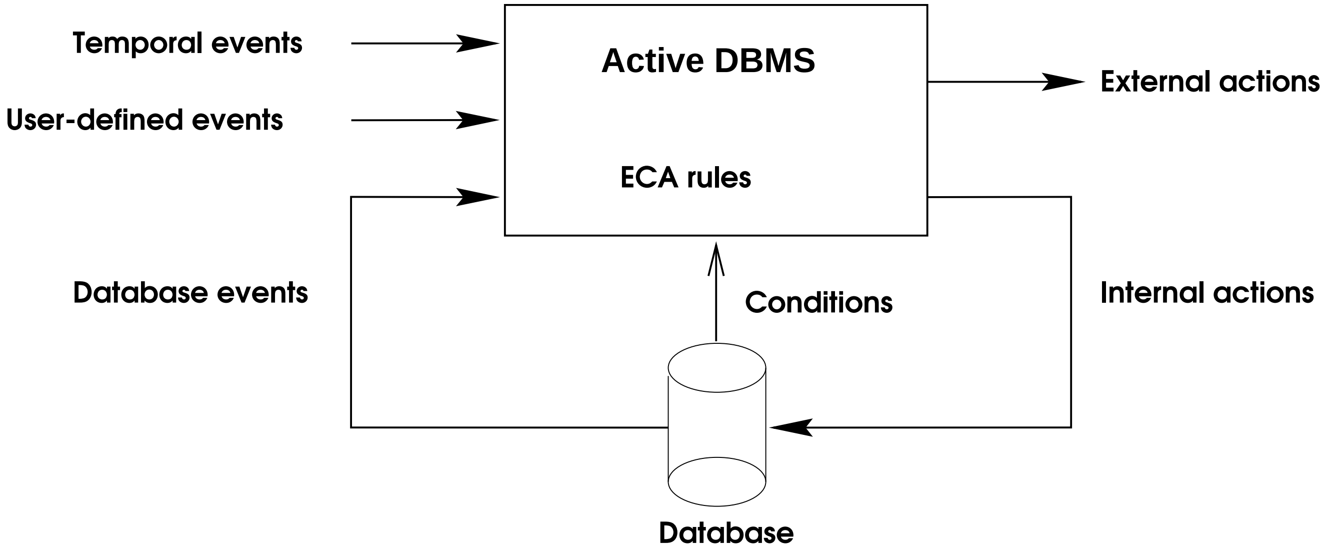
- **Enhanced DBMS functionality:**
  integrity checking and *enforcement* (e.g. *referential actions*), access control, version control, view maintenance (data warehouses), inheritance in OO systems, ...
- **Applications with active requirements:**
  monitoring applications (e.g. traffic control), plant control, workflow management, inventory control, financial applications (e.g. portfolio management), ...

Active features (event detection, action execution) can be encoded in application programs, but ...

- ... same/similar functions have to be reimplemented in several places $\Rightarrow$ loss of modularity and increased maintenance effort (e.g. frequent recompilation of applications)
- ... complex events coming from different subsystems are difficult to handle
- ... monitoring of events by **polling** is often less efficient than active event processing (polling frequency is critical)

In contrast, in an active system ...

- ... rules can be shared by many applications
- ... complex events can be detected in a central place
- ... and may be optimized



- Event detection
- Condition evaluation
- Action execution

$\Rightarrow$ specification of active behavior by **ECA** (Event-Condition-Action) **rules**

Syntax

$$\textbf{on } \langle event \rangle \textbf{ if } \langle condition \rangle \textbf{ then } \langle action \rangle$$

"Semantics"

Whenever the specified $\langle event \rangle$ occurs, the rule is triggered and the $\langle action \rangle$ executed provided the $\langle condition \rangle$ is satisfied

Example

on deletion from `stock-rel` **if** `stock-rel` $= \emptyset$ **then** order items

- CA rules are sometimes called **production rules**
- EA rules are sometimes called **triggers**

Separation of **event** vs. **condition** (HiPAC):

- the event specifies **when** to check the condition
- the condition specifies **what** to check

(to avoid loss of events asynchronous processing can be used;
but from a more abstract viewpoint EC may be integrated)

