A Logic-Based Integration of Active and Deductive Databases

Alvaro A.A. Fernandes

Department of Mathematical and Computing Sciences

Goldsmiths
UNIVERSITY OF LONDON

Agenda

1.	**Background**	Active Rule Syntax and Semantics Basic Notions About DDBs
2.	**Foundations**	Kowalski-Sergot Event Calculi Embedding DDBs into an EC
3.	**Development**	An ADB Architecture Event, Condition and Action Specification An Active Rule Language
4.	**Some Conclusions**	
5.	**Further Information**	

[b] Active Rule Syntax (1)

An *active rule* (AR) ρ is a pair $\rho = \langle \kappa, \sigma \rangle$.

The *control-part* κ determines the execution conditions of the *specification-part* σ.

The interpreter for κ implements *the execution model* of the ADBS. There is *no* widespread agreement on κ's general form.

The interpreter for σ implements *the knowledge model* of the ADBS. There *is* widespread agreement on σ's general form.

[b] Active Rule Syntax (2)

The specification-part σ of an AR ρ is a triple $\langle E, C, A \rangle$, written as '$E : C \leadsto A$', and is often referred to as an *ECA rule*.

E is an *event specification* in an event-specification language (ESL), C is a *condition specification* in a condition-specification language (CSL), and A is an *action specification* in an action-specification language (ASL).

An ECA rule $E : C \leadsto A$ is read as 'when an occurrence of E has been detected and C holds in the underlying database, execute A'.

[b] Active Rule Syntax (3)

Let B be a database, whose definition, manipulation and query languages are, respectively, $\mathcal{D}$, $\mathcal{M}$, $\mathcal{Q}$. Similarly, let $\mathcal{E}$, $\mathcal{C}$, $\mathcal{A}$ be the event-, condition- and action-specification languages whose application-dependent lexicon is induced from that of $\mathcal{D}$. Typically, $\mathcal{C} \supseteq \mathcal{Q}$ $\mathcal{A} \supseteq \mathcal{M}$.

An *ECA rule-set* B^+ *over* B is a set of expressions of the form $E : C \leadsto A$ such that $E \in \mathcal{E}$, $C \in \mathcal{C}$ and $A \in \mathcal{A}$.

An *AR-program* B^* *over a database* B is a set of pairs $\langle \kappa, \sigma \rangle$ where κ is a set of conditions for the execution of σ and $\sigma \in B^+$. An *active database* is a pair $\langle B, B^* \rangle$ where B is a database and B^* is an AR-program over B.

[b] Active Rule Semantics (1)

proc $evaluateECASet$
  $\Leftarrow RS, \prec_R, \prec_E, DB, EQ$
  $\Rightarrow DB, EQ :=$
  begin
  while $undriven(EQ) \neq \emptyset$ **do**
    $driveEvent(RS, \prec_R, \prec_E, DB, EQ, NewDB, NewEQ);$
    $DB := NewDB;$
    $EQ := NewEQ$
  od
  end
endproc

[b] Active Rule Semantics (2)

$$
\begin{array}{l}
\underline{\textbf{proc}}\ \ driveEvent \\
\quad \Leftarrow RS, \prec_R, \prec_E, DB, EQ \\
\quad \Rightarrow NewDB, NewEQ := \\
\quad \underline{\textbf{begin}} \\
\quad DrivingEvent := earliest(undriven(EQ), \prec_E); \\
\quad RemainingRs := RS; \\
\quad \underline{\textbf{while}}\ \ RemainingRs \neq \emptyset \quad \underline{\textbf{do}} \\
\quad\quad R := first(RemainingRs, \prec_R); \\
\quad\quad touchRule(R, DrivingEvent, DB, EQ, NewDB, NewEQ); \\
\quad\quad RemainingRs := RemainingRs \setminus \{R\}; \\
\quad\quad DB := NewDB; \\
\quad\quad EQ := NewEQ \\
\quad \underline{\textbf{od}}\ \ ; \\
\quad logAsDriven(DrivingEvent) \\
\quad \underline{\textbf{end}} \\
\underline{\textbf{endproc}}
\end{array}
$$

