Tutorial

Reaction RuleML

Nov. 11th, 2006

Athens, GA, USA

at RuleML'06

Tutorial

Reaction RuleML

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

Nov. 11th, Athens, GA, USA at RuleML'06

Adrian Paschke (Co-Chair Reaction RuleML Technical Group)
IBIS, Technical University Munich

Reaction RuleML Technical Group

Agenda

1. Introduction to Reaction RuleML
   - Intention of Reaction RuleML
   - Goals of Reaction RuleML
   - Relation to RuleML language family
   - Scope of Reaction RuleML

2. Reaction RuleML 0.1
   - Introduction
   - Examples

Reaction RuleML is …

- An open, general, practical, compact and user-friendly **XML-serialization language** for the **family of reaction rules** including:
  - ECA rules and variants such as ECAP rules and triggers (EA rules)
  - Production rules (CA rules)
  - Active rules (rule execution sequences)
  - Event notification and messaging rules including agent communications, negotiation and coordination protocol rules
  - Temporal event / action and state/fluent processing logics
  - Dynamic, update, transaction, process and transition logics

… but not limited to these, due to extensible language design

Reaction RuleML is intended for e.g., ...

- Event Processing Networks
- Event Driven Architectures (EDAs)
- Reactive, rule-based Service-Oriented Architectures (SOAs)
- Active Semantic Web Applications
- Real-Time Enterprise (RTE)
- Business Activity Management (BAM)
- Business Performance Management (BPM)
- Service Level Management (SLM) with active monitoring and enforcing of Service Level Agreements (SLAs) or e-Contracts
- Supply Chain Event Management
- Policies
- …

… where reaction rules of the various kinds can be … (1)

- serialized in a homogeneous combination with other rule types such as conditional derivation rules, normative rules, exceptional, default, prioritizied rules or integrity constraints;

- managed, maintained and interchanged in a common rule markup and interchange language;

- internally layered and unitized to capture sublanguages such as production rules, ECA rules, event notification rules, KR event/action/state processing and reasoning rules;

… where reaction rules of the various kinds can be … (2)

- managed and maintained distributed in closed or open environments such as the (Semantic) Web including different domain-specific vocabularies which must be dynamically mapped into domain-independent rule specifications during runtime
- interchanged, translated and executed in different target environments with different operational, execution and declarative semantics;
- engineered collaboratively and verified/validated statically and dynamically according to extensional but also intensional knowledge update actions which dynamically change the behavioral logic of the event-driven rules systems

