MARS: Modular Active Rules in the Semantic Web

Erik Behrends, Oliver Fritzen, Wolfgang May, Franz Schenk

Institut für Informatik, Universität Göttingen, Germany

Supported by the EU Network of Excellence

Further Contributors:
Heiko Kattenstroth, Tobias Knabke, Elke von Lienen, Daniel Schubert, Sebastian Spautz, Thomas Westphal
Joint Work with: José Júlio Alferes, Ricardo Amador
Note

Note: this is not a single talk, but a partially redundant collection of slides from different talks.
Background: REWERSE NoE

- “Reasoning on the *Web with Rules and Semantics*”
- one out of several NoEs (with different focuses) in the area of the “Semantic Web”:
  - REWERSE: *rule-based methods*
- about 30 research groups, 150 participating researchers
- in 8 “Working Groups” I1-I5 (Rule Markup, Policies, Typing & Composition, Querying, Dynamics), A1-A3 (Applications: spatial/temporal, personalization, bioinformatics and 2 “Activities”: Education & Training, Technology Transfer
REWERENCE Working Group I5: "Dynamics"

Behavior in the Semantic Web

- General Framework for Evolution and Reactivity in the Semantic Web (Göttingen, Lisbon)
- RuleCore (Skövde)
- Xcerpt/XChange (LMU München)
Excerpts of this talk ...

... have been given on different aspects at the following events in 2005:


- ODBASE 2005, Agia Napa, Cyprus, Okt. 31 - Nov. 4, 2005: An Ontology- and Resources-Based Approach to Evolution and Reactivity in the Semantic Web (Ontology of rules, rule components and languages, and the service-oriented architecture)

- RuleML 2005, Galway, Ireland, Nov. 10-12, 2005: Active Rules in the Semantic Web: Dealing with Language Heterogeneity (Languages and their markup, communication and rule execution model)

- REWERSE A3-I4 Meeting, Hannover, Germany, Nov. 21/22, 2005: A General Framework for Evolution and Reactivity in the Semantic Web
... in the first half of 2006:


- EDBT-Colocated Workshop “Reactitivity in the Semantic Web”, Munich, March 31, 2006: An ECA Engine for Deploying Heterogeneous Component Languages in the Semantic Web (ECA Level + Prototype)


- EID 2006, Brixen-Bressanone, Italy, June 11/12, 2006: An Ontology-Based Approach to Integrating Behavior in the Semantic Web
Excerpts of this talk ... (Cont’d)

... in the second half of 2006:

- Dagstuhl Seminar “Scalable Data Management in Evolving Networks”, IBFI Dagstuhl, Oct. 23-27, 2006: Distributed Processing of Active Rules over Heterogeneous Component Languages in the Semantic Web


... in 2007:

- RR 2007, Innsbruck, Austria, June 7/8, 2007: Rule-Based Active Domain Brokering for the Semantic Web
Further Contributors

At DBIS, Universität Göttingen, Germany:
Erik Behrends, Oliver Fritzen, Franz Schenk
Students: Carsten Gottschlich, Heiko Kattenstroth, Tobias Knabke, Elke von Lienen, Daniel Schubert, Frank Schwichtenberg, Sebastian Spautz, Thomas Westphal

At CENTRIA, Universidade Nova de Lisboa, Portugal:
Ricardo Amador
Students:
Thesis: There is not a single formalism/language for describing and implementing behavior in the Semantic Web.

Hypothesis: Semantical approaches (i.e., not “programming”, but based on an ontology of behavior) follow the Event-Condition-Action paradigm.

Justification: We show that a general framework approach with modular components covering many existing concepts will prove useful for behavior in the Semantic Web.
Part I: Overview and Situation
Semantic Web

“Computer-understandable semantics” of data
(information vs. data)

Independence from the actual data model, (query) language or formalism, and location

Independence from the local schema and terminology

global concepts and names, oriented at a “natural terminology”

ideas of the static (data) level and queries already quite specific (RDF, OWL)
Motivation and Goals

(Semantic) Web:

- XML: bridge the heterogeneity of data models and languages
- RDF, OWL provide a computer-understandable semantics

... same goals for describing behavior:

- description of behavior *in the Semantic Web* expressed in the terminology of the applications,
- semantic description *of behavior* in a meta-ontology

Event-Condition-Action Rules are suitable for both goals:

- operational semantics
- ontology of rules, events, actions
Behavior

- evolution of *individual* nodes (updates + reasoning)
- *cooperative* evolution of the Web (local behavior + communication)
- different abstraction levels and languages
Behavior

- Decentral P2P structure, autonomous nodes
- Communication
- Behavior located in nodes
  - Local level:
    - Based on local information (facts + received messages)
    - Executing local actions (updates + sending messages + raising events)
  - Semantic Web level (in a given application area):
    - Execution located at a certain node, but “acting globally”:
      - Global information base
      - Global actions (including intensional RDF/OWL updates)
Overlapping ontologies and information between different sources:

- updates: in the same way as there are semantic query languages, there must be a semantic update language.

- updating OWL data: just tell (a portal) that a property of a resource changes intensional, global updates ⇒ must be correctly realized in the Web!

- reactivity – see such updates as events where sources must react upon.
Cooperative Evolution of the Semantic Web

There are not only queries, but there are activities going on in the Semantic Web:

- Semantic Web as a base for processes
  - Business processes, designed and implemented in participating nodes: banking, ...
  - Predefined cooperation between nodes: travel agencies, ...
  - Ad-hoc rules designed by users

The less standardized the processes (e.g. human travel organization), the higher the requirements on the Web assistance and flexibility

⇒ local behavior of nodes and cooperative behavior in “the Web”
Communication

⇒ specify and implement propagation by communication/propagation strategies

Propagation of Changes

Information dependencies induce communication paths:

- direct communication: subscribe – *push*
  based on registration; requires activity by provider

- direct communication: polling – *pull*
  regularly evaluate remote query
  – yields high load on “important” sources
  – outdated information between intervals
  + mapping into local data, *view maintenance*
Abstraction Levels

OWL View

+ Reasoning

RDF View

Mapping + Union

XML View

XML View

XML View

RDF

SQL

SQL

XML

XML

Databases

local logical

XML Web

RDF

MARS
Individual Semantic Web Node

- local state, fully controlled by the node
- [optional: local behavior; see later]
- stored somehow: relational, XML, RDF databases
- local knowledge: KR model, notion of integrity, logic Description Logics, F-Logic, RDF/RDFS+OWL
- query/data manipulation languages:
  - database level, logical level
- mapping? – logics, languages, query rewriting, query containment, implementation
- For this local state, a node should guarantee consistency
A Node in the Semantic Web