- **Database events:**
  - state transition events/updates (R.insert, R.delete, R.modify)
  - retrieval (R.select)
- **Transactional events:** bot, eot, abort, commit, checkpoint
- **Temporal events:**
  - absolute (`@ Fri Jun 6 10:10:25 MET DST 1997`)
  - relative (`@ now + 1hour`)
  - periodic time events (`@ now every 2nd day`)
- **External events:** application-specific events (temperature exceeds threshold, runway clear)

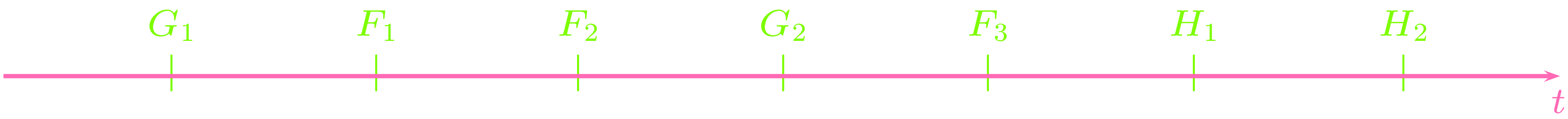
from an abstract/logical point of view, events can be incorporated as special database events

Composite (or **complex**) events are built form primitive events using an **event algebra**, e.g. HiPAC [DBB+88, DBC96], SAMOS [GD93], ODE [GJS92], Sentinel/SNOOP [CKAK94]:

- **Sequence:** $(E1 \; ; E2)$
  occurs when $E2$ is raised after $E1$
- **Disjunction:** $(E1 \mid E2)$
  occurs when $E1$ or $E2$ is raised
- **Conjunction:** $(E1 \; , E2) := (E1 \; ; E2) \mid (E2 \; ; E1)$
  occurs when $E1$ and $E2$ occur in either order
- **Negation: not** $E1$ **within** $[E2, E3]$
  occurs when $E1$ does not occur in the interval $[E2, E3]$
- **Any operator: any**$(k, E1, \ldots, En)$
  occurs when $k \leq n$ *distinct* events occur
- **Aperiodic (between) event: A**$(E1, E2, E3)$
  occurs when $E2$ occurs after $E1$ but before $E3$,
  A* **cumulates** the occurrences of $E2$ and is raised only once at $E3$
- **Periodic event: P**$(E1, T[params], E3)$
  occurs every T interval between $E1$ and $E2$, **P*** **cumulates** the $params$ and occurs only at $E3$

Problem: which specific constituent event occurrences make up the composite event?

Example: $E = (F, G)\ ;\ H$



Using an **unrestricted context**, 12 occurrences of E are signaled:

$$E_i = F_j \times G_k \times H_l \qquad \text{where } t(l) > t(j), t(k)$$

Handling all of these may be too costly and/or not meaningful in the application
$\Rightarrow$ specific **event consumption policies:**

- **Recent context:** $E_i = \{G_2, F_3, H_1\}, \{G_2, F_3, H_2\}$
- **Chronicle context:** $E_i = \{G_1, F_1, H_1\}, \{F_2, G_2, H_2\}$
- **Continuous context:** $E_i = \{G_1 : F_1, H_1\}, \{F_1 : G_2, H_1\}, \{F_2 : G_2, H_1\}, \{G_2 : F_3, H_1\}$
- **Cumulative context:** $E_i = \{G_1, F_1, F_2, G_2, F_3, H_1\}$

Which parameter context is appropriate depends on the type of application:

- **Recent context:**
  if events arrive at a high rate and new occurrence of the same event type refine previous occurrences; e.g. sensor monitoring applications
- **Chronicle context:**
  if events have to be processed in their chronological order (first come, first served), e.g. to maintain dependencies, fairness (queue processing, workflows)
- **Continuous context:**
  each occurrence starts an new window of continuous monitoring: e.g. trend analysis and forecasting applications (report for 2 hours after DJ index changes by more then 20%)
- **Cumulative context:**
  when grouping of multiple events is needed/meaningful, e.g. on occurrence of a deadline (bank closes), all constituent events (money transfers) are used (e.g. to compute the overall balance)

A **condition** is query on the current database state. Typically **first-order** (**FO**) queries are supported:

- SQL: `select ... from ... where ...`
- Relational algebra: $\{\sigma, \pi, \bowtie, \delta, \cup, -\}$
- Relational (first-order) calculus: $\{\exists, \forall, \wedge, \vee, \neg\}$

Events which are related to database operations require the expression of **transition conditions** $\Rightarrow$

- different database **states** (R.old, R.new) and/or
- **delta relations** (R.inserted, R.deleted)

have to be provided in the rule language. Since in principle all primitive events can be mapped to database updates, the possibility to referrence different db-states and/or delta relations is essential!

