A General Language for Evolution and Reactivity in the Semantic Web

José Júlio Alferes **Ricardo Amador**

Wolfgang May

CENTRIA, Universidade Nova de Lisboa, Portugal
Institut für Informatik, Universität Göttingen, Germany

PPSWR 2005, Dagstuhl, Sept. 12-16, 2005

Motivation and Goals

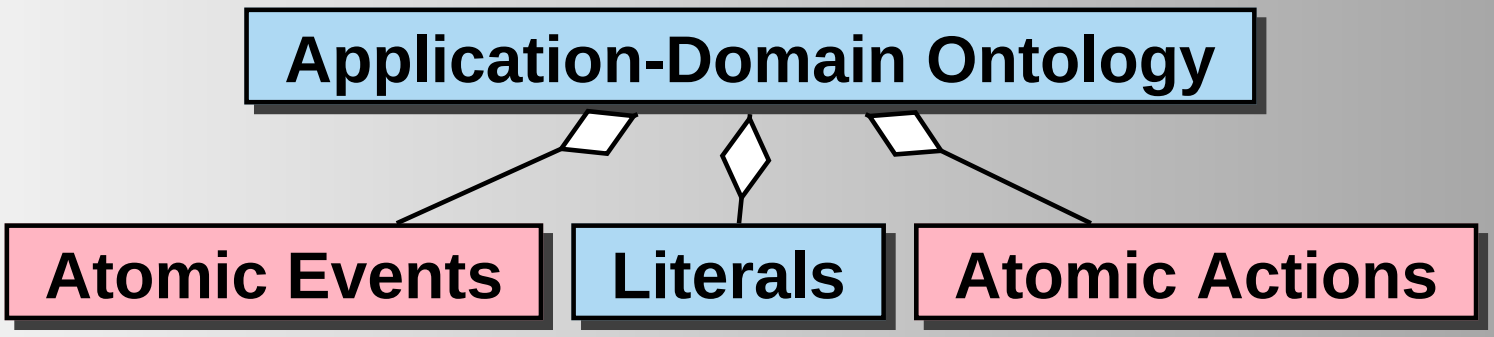
- Description of behavior *in* the Semantic Web
- semantic description *of* behavior

Scope

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

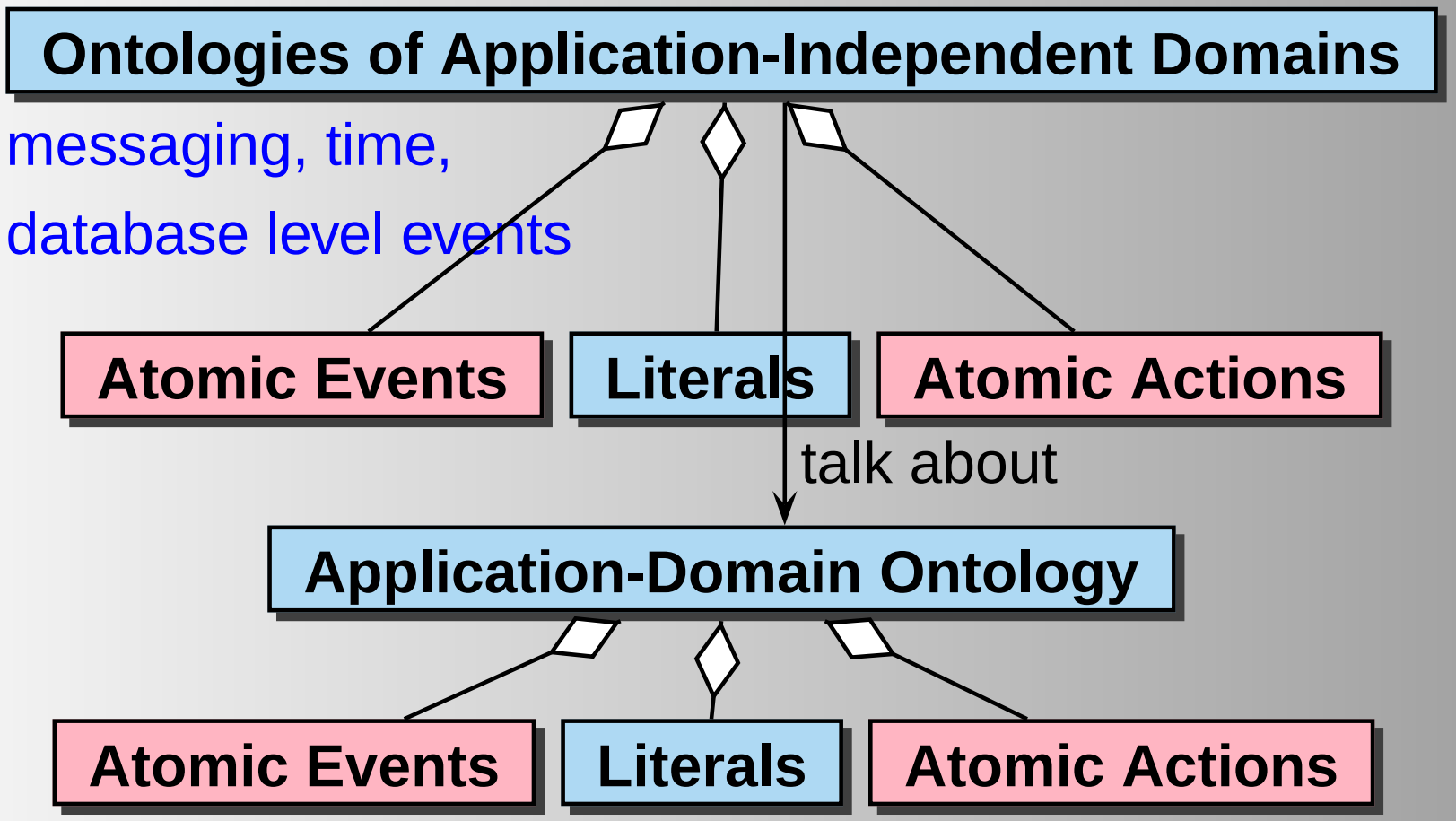
⇒ use Event-Condition-Action Rules as a well-known paradigm.
⇒ ontologies must also describe actions and events.