[b] Active Rule Semantics (3)

$\underline{\mathbf{proc}}$ $touchRule$

$\Leftarrow R, DrivingEvent, DB, EQ$

$\Rightarrow NewDB, NewEQ :=$

$\underline{\mathbf{begin}}$

$E := eventSpec(R);$

$C := conditionSpec(R);$

$A := actionSpec(R);$

$detect(EQ, DrivingEvent, E, DetectionBindings);$ $\ldots$

[b] Active Rule Semantics (4)

... **if** $DetectionBindings \neq \{\triangledown\}$
  then
    begin
      $bind(C, DetectionBindings, boundC);$
      $verify(DB, boundC, VerificationBindings);$
      if $VerificationBindings \neq \{\triangledown\}$
        then
          begin
            $bind(A, VerificationBindings, boundA);$
            $execute(DB, boundA, NewDB, NewEQ)$
          end
      fi
    end
  fi
  end
endproc

[b] Active Rule Semantics (5)

This is *one* among many execution models.

An alternative formalization is needed for analysis and optimization, but there seems to be no single formalism that is adequate for both the execution and the knowledge models.

The knowledge model manifests itself in the execution model in the calls to *detect*, *verify* and *execute*, as well as in supporting the bindings resulting from, and assumed by, those calls.

This work shows how a logic-programming-based knowledge model can be formalized using an event calculus.

[b] Basic Notions about DDBs (1)

A deductive database (DDB) D has an extensional part $EDB(D)$ and an intensional part $IDB(D)$. The language of D is denoted $L(D)$.

The extensional part of a DDB is called an *extensional database* (EDB), and the intensional part of a DDB is called an *intensional database* (IDB).

The Herbrand universe of a language L is the set of all ground terms induced by L. The Herbrand base of a language L is the set of all ground atoms induced by L. The Herbrand universe and Herbrand base of $L(D)$ are denoted $HU(D)$ and $HB(D)$, respectively.

[b] Basic Notions about DDBs (2)

$EDB(D)$ is a subset, denoted $EHB(D)$, of positive atoms in $HB(D)$. $IDB(D)$ is a set of deductive rules (i.e., function-free, range-restricted Horn clauses of the form $A \leftarrow B_1, \ldots B_n$ where $n > 0$).

The semantics of DDBs is given in terms of a mapping that takes $EHB(D)$ as input and computes the subset, denoted $IHB(D)$, of $HB(D)$, every element of which is a logical consequence of $IDB(D) \cup EHB(D)$.

Hence, one meaning of a DDB D is the set of all ground facts that follow from it by logical consequence.

[f] Kowalski-Sergot Event Calculi (1)

A *Kowalski-Sergot event calculus* (EC) is a system of Horn logic for the derivation of the information states induced by a set of time-stamped event descriptions.

It is a calculus insofar as states are intensionally derived through a logical-consequence relation based on a set of axioms, rather than explicitly and destructively manipulated.

A state is a set of facts in the language of the calculus.

[f] Kowalski-Sergot Event Calculi (2)

The axiomatic content of an EC comprises:

1. A *history*, denoted by **H**, which is a set of *event descriptions* ordered by an explicit *time line*.

2. A set of *event axioms*, denoted by $EvAx$, which determine how relationships among entities in the domain arise or change as a result of *event occurrences*, recorded in the form of event descriptions in the history.

When a new event occurs, the event description corresponding to that occurrence is appended to **H**. By the force of the axioms in $EvAx$, this causes relationships to arise or change, thereby implicitly characterizing a new state.

[f] Kowalski-Sergot Event Calculi (3)

The crucial idea behind an EC is, then, that states are not explicitly asserted, rather they follow logically from the known succession of event occurrences.

An EC can also be seen as the definition of a predicate $*holdsAt/2$.

If $*holdsAt(Atom, T)$ is derivable in its corresponding EC then the atom $Atom$ holds in the application domain at the time point T.