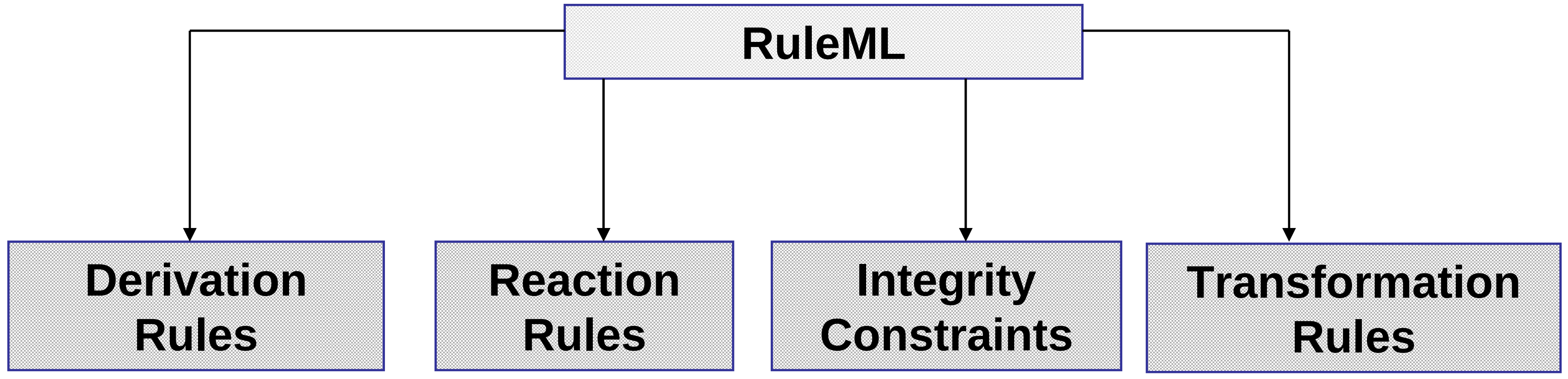
Our goals are …

- to enable interoperation between various domains of event/action definition and processing such as:
  - Active Databases, Production Rules Systems, (Multi) Agent Systems, KR Event/Action Logics and Transactional Dynamic Update Logics, Transition and State Process Systems
- to be an general and open intermediary between various “specialized” vendors, applications, industrial and research working groups and standardization efforts such as:
  - OMG PRR
  - W3C RIF
  - Rewerse (e.g. XChange, R2ML, ECA-ML)

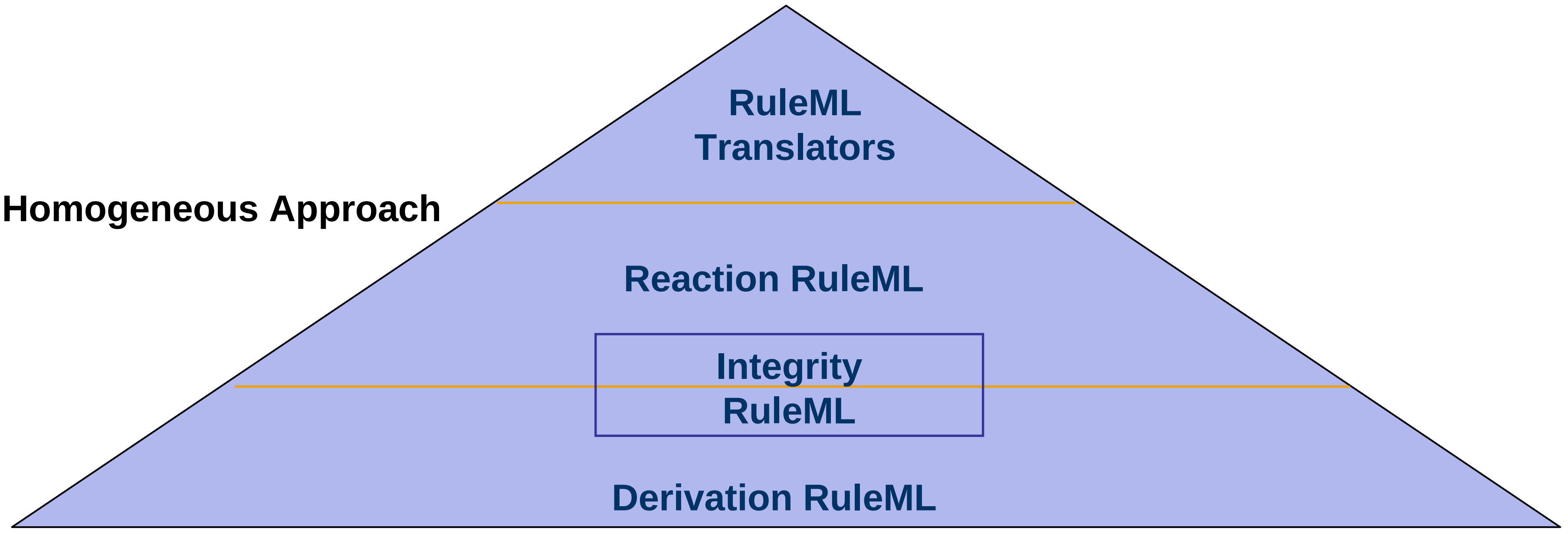
Reaction RuleML as “GLUE” between previously separated approaches to event/action/state definitions and processing/reasoning techniques

Bridging the gap between the divergent notations and terminologies via a general syntactic and semantic design