A Web node has not only its own data, but also “sees” other nodes:

- agreements on ontologies (application-dependent)
- agreement on languages (e.g., RDF/S, OWL)
- how to deal with inconsistencies?
  - accept them and use appropriate model/logics, reification/annotated statements (RDF), fuzzy logics, disjunctive logics
  - or try to fix them ⇒ evolution of the Semantic Web

- tightly coupled peers: sources are known
- predefined communication
- “open” world: e.g. travel planning
A Node in the Semantic Web (Cont’d)

- Non-closed world
- incomplete view of a part of the Web
  - how to deal with incompleteness?
    - different kinds of negation
    - queries, information about events
- how to extend this view?
  - find appropriate nodes
    - information brokers, recommender systems
    - negotiation, trust
  - ontology querying and mapping
- static (model theory) vs. dynamic (query answering in restricted time; detection of changes/events)
- different kinds of logics, belief revision etc.
Global Evolution

Semantic Web as a network of *communicating nodes*.

- Dependencies between different Web nodes,
- global Semantic Web model is an integrating view, overlapping sources → consistency
- (the knowledge of) every node presents an excerpt of it
  - view-like with explicit reference to other sources
    + always uses the current state
    - requires permanent availability/connectivity
    - temporal overhead
- materialize the used information
  + fast, robust, independent
  - potentially uses outdated information
- view maintenance strategies (web-wide, distributed)
Evolution and Behavior

Behavior is ...
... doing something

- when it is required
  - upon user interaction, a message, or a service call
  - as a reaction to an internal event (temporal, update)
  - upon some events/changes in the “world”

Working Hypothesis

⇒ use Event-Condition-Action Rules as a well-known paradigm.
Part II: The Approach
ECA Rules

“On Event check Condition and then do Action”

- Active Databases
- paradigm of *Event-Driven Behavior*,
- modular, declarative specification in terms of the domain ontology
- sublanguages for specifying *Events, Conditions, Actions*
- simple kind (database level): triggers
- high level: Business Processes, described in terms of the domain ontology
  - react on an event “somewhere in the Web”
ECA Rules

“On Event check Condition and then do Action”

- paradigm of *Event-Driven Behavior*,
- modular, declarative specification in terms of the domain ontology
- sublanguages for specifying *Events, Conditions, Actions*
- *global* ECA rules that act “in the Web”

Requirements

- ontology of behavior aspects
- modular markup definition
- implement an operational and executable semantics
Applications do not only have an ontology that describes static notions:
- cities, airlines, flights, hotels, etc., relations between them ...

But also an ontology of events and actions:
- cancelling a flight, cancelling a (hotel, flight) booking,

Allows for correlating actions, events, and derivation of facts:
- intensional derived events are described in terms of actual events.
  e.g., “economy class of flight X is now 50% booked”
  (derived by “if simple event and condition then (raise) derived event”)

MARS
MARS’ Underlying Paradigm: ECA Rules

“On Event check Condition and then do Action”

- paradigm of Event-Driven Behavior,

- modular, declarative specification in terms of the domain ontology

![Diagram of ECA-ML Language and its components]

ECA-ML Language

Active Concepts Ontologies

Event Language

Query Language

Test Language

Action Language

Application-Domain Language

Domain Ontologies

Event Language

Query Language

Test Language

Action Language

Application-Domain Language

ECA-ML Language embeds Event Language

ECA-ML Language embeds Query Language

ECA-ML Language embeds Test Language

ECA-ML Language embeds Action Language

Composite Events

Queries

Conditions

Complex Reactions

Atomic Events

Literals

Atomic Actions
Events and Actions in the Semantic Web

Applications do not only have an ontology that describes static notions
- cities, airlines, flights, etc., relations between them...

but also an ontology of events and actions
- cancelling a flight, cancelling a (hotel, flight) booking,

Domain languages also describe behavior:
Adding Events and Actions to the Ontologies

- Domain languages also describe behavior:

  - Events
    - `<trvl:canceled-flight flight="LH123">`
    - `<trvl:reason>...</trvl:reason>`
    - `</trvl:canceled-flight>`
  
  - Actions
    - `<trvl:cancel-flight flight="AF456">`
    - `<trvl:reason>...</trvl:reason>`
    - `</trvl:cancel-flight>`

- Ontology of behavior aspects
- correlate and axiomatize actions, events and state
- combine application-dependent semantics with generic concepts/patterns of behavior
Ontologies with Active Notions (Cont’d)

There are not only atomic events and actions.

Ontologies also define the following:

- Derived/complex events, specified by some formalism over simpler events (usually an event algebra, e.g., SNOOP)
- Composite actions = processes, specified by a process algebra over simpler actions, e.g. CCS
Abstraction Levels and Types of Rules

- **Global Level Events**
  - ECA Rules
  - ACE Mapping

- **Global Level Actions**

- **Web-Wide**
  - **Local to the Node**

- **Domain (RDF)**
  - **Local Level Events**
    - ECE Deriv.

- **Domain Ontology**

- **Local Data Model**
  - **Local Level Events**
  - ECA triggers
  - database level: actions=events

- **Local (RDF,XML,SQL)**
  - **Local Level Actions**
  - ACA Reduct.

- **MARS**
  - 24
Behavior on the Web: Abstraction Levels

- OWL ontology level: *Business Processes*
- XML/RDF level:
  - cooperation and communication between closely coupled nodes on the XML Web level
  - local behavior of an application on the logical level
- database level: internal behavior (cf. SQL triggers) in terms of database items

**Additional Derivation and Implementation Rules**

- high-level actions are translated to lower levels
- events are derived from
  - lower-level events, same-level events
  - same-level actions
Sources of Events

- local events: updates on the local knowledge
  - database level: updates of tuples, insertion into XML data
  - actions on the ontology level (e.g., banking:transfer(Alice, Bob, 200) or cancel-flight(LH0815))

- application-independent events: communication events, system events, temporal events
Ontologies including Dynamic Aspects

