Reaction RuleML: A Reaction Rule Extension of RuleML

Agenda

- Basics / Preliminaries
- Core Syntax Constructs
- Examples
  - Active Global Reaction Rules
  - Active Local Reaction Rules
  - Passive Global Reaction Rules
  - Passive Local Reaction Rules
  - KR Event/Action Logic Reaction Rules
- Complex Event Processing

Adrian Paschke, Alexander Kozlenkov and Harold Boley
Reaction RuleML, Telephone Conference, 2006-10-19
Different Styles of Event/Action Definition and Processing

Active databases
- Instantaneous occurrences of atomic or complex events (defined by event algebra operators)
- Short term perspective (event sequence history to detect complex events)
- ECA paradigm

Production rule systems:
- Implicit sequences of knowledge updates
- Condition $\rightarrow$ Action (mostly Assert / Retract)

Event Messaging / Notification
- Event messages (inbound / outbound messages)
- Follow some protocol, e.g. realized by state machine

KR Temporal / Event / Action Logics
- Persistent non-transient past or planned events
- Long term perspective
- Axioms to formalize events/actions and their effects on knowledge states
- Transitions from one state to another
General reaction rule form that can be specialized as needed

- **Three general execution styles:**
  - **Active:** 'actively' polls/detects occurred events, e.g. by a ping on a service/system or a call on an event database
  - **Passive:** 'passively' waits for incoming events, e.g. an event message
  - **Reasoning:** KR event/action logic reasoning and transitions (as e.g. in Event Calculus, Situation Calculus, ACTL formalizations)

- **Appearance**
  - **Global:** 'globally' defined reaction rule
  - **Local:** 'locally' defined (inline) reaction rule nested in a outer rule

- **Event:** event of reaction rule
  - Production rule systems: Event implicit in starting next cycle
  - Active execution: Actively detect / listen to events (possibly clocked by a time function / monitoring schedule)
  - Passive execution: Passively wait / listen for matching event pattern (e.g. event message)

- **Condition**
  - Production rule system: trigger for action
  - Backward reasoning: top-down goal proof attempt based on derivation rules
  - **Strong condition:** on failure completely terminates the execution, e.g. the message sequence or the derivation process
  - **Weak condition:** on failure proceeds with the derivation or waits for further messages without execution of the action

- **Action**
  - Executes action either as internal knowledge update or externally, e.g. as sendMessage

- **Postcondition**
  - Evaluated after action has been performed
  - **Transactional postcondition:** rolls back action (knowledge update) if failed

- **Alternative Action**
  - Executes alternative action if condition or action fails (akin to "if then else" logic)
Reaction RuleML Syntax – Basic Constructs

- `<Reaction>` General reaction rule construct
- `@exec = "active | passive | reasoning"; default = "passive"`
  - Attribute denoting “active”, “passive” or “reasoning” execution style
- `@kind` Attribute denoting the kind of the reaction rule, i.e. its combination of constituent parts, e.g. „eca“, „ca“, „ecap“
- `@eval` Attribute denoting the interpretation of a rule: “strong | weak”
- `<event>, <body>, <action>, <postcond>, <alternative>`
  - role tags; may be omitted when they can be uniquely reconstructed from positions
- `<Message>` Defines an inbound or outbound message
- `@mode = inbound | outbound`
  - Attribute defining the type of a message
- `@directive = [directive, e.g. FIPA ACL]`
- `<Assert> | <Retract>` Performatives for internal knowledge updates
General Syntax for Reaction Rules

```xml
<Reaction exec="active" kind="ecapa" eval="strong">

  <event>
    <!-- event -->
  </event>

  <body>
    <!-- condition -->
  </body>

  <action>
    <!-- action -->
  </action>

  <postcond>
    <!-- postcondition -->
  </postcond>

  <alternative>
    <!-- alternative/else action -->
  </alternative>

</Reaction>
```
Example 1: Active Global Reaction Rule (ECA) (1)

```xml
<Reaction kind="eca" exec="active">
  <event> <!-- the role tag might be omitted if still unambiguous -->
    <Reaction kind="ea">
      <event>
        <Atom><Rel>everyMinute</Rel><Var>T</Var></Atom>
      </event>
      <action>
        <Atom><Rel>detect</Rel> <Var type="event:EventTypel" mode="-">TroubleTicket</Var>
        <Var>T</Var>
      </Atom>
    </action>
  </event>
</event>

<body>
  <Atom><Rel>maintenance</Rel><Var>T</Var></Atom>
</body>

<action>
  <!-- Boolean-valued procedural attachment -->
  <Atom><oid><Ind uri="rbsla.utils.TroubleSystem,\"/</Ind> <!-- class/object -->
    <Rel in="effect" lang="java">processTicket</Rel> <!-- method -->
    <Var type="event:EventTypel" mode="+">TroubleTicket</Var> <!-- parameter -->
  </Atom>
</action>
</Reaction>
```
ECA-LP/Prova Syntax (related to ISO Prolog notation)

```prolog
eca(
    everyMinute(T), % time precond (clock)
    detect(TroubleTicket:event_EventType1,T), % event
    maintenance(T), % condition
    rbsla.utils.TroubleSystem.ProcessTicket(TroubleTicket:event_EventType1) % action
).

% Formalization of time function „everyMinute(T)“ – omitted in slide 6
everyMinute(T):-
    sysTime(T), % get actual system time/date
    interval(timespan(0,0,1,0), T). % interval function
```
Example 2: Active Global Reaction Rule (CA / Production) (1)

