ECA-LP/ECA-RuleML

2006-09-21

IBIS, TU München
+49 89 289 17504
http://ibis.in.tum.de

ECA-LP / ECA-RuleML: A Homogeneous ECA Logic Programming Language

Agenda

- ECA-LP Syntax
- ECA-LP Semantics
- Update Actions
- Complex Event / Action Algebra
- ECA-RuleML
- Event Notification / Event Messaging
- Discussion

Adrian Paschke (TUM)

RuleML Reaction Rules, Telephone Conference, 2006-09-21

IBIS TUM

Event Condition Action Logic Programming Language (ECA-LP) – Syntax (1)

- ECA-LP Core Syntax: Extended (T)ECA(P)(EL) Rules

*eca(Time,**Event,Condition,Action**,Post-Condition,Else).*

- **Optional Time Part**
  - Interval validity period of rule, e.g. "*from 1-1-2005 to 10-3-2006*"
  - Periodical monitoring schedules, e.g. " *from 9 a.m. to 12 p.m. every 10 seconds"*
  - Absolute time events, e.g. "*at 25-2-2006 at 9:00 a.m.*"
  - Relative event, e.g. " *5 minutes after event X*"

- **Post-Condition**
  - Cuts and counters might be set to prevent backtracking of variable bindings
  - Post conditional test, e.g. test integrity constraints, test special predefined test case etc.
  - If Post-condition test fails, then rollback (internal) update transactions (actions)

- **Else**
  - Defines alternative action (exception) in case the normal execution sequence fails
  - Leads to a more compact syntax:

    "*On **event** and **condition** do **action 1** else do **action 2***"

Event Condition Action Logic Programming Language (ECA-LP) – Syntax (2)

Homogeneous Representation with Derivation Rules

ECA rule:

```
eca (
    everyMinute(),                                         % time
    detect(request(Customer, FlightDestination),T),        % event
    find(FlightDestination,Flight),                        % condition
    book(Customer, Flight), ! ,                            % action
    notify(Customer, bookedUp(FlightDestination) ).        % else
```

Derivation Rules:

Time:

```
everyMinute() :- sysTime(T), interval(timespan(0,0,1,0),T).
```

Event:

```
detect(request(Customer, FlightDestination),T):-
        occurs(request(Customer,FlightDestination),T),
        consume(request(Customer,FlightDestination)).
```

Condition:

```
find(Destination,Flight) :-
        on_exception(java.sql.SQLException,on_db_exception()),
        dbopen("flights",DB),
        sql_select(DB,"flights", [flight, Flight], [where, "dest=Destination"]).
```

Action:

```
book(Cust, Flight) :-
        flight.BookingSystem.book(Flight, Cust),
        notify(Cust, flightBooked(Flight)).
```

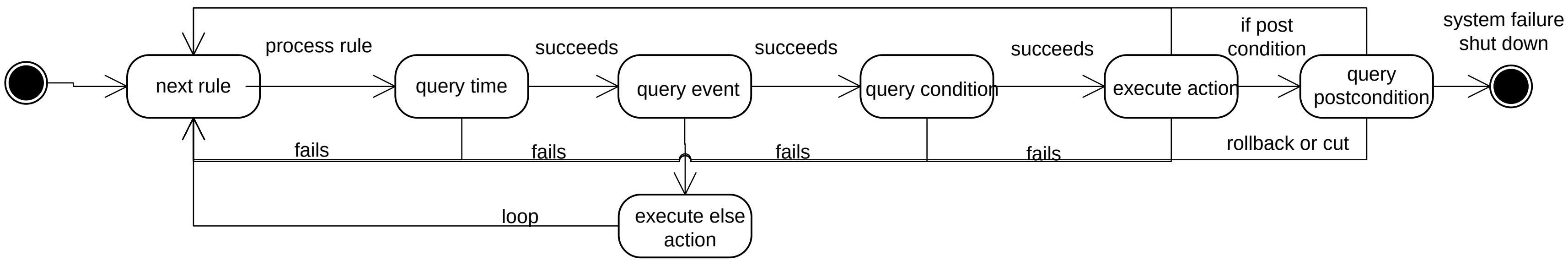
Post-Condition

Else

```
notify(Customer, Message):-
        sendMessage(Customer, Message).
```