- Application-Domain Ontology
  - Atomic Events
  - Concepts
  - Atomic Actions

- correlate actions, state, and events
Ontologies including Dynamic Aspects

Ontologies of Application-Independent Domains

- messaging, time,
- database level events

Atomic Events \(\rightarrow\) Concepts \(\rightarrow\) Atomic Actions

talk about

Application-Domain Ontology

Atomic Events \(\rightarrow\) Concepts \(\rightarrow\) Atomic Actions

correlate actions, state, and events
Example: Travel Domain

- defines an ontology

**Individual Nodes**

- access to train/flight schedules, hotels etc.
- allow for actions (book a ticket, cancel a flight)
- emit events (delayed or cancelled flights)

```xml
<travel:canceled-flight flight="LH123">
  <travel:reason>bad weather</travel:reason>
</travel:canceled-flight>
```

- rules for deriving events are also part of the domain ontology ("flight fully booked")
Triggers on the XML Level

- similar to SQL triggers:
  
  ON event WHEN condition BEGIN action END

- event is an (update) event on the XML level
  
  - immediately caused and identical with an update action
  
  - native storage: DOM Level 2/3 events
  
  - relational storage: must be raised/detected internally

Tasks of triggers:

- local behavior of a node (including consistency preservation),

- raise (=derive) application-level events.
Events on the XML Level

- **ON** \{DELETE | INSERT | UPDATE\} **OF** `xsl-pattern`: operation on a node matching the `xsl-pattern`,

- **ON MODIFICATION OF** `xsl-pattern`: update anywhere in the subtree,

- **ON INSERT INTO** `xsl-pattern`: inserted (directly) into a node,

- **ON** \{DELETE | INSERT | UPDATE\} \{SIBLING \{IMMEDIATELY\}\} \{BEFORE | AFTER\} `xsl-pattern`: insertion of a sibling

$\Rightarrow$ extension to the local database (e.g., eXist), easy to combine with XUpdate “events”
Sample Rule on the XML Level

- reacts on an event in the XML database
- here: maps it to an event on the RDF level
- actually an *ECE derivation rule*

```xml
ON INSERT OF department/professor
let $prof:= :NEW/@rdf-uri,
    $dept:= :NEW/parent::department/@rdf-uri
RAISE RDF_EVENT(INSTALL OF has_professor OF department)
with $subject:= $dept, $property:= has_professor, $object:= $prof;
```
Triggers on the RDF Level

Events on the RDF Level

**ON** \{INSERT\|DELETE\|UPDATE\} **OF** property [OF INSTANCE OF **class**].

**ON** \{CREATE\|UPDATE\|DELETE\} **OF** INSTANCE **OF** **class**: if a resource of a given class is created/updates/deleted.

On the RDF/RDFS level, also metadata changes are events:

**ON** NEW **CLASS**,

**ON** NEW PROPERTY [OF **CLASS** **class**]
Sample Rule on the RDF Level

- reacts on an event on the RDF view level
- again an *ECE derivation rule*: derives an event of the domain ontology

```
ON INSERT OF has_professor OF department
  % (comes with parameters $subject=dept,
  %   $property:=has_professor and $object=prof)
  % $university is a constant defined in the (local) database
RAISE EVENT
  (professor_hired($object, $subject, $university))
```
Actions and Events

Logical events differ from actions: an event is an observable (and volatile) consequence of an action.

- **action**: “book flight LH0815 FRA-LIS for Alice on 20.3.2006”
  
  `<travel:book-flight person="Alice" flight="LH0815" date="20.3.2006"/>
  
  **effect**: an update in the Lufthansa database

  **directly resulting event**:
  
  `<travel:booked-flight person="Alice" flight="LH0815" date="20.3.2006" seat="18A"/>
  
  **Ontology**:  
  
  travel:flight rdf:type mars:Class  
  travel:book-flight rdf:type mars:Action  
  travel:booked-flight rdf:type mars:Event

MARS
Derived Events

Other events can “result” from the above change:

```xml
<travel:fully-booked flight="LH0815" date="20.3.2006"/>
<travel:all-flights-fully-booked from="FRA" to="LIS" date="20.3.2006"/>
```

- can be raised from the database updates (triggers), or
- can be *derived* by a local rule:

  second is more semantical and allows for reasoning:

  - on `<book-flight flight=X date=D/>` if ...
  - then raise `<fully-booked flight=X date=D/>` domain-inherent and local to the node;
  - on `<book-flight flight=X date=D/>` if ...
  - then raise `<all-flights-fully-booked from=F to=T/>` domain-inherent and involves many nodes.
Global and Remote Events

Events are caused by updates to a certain Web source. Applications are not actually interested where this happens. Global application-level events “somewhere in the Web”

- “on change of VAT do ...”
- “if a flight is offered from FRA to LIS under 100E”

⇒ requires detection/communication strategies

... so far to the analysis of events and actions. Let’s continue with the rules.
Analysis of Rule Components

... have a look at the clean concepts:
“On Event check Condition and then do Action”

- **Event**: specifies a rough restriction on what *dynamic* situation probably something has to be done. Collects some parameters of the events.

- **Condition**: specifies a more detailed condition, including *static* data if actually something has to be done. ⇒ evaluate a ((Semantic) Web) query.

- **Action**: actually *does* something.

**Example**

“if a flight is offered from FRA to LIS under 100E and I have no lectures these days then do ...”
SQL Triggers

ON {DELETE|UPDATE|INSERT} ... WHEN where-style condition
BEGIN
    // imperative code that contains
    // - SQL-queries into PL/SQL variables
    // - if ... then ...
END;

- only very simple events (atomic updates)
- WHEN part can only access information from the event
- large parts of evaluating the condition actually happen in the non-declarative PL/SQL program part
  ⇒ no reasoning possible!
the event should just be the dynamic component

“if a flight is offered from FRA to LIS under 100E and I have no lectures these days then do ...”

“100E” is probably contained in the event data (insertion of a flight)

my lectures are surely not contained there

⇒ includes another query before evaluating a condition

SQL: would be in an select ... into ... and if in the action part.
Clean, Declarative “Normal Form”

“On Event check Condition and then do Action”

Rule Components:

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic</td>
<td>static</td>
<td>dynamic</td>
</tr>
<tr>
<td>event</td>
<td>query</td>
<td>action</td>
</tr>
<tr>
<td>collect</td>
<td>test</td>
<td>act</td>
</tr>
</tbody>
</table>

- **Event**: detect just the dynamic part of a situation,
- **Query**: then obtain additional information by queries,
- **Test**: then evaluate a *boolean* condition,
- **Action**: then actually do something.