[f] Kowalski-Sergot Event Calculi (4)

The **persistence axiom** $AxP \in EvAx$ is the root of the definition of $*holdsAt/2$ as follows:

*holdsAt(Atom, T2) $\leftarrow$
starts(Event, Atom),
happenedAt(Event, T1),
T1 $\leq$ T2,
$\sim$(stoppedHolding(Event, Atom, T1, T2))

The other predicates in AxP, i.e., *starts/2, happenedAt/2, stoppedHolding/4*, are defined by the remaining axioms in $EvAx$.

[f] Kowalski-Sergot Event Calculi (5)

In particular, a group of axioms defines how atoms hold, cease to hold, or exclude others from holding as a result of event occurrences.

This group of axioms, therefore, binds event descriptions to the start and the end of the relationships denoted by atoms.

[f] Kowalski-Sergot Event Calculi (6)

This work proposes the embedding of DDBs into an EC.

Then, an EDB ceases to be asserted explicitly, and becomes an element in a time-parameterized sequence of EDBs.

Each element in this sequence corresponds to the extensional state of the database at a particular point in time and contains (in principle, notionally) all the logical consequences that are implicitly derived from the history of event occurrences, up to that point, through the application of $EvAx$.

On top of the EDB sequence, one can write the rules that make up an IDB.

[f] Embedding DDBs into an EC (1)

The crucial relationship that needs to be established between a DDB D and an EC is: *in EC-atoms of the form '$*holdsAt(\alpha, \tau)$' the EC-term 'α' is an atom in $L(D)$.*

This technique, which maps an atom (i.e., a truth-value-denoting expression) in the weaker theory (a DDB) into a term (i.e., an individual-denoting expression) in the stronger theory (an EC), is a form of *embedding*.

The language and rules of inference of the resulting logical system are those of the EC.

[f] Embedding DDBs into an EC (2)

Each element $\alpha \in EHB(D)$ is rewritten as an element of AxP of the form $*holdsAt(\alpha, \tau) \leftarrow$, where τ is the time point where α is first known to hold, with $\tau = 0$ for all the atoms determining the initial state.

[f] Embedding DDBs into an EC (3)

Each element $\alpha \in IHB(D)$ must also be embedded. This is achieved by rewriting a rule of the form $H \leftarrow L_1, \ldots, L_n$ into an element of AxP of the form

$$
\begin{array}{l}
*holdsAt(H, \tau_H) \leftarrow \\
\qquad *holdsAt(L_1, \tau_{L_1}), \\
\qquad \ldots, \\
\qquad *holdsAt(L_n, \tau_{L_n})
\end{array}
$$

where, typically, $\tau_H = \tau_{L_1} = \ldots = \tau_{L_n}$.

The group of axioms in $EvAx$ that binds event descriptions to the start and the end of the relationships denoted by atoms is particularized to $L(D)$.