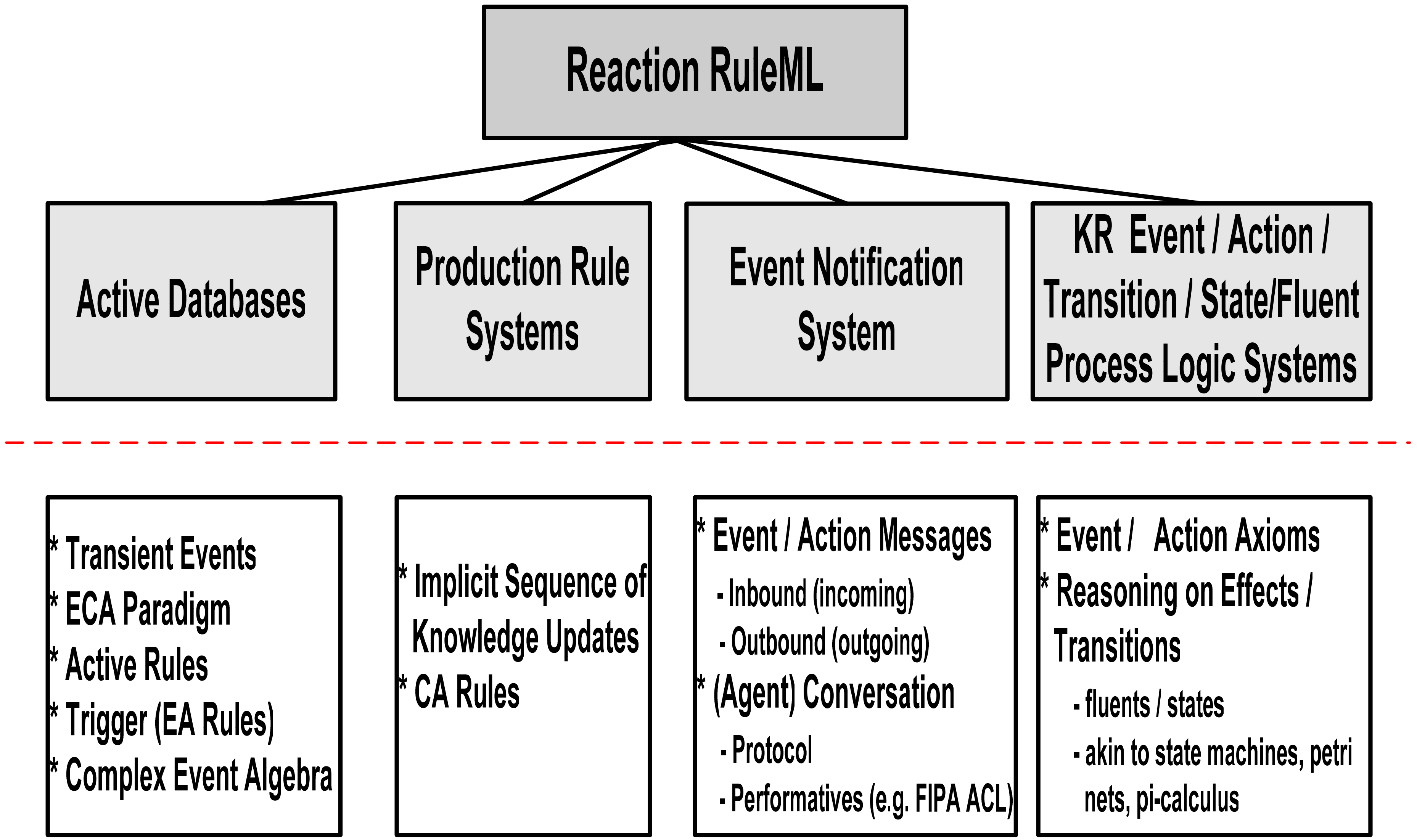
How does Reaction RuleML relate to RuleML?