Component sublanguages: heterogeneous
Modular ECA Concept: Rule Ontology

Rule Model

ECARule

EventComponent
  ↓ uses
  Event Ontology
  ↓ expr.by
  Event Language

ConditionComponent
  ↓ uses
  State Ontology
  ↓ expr.by
  Query Language

ActionComponent
  ↓ uses
  Action Ontology
  ↓ expr.by
  Test Language

Languages Model
(such languages are needed to describe the above things)

Language
  Name
  URI
Rule Markup: ECA-ML

<!ELEMENT rule (event,query*,test?,action+)>
<eca:Rule rule-specific attributes>
  <eca:Event identification of the language>
    event specification, probably binding variables
  </eca:Event>
  <eca:Query identification of the language> <!-- there may be several queries -->
    query specification; using variables, binding others
  </eca:Query>
  <eca:Test identification of the language>
    condition specification, using variables
  </eca:Test>
  <eca:Action identification of the language> <!-- there may be several actions -->
    action specification, using variables, probably binding local ones
  </eca:Action>
</eca:Rule>
Example

Sample Event:  

```xml
<travel:canceled-flight flight="LH123">
  <travel:reason>bad weather</travel:reason>
</travel:canceled-flight>
```

```xml
<eca:Rule>
    <eca:Atomic>
      <travel:canceled-flight flight="{$flight}"/>
    </eca:Atomic>
  </eca:Event>
  <eca:Query>get $email of all passengers of $flight </eca:Query>
  <eca:Test>... </eca:Test>
  <eca:Action>tell each $email that $flight is cancelled</eca:Action>
</eca:Rule>
```
Combination of Ontologies

ECA Ontology

Event Ontology

State Ontology

Action Ontology

Ontologies for Application-Independent Domains

Atomic Events

Literals

Atomic Actions

Application-Domain Ontologies

Atomic Events

Literals

Atomic Actions
Embedding of Languages

... there are not only atomic events and actions.
Embedding of Languages

ECA Language:
<Event/> <Query/> <Test/> <Action/>

Domain Languages:

Languages for Application-Independent Domains

Atomic Events

Application-Domain Language

Atomic Events

MARS
Active Concepts Ontologies

- Domains specify atomic events, actions and static concepts

Composite [Algebraic] Active Concepts

- Event algebras: composite events
  - (when) $E_1$ and some time afterwards $E_2$ (then do $A$)
  - (when) $E_1$ happened and then $E_2$, but not $E_3$ after at least 10 minutes (then do $A$)
  - well-investigated in Active Databases (e.g. SNOOP).

- Process algebras (e.g. CCS)

⇒ See concepts defined by these formal methods as defining ontologies.
**Active Concepts Ontologies**

- **Domains**: atomic events, actions and static concepts
- **Event algebras**: composite events (e.g. SNOOP)
- **Process algebras**: composite actions and processes (e.g. CCS)

Consist of *composers/operators* to define composite events/processes,
leaves of the terms are atomic domain-level events/actions,
as operator trees: “standard” XML markup of terms
RDF markup as languages,
every expression can be associated with its language.
⇒ See concepts defined by these *formal methods* as defining *ontologies*. 
Algebraic Sublanguages

ComponentLanguage

DomainEngine

DomainLanguage

name

Primitive

arity

AlgebraicLanguage

name

Composer

/arity

cardinality

Parameter

name

Semantics

↓impl

Processor

1..*

*
Opaque Components

Compatibility with current Web standards:

- current (query) languages do in general not use markup, but program code
- allow *opaque* components:
  - query component: XQuery, XPath, SQL
  - action component: updates in XQuery, XUpdate, SQL
Syntactical Structure of Expressions

- RuleComponent
  - Expression
    - AtomicExpr
      - Variable
        - Parameter
          - name
    - CompositeExpr
      - OpaqueSpec
        - Composer
          - DomainLanguage
            - Language
              - AlgebraicLanguage

as operator trees: “standard” XML markup of terms
RDF markup as languages
every expression can be associated with its language
Subconcepts and Sublanguages

- different languages, different expressiveness/complexity
- common structure: algebraic languages

- e/q/t/a subelements contain a language identification, and appropriate contents
- embedding of languages according to language hierarchy:
  - algebraic languages have a natural term markup.
  - every such language “lives” in an own namespace,
  - domain languages also have an own namespace,
- information flow between components by logical variables,
- (sub)terms must have a well-defined result.
ECA Rule Markup

Ontology of behavior:
- ECA rules
  - (composite) events
  - queries/conditions
  - (composite) actions

Define overall structure

Rules as tree structure **patterns**

Rules, components, expressions as resources

Domain ontology:
- atomic events
- atomic actions
- individuals

Logical Variables

Define overall structure from domain ontologies and extend
Rule Semantics/Logical Variables

Deductive Rules: \[ \text{head}(X_1, \ldots, X_n) : \neg \text{body}(X_1, \ldots, X_n) \]

- bind variables in the body
- obtain a set of tuples of variable bindings
- “communicate” them to the head
- instantiate/execute head for each tuple
Rule Semantics/Logical Variables

Deductive Rules:  \[ \text{head}(X_1, \ldots, X_n) : \neg \text{body}(X_1, \ldots, X_n) \]

- bind variables in the body
- instantiate/execute head for each tuple

ECA Rules

- initial bindings from the event
- additional bindings from queries
- restrict by the test
- execute action for each tuple

\[ \text{action}(X_1, \ldots, X_n) \leftarrow \\
\text{event}(X_1, \ldots, X_k), \text{query}(X_1, \ldots, X_k, \ldots X_n), \text{test}(X_1, \ldots, X_n) \]
Rule Semantics

- Deductive rules: variable bindings Body → Head
- Communication/propagation of information by *logical variables*: $E \lor Q \rightarrow T & A$
- Safety as usual (extended with technical details ...)