Ontologies including Dynamic Aspects



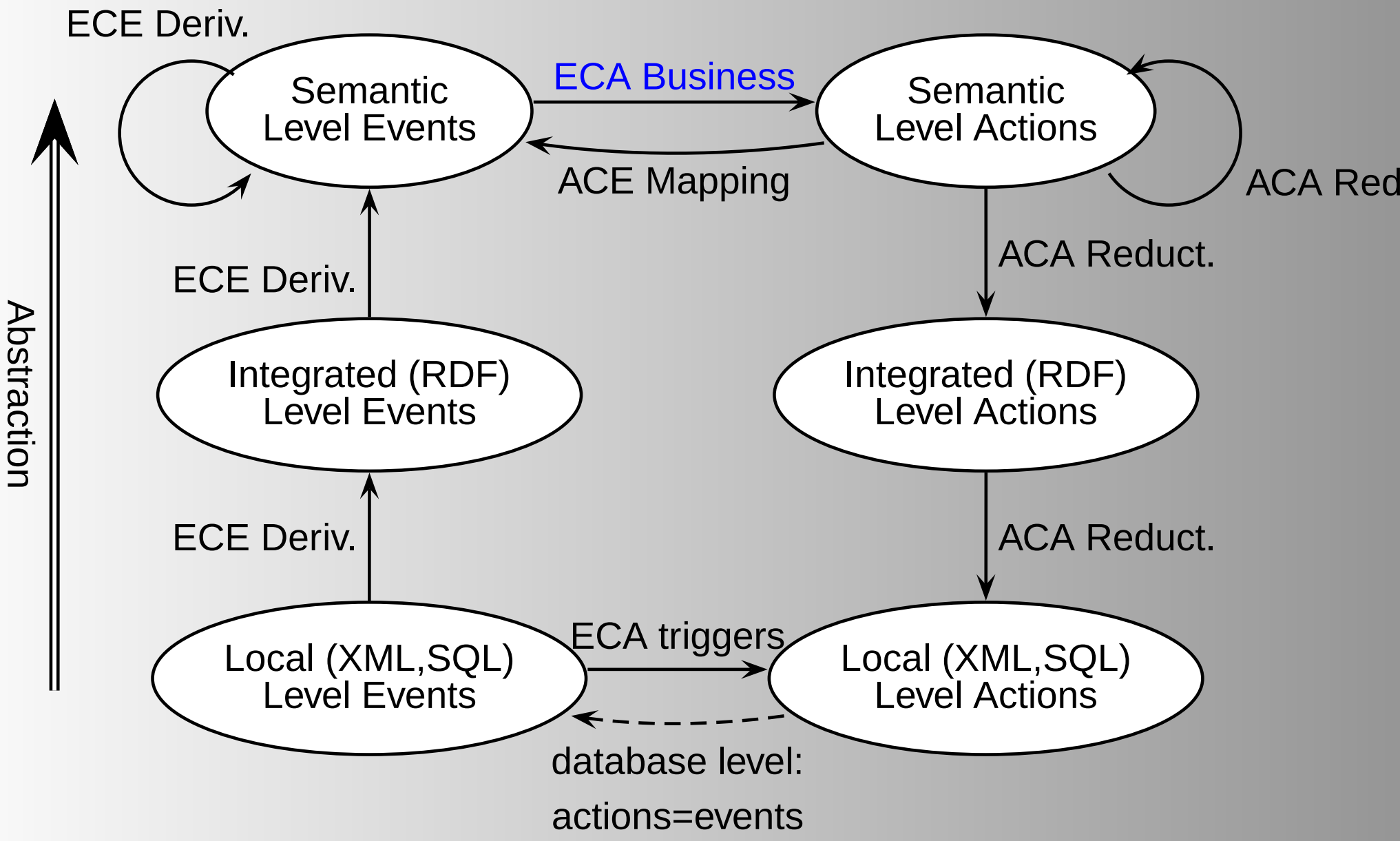
- correlate actions, state, and events

Ontologies including Dynamic Aspects



- correlate actions, state, and events

Abstraction Levels and Types of Rules



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*

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

Sample Rule on the RDF Level

- reacts on an event on the RDF view level
- here: maps it to an event on the OWL level
- again an *ECE derivation rule*

```
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))
```

... which is then an event of the domain ontology.

Analysis of Rule Components

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

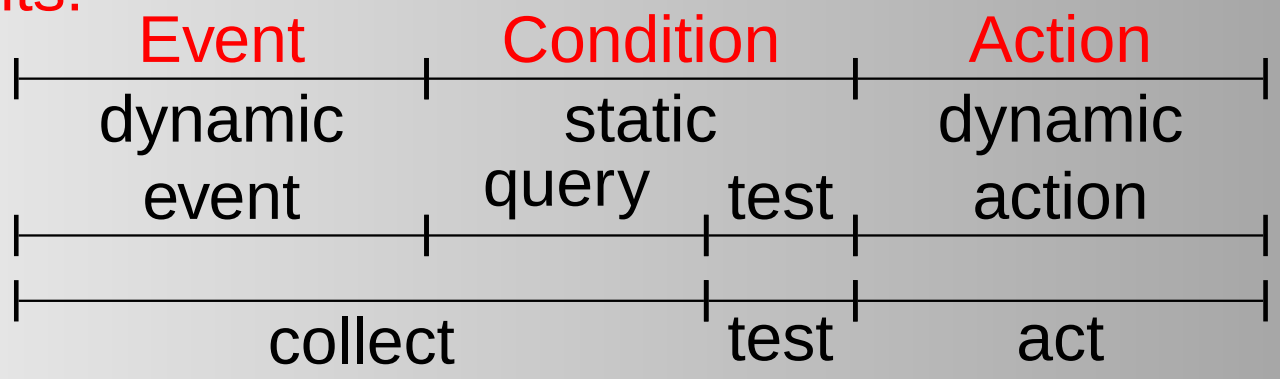
- Event: specifies a rough restriction on what *dynamic* situation probably something has to be done.
  Collects some parameters of the events.
- Condition: specifies a more detailed condition, including *static* data if actually something has to be done.
  $\Rightarrow$ evaluate a ((Semantic) Web) query.
- 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 ..."

Clean, Declarative “Normal Form”