Homogeneous Approach



Scope of Reaction RuleML (1)


Reaction RuleML
Active Databases
Production Rule Systems
Event Notification System
KR Event / Action / Transition / State/Fluent Process Logic Systems
* Transient Events
* ECA Paradigm
* Active Rules
* Trigger (EA Rules)
* Complex Event Algebra
* Implicit Sequence of Knowledge Updates
* CA Rules
* Event / Action Messages
- Inbound (incoming)
- Outbound (outgoing)
* (Agent) Conversation
- Protocol
- Performatives (e.g. FIPA ACL)
* Event / Action Axioms
* Reasoning on Effects / Transitions
- fluents / states
- akin to state machines, petri nets, pi-calculus

Classification of Event Space – 1st Dimension (1)

- Processing (a.k.a. situation detection or event/action computation / reasoning)
  - **Short term**: Transient, non-persistent, real-time selection and consumption (e.g. triggers, ECA rules): ***immediate reaction***
  - **Long term**: Transient, persistent events, typically processed in retrospective e.g. via KR event reasoning or event algebra computations on event sequence history; but also prospective planning / proactive, e.g. KR abductive planning: ***deferred or retrospective/prospective***
  - **Complex event processing**: computation of complex events from event sequence histories of previously detected raw or other computed complex event (event selection and possible consumption) or transitions (e.g. dynamic LPs or state machines); typically by means of event algebra operators (event definition) (e.g. ECA rules and active rules, i.e. sequences of rules which trigger other rules via knowledge/state updates leading to knowledge state transitions)

Derived from: Paschke, A.: **ECA-RuleML: An Approach combining ECA Rules with temporal interval-based KR Event/Action Logics and Transactional Update Logics**, Internet-based Information Systems, Technical University Munich, Technical Report 11 / 2005.

Classification of Event Space – 1st Dimension (2)

…

- **Deterministic vs. non-deterministic**: simultaneous occurred events give rise to only one model or two or more models
- **Active vs. Passive**: actively detect / compute / reason event (e.g. via monitoring, sensing akin to periodic pull model or on-demand retrieve queries) vs. passively listen / wait for incoming events or internal changes (akin to push models e.g. publish-subscribe)

Classification of Event Space – 2nd Dimension

- Type
  - **Flat vs. semi-structured compound data structure/type**, e.g. simple String representations or complex objects with or without attributes, functions and variables
  - **Primitive vs. complex**, e.g. atomic, raw event or complex derived/computed event
  - **Temporal**: Absolute (e.g. calendar dates, clock times), relative/delayed (e.g. 5 minutes after …), durable (occurs over an interval), durable with continuous, gradual change (e.g. clocks, countdowns, flows)
  - **State or Situation**: flow oriented event (e.g. “server started”, “fire alarm stopped”)
  - **Spatio / Location**: durable with continuous, gradual change (approaching an object, e.g. 5 meters before wall, “bottle half empty”)
  - **Knowledge Producing**: changes agents knowledge belief and not the state of the external world, e.g. look at the program → effect

Classification of Event Space – 3rd Dimension

- Source
  - **Implicit** (changing conditions according to self-updates) vs. **explicit** (**internal** or **external** occurred/computed/detected events) (e.g. production rules vs. ECA rules)
  - **By request** (query on database/knowledge base or call to external system) vs. **by trigger** (e.g. incoming event message, publish-subscribe, agent protocol / coordination)
  - **Internal database/KB update events** (e.g. add, remove, update, retrieve) or **external explicit events** (inbound event messages, events detected by external systems): **belief update and revision**
  - **Generated/Produced (e.g. phenomenon, derived action effects) vs. occurred (detected or received event)**

Classification of the Action Space (1)

- **Similar dimensions as for events**
- Temporal KR event/action perspective: (e.g. Event, Situation, Fluent Calculus, TAL)
  - Actions with effects on changeable properties / states, i.e. actions ~ events
  - ➔ Focus: reasoning on effects of events/actions on knowledge states and properties
- KR transaction, update, transition and (state) processing perspective: (e.g. transaction logics, dynamic LPs, LP update logics, transition logics, process algebra formalism)
  - Internal knowledge self-updates of extensional KB (facts / data) and intensional KB (rules)
  - Transactional updates possibly safeguarded by post-conditional integrity constraints / test case tests
  - Complex actions (sequences of actions) modeled by action algebras (~event algebras), e.g. delayed reactions, sequences of bulk updates, concurrent actions
  - ➔ Focus: declarative semantics for internal transactional knowledge self-update sequences (dynamic programs)
  - External actions on external systems via (procedure) calls, outbound messages, triggering/effecting