Diagram:

- ECARule
  - RuleComponent
    - Expression
      - Variable
        - name
      - repr. by 1
      - *pos, neg*
        - free, bound
      - *scopes*

MARS 55
Binding and Use of Variables in ECA Rules

\[
\text{action}(X_1, \ldots, X_n) \leftarrow \\
\text{event}(X_1, \ldots, X_k), \text{query}(X_1, \ldots, X_k, \ldots X_n), \text{test}(X_1, \ldots, X_n)
\]

Semantic Web: Domain Brokers and Domain Nodes
Operational Semantics of Rules

- **Event**: fires the rule
  - returns the sequence that matched the event
  - optional: variable bindings

- **Query**: obtain additional static information
  - returns the answer/set of answers
  - optional: for each answer, restrict/extend variable bindings (join semantics)

- **Condition**:
  - check a boolean condition, constrain variable bindings

- **Action**:
  - do something by using the variable bindings.
Binding and Use of Variables

- Variables can be bound to values, XML fragments, RDF fragments, and (composite) events.
- Logic Programming (Datalog, F-Logic): variables occur free in patterns. Markup uses XSLT-style
  \[<\text{variable name="var-name"}>\text{language-expr}</\text{variable}>\]
  and $\text{var-name}$ inside component expressions.
- Functional style (event algebras, SQL, OQL, XQuery): expressions return a value/fragment. ⇒ must be bound to a variable to be kept and reused.
  \[<\text{Element}
  \text{bind-to-variable="var-name"}>\text{language-expr}</\text{Element}>\]
  on the rule level around a component expression.
<!ELEMENT Rule (Event, Query*, Test?, Action+) >
  <eca:Event
    xmlns:snoop="http://www.semwebtech.org/languages/2006/snoopy#">
    <snoop:Sequence>
      <travel:delayed-flight flight="{$flight}"/>
      <travel:canceled-flight flight="{$flight}"/>
    </snoop:Sequence>
  </eca:Event>
  <eca:Query bind-to-variable="email">
    <eca:Opaque language="http://www.w3.org/xpath">
      doc("http://xml.lh.de")/flights[code="{$flight}"]/passenger/@e-mail
    </eca:Opaque>
  </eca:Query>
  <eca:Action xmlns:smtp="...">
    <smtp:send-mail to="$email" text="..."/>
  </eca:Action>
</eca:Rule>
Event Algebras

... up to now: only simple events. Atomic events can be combined to form composite events. E.g.:

- (when) $E_1$ and some time afterwards $E_2$ (then do $A$)
- (when) $E_1$ happened and then $E_2$, but not $E_3$ after at least 10 minutes (then do $A$)

*Event Algebras* allow for the definition of composite events.

- specifying composite events as terms over atomic events.
- well-investigated in Active Databases (e.g., the SNOOP event algebra of the SENTINEL ADBMS)
Events Subontology

- Event
  - AtomicEvent
    - Application Domain Atomic Event
      - Application Domain
        - identifier
      - EventLanguage
        - EventAlgebra
          - identifier
  - Composite Event Spec
    - Event Operator
      - arity = $k$
Atomic Event Specifications

Sample Event:

```
<travel:canceled-flight flight="LH123">
  <travel:reason>bad weather</travel:reason>
</travel:canceled-flight>
```

Event expressions require an auxiliary formalism for specifying relevant events:

- type of event ("travel:canceled-flight"),
- constraints ("must have a travel:reason subelement"),
- extract data from events ("bind @flight to variable flight")

Sample: XML-QL-style matching

```
<Atomic language="xmlqlmatch">
  <travel:canceled-flight flight="{$flight}"/><travel:reason/></travel:canceled-flight></Atomic>
```
Event Expressions: Languages

- EventExpression
  - Atomic Event Description
  - Composite Event Specification

- Domain Event
  - Domain Ontology
    - Domain Broker
  - Atomic Event Description Formalism
  - EventComposer
    - cardinality
    - EventAlgebra
      - identifier
      - Processor

- Rule Model
  - Ontologies/Languages
  - Engines/Processors

- descripts
- uses
- from

MARS
Event Detection Communication

users, clients

register ECA rule

Event Sources, Domain Brokers

register events

Atomic Event Matcher for formalism AESL

upon matching: varbdgs as <logvars:answers>

Composite Event Detection Service for CEL:

contains atomic event spec in formalism AESL

upon detection: varbdgs as <logvars:answers>

ECA Engine

<eca:Rule>
<eca:Event>
composite event spec in event algebra CEL
</eca:Event>
</eca:Rule>

register composite event spec

register atomic event spec
Sample Markup (Event Component)

```
  <eca:Event bind-to-variable="theSeq"
    xmlns:snoop="http://www.semwebtech.org/languages/2006/snoopy#">
    <snoop:Sequence>
      <snoop:Atomic language="xmlqlmatch">
        <travel:delayed-flight flight="\{"$Flight\}" minutes="\{"$Minutes\}"/>
      </snoop:Atomic>
      <snoop:Atomic language="xmlqlmatch">
        <travel:canceled-flight flight="\{"$Flight\}"/>
      </snoop:Atomic>
    </snoop:Sequence>
  </eca:Event>
</eca:Rule>
```

binds variables:
- Flight, Minutes: by matching
- theSeq is bound to the sequence of events that matched the pattern
Example as RDF
Rule components, subexpressions etc. are resources associated with languages corresponding to the ontologies (event languages, action languages, (auxiliary languages), domain languages)

each language is a resource, identified by a URI.

DTD/XML Schema/RDF description of the language

Algebraic and auxiliary languages: processing engines

Domain Languages: Domain Nodes and Domain Broker Services
Detection of Atomic Events

- **Atomic Data Level Events** [database system ontology; local]

- **Appl.-indep. Domain Events**
  - receive message [common ontology; local] with contents [contents: own ontology] as parameter
  - transactional events [common ontology; local]
  - temporal events [common ontology] provided by services (upon registration)

- **Application-Level Events** [domain ontology]
  - derived/raised by appropriate ECE/ACE rules, (probably also derived from other facts)

- **Composite Events**: event detection algorithm; fed with detection messages from atomic events
Event Component: Event Algebras

- a composite event is detected when its “final” subevent is detected:

\[(E_1 \triangledown E_2)(x, t) \iff E_1(x, t) \lor E_2(x, t),\]
\[(E_1; E_2)(x, y, t) \iff \exists t_1 \leq t : E_1(x, t_1) \land E_2(y, t)\]
\[\neg(E_2)[E_1, E_3](t) \iff \text{if } E_1 \text{ and then a first } E_3 \text{ occurs, without occurring } E_2 \text{ in between.}\]

- “join” variables between atomic events
- “safety” conditions similar to Logic Programming rules

**Result:**
- the sequence that matched the event
- optional: additional variable bindings
Advanced Operators (Example: SNOOP)

- ANY \( (m, E_1, \ldots, E_n)(t) \) :⇔
  \[ \exists t_1 \leq \ldots \leq t_{m-1} \leq t, \ 1 \leq i_1, \ldots, i_m \leq n \text{ pairwise distinct s.t. } E_{i_j}(t_j) \text{ for } 1 \leq j < m \text{ and } E_{i_m}(t) \]