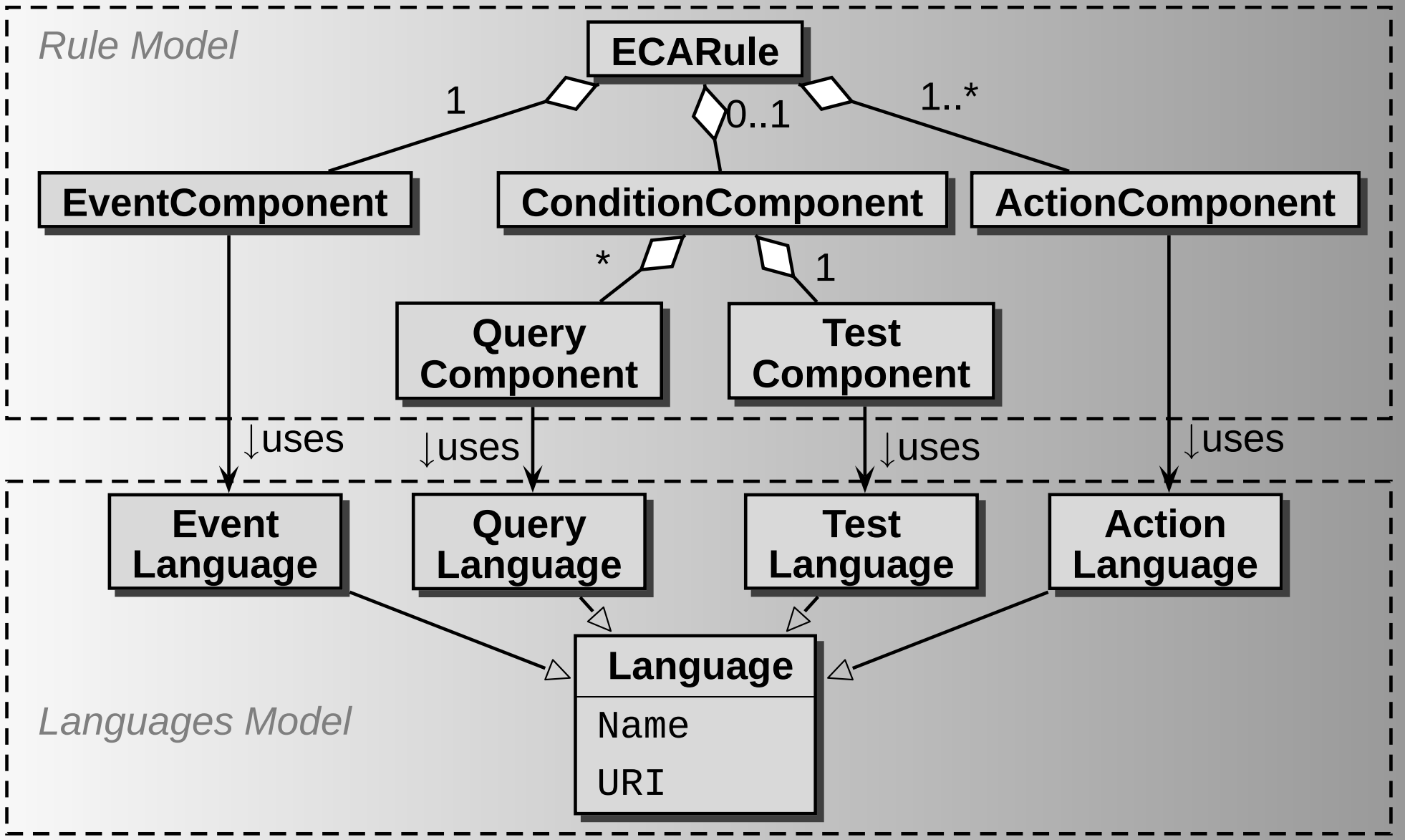
Rule Components:



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

- Component sublanguages: heterogeneous
- Communication between components: logical variables