Classification of the Action Space (2)

- Event Messaging / Notification System perspective
  - Event/action messages (inbound / outbound messages)
  - Often: agent / automated (web) service communication; sometimes with broker, distributed environment, language primitives (e.g. FIPA ACL) and protocols; event notification systems, publish / subscribe
  - Focus: often follow some protocol (negotiation and coordination protocols such as contract net) or publish-subscribe mechanism

Classification of the Action Space (3)

- Production rules (OPS5, Clips, Jess, JBoss Rules/Drools, Fair Isaac Blaze Advisor, ILog Rules, CA Aion, Haley, ESI Logist, …)
  - Mostly forward-directed operational semantics for Condition-Action rules
  - Primitive update actions (assert, retract); update actions (interpreted as implicit events) lead to changing conditions which trigger further actions, leading to sequences of triggering production rules
  - But: approaches to integrate negation-as-failure and declarative semantics exist:
    - E.g. for subclasses of production rules systems such as stratified production rules with priority assignments or transformation of the PR program into a normal LP
    - Related to serial Horn Rule Programs

Classification of the Action Space (4)

- Active Database perspective (e.g. ACCOOD, Chimera, ADL, COMPOSE, NAOS, HiPac)
  - ECA paradigm: “*on Event and Condition do Action*”; mostly operational semantics
  - Instantaneous, transient events/actions according to their detection time
  - Complex events: event algebra (e.g. Snoop, SAMOS, COMPOSE) and active rules (sequences of self-triggering ECA rules)

Classification of the Event / Action / State Definition and Processing / Reasoning Space (1)

1. Event/Action Definition
   - Definition of event/action pattern by event algebra
   - Based on declarative formalization or procedural implementation
   - Defined over an atomic instant or an interval of time, events/actions, situation, transition etc.

2. Event/Action Selection
   - Defines selection function to select one event from several occurred events (stored in an event instance sequence e.g. in memory, database/KB) 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
   - **KR view**: Derivation over event/action history of happened or future planned events/actions

3. Event/Action Consumption / Execution
   - 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
   - **KR view**: events/actions are not consumed but persist in the fact base

Classification of the Event / Action / State Definition and Processing / Reasoning Space (2)

4. State / Transition Processing
   - Actions might have an internal effect i.e. change the knowledge state leading to state transition from (pre)-condition state to post-condition state.
   - The effect might be hypothetical (e.g. a hypothetical state via a computation) or persistent (update of the knowledge base),
   - Actions might have an external side effect

➔ Separation of these phases is crucial for the outcome of a reaction rule base, since typically events occur in a context and interchange context data with the condition or action (e.g. via variables, data fields)

➔ Declarative configuration and semantics of different selection and consumption policies is desirably (also in the syntax layer)

Design Principles of Reaction RuleML (1)

- XML Schema + EBNF Syntax
- Full RDF compatibility via type and role tags (akin to triple syntax); but certain role tags can be omitted
- Reaction RuleML is intended to be transformed into a target execution language of an underlying rule-based or event/action-driven systems
- XML Schema Modularization: Layered and uniform design
  - The layers are organized around increasing expressiveness levels
  - Benefits:
  - easier to learn the language and to understand their relationships
  - facilitates reusability and complex language assemblies from modules
  - provides certain guidance to vendors who might be interested only in a particular subset of the features
  - easier to maintain, manage and extend in a distributed environment

Design Principles of Reaction RuleML (2)

- Reaction RuleML also facilitates **declarative programming with state / event / action processing rules;** it is not just a specification language;

- Fulfils typical criteria for good language design such as *minimality*, *symmetry* and *orthogonality*

- Satisfies typical KR adequacy criteria such as *epistemological adequacy* in view of expressiveness of the language

Part II: Reaction RuleML 0.1

Examples

General Concepts (1)