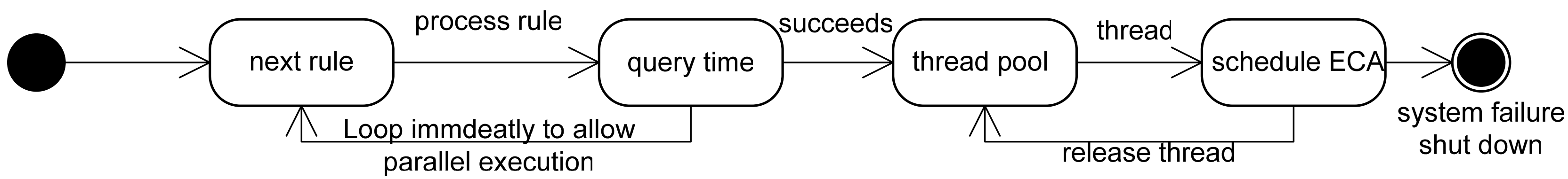
ECA-LP Procedural Semantics

- Goal-driven Backward-Reasoning
  - ECA rules are meta interpreted
  - Active forward-directed operational semantics of ECA paradigm is simulated via queries



- ECA interpreter provides general Wrapper interface on query API

- Parallel execution via multi-threading
  - safeguarded by Event/Action Context, Event Calculus States and Integrity Constraints



- Implements Typed Variable Unification, Backtracking, Procedural Attachments

Dynamic ID-based Updates

- Declarative Semantics
  - Inherits semantics and properties from underlying inference system
  - Global definition of ECA rules
  - Parallel execution of ECA rules

- Labeled, Unitized Logic
  - Clauses are labeled with ID (rule name)
  - Bundled to clause sets (modules) with module ID

- Dynamic Logic
  - Updates: Add / Remove / Change extensional and intensional knowledge
  - Transition to new knowledge state: $P' = P \cup U_{oid}^{pos}$ or $P' = P \setminus U_{oid}^{neg}$
  - Transition: <P,E,U> → <P',U,U'>

- Transactional Updates
  - Safeguarded by Integrity Constraints / Test Cases
  - Transition into hypothetical / pending state: P → P^{hypo}
  - If test fails rollback P'=P else commit $P'=P^{hypo}$

Complex Events / Actions

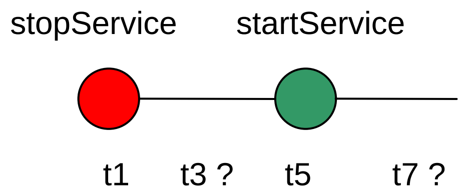
- Different Event / Action Definitions
  - Active Database
    - Complex Event Algebra: Transient complex event occurrences ~ detection time of terminating event
  - Event Notification Systems
    - Sequence of event messages (following a protocol)
  - KR Event / Action Logics
    - Formalized axioms to represent happened or planned non-transient events
    - Temporal reasoning over effects of events / actions

- Interval-based Event Calculus
  - Classical Event Calculus

<u>Example:</u>

initiates*(stopService,serviceUnavailable,T)*
terminates*(startService,serviceUnavailable,T)*
happens*(stopService,t1);* ***happens****(startService,t5)*

holdsAt*(serviceUnavailable,t3)?* → ***<u>true</u>***
holdsAt*(serviceUnavailable,t7)?* → ***<u>false</u>***



  - Interval-based Event Calculus
    - Event interval: [E1,E2], E1 = Initiator, E2 = Terminator
    - Time interval: [T1,T2]

Examples:

occurs(e1,[t1,t1]). **occurs**(e2,[t2,t2])

holdsInterval([e1,e2],[t1,t2])?