$\Rightarrow$ (later) **State**log

- **Database actions:**
  - updates (R.insert, R.delete, R.modify)
  - retrieval (select)
- **Transactional commands:**
  bot, eot, abort, commit
- **External actions:**
  application-specific actions (close gate, send mail, print order, ...)

the database actions issued by an ECA rule can trigger other rules

$\Rightarrow$ cascaded execution

$\Rightarrow$ problem of nontermination

Basic algorithm:

while there are triggered rules $\{$

1. select a triggered rule r
2. evaluate r's condition
3. if this condition is true, execute r's action

$\}$

- similar to the **recognize-act cycle** of OPS-5 [BFKM85] (apart from distinction event / condition)

clearly, this leaves open many issues:

- **granularity** of rule processing, e.g. for SQL commands
  - **instance-oriented:** the rule is triggered for each updated tuple
  - **set-oriented:** the rule is triggered once for the whole statement, i.e. once for all involved tuples
- **rule selection:** which rule should be selected for execution if several are eligible?
  $\Rightarrow$ **conflict resolution policies** e.g. static/dynamic priorities given by the system/user
- **how many rules are fired?** all/some selected rules, sequentially/in parallel

$\Rightarrow$ determinism and confluence problems

Coupling modes (cf. HiPAC [DBC96]) describe the relationship between rule processing and database transactions:

- **When does rule execution begin?**
  immediately after event detection, just before commit, after commit of the triggering transaction, in parallel?
- **When does it end?**
  before triggering transaction resumes/commits, after commit/abort of the triggering transaction?
- **In which transaction is the rule executed?**
  within the triggering transaction, as a subtransaction, separately

Suppose transaction T executes an operation which raises event E, rule r defines action A provided condition C holds.

If E-C coupling is ...

- **immediate** then C is evaluated within the same transaction T immediately after E was detected. The remaining operations of T are suspended until r has been processed.
  (e.g. if timely responses are needed)
- **deferred** then C is still evaluated within T but just before T commits
  (e.g. for checking integrity constraints
- **decoupled (detached)** then C is evaluated in a separate transaction T'.
  - **independent:** T' does not depend on T's commit
  - **dependent:** T' can only commit if T commits

  (e.g. inventory control: restocking transaction can be executed independently; also: shorter transactions $\Rightarrow$ necessary for timely responses)

(analogously for C-A coupling)

Termination: in general, rules may trigger each other recursively $\Rightarrow$ nontermination may arise (often termination is undecidable at compile-time and at least computationally hard at run-time)

- conservative analysis technique based on **triggering graph** [CW90, AWH92, AWH95]:
  $r_i \rightarrow r_j$ if r_i's action may trigger r_j
  $\Rightarrow$ termination is guaranteed if triggering graph is acyclic
- improved analysis [BCW93, BW94] based on "semantic" properties of rules
  (i.e. looking closer at r_i, r_j):
  r_i: **del:**client(C) $\leftarrow$ φ(client,...)
  r_j: **ins:**client(C) $\leftarrow$ ψ(client,...)
  Idea: if $\mathit{effect}(\varphi \wedge$ **del:**client(C)$) \Rightarrow \neg\,\psi$, then do **not** consider $r_i \rightarrow r_j$
  (here, φ, ψ correspond to the conjunction of *event* and *condition*)

Confluence: a rule program is called **confluent** if the final state is unique

- confluence is guaranteed if semantics is deterministic
- `Starburst`: [AWH92] use dependencies, priorities, and **commutativity** of rules to analyse confluence

Semantics:

A serious problem often mentioned is the difficult predictability of rule effects and the lack of declarative and intuitive semantics. Indeed, unless simpler semantic frameworks are available

> *". . . rule designers will have to be four-star wizards in order to ensure that the rule system does what they actually want"* [Wid94]

> *"The unstructured, unpredictable, and often nondeterministic behavior of rule processing can become a nightmare for the database rule programmer"* [AWH95]

The *active database manifesto* [DGG95] requires as an essential feature that

> *". . . rule execution must have a clear semantics, i.e. must define when, how, and on what database state conditions are evaluated and actions executed"*

See also [WC96, PCFW95, DHW95, FT96]

$\Rightarrow$ more "high-level" **production rules**

References

[AWH92] A. Aiken, J. Widom, and J. M. Hellerstein. Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism. In M. Stonebraker, editor, *Proc. ACM SIGMOD Intl. Conference on Management of Data*, San Diego, California, 1992.

[AWH95] A. Aiken, J. Widom, and J. M. Hellerstein. Static Analysis Techniques for Predicting the Behavior of Active Database Rules. *ACM Transactions on Database Systems*, 20(1):3–41, March 1995.

[BCW93] E. Baralis, S. Ceri, and J. Widom. Better Termination Analysis for Active Databases. In Paton and Williams [PW93], pages 163–179.

[BFKM85] L. Brownston, R. Farrel, E. Kant, and N. Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*. Addison-Wesley, 1985.

[BW94] E. Baralis and J. Widom. An Algebraic Approach to Rule Analysis in Expert Database Systems. In *Proc. Intl. Conference on Very Large Data Bases*, pages 475–486, Santiago, Chile, 1994.

[CKAK94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. Intl. Conference on Very Large Data Bases*, pages 606–617, Santiago de Chile, 1994.

[CW90] S. Ceri and J. Widom. Deriving Production Rules for Constraint Maintenance. In *Proc. Intl. Conference on Very Large Data Bases*, pages 566–577, 1990.

[DBB+88] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M. J. Carey, M. Livny, and R. Jauhari. The HiPAC project: Combining Active Databases and Timing Constraints. In *Proc. ACM SIGMOD Intl. Conference on Management of Data*, pages 51–70, March 1988.

[DBC96] U. Dayal, A. Buchmann, and S. Chakravarthy. The HiPAC Project. In Widom and Ceri [WC96], chapter 7.

[DGG95] K. R. Dittrich, S. Gatziu, and A. Geppert. The Active Database Management System Manifesto: A Rulebase of ADBMS Features. In Sellis [Sel95].

[DHW95] U. Dayal, E. Hanson, and J. Widom. Active Database Systems. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond*, chapter 21. ACM Press, 1995.

[FT96] P. Fraternali and L. Tanca. A Structured Approach for the Definition of the Semantics of Active Databases. *ACM Transactions on Database Systems*, 1996. to appear.

[GD93] S. Gatziu and K. R. Dittrich. Events in an Active Object-Oriented Database System. In Paton and Williams [PW93].

[GJS92] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event Specification in an Active Object-Oriented Database. In *Proc. ACM SIGMOD Intl. Conference on Management of Data*, 1992.

[PCFW95] N. W. Paton, J. Campin, A. A. A. Fernandes, and M. H. Williams. Formal Specification of Active Database Functionality: A Survey. In Sellis [Sel95].

[PW93] N. W. Paton and M. H. Williams, editors. *Proc. of the 1st Intl. Workshop on Rules in Database Systems (RIDS)*, Workshops in Computing, Edinburgh, Scotland, 1993. Springer.

[Sel95] T. K. Sellis, editor. *Proc. of the 2nd Intl. Workshop on Rules in Database Systems (RIDS)*, number 985 in LNCS, Athens, Greece, 1995. Springer.

[WC96] J. Widom and S. Ceri, editors. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.

[Wid93] J. Widom. Deductive and Active Databases: Two Paradigms or Ends of a Spectrum. In Paton and Williams [PW93].

[Wid94] J. Widom. Active Databases. In M. Stonebraker, editor, *Readings in Database Systems, 2nd edition*. Morgan Kaufmann, 1994. Introduction to chapter 4.