- 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 query on an internal or external 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, TAL formalizations)
- Appearance
  - **Global**: ‘globally’ defined reaction rule
  - **Local**: ‘locally’ defined (inline) reaction rule nested in an outer rule

General Concepts (2)

- Event: event of reaction rule
  - 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
  - Forward-directed production rule system: trigger for action
  - Backward-reasoning: top-down goal proof attempt based on derivation rules or query on external data source
  - **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 or procedural call on an external system, which executed the action.
- Postcondition
  - Formalizes the knowledge state after the action execution and it is evaluated after the action has been performed
  - **e.g. transactional postcondition test** (e.g. an integrity constraint): 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“ **exec**ution style

- **@kind** = Attribute denoting the **kind** of the reaction rule, i.e. its combination of constituent parts, e.g. „*eca*“, „ca“, „*ecap*“; is used to **select subsets of full reaction rules**

- **@eval** = Attribute denoting the interpretation of a rule: “*strong* | *weak*”

- **<event>,<body>,<action>,<postcond>, <alternative>**
  - role tags; in certain cases may be omitted

Reaction RuleML Syntax – Basic Constructs (2)

- **<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

… glossary on further constructs such as complex event/action algebra operators on the Reaction RuleML website (http://ibis.in.tum.de/research/ReactionRuleML)

General Syntax for Reaction Rules

```
<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>
```

Examples

Example 1: Production Rule (CA Rule) (1)

“If a heartbeat of a service is recorded to occur at some time then the service is asserted to be alive at that time.”

```
Condition          →          Action
```

Example 1: Active Global Reaction Rule (CA / Production) (2)

```
<Reaction kind=“ca“ exec=“active”>
   <body>
        <Atom>
           <Rel>occurs</Rel>
            <Expr>
                 <Fun in=“no”>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 1: Active Global Reaction Rule (Production Rule) (3)

- Production Rule (forward-directed):

```
(occurs (heartbeat ?service), ?t) => (assert ( alive ?service, ?t))
```

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

```
eca(
  occurs(heartbeat(Service),T),    % condition
  add(“availability values”,”alive(_0,_1).”, [Service, T]) % action
).
```

Example 2: Active Trigger Rule (EA Rule) (1)

“If a heartbeat of a service is detected to occur at some time then the service is asserted to be alive at that time.”

```
Event        →        Action
```

Example 2: Active Global Reaction Rule (Trigger EA Rule) (2)

```
<Reaction kind=“ea“ exec=“active”>
   <event>
        <Atom>
           <Rel>occurs</Rel>
           <Expr>
                 <Fun in=“no”>heartbeat</Fun><Var>Service</Var>
           </Expr>
           <Var>T</Var>
        </Atom>
   </event>
   <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 (Trigger) (3)

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

```
eca(
 _,             % empty time part
 occurs(heartbeat(Service),T),   % event
  _,            % empty condition
 add(“availability values”,”alive(_0,_1).”, [Service, T]),
 _,_            % empty post-cond. and alternative action
).
```

Example 3: Event Notification Trigger (EA Rule) (1)

“**on** *ACL:inform(XID, Protocol, From, Payload)*

do *assert(opinion(From, Payload)”*

on *Event Notification* **do** *Update Action*

Example 3: Passive Global Notification Reaction Rule (2)

```
<Reaction kind=“ea“ exec=“passive” eval="strong”>

   <event>
      <Message mode="inbound” directive=“ACL:inform">
         <oid><Var>XID</Var></oid>
         <protocol><Var>Protocol</Var>
         <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 3: Passive Global Notification Reaction Rule (3)

- Prova Agent Architecture Syntax (related to ISO Prolog notation)

```
rcvMsg(XID,Protocol,From,“ACL:inform",Payload) :-
      add(opinions,"opinion(_0,_1).",[From,Payload]).
```

Example 4: Timed Event Condition Action (ECA Rule) (1)

Every minute **on** *detection of a new trouble ticket* and currently *not maintaining* **do** call the *Trouble Ticket System and process the trouble ticket*.

`on` *`Timer`* **`detect`** *`Event`* **`and`** *`Condition`* **`do`** *`Action`*

Example 4: Active Global Reaction Rule (ECA) (2)