Complex KR Event / Action Algebra

- Event / Action Algebra based on Interval-based Event Calculus

```
(A;B;C) ≡ detect(e,[T1,T3]) :-  holdsInterval([a,b],[T1,T2],[a,b,c]),
                                holdsInterval([b,c],[T2,T3],[a,b,c]),
                                [T1,T2]<=[T2,T3].
```

- Meta Program for Algebra Operators
  - Sequence, conjunction, or, xor, concurrent, neg, any, aperiodic

```
detect(e,T):-  event(sequence(a,b),T),              % detection condition for the event e
               update(eis(e), "occurs(e,_0).“, [T]), % add e with key eis(e)
               consume(eis(a)), consume(eis(b)).% consume all a and b events
```

- Transient events: occurs(E,T) vs. Non-Transient Events: happens(E,T)
- Consume: Remove transient events from event instance sequence (managed by ID)

SLA Scenario with States, Rights and Obligations

Service Level Agreement

Schedule	Time	Availability	Response Time
Prime	8 a.m. -18 p.m.	98%[99%]100% ; pinged every 10s	4 sec.; pinged every 10s
Standard	18 p.m. -8 a.m.	95%[97%]99%; pinged every min.	10[14]16 sec.; pinged every min.
Maintenance	0 a.m.- 4 a.m.*	20%[50%]80%;pinged every 10 min	No monitoring

Price	Base	Bonus	Malus
Prime	p_{prime}	$p_{prime} + (x_{high}-x_{median}) * p_{bonus}$ %	$p_{prime} - (x_{median}-x_{low}) * p_{malus}$ %
Standard	$p_{standard}$	$p_{standard} + (x_{high}-x_{median}) * p_{bonus}$ %	$p_{standard} - (x_{median}-x_{low}) * p_{malus}$ %
Maintenance	$p_{maintenance}$	$p_{maintenance} + (x_{high}-x_{median}) * p_{bonus}$ %	$p_{maintenance} - (x_{median}-x_{low}) * p_{malus}$ %

Level	Role	Time-to-Repair (TTR)	Rights / Obligations
1	Process Manager	10 Min.	Start / Stop Service
2	Quality Manager	Max. Time-to-Repair (MTTR)	Change Service Levels to max values
3	Control Committee	No Limit	All rights

Formalization (1)

…

ECA rule: "If the ping on the service fails and not maintenance then trigger escalation level 1 and notify process manager, else if ping succeeds and service is down then update with restart information and inform responsible role about restart".

```
eca( schedule(T,S), not(available(S)), not(maintenance(S)), escalate(S),_, restart(S)). % ECA rule
available(S) :- WebService.ping(S).          % ping service
maintenance(S) :- sysTime(T), holdsAt(maintenance(S),T).
escalate(S) :- sysTime(T),
        not(holdsAt(unavailable(S),T)),          % escalate only once
        update("outages","happens(outage(_0),_1).",[S,T]),          % add outage event
        role(R), notify (R, unavailable(S)).  % notify
restart(S) :-
        sysTime(T), holdsAt(unavailable(S),T),
        update("outages","happens(restart(_0),_1).",[S,T]),          % add restart event
        role(R), notify(R,restart(S)).          % update + notify
..
% initiate escalation level 1 if outage event happens
terminates(outage(S),escl_lvl(0),T).
initiates(outage(S),escl_lvl(1),T).
…
```

Formalization (2)

…

% define time-to-repair deadline and trigger escalation level 2 if deadline is elapsed

1 time_to_repair(tdeadline). % relative time to repair value (fact)

2 trajectory(escl_lvl(1),T1,deadline,T2,(T2 - T1)) . % deadline function (countdown)

3 derivedEvent(elapsed).

4 happens(elapsed,T) :- time_to_repair(TTR), valueAt(deadline,T, TTR).