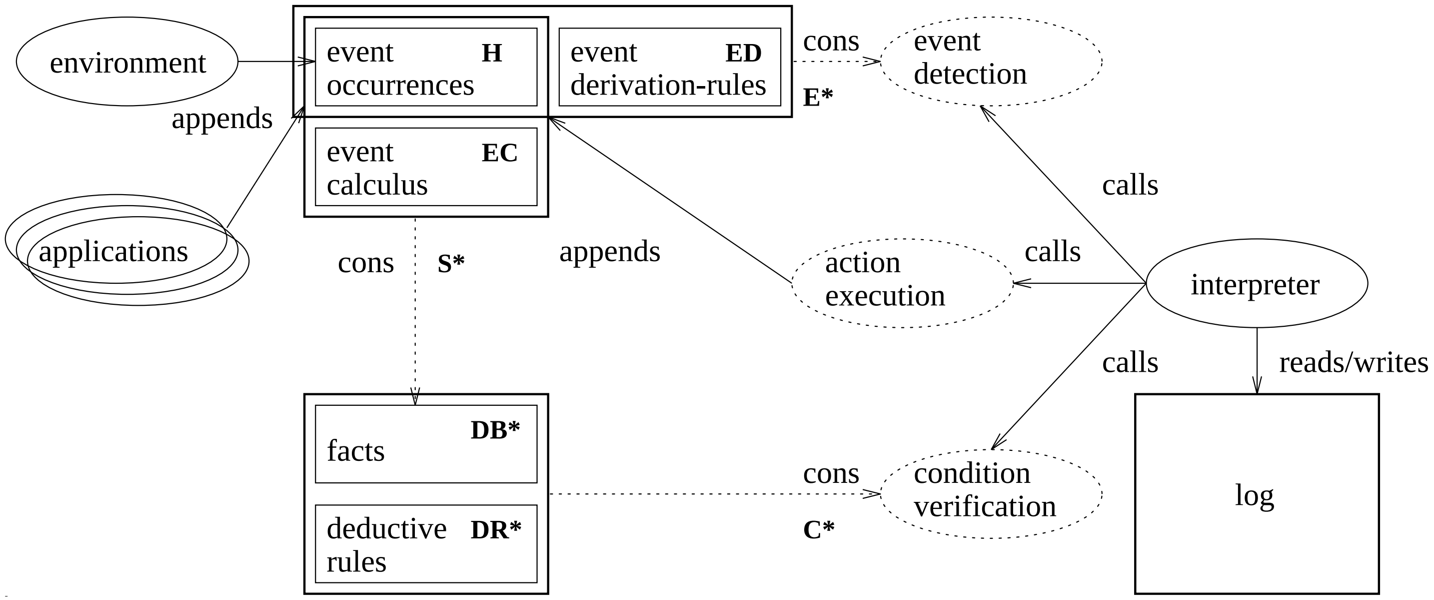
[d] An ADB Architecture (2)

An ADB architecture is proposed in which the role of the history is more significant than in the standard case.

The notion of history that is used is the one that underlies event calculi.

Event detection, condition verification and action execution are all definable in terms of that notion.

[d] An ADB Architecture (1)



Goldsmiths UNIVERSITY OF LONDON

[d] An ADB Architecture (3)

Note that two IDBs are defined on top of the history:

1. The first IDB, denoted by **ED**, is made up of *event derivation-rules* over **H**. This determines a first DDB $\mathbf{E}^\star = \mathbf{H} \cup \mathbf{ED}$. Then, the set **E** of *known events* is the set of all logical consequences of $\mathbf{E}^\star$, denoted by $cons(\mathbf{E}^\star)$.

2. The second IDB, denoted by **EC**, over **H** is an EC. This determines a second DDB $\mathbf{S}^\star = \mathbf{H} \cup \mathbf{EC}$. Then, the set $\mathbf{DB}^\star$ of *known EDBs* is the set of all logical consequences of $\mathbf{S}^\star$.

One can then write a set $\mathbf{DR}^\star$ of *data derivation-rules* over $\mathbf{DB}^\star$. This characterizes a user-visible DDB $\mathbf{C}^\star = \mathbf{DB}^\star \cup \mathbf{DR}^\star$, against which conditions are evaluated.

[d] An ADB Architecture (4)

By being used extensionally both by event derivation-rules and by the event axioms, the history determines the set of known event occurrences and the set of known EDBs, and from the latter, the set of all known database states can be derived.

It follows that for action executions to make events known or to effect state transitions they need only append event occurrences to the history.

[d] An ESL (1)

An event-specification language $\mathcal{E}$ is the set of all queries over the DDB $\mathbf{E}^\star$, whose extensional part **H** is a history and whose intensional part is a set **ED** of *event-derivation rules*.

In standard DDB fashion, information explicitly asserted in the extensional part **H** is made available in the intensional part **ED** by including in the latter the following rules:

event(EventId) $\leftarrow$
    *event(EventId)
at(EventId, TimePoint) $\leftarrow$
    *at(EventId, TimePoint)
happened(ReferringEventId, Role, Referred) $\leftarrow$
    *happened(ReferringEventId, Role, Referred)

[d] An ESL (2)

The set of derived events is extended by including rules in **ED** that use in their bodies:

- predicates in the set $\{event/1, at/2, happened/3\}$, and
- as expressive a set of auxiliary predicates as one chooses in the light of the trade-offs in semantic complexity that they imply. Typical auxiliary predicates are negation (e.g., $\sim /1$ interpreted as finite failure to prove), aggregates (e.g., $setof$, $count$, sum, max, min, $average$, all binary), arithmetic (e.g., $+, -, /, *, mod$, all ternary) and comparison (e.g., $<, >, \leq, \geq, =$, all binary) predicates. Access to the system clock through a predicate, e.g., $clock/1$, is also of obvious interest.