<Reaction kind="ca" exec="active">

  <body>
    <Atom>
      <Rel>occurs</Rel>
      <Expr in="no">
        <Fun>heartbeat</Fun>
        <Var>Service</Var>
      </Expr>
      <Var>T</Var>
    </Atom>
  </body>

  <action>
    <Assert>
      <oid><Ind>availability values</Ind></oid>  <!– OID of update -->
      <Atom>
        <Rel>alive</Rel>
        <Var>Service</Var>
        <Var>T</Var>
      </Atom>
    </Assert>
  </action>

</Reaction>
Example 2: Active Global Reaction Rule (Production Rule) (2)

Production Rule (forward-directed):

\[
\text{occurs}(\text{heartbeat} (\text{Service}), T) \rightarrow \text{assert} ( \text{alive} (\text{Service}, T) )
\]

ECA-LP/Prova Syntax (related to ISO Prolog notation)

\[
\text{eca} (\text{occurs}(\text{heartbeat} (\text{Service}), T), \% \text{condition} \newline \text{add}(\text{"availability values"}, \text{"alive(_0,_1)."}, \text{[Service, T]}) \% \text{action})
\]
Example 3: Active Local Reaction Rule (EA) (1)

<Implies>
  <head>
    <Atom>
      <Rel>available</Rel>
      <Var>Service</Var>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <Rel>service</Rel>
        <Var>Service</Var>
      </Atom>
      <Atom>
        <Rel>sysTime</Rel>
        <Var>T</Var>
      </Atom>
    </And>
  </body>
  <Reaction kind="ea" exec="active" eval="strong">
    ...
    next slide
  </Reaction>

Example 3: Active Local Reaction Rule (EA) (2)

<event>
  <Atom>
    <oid><Ind uri="rbsla.utils.WebService"/> </oid> <!-- object / class-->
    <Rel in="effect" lang="java">ping</Rel> <!-- Boolean-valued method -->
    <Var mode="+">Service</Var>
  </Atom>
</event>

<action>
  <Assert>
    <oid><Ind>id1</Ind></oid> <!-- ID of update -->
    <Atom>
      <Rel>occurs</Rel>
      <Expr in="no">
        <Fun>alive</Fun>
        <Var>Service</Var>
      </Expr>
      <Var>T</Var>
    </Atom>
  </Assert>
</action>

</Reaction>
</And>
</body>
</Implies>
Example 3: Active Local Reaction Rule (EA) (3)

ECA-LP/Prova Syntax (related to ISO Prolog notation)

```prolog
available(Service) :- % a mixed rule with a local reaction rule
    service(Service),
    sysTime(T),
    eca(
        rbsla.utils/WebAPI.ping(Service), % ping service
        _, % no condition
        add(id1,"occurs(alive(_0),_1).",[Service,T]) % add action
    ).
```

---

---

---
Example 4: Passive Global Notification Reaction Rule (1)