5 terminates(elapsed, escl_lvl(1),T). % terminate escalation level 1

6 initiates(elapsed, escl_lvl(2),T). % initiate escalation level 2

…

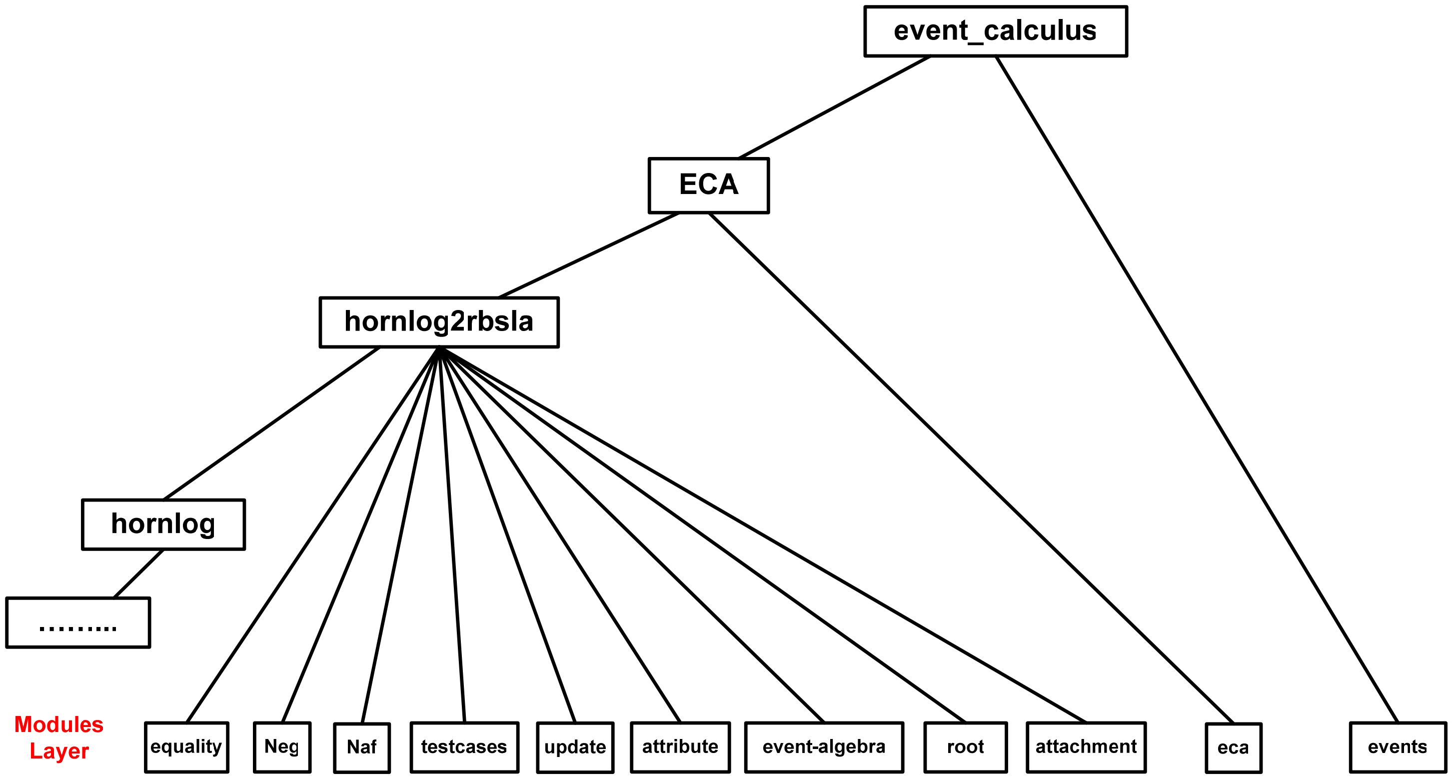
% terminate escalation level 1/2/3 if servicing is started

7 initiates(startServicing(S),escl_lvl(0),T). terminates(startServicing(S), escl_lvl(1),T).
terminates(startServicing(S), escl_lvl(2),T). terminates(startServicing(S),escl_lvl(3),T).