- “aperiodic event”
  \[ \mathcal{A} (E_1, E_2, E_3)(t) :⇔ \]
  \[ E_2(t) \land (\exists t_1 : E_1(t_1) \land (\forall t_2 : t_1 \leq t_2 < t : \neg E_3(t_2))) \]
  “after occurrence of \( E_1 \), report each \( E_2 \), until \( E_3 \) occurs”

- “Cumulative aperiodic event”:
  \[ \mathcal{A}^* (E_1, E_2, E_3)(t) :⇔ \exists t_1 \leq t : E_1(t_1) \land E_3(t) \]
  “if \( E_1 \) occurs, then for each occurrence of an instance of \( E_2 \), collect its parameters and when \( E_3 \) occurs, report all collected parameters”.
  (Same as before, but now only reporting at the end)
Examples of Composite Events

- A deposit (resp. debit) of amount $V$ to account $A$:
  \[ E_1(A, V) := deposit(A, V) \quad (\text{resp. } E_2(A, V) := debit(A, V)) \]

- A change in account $A$:
  \[ E_3 := E_1(A, V) \nabla E_2(A, V). \]

- The balance of account $A$ goes below 0 due to a debit:
  \[ E_4(A) := debit(A, V) \land balance(A) < 0 \]
  [note: not a clean way: includes a simple condition]

- A deposit followed by a debit in Bob’s account:
  \[ E_5 := E_1(bob, V_1); E_2(bob, V_2). \]

- There were no deposits to an account $A$ for 100 days:
  \[ E_6(A) := (\neg (\exists X : deposit(A, X))) \]
  \[ [deposit(A, Am) \land t = date; date = t + 100\text{days}] \]
Examples of Composite Events (Cont’d)

- The balance of account $A$ goes negative and there is another debit without any deposit in-between:
  $$E_7 := \mathcal{A}(E_4(A), E_2(A, V_1), E_1(A, V_2))$$

- After the end of the month send an account statement with all entries:
  $$E_8(A, list) := \mathcal{A}^*(first_{of\ month}, E_3(A), first_{of\ next_{month}})$$
Query Component

... obtain additional information:

- local, distributed, OWL-level

Result:

- the answer to the query XQuery, XPath, SQL
- bindings of free variables Datalog, F-Logic, XPathLog, SPARQL

Test Component

evaluate (locally) a test over the collected information
The Action Component

- invoked for a set of tuples of variable bindings
  - Atomic actions:
    - ontology-level local actions
    - data model level updates of the local state
    - explicit calls of remote procedures/services
    - explicit sending of messages
    - ontology-level *intensional* actions (e.g. in *business processes*)
  - Composite actions: e.g. a process algebra like CCS
  - Opaque code
Composite Actions: Process Algebras

e.g., CCS - Calculus of Communicating Systems [Milner‘80]
operational semantics defined by transition rules, e.g.
- a sequence of actions to be executed,
- a process that includes “receiving” actions,
- guarded (i.e., conditional) execution alternatives,
- the start of a fixpoint (i.e., iteration or even infinite processes), and
- a family of communicating, concurrent processes.

originally only over atomic processes/actions

reading and writing simulated by communication
a (send), ¯a (receive) “match” as communication

... extend this to the (Semantic) Web environment with autonomous nodes.
Composite Actions: Process Algebras

- e.g., CCS - Calculus of Communicating Systems [Milner‘80]
- composers; operational semantics defined by transition rules
- originally only over atomic processes/actions
- reading and writing simulated by communication $a$ (send), $\bar{a}$ (receive) “match” as communication
Composite Actions: Overview

- a sequence of actions to be executed (as in simple ECA rules),
- a process that includes “receiving” actions (which are actually events in the standard terminology of ECA rules),
- guarded (i.e., conditional) execution alternatives,
- the start of a fixpoint (i.e., iteration or even infinite processes), and
- a family of communicating, concurrent processes.
Action Component: Process Algebras

- example: CCS (Calculus of Communicating Systems, Milner 1980)
- describes the execution of processes as a transition system:
  (only the asynchronous transitions are listed)

\[
\begin{align*}
  a : P & \xrightarrow{a} P \\
  P_i & \xrightarrow{a} P \\
  \sum_{i \in I} P_i & \xrightarrow{a} P \\
  P & \xrightarrow{a} P' \\
  Q & \xrightarrow{a} Q' \\
  P|Q & \xrightarrow{a} P'|Q \\
  P|Q & \xrightarrow{a} P|Q' \\
  P_i\{\text{fix } \vec{X} \vec{P} / \vec{X}\} & \xrightarrow{a} P' \\
  \text{fix}_i \vec{X} \vec{P} & \xrightarrow{a} P'
\end{align*}
\]
Adaptation of Process Algebras

Goal: specification of reactions

- liberal asynchronous variant of CCS: go on when possible, waiting and delaying possible
- extend with variable bindings semantics
- input variables come bound to values/URIs
- additional variables can be bound by “communication”
- queries as atomic actions: to be executed, contribute to the variable bindings
- event subexpressions as atomic actions: like waiting for \( \bar{a} \) communication

\( \Rightarrow \) subexpressions in other kinds of component languages
Languages in the Action Component

- Process Engine
- Action Component Language, e.g. CCS
- Composer
  - * name
- Event Detector
- Query Engine
- Event Language
- Query/Condition Language
- Domain Broker
- Domain Nodes
- Domain Language
  - uri
- Atomic Events
- Literals
- Atomic Actions

Relationships:
- Process Algebra Responsibility
- Other Responsibilities

- Composer implements Action Component Language, e.g. CCS
- Action Component Language embeds 1..*
- Domain Language embeds *
- Event Language uses *
- Query/Condition Language uses *
- Event Detector uses *
- Query Engine uses *
- Domain Broker uses Literals
- Domain Nodes uses Atomic Events
- Atomic Actions uses Literals
Embedding Mechanisms: Same as in ECA-ML

- communication by logical variables
- namespaces for identifying languages of subexpressions
Example

Consider the following scenario:

- if a student fails twice in a written exam (composite event), it is required that another oral assessment takes place for deciding upon final passing or failure.