[d] An ESL (3)

With these atoms, one can write rules that relate event occurrences and thus model composite events. For example, conjunction:

$$
\begin{array}{l}
\text{conjunction}(\text{E1}, \text{E2}) \leftarrow \\
\qquad \text{event}(\text{E1}), \\
\qquad \text{event}(\text{E2})
\end{array}
$$

[d] An ESL (4)

Disjunction and sequence can be similarly defined:

disjunction(E1, E2) $\leftarrow$
    event(E1)
disjunction(E1, E2) $\leftarrow$
    event(E2)

sequence(E1,E2) $\leftarrow$
    at(E1, TimePoint1),
    at(E2, TimePoint2),
    TimePoint2 $>$ TimePoint1

[d] An ESL (5)

This is essentially a process of adding new predicates to the intensional part of $\mathbf{E}^{\star}$, and hence enlarging the set of its logical consequences.

An event specification is then a query over $\mathbf{E}^{\star}$, i.e., an ESL can be formally defined as

$$\mathcal{E} = \{\leftarrow E_1, \ldots, E_n : \forall 1 \leq i \leq n[F \sqsubseteq E_i \wedge F \in IHB(\mathbf{E}^{\star})]\}$$

where $X \sqsubseteq X'$ denotes that X is a substitution-instance of X'.

[d] An ESL (6)

As an example, consider a change in an employee's salary:

happened(EventId, referencing-act, put),
happened(EventId, referenced-property, salary),
happened(EventId, reference-source, Employee),
happened(EventId, reference-target, EmployeeSalary)

When an event specification holds against $\mathbf{E}^{\star}$ the corresponding event occurrence is said to have been *detected*.

[d] A CSL (1)

A CSL $\mathcal{C}$ can be characterized as the query language over a DDB $\mathbf{C}^\star$, whose EDB is the set of logical consequences $cons(\mathbf{S}^\star) = \mathbf{DB}^\star$ of $\mathbf{S}^\star$ and whose IDB is a set $\mathbf{DR}^\star$ of *data derivation-rules*.

- The EDB of $\mathbf{C}^\star$ is obtained by computing the logical consequences of another DDB, viz., $\mathbf{S}^\star$. This means that to ascertain that a ground atom F is a fact in $EHB(\mathbf{C}^\star)$ one has to ascertain that F is in $IHB(\mathbf{S}^\star)$.
- It follows from this and from the definition of event calculi that the EDB of $\mathbf{C}^\star$ can be construed as a sequence of time-stamped EDBs, so that there is an EDB at each time point τ for which an event occurrence is recorded in the history $\mathbf{H}$.

[d] A CSL (2)

Information inferred from **S*** is made available in **DR*** by including in the latter the following rule:

holdsAt(Atom, TimePoint) $\leftarrow$
    *holdsAt(Atom, TimePoint)

The set of specifiable conditions can be extended by including rules in **ED** that use in their bodies the $holdsAt$ predicate and as expressive a set of auxiliary predicates as one chooses, with the same caveat as in the case of event-derivation rules.

[d] A CSL (3)

As an example, consider testing whether an employee has a larger salary than that of his/her manager:

holdsNow(property(Employee, manager, Manager)),
holdsNow(property(Manager, salary, ManagerSalary)),
ManagerSalary $<$ EmployeeSalary

A condition specification is then a query over $\mathbf{C}^*$, i.e., a CSL can be formally defined as

$$\mathcal{C} = \{\leftarrow C_1, \ldots, C_n : \forall 1 \leq i \leq n[F \sqsubseteq C_i \wedge F \in IHB(\mathbf{C}^\star)]\}$$

[d] An ASL (1)

An ASL $\mathcal{A}$ can be minimally defined in terms of monotonic appendage of event occurrences in clausal form to the history.