…

ECA-RuleML

- Serialization Syntax for ECA Rules and Event / Action Algebra
  - URL: http://ibis.in.tum.de/staff/paschke/eca-ruleml/index.htm
- Based on RuleML 0.9



ECA-RuleML Example

```
<ECA>
        <time>
                <Cterm>
                        <op><Ctor>everySecond</Ctor></op>        <arg><Var>T</Var></arg>
                </Cterm>
        </time>
        <event>
                <Sequence>
                        <operator>   <Concurrent>
                                             <event>
                                                     <Ind>a</Ind>
                                                     <Ind>b</Ind>
                                             </event>
                        </Concurrent>   </operator>
                        <event>   <Ind>c</Ind>  </event>
                </Sequence>
        </event>
        <condition>
                <HoldsAt>
                        <fluent>
                                <Cterm>
                                        <op><Ctor>state</Ctor></op>
                                        <arg><Var type="java.lang.Integer">1</Var></arg>
                                </Cterm>
                        </fluent>
                </HoldsAt>
        </condition>
        <action>
                <Assert>
                        <oid><Ind>state1</Ind></oid>
                        <formula><Happens>
                                <event><Ind>ab</Ind></event>
                                <time><Var>T</Var></time>
                        </Happens></formula>
                </Assert>
        </action>
</ECA>
```

Event Notification / Messaging Style

- Prova Agent Architecture: Event Notification / Messaging (Alex Kozlenkov)
  - Core concept: Serial Horn Rules
    - combination of updates and conditional body literals (see transaction logics)
  - Communication / Protocol oriented reaction patterns
  - Message reception and variety of outbound / inbound communication actions
  - Distinguish conversations and protocol states
    - Built-in management of correlation, conversation ids, session ids

- Combination of
  - Global ECA Rules (active database ECA paradigm)
  - Event Notification / Messaging
  - KR Event / Action Logics (Event Calculus)

  - General denominator: “Extended Logic Programming”
  - Promises to combine benefits and overcome drawbacks in each domain
  - Questions:
    - Homogeneous syntax?
    - Heterogeneous approach?
    - Minimal extensions to common logic programming vs. extensive built-ins?
    - Reuse standard LP inference engines (Prolog derivates) – What about procedural attachments?

Discussion

- Pros
  - Homogeneously represent ECA rules with derivation rules, integrity rules, defeasible rules etc.
  - Compact ECA syntax, but nevertheless full expressiveness of logic programming
  - KR event / action formalisms such as Event Calculus, Situation Calculus
  - Light-weight ECA interpreter as add-on to arbitrary LP inference systems
    - Forward-directed production rule systems can use ECA terms as final matching constraints
  - Adopts procedural and declarative semantics of logic programming
    - Additional semantics needed to safeguard updates and parallel execution in dynamic LPs (e.g. test cases / integrity constraints)
  - Enables active event processing via procedural attachments (e.g. ping service) or passive event processing (query facts/event data)
  - Reusability of global rules
  - Event context represented by variables might be derived by more or less complex (derivation) rule sets
  - Dynamic changes with update actions safeguarded by integrity constraints
  - Traceability and verifiability of conclusions resp. reactions due to formal semantics
  - Different event/action definitions due to typed logic, e.g. simple test value, complex term/function, Object-oriented (Java) class
  - Supports short and long-term perspective with transient, non-transient and planned events

- Cons
  - Trade-off between expressiveness and complexity
  - Procedural problems such as event storms, non-terminating event answers (Internet), side effects are out of the scope of the logical formalisms and need procedural treatment, e.g. termination of processing threads, event queues.
  - Event Notification / Messaging not directly supported by global ECA rules – Conversation ID /State needs to be managed
  - Extra effort needed to safeguard parallel execution with update actions, e.g. by Event Calculus states and integrity constraints

➔ ECA-LP suitable for higher-level behavioral logic which amounts for formal reasoning and predictability / traceability of triggered actions and concluded results.

Thanks

Questions / Discussion ?