- Action component of the rule: Ask the responsible lecturer for a date and time. If a room is available, the student and the lecturer are notified. If not, ask for another date/time.

```
fix X.(ask_appointment($Lecturer,$Subj,$StudNo) :
   ∂ proposed_appointment($Lecturer,$Subj,$DateTime) :
   (available(room,$DateTime) +
    (∼ available(room,$DateTime) : X)))) :
inform($StudNo,$Subj,$DateTime) :
inform($Lecturer,$Subj,$DateTime)
```
<eca:Rule xmlns:uni="http://www.education.de">
<eca:Event> failed twice – binds $student ID and $course </eca:Event>
<eca:Query> binds e-mail addresses of the student and the lecturer </eca:Query>
  <ccs:Sequence>
    <ccs:Fixpoint variables="X" index="1" localvars="$date $time $room">
      <ccs:Sequence>
        <ccs:Atomic> send asking mail to lecturer </ccs:Atomic>
        <ccs:Event> answer binds $date and $time </ccs:Event>
        <ccs:Query> any room $room at $date $time available? </ccs:Query>
        <ccs:Alternative>
          <ccs:Test> yes </ccs:Test>
          <ccs:Sequence>
            <ccs:Test> no </ccs:Test>
            <ccs:ContinueFixpoint withVariable="X"/>
          </ccs:Sequence>
        </ccs:Alternative>
      </ccs:Sequence>
    </ccs:Fixpoint>
    <ccs:Atomic> send message ($date, $time, $room) to student </ccs:Atomic>
    <ccs:Atomic> send message ($date, $time, $room) to lecturer </ccs:Atomic>
  </ccs:Sequence>
</eca:Action>
</eca:Rule>
Comparison

- CCS (extended with events and queries) strictly more expressive than ECA rules alone:
  ECA pattern in CCS: \textit{event} : \textit{condition} : \textit{action},
- many ECA rules have much simpler actions and do not need CCS,
- useful to have CCS as an \textit{option} for the action part.
Part III: The Architecture
ECA Rules

- Event language: dynamic event
- Condition language: static query
- Action language: dynamic action
- Collect test act

- Each ECA Rule language uses:
  - A (composite) event language (mostly an event algebra)
  - A query language
  - A condition language
  - A language for specification of actions/transactions

- Different languages, different expressiveness/complexity
- Different locations where the evaluation takes place

⇒ Modular concepts with Web-wide services
Languages and Resources

Each language is a resource, identified by a (namespace) URI. Connected to the following resources:

ECA and Generic Sublanguages
- DTD/XML Schema/RDF description of the language
- processing engine (according to a communication interface)
- [semantics description by a formal method for reasoning about it]

Application Languages/Ontologies
- DTD/XML Schema/RDF description of the language
- Domain Broker Services (subscribe)
Service-Based Architecture

Language Processors as Web Services:

- ECA Rule Execution Engine employs other services for E/Q/T/A parts
- dedicated services for each of the event/action languages e.g., composite event detection, process algebras
- Auxiliary services: Atomic Event Matchers
- Domain Brokers
- Domain Services: raise events, serve as data sources, execute actions/updates
- query languages often implemented directly by the Web nodes (portals and data sources)
Architecture

1.1: register rule
eca: travel: match: snoop: ccs: smtp:

1.2: register event
travel: match: snoop:

1.3: atomic event patterns
match: travel:

1.4: register me
travel:

2.1a: atomic events
travel:

2.1b: atomic events
travel:

2.2: atomic events
travel:

3: detected parameters

4: detected parameters

5.1: action
ccs: travel: smtp:

5.2a: atomic actions
travel:

5.2b: atomic actions
smtp:

5.3a: booking
travel:

5.3b: message
(here: confirm)
by url

Domain Broker
tavel:

Lufthansa
tavel:

SNCF
tavel:

Event Detection
snoop:

Action Engine
ccs:

SMTP Mail Service
smtp:

ECA Engine
teca:

Client C:
Travel Agency
tavel:

Atomic Event Matcher
match:

Language Services Application Domain

Application Domain

Language Services
Tasks

- ECA Engine: Rule Semantics
  - Control flow: registering event component, receiving “firing” answer, continuing with queries etc.
  - Variable Bindings, Join Semantics
- Component Engines: dedicated to certain Event Algebras, Query Languages, Action Languages
- Generic Request Handler: Mediator towards Component Engines
  - depending on Service Descriptions
- Domain Services: atomic events, queries, atomic actions
- Domain Brokers: ECE composite event derivation rules, ACA action reduction rules, query and action brokering
ECA Architecture

ECA Engine:
<Rule>
<Event xmlns:ev="...">...</Event>
<Query xmlns:ql="...">...</Query>
<Test xmlns:tst="...">...</Test>
<Action xmlns:act="...">...</Action>
</Rule>

Generic Request Handler

Languages & Services Registry

Component Language Services

Domain Brokers

Domain Services

travel:
banking:
LH
SNCF

uni:
Communication

ECA engine sends component to be processed together with bindings of all relevant variables to GRH.

Generic Request Handler (GRH)

- Submits component (with relevant input/used variable bindings) to appropriate service (determined by namespace/language used in the component)
- if necessary: does some wrapping tasks (for non-framework-aware services)
- receives results and transforms them into flat variable bindings and sends them back to the ECA engine ...
- ... where they are joined with the existing tuples ...
- ... and the next component is processed.
MARS Metalevel & Infrastructure Ontology

The LSR is based on a metalevel infrastructure ontology:

- Ontology of language and service types
- Ontology of service types and tasks
- the LSR database: mars:Languages, mars:implemented-by, mars:Services, mars:TaskDescriptions
- give the URLs where certain services provide certain tasks for handling certain languages.
MARS Rule Semantics Ontologies

The Language Structure and Semantics

- Expressions
- Algebraic Expressions
- Use of Variables

The Languages

- ECA-ML
- SNOOP, CCS, ...

the XML markup is a stripped variant of a canonical RDF/XML-serialization of the OWL representation of rules and their component
Part IV: Domain Issues
General Architecture (Domain Aspects)

- ECA Rule Engine
- Sublanguage Services (Composite Event Detection, Complex Process Engines, Atomic Event Matchers)
- Domain Broker
- Domain Node
  - events
  - queries
  - actions
  - answers
- Domain Node
MARS: General Architecture (simplified)

ECA Rule Engine

Sublanguage Services (Composite Event Detection, Complex Process Engines)

Domain brokers forward actions and events, and process queries
- Derived Event Specifications: EC(raise-E)-Rules
- Composite Action Specifications: (on-A)CA-Rules

Domain nodes execute actions, raise events, and answer queries
- Composite Action Specifications: local (on-A)CA-Rules
Domain Broker

Initialize with an Ontology

- complete ontology in terms of mars:Class, mars:Property, mars:Event, mars:Action
- the ontology’s ECE and ACA rules (using the ECA-ML ontology+markup)
- domain broker registers ECE+ACA rules at the ECA Engine