```
<Reaction kind="eca" exec=“active”>
   <event>
       <Reaction kind="ea">
        <event>
          <Atom>
              <Rel>everyMinute</Rel>
            <Var>T</Var>
          </Atom>
        </event>
        <action>
          <Atom>
            <Rel>detect</Rel>
            <Var type="event:EventType1”
                 mode=“-”>TroubleTicket</Var>
            <Var>T</Var>
          </Atom>
        </action>
       </Reaction>
    </event>
... next slide
```

Example 4: Active Global Reaction Rule (ECA) (3)

```
   <body>
     <Naf>
        <Atom>
            <Rel>maintenance</Rel>
            <Var>T</Var>
         </Atom>
      </Naf>
   </body>
    <action>
       <!– Boolean-valued "procedural attachment" on
           incident management system -->
       <Atom>
             <!-- class/object -->
             <oid><Ind uri="rbsla.utils.TroubleTicketSystem"/></oid>

             <!-- method -->
             <Rel in="effect" lang="java">processTicket</Rel>
             <!-- parameter -->
             <Var type="event:EventType1“
                  mode="+">TroubleTicket</Var>
       </Atom>
    </action>
</Reaction>
```

Example 4: Active Global Reaction Rule (ECA) (4)

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

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


% Formalization of time function „everyMinute(T)“
everyMinute(T):-
   sysTime(T),   % get actual system time/date
   interval(timespan(0,0,1,0), T).% interval function
```

Example 4: Active Global Reaction Rule (ECA) (5)

```
% Formalization of event detection
detect(TroubleTicket:event_EventType1,T) :-
      occurs(TroubleTicket:event_EventType1,T),
      consume(TroubleTicket:event_EventType1,T).


% Formalization of condition
maintenance(T) :- holdsAt(maintenance,T).

% Event Calculus state processing rules
initiates(startingMaintenance,maintenance,T).
terminates(stopingMaintenance,maintenance,T).
```

Example 5: A complex Timed Reaction Rule

"E*very 10 seconds it is checked (**timer event**) whether there is an incoming request by a customer to book a flight to a certain destination (**event**).*

*Whenever this event is detected, a database look-up selects a list of all flights to this destination (**condition**) and tries to book the first flight (**action**).*

*In case this action fails, the system will backtrack and try to book the next flight in the list otherwise it succeeds sending a "flight booked" notification and terminates processing (**post-condition cut**).*

*If no flight can be found to this destination, i.e. the condition fails or the found flights could not be booked, the **alternative action** is triggered, sending a "booked up" notification back to the customer."*

Example 5: Active Global Reaction Rule (ECAP) (2)

```
<Reaction kind="ecapa" exec=“active”>
    <event>
       <Reaction kind="ea">
         <event>  <!–  Timer Event -->   </event>
         <action> <!- Detect Request --> </action>
       </Reaction>
    </event>
   <body> <!– select list of flights --> </body>
   <action> <!– Book flight -->   </action>
   <postcond> <!– Apply cut -->   </postcond>
   <alternative> <!- Send Failure --> </alternative>
</Reaction>
```

Example 5: ECA-LP Formalization (3)

```
eca(
 every10Sec(),
 detect(request(Customer, Destination),T),
 find(Destination, Flight),
 book(Customer, Flight),
 !,
 notify(Customer, bookedUp(Destination))
).
```

... next slide

Example 5: ECA-LP Formalization (4)

```
% time derivation rule
every10Sec() :-  sysTime(T), interval(
 timespan(0,0,0,10),T).


% event derivation rule
detect(request(Customer, FlightDestination),T):-
      occurs(request(Customer,FlightDestination),T),
      consume(request(Customer,FlightDestination)).


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

... next slide

Example 5: ECA-LP Formalization (5)

```
% action derivation rule
 book(Cust, Flight) :-
     flight.BookingSystem.book(Flight, Cust),
     notify(Cust,flightBooked(Flight)).