Modular ECA Concept: Rule Ontology



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**>

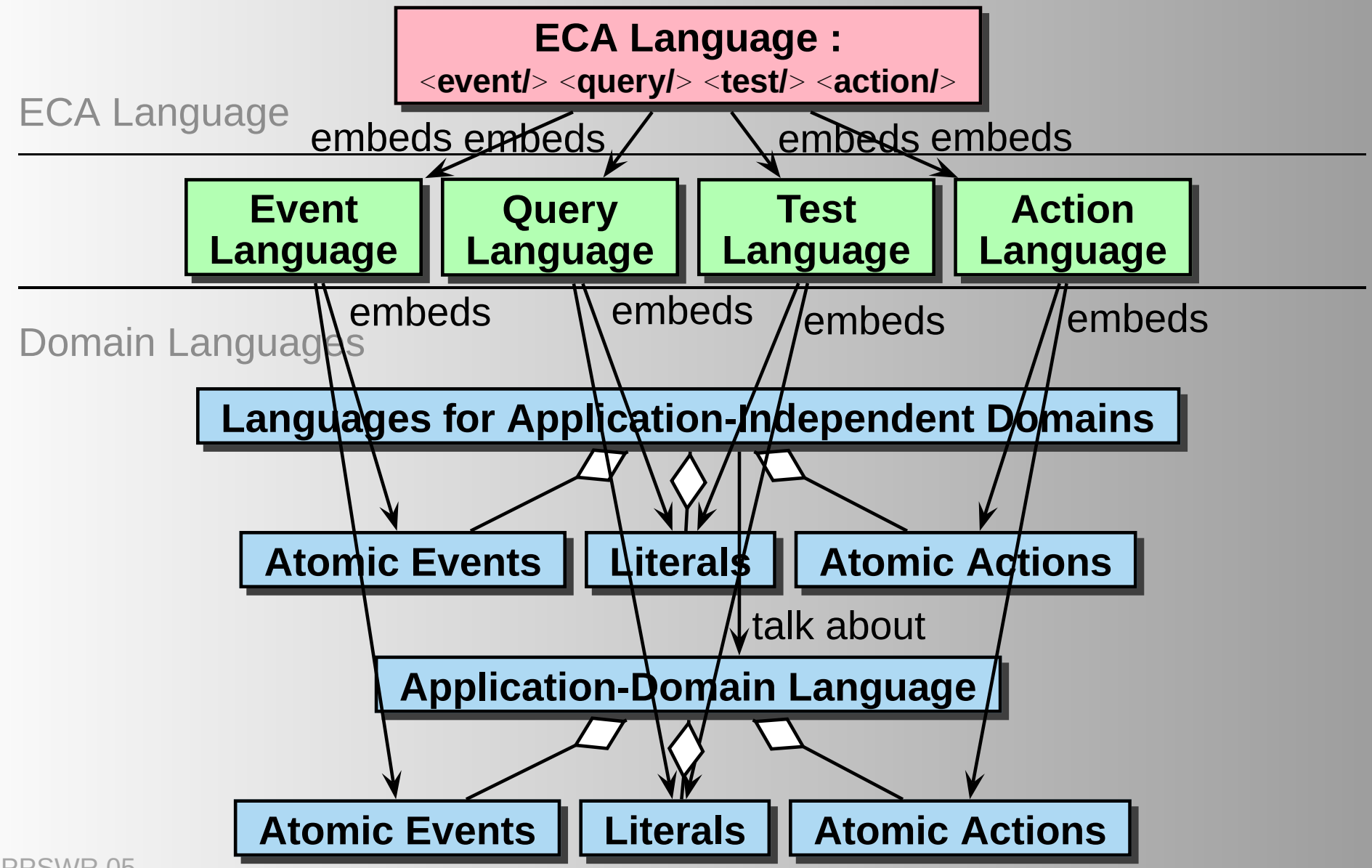
<**eca:action** *identification of the language* > <!-- there may be several actions -->