<Reaction kind="ea" exec="passive" eval="strong">
  <event>
    <Message mode="inbound" directive="ACL:inform">
      <oid><Var>XID</Var></oid>
      <protocol><Var>Protocol</Var></protocol>
      <sender><Var>From</Var></sender>
      <content><Var>Payload</Var></content> <!--message payload-->
    </Message>
  </event>
  <action>
    <Assert>
      <oid><Ind>opinions</Ind></oid> <!-- OID of update -->
      <Atom>
        <Rel>opinion</Rel>
        <Var>From</Var>
        <Var>Payload</Var>
      </Atom>
    </Assert>
  </action>
</Reaction>
Example 4: Passive Global Notification Reaction Rule (2)

Prova AA Syntax (related to ISO Prolog notation)

rcvMsg(XID, Protocol, From, “inform”, Payload) :-
  add(opinions, "opinion(_0,_1).", [From, Payload]).
Example 5: Passive Local Notification Reaction Rule (1)

<Implies>
  <head>
    <!-- Standard derivation rule head or reaction rule (receive) -->
  </head>
  <body>
    <And>
      <Reaction kind="e" exec="passive" eval="strong">
        <event>
          <Message mode="inbound" directive="ACL:inform">
            <oid><Var>XID</Var></oid>
            <protocol><Var>Protocol</Var></protocol>
            <sender><Var>From</Var></sender>
            <content><Var>Payload</Var></content>
          </Message>
        </event>
      </Reaction>
    </And>
  </body>
</Implies>

... next slide →
Example 5: Passive Local Notification Reaction Rule (2)

<Atom> ... </Atom>

<Reaction kind="a" exec="passive" eval="weak">
  <action>
    <Message mode="outbound" directive="ACL:inform">
      <oid><Var>XID</Var></oid>
      <protocol><Var>Protocol</Var></protocol>
      <sender><Var>From</Var></sender>
      <content><Var>Payload</Var></content>
    </Message>
  </action>
</Reaction>

<Atom> ... </Atom>

</And>
</body>
</Implies>
Example 5: Passive Local Notification Reaction Rule (3)

- Prova AA Syntax (related to ISO Prolog notation)

<normal derivation rule head> :-
rcvMsg(XID, Protocol, From, inform, Payload), %inline event
...,% normal literals
sendMessage(XID, Protocol, From, inform, Payload), % inline action
... .
Example 6: KR Temporal Event/Action Logics

<Reaction kind="ea" exec="reasoning">
  <event>
    <Atom>
      <Rel>happens</Rel>
      <Ind>StartMaintenance</Ind>
      <Var>T</Var>
    </Atom>
  </event>
  <action>
    <Initiates>
      <state>
        <Ind>maintenance</Ind> <!-- fluent / state -->
      </state>
    </Initiates>
  </action>
</Reaction>
Example 6: KR Event/Action Logics (3)

- ECA-LP/Prova Syntax (related to ISO Prolog notation)
  - Event Calculus formalization

\[
\text{happens} (\text{startMaintenance}, t1). \quad \% \text{ fact (omitted in slide 18)}
\]

\[
\text{initiates} (\text{startMaintenance}, \text{maintenance}, T). \quad \% \text{ initiate state}
\]
## Complex Events

### Atomic Events
- simple flat constants `<Ind>`
- nested complex functions `<Expr>`
- complex external objects `<Attachment>` in `<Atom>`; might be bound to variables
- messages `<Message>`

### Complex Events
- Defined by event algebra operators
- Example: `sequence(concurrent(a,b),c)`

```xml
<event>
  <Sequence>
    <Concurrent>
      <Ind>a</Ind>
      <Ind>b</Ind>
    </Concurrent>
    <Ind>c</Ind>
  </Sequence>
</event>
```
Complex Event Processing

■ Event Definition
  ■ Definition of event pattern by event algebra

■ Event Selection
  ■ Defines selection function to select one event from several occurred events (stored in an event instance sequence) of a particular type, e.g. “first”, “last”
  ■ Crucial for the outcome of a reaction rule, since the events may contain different (context) information, e.g. different message payloads or sensing information

■ Event Consumption
  ■ Defines which events are consumed after the detection of a complex event
  ■ An event may contribute to the detection of several complex events, if it is not consumed
  ■ Distinction in event messaging between “multiple receive” and “single receive”
  ■ Events which can no longer contribute, e.g. are outdated, should be removed

Separation of this phases is crucial for the outcome of a reaction rule base in the context of complex events

Declarative configuration of different selection and consumption policies is desirably (also on the syntax layer)
Thank you!

Discussion?

Reaction RuleML Homepage:

http://ibis.in.tum.de/research/ReactionRuleML