% alternative action derivation rule
 notify(Customer, Message):-
     sendMessage(Customer, Message).
```

Example 6: Complex Event Definition

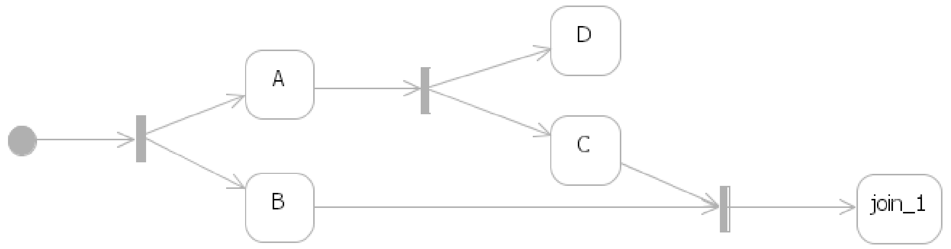
Reaction RuleML

```
<event>
  <Sequence>
      <Concurrent>
           <Ind>a</Ind>
           <Ind>b</Ind>
      </Concurrent>
      <Ind>c</Ind>
  </Sequence>
</event>
```

ECA-LP

```
detect(ce,T):-
         event(sequence(concurrent(a,b),c),T),
         consume(eis(a)), consume(eis(b), consume(eis(c))).
```

Example 7: Prova as a pattern- and rule-based workflow language



```
process_join() :-
     iam(Me),
     init_join(XID,join_1,[c(_),b(_)]),
     fork_a_b(Me,XID).
fork_a_b(Me,XID) :-
     rcvMsg(XID,self,Me,reply,a(1)),
     fork_c_d(Me,XID).
fork_a_b(Me,XID) :-
     rcvMsg(XID,self,Me,reply,b(1)),
     join(Me,XID,join_1,b(1)).
fork_c_d(Me,XID) :-
     rcvMsg(XID,self,Me,reply,c(1)),
     % Tell the join join_1 that a new pattern is ready
     join(Me,XID,join_1,c(1)).

% The following rule is invoked by join once all the inputs are assembled.
join_1(Me,XID,Inputs) :-
     println(["Joined for XID=",XID," with inputs: ",Inputs]).

% Prints
% Joined for XID=agent@hostname001 with inputs [[b,1],[c,1]]
```

Reaction RuleML brings the following benefits …

- Compared to traditional event-driven systems, this approach has the following major advantages:
  - rules are externalized and easily shared among multiple applications (avoiding vendor lock-in) ;
  - encourages reuse and shortens development time;
  - changes can be made faster and with less risk;
  - lowers cost incurred in the modification of business and reaction logic;
  - Allows to continuously adapt the rule-based behavioral logic to a rapidly changing business environments, and overcomes the restricting nature of slow IT change cycles;

“Reaction rules constitute the next step in the application of rule-based information system (IS) and decision support systems (DSS) technology aimed at automating reactions to events occurring in open service-oriented Web applications (SOAs)”

Summary

Reaction RuleML offers

- ✓ open, general, practical, compact and user-friendly XML-serialization language for reaction rules of various kinds
- ✓ XML schema, EBNF syntax and RDF syntax compatibility;
- ✓ layered, extensible design with adjustable expressiveness and identification of specialized rule types via characterizing attributes
- ✓ homogeneous representation with other rule types;
- ✓complex event / action / state / transition / process definitions to describe e.g. state machines, Petri nets, or pi-calculus based rule systems or conversation protocols
- ✓ transactional internal and external updates or update sequences including intermediate post-conditional testing and compensating actions / rollbacks
- ✓ homogeneous combination of derivation rules, reaction rules and other rule types
- ✓ support for messaging and notification (e.g. multi agent communication, event notification systems, web service communication, XML based event queries or action triggers)
- ✓ integration of procedural object-oriented functionalities and data via expressive “procedural attachments” which allow to bind dynamically instantiated objects to the rule variables and use the functions and data during the execution or reasoning process, e.g. to query or call relational or XML databases, data warehouses, middleware applications, enterprise beans and other APIs for sensing and effecting
- ✓ tool support with validators, editors and translators for transforming Reaction RuleML into executable languages and applications

… and much more