action specification, using variables, probably binding local ones

<**/eca:action**>

/eca:rule>

Embedding of Languages



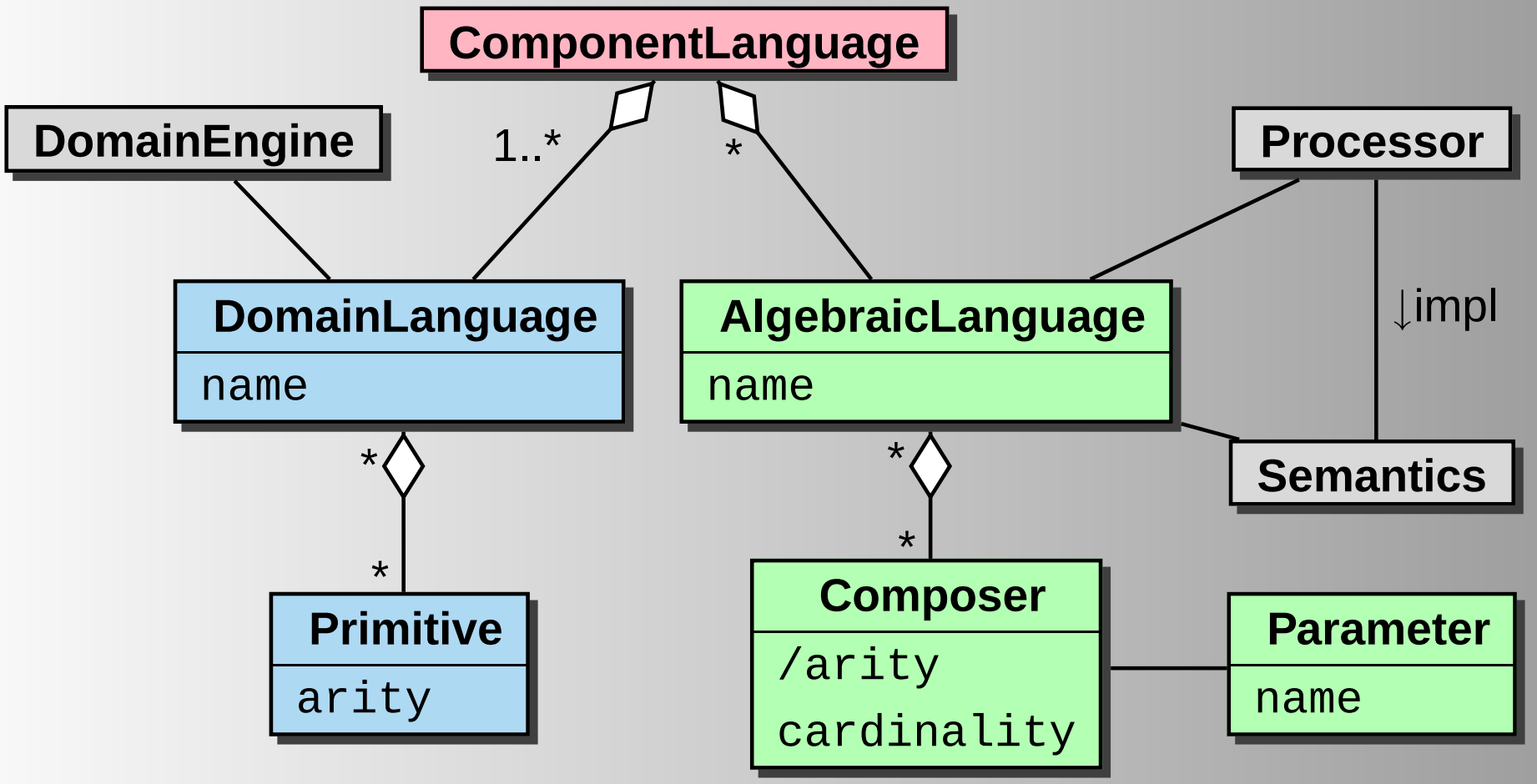
Sublanguages: Algebraic Languages

- Domains specify atomic events, actions and literals