This suffices both:

- to change the set of known events by changing the extensional base on which the logical consequences of $\mathbf{E}^\star$ are computed and
- to induce a new state in $\mathbf{DB}^\star$ and hence change the logical consequences of $\mathbf{C}^\star$ that can be computed using $\mathbf{DR}^\star$.

[d] An ASL (2)

An ASL $\mathcal{A}$ can be formally defined as the set of all expressions of the form $append(\delta)$, where δ is an event description that is mappable, through a function θ, into a set of clauses of **H**.

Unlike ESL and CSL expressions, whose semantics is that of DDB queries, the semantics of ASL is given by a simple set-theoretic expression. Given a history **H** and an event description δ, the result of evaluating an ASL expression $append(\delta)$ is the set $\mathbf{H}'$ of clauses defined by

$$\mathbf{H}' = \mathbf{H} \cup (\theta(\delta))$$

[d] An ASL (3)

As an example, consider setting the salary of an employee's manager to be identical to that of the said employee:

append(clausal(put_salary(EmployeeSalary)@Manager))

In the above, $\delta =$ *put_salary(EmployeeSalary)@Manager* is a command in ROCK & ROLL, a DOOD developed at Heriot-Watt University, and $\theta =$ *clausal*, which applied to *put_salary(EmployeeSalary)@Manager* is assumed to yield its corresponding well-formed event description.

[d] An Active Rule Language (1)

Given an AR $\rho = \langle \kappa, \sigma \rangle$, the specification part σ of ρ is an expression of the form $E : C \leadsto A$ where $E \in \mathcal{E}$, $C \in \mathcal{C}$, and $A \in \mathcal{A}$, and $Var(\mathcal{E}) = Var(\mathcal{C}) = Var(\mathcal{A})$.

As an example, consider the ECA rule:

happened(EventId, referencing-act, put),
happened(EventId, referenced-property, salary),
happened(EventId, reference-source, Employee),
happened(EventId, reference-target, EmployeeSalary)
:
    holdsNow(property(Employee, manager, Manager)),
    holdsNow(property(Manager, salary, ManagerSalary)),
    ManagerSalary $<$ EmployeeSalary
$\leadsto$
        append(clausal(put_salary(EmployeeSalary)@Manager))

Goldsmiths UNIVERSITY OF LONDON

[d] An Active Rule Language (12)

The ECA rule is the example just given can be read as follows:

> *when some employee's salary has changed and that employee has a larger salary than that of his/her manager, set the salary of the latter to be identical to that of the former*

and one can think of adding as much syntactic sugar as does not rotten one's semantic teeth.

[d] An Active Rule Language (3)

In terms of the AR semantics previously presented:

- *bind* becomes the unification procedure over terms in $\mathcal{E}$, $\mathcal{C}$ and $\mathcal{A}$.
- *detect* becomes the query evaluation procedure associated with the DDB $\mathbf{E}^{\star}$.
- *verify* becomes the query evaluation procedure associated with the DDB $\mathbf{C}^{\star}$.
- *execute*, instead of effecting transitions by directly mapping a database state into another while generating new events in the process can achieve the same effect simply by adding event occurrences to the history.

Some Conclusions

- (+) The knowledge model is simple, well-understood, can be reasoned about, implementation can use well-known optimization and evaluation techniques.
- (+) There is a lot of freedom with respect to choosing an execution model, since the knowledge *module* is strongly-cohesive and loosely-coupled.
- (-) The execution model cannot easily be reasoned about.
- (?) Could it be specified with a modal logic? E.g., Pratt-Harel's dynamic logic, Bonner-Kifer transaction logic, an executable intensional temporal-logic?
- (?) If it could, why couldn't it be specified with an executable *extensional* temporal-logic? An event calculus?!

Further Information

For more information, please start from

`http://www.gold.ac.uk/~mas01aaf/`.

See also

Fernandes, A.A.A., Williams, M.H. and Paton N.W. **A Logic-Based Integration of Active and Deductive Databases**. *New Generation Computing*, 15(2), 1997, pp. 205–244.