Domain Nodes

- Each domain node registers at the domain broker which notions (classes, properties, actions) it mars:supports,
- runtime behavior: next slide ...
Domain Broker: Initialization

- complete ontology in terms of mars:Class, mars:Property, mars:Event, mars:Action
- the ontology’s ECE and ACA rules (using the ECA-ML ontology+markup)
  - Derived Event Specifications (ECE): register as EC(raise-E)-Rules at the ECA Engine
  - Composite Action Specifications: register as (on-A)CA-Rules at the ECA Engine
- “outsourcing” of these tasks
- allows ontology designer to use any E/C/A languages!
Architecture of the Domain Node

register for classes, properties, actions

- Domain Broker
- ACA Mapper matches actions against mappings
- ACA Mappings Repository
- Jena-based core module with Active Functionality
- PostgreSQL Database: RDF facts
- DL Reasoner (e.g., Pellet)
  - RDF graph
  - model answers
  - facts queries
  - updates
- actions
- queries
- answers
- event occurrences
Sample Local ACA Rule of the Domain Node

- in: an action in XML
- or RDF (graph fragment containing one
  \{ ?A rdf:type mars:Action \}
- implement the action on the local RDF database

```xml
## sample rule using XQuery-style
IMPLEMENT <travel:schedule-flight/> BY
let $flight := /travel:schedule-flight/@flight
let $captain := /travel:schedule-flight/@captain
return concat("INSERT ($flight has-captain $captain);",
for $name in /travel:schedule-flight/cabincrew/@name
let $cabincrew := local:make-person-uri($name)
return "INSERT ($flight has-cabincrew $cabincrew);")
```

MARS
Summary

- describe events and actions of an application within its RDF/OWL ontology
- rules on different levels of abstraction/locality
- architecture: functionality provided by specialized nodes
- outsourcing ECE+ACA rules as much as possible to existing ECA infrastructure.
Part V: Syntax Details and Implementation
Communication of Variable Bindings

XML markup for communication of variable bindings:

```xml
<logvars:variable-bindings>
  <logvars:tuple>
    <logvars:variable name="name" ref="URL"/>
    <logvars:variable name="name"> any value </logvars:variable>
  </logvars:tuple>
  ...
  ...
</logvars:variable-bindings>
```
Communication ECA → GRH

- the component to be processed
- bindings of all relevant variables

[Sample: a query component]

```xml
<eca:Query xmlns:ql="url"
  rule="rule-id" component="component-id">
  <!-- query component -->
  <eca:Query>
    <logvars:variable-bindings>
      <logvars:tuple> ... </logvars:tuple>
      <logvars:tuple> ... </logvars:tuple>
    </logvars:variable-bindings>
  </eca:Query>
</eca:Query>
```

- `url` is the namespace used by the component language
- identifies appropriate service
Communication of Variable Bindings

Sample XML markup for communication of a query and variable bindings:

```xml
<eca:Query xmlns:ql="url"
rule="rule-id" component="component-id">
<!-- query component -->
<eca:Query>
<logvars:variable-bindings>
  <logvars:tuple>
    <logvars:variable name="name" ref="URI"/>
    <logvars:variable name="name" any value</logvars:variable>:
  </logvars:tuple>
  <logvars:tuple> . . </logvars:tuple>
  <logvars:tuple> . . </logvars:tuple>
</logvars:variable-bindings>
</eca:Query>
```
result-bindings-pairs (semantics of expression)

```xml
<logvars:answers rule=""rule-id"" component=""component-id"">  
  <logvars:answer>  
    <logvars:result>  
      <!-- functional result -->  
    </logvars:result>  
    <logvars:variable-bindings>  
      <logvars:tuple> ... </logvars:tuple>  
      :  
      <logvars:tuple> ... </logvars:tuple>  
    </logvars:variable-bindings>  
  </logvars:answer>  
  <logvars:answer> ... </logvars:answer>  
  :  
  <logvars:answer> ... </logvars:answer>  
</logvars:answers>
```
Communication GRH → ECA

- set of tuples of variable bindings (i.e., input/used variables and output/result variables)
- is then joined with tuples in ECA engine
- ... and next component is processed
Special Issue: Functional Results

Example: Event Component

```xml
<eca:Query bind-to-variable="name" xmlns:ql="uri">
  event specification
</eca:Query>
```

- GRH submits *event specification* to processor associated with *uri*
- GRH receives `answer(result,variable-bindings*)` elements from event detection engine
- binds `<result>` to *name* and extends `<variable-bindings>`
Example: wrapped, framework-aware XQuery engine

```xml
<eca:Query>
  <eca:Opaque language="uri or shortname">
    <eca:has-input-variable name="varname" use="$localname"/>
    code fragment in language language
  </eca:Opaque>
</eca:Query>
```

- GRH submits *event specification* to processor associated with *lang*
- GRH receives `answer(result,variable-bindings*)` elements from event detection engine
- and returns them to ECA engine
Further Issues

Normal Form vs. Shortcut

- note that parts of the condition can often already checked earlier during event detection
- most event formalisms allow for small conditions already in the event part (e.g., state-dependent predicates and functions; cf. Transaction Logic)
Summary

- First: diversity looked like a problem, lead to the Web (XML) and the Semantic Web (RDF and OWL data);
- Heterogeneous data models and schemata:
  \[ \Rightarrow \text{RDF/OWL as integrating semantic model in the Semantic Web} \]
- Extend these concepts to describe behavior
- Describe events and actions of an application domain within its RDF/OWL model
- Diversity + unified Semantic-Web-based framework has many advantages
- Languages of different expressiveness/complexity available
- Markup+ontologies make expressions accessible for reasoning about them
Summary

- architecture: functionality provided by specialized nodes
- Local: triggers (SQL, XML, RDF/Jena, ...)
  - local updates
  - raise higher-level events
- Global: ECA rules
  - components
  - application-level atomic events and atomic actions
  - specific languages (event algebras, process algebras)
    - opaque (= non-markup, program code) allowed
- Communication: events, event broker services, registration
- Identification of services via namespaces
Further Information

REWERSE Deliverable I5-D4: “Models and Languages for Evolution and Reactivity”

REWERSE Deliverable I5-D5: “A First Prototype on Evolution and Behavior at the XML Level”


Prototypes:

- MARS Prototype: http://www.semwebtech.org
- Jena+Triggers (GOE/CLZ Diploma)
- Cooperation within REWERSE I5 with r³ (U Nova de Lisboa, Portugal), RuleCore (U Skövde/Sweden) and XChange (LMU München/Germany)