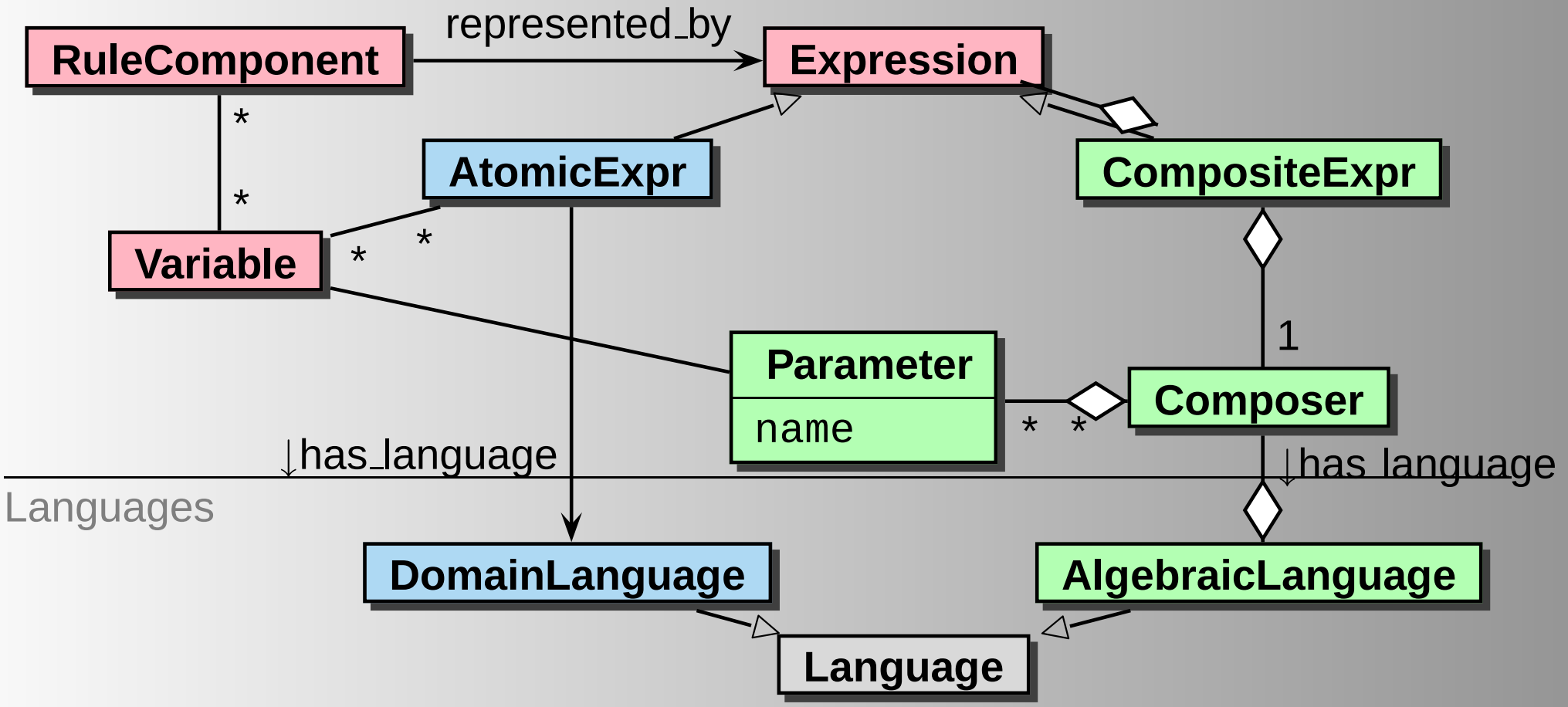
Algebraic Languages

- 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).
- algebraic query languages (e.g. SQL, XQuery)
- tests: boolean algebra
- Process algebras (e.g. CCS)

Algebraic Sublanguages



Syntactical Structure of Expressions



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

Event Component: Event Algebras

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

$$
\begin{aligned}
(E_1 \nabla E_2)(x,t) &:\Leftrightarrow E_1(x,t) \vee E_2(x,t)\ , \\
(E_1;E_2)(x,y,t) &:\Leftrightarrow \exists t_1 \leq t : E_1(x,t_1) \wedge E_2(y,t) \\
\neg(E_2)[E_1,E_3](t) &:\Leftrightarrow \text{if } E_1 \text{ and then a first } E_3 \text{ occurs,} \\
&\quad\ \text{without occurring } E_2 \text{ in between.}
\end{aligned}
$$

- “join” variables between atomic events
- “safety” conditions similar to Logic Programming rules
- Result:
  - the sequence that matched the event
  - optional: additional variable bindings

Query Component

... obtain additional information:

- local, distributed, OWL-level
- Result:
  - the answer to the query
  - optional: additional variable bindings

Condition and Action Component

- Condition: check a boolean condition (including predicates of the application domain),
- Action: do something by using the variable bindings.

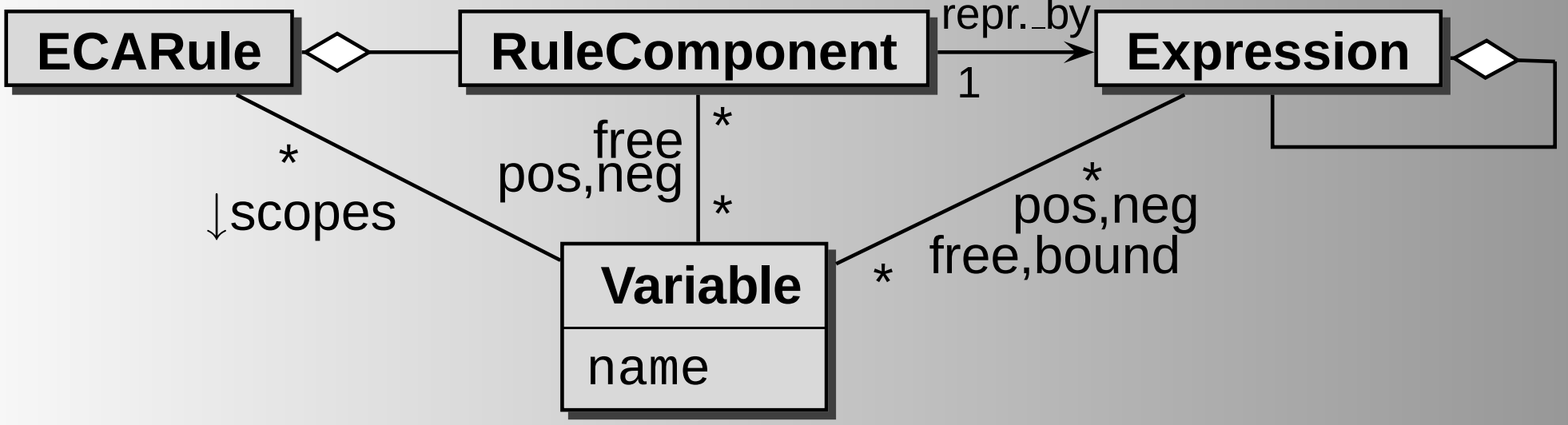
Subconcepts and Sublanguages

- 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,
- (sub)terms must have a well-defined result.

Rule Semantics

- Deductive rules: variable bindings Body$\rightarrow$Head
- communication/propagation of information by *logical variables*:
  $E \overset{+}{\rightarrow} Q \rightarrow T$ & A
- safety as usual ...



Binding Variables in the Collect Part

concerns: (Composite) Events and Queries

- to values, XML fragments, RDF fragments, and (composite) events
- Logic Programming (Datalog, F-Logic): binding variables by patterns.
  proposal: markup of E and Q languages uses XSLT-style
  <variable name="*var-name*"> and $*var-name*
- functional style (SQL, OQL, XQuery): expressions return a value/fragment.
  ⇒ must be bound to a variable to be kept and reused
  proposal:
  <variable name="*var-name*">*event-spec*</variable>

Sample Markup (Event Component)

```
<eca:rule xmlns:uni="...">
  <eca:variable name="theSeq">
    <eca:event xmlns:snoop="..."
    <snoop:sequence>
       <eca:atomic-event>
         <uni:reg_open subject="$Subj"/>
       </eca:atomic-event>
       <eca:atomic-event>
         <uni:register subject="$Subj" name="$Name"/>
       </eca:atomic-event>
     </snoop:sequence>
    </eca:event>
  </eca:variable>
    :
</eca:rule>
```

binds variables:

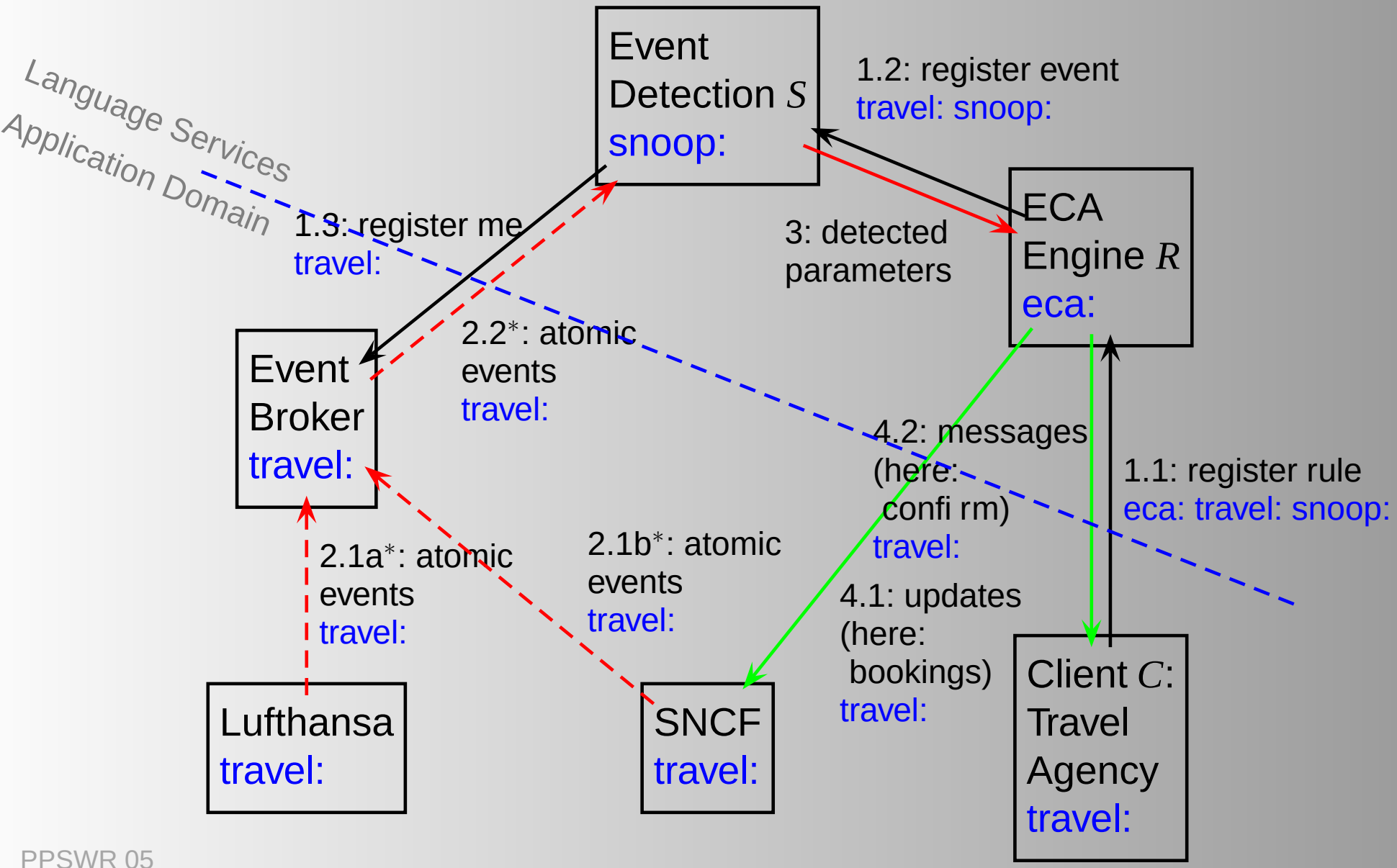
- Subj, Name: by matching
- theSeq returns the sequence of events that matched the pattern

Engines – Service-Based Architecture

Language Processors as Web Services:

- ECA Rule Execution Engine employs other services for E/Q/T/A parts:
  nodes register their rules at the engines; processing is done by the engine
- dedicated services for each of the event/action languages
  e.g., composite event detection engines
- dedicated services for domain-specific issues:
  raising and communicating events, predicates, executing actions/updates
- query languages often implemented directly by the Web nodes (portals and data sources)

Architecture



Summary

- unified & flexible Semantic-Web-based framework for specifying behavior
- languages of different expressiveness/complexity
- describe events and actions of an application within its RDF/OWL model
- architecture: functionality provided by specialized nodes

Publications & Details

- REWERSE I5-D4: “Models and Languages for Evolution and Reactivity”: Everything + examples
- PPSWR05: preliminary workshop paper
- ODBASE05: ontology of rules, rule components and languages, and the service-oriented architecture
- RuleML05: languages and their markup, communication and rule execution model