Semantic Business Process Management

Lecture 8 – Semantic Technologies IV
Rules: RuleML, Prova

Prof. Dr. Adrian Paschke

Corporate Semantic Web (AG-CSW)
Institute for Computer Science, Freie Universitaet Berlin
paschke@inf.fu-berlin.de
http://www.inf.fu-berlin.de/groups/ag-csw/
Overview

- Overview Semantic Technologies
- Ontologies
  - OMG Ontology Definition Metamodel
  - W3C Web Ontology Language
- Rules
  - OMG SBVR
  - OMG PRR
  - W3C RIF
  - RuleML
  - Prova
Semantic Computing Technologies

4. Software Agents and Web-based Services
   - Rule Responder, FIPA, Semantic Web Services, …

3. Rules and Event/Action Logic & Inference
   - RIF, SBVR, PRR, RuleML, Logic Programming
     Rule/Inference Engines,…

2. Ontologien
   - RDFS, OWL Lite|DL|Full, OWL 2, ODM, …

1. Explicit Meta-data and Terminologies
   - vCard, PICS, Dublin Core, RDF, RDFa, Micro Formats,
     FOAF, SIOC …
RuleML

- Rule Markup and Modeling Initiative (RuleML) ([www.ruleml.org](http://www.ruleml.org))
  - representatives from academia, industry and government
  - promotion of the modern and future generations of Web rule technology
- RuleML is currently the **de facto open language standard** for Web Rules
  - W3C Rule Interchange Format in preparation
- Collaborating with W3C ([RIF](http://www.w3.org/RIF/)), OMG (PRR, SBVR), OASIS, DARPA and other standards/gov't bodies
RuleML Enables ... 

Rule modelling markup translation interchange execution publication archiving 
in UML RDF XML ASCII
RuleML Language Family

Layered Approach

- Derivation Rules
- Reaction Rules
- Integrity Constraints
- Transformation Rules
Schema Modularization

- RuleML is specified by a set of modular XSDs
- XML Schema + EBNF Syntax
- Full RDF compatibility via type and role tags (akin to triple syntax);
- XML Schema Modularization: Layered and uniform design
"The **discount** for a **customer** buying a **product** is **5.0 percent** if the **customer** is **premium** and the **product** is **regular**."
Striped Syntax vs. Stripe-skipped Syntax

<Implies>
  <body>
    <Atom>
      <Rel>spending</Rel>
      <Var>customer</Var>
      <Ind>min 5000 euro</Ind>
      <Ind>previous year</Ind>
    </Atom>
  </body>
  <head>
    <Atom>
      <Rel>premium</Rel>
      <Var>customer</Var>
    </Atom>
  </head>
</Implies>

<Implies>
  <Atom>
    <Rel>spending</Rel>
    <Var>customer</Var>
    <Ind>min 5000 euro</Ind>
    <Ind>previous year</Ind>
  </Atom>
  <Atom>
    <Rel>premium</Rel>
    <Var>customer</Var>
  </Atom>
</Implies>
RuleML 0.91 - Constants

- Logical constant terms are represented with `<Ind>` tags in RuleML.

<table>
<thead>
<tr>
<th>Prova</th>
<th>RuleML</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td><code>&lt;Ind&gt;abc&lt;/Ind&gt;</code></td>
</tr>
<tr>
<td>“Adrian Paschke”</td>
<td><code>&lt;Ind&gt;Adrian Paschke&lt;/Ind&gt;</code></td>
</tr>
<tr>
<td>42</td>
<td><code>&lt;Ind&gt;42&lt;/Ind&gt;</code></td>
</tr>
</tbody>
</table>
RuleML 0.91 - Variables

- Logical variable terms are represented with `<Var>` tags in RuleML.

Prova  

X  

RuleML  

<Var>X</Var>

<Var/></Var>
RuleML 0.91 - Functions

- Function expressions / complex terms are represented using the `<Expr>` tag
- The function name is represented by embedded `<Fun>` tag
- Argument can be `<Ind>`, `<Var>`, `<Expr>` or `<Plex>`

Prova: $f(X)$

RuleML: `<Expr>
  <Fun>f</Fun>
  <Var>X</Var>
</Expr>`
RuleML 0.91 - Lists

- Lists are represented using the `<Plex>` tag

Prova

[p,1,2]  
p(1,2)

RuleML

```xml
<Plex>
  <Rel>p</Rel>
  <Ind>1</Ind>
  <Ind>2</Ind>
</Plex>
```
RuleML 0.91 - Lists

- The RuleML `<repo>` (rest, positional) tag for the Prolog/Prova “|” operator

Prova

[ Head | Rest ]

RuleML

```
<Plex>
  <Var>Head</Var>
  <repo>
    <Var>Rest</Var>
  </repo>
</Plex>
```
An equational formula consisting of two expressions is represented by an `<Equal>` tag.

Prova  
X=1

RuleML  
`<Equal>`  
`<Var>X</Var>`  
`<Ind>1</Ind>`  
`</Expr>`
RuleML-0.91 Types

- Types can be assigned to terms using the type attribute.
- Types are valid for `<Ind>`, `<Var>` and `<Expr>` terms.

```xml
<Var type="Vehicle">Car</Var>
<Ind type="Sedan ">2000 Toyota Corolla</Ind>
```
RuleML 0.91 – Atomic Formula

- Atomic formulas are represented using the `<Atom>` tag
- Relation name is represented as an embedded `<Rel>` tag
- Arguments are embedded in `<Atom>` - these can be `<Ind>`, `<Var>`, `<Expr>`, and `<Plex>`

Prova

spending(“Peter Miller”, “min 500 euro”, “previous year”).

RuleML

<Atom>
  <Rel>spending</Rel>
  <Ind>Peter Miller</Ind>
  <Ind>min 500 euro</Ind>
  <Ind>previous year</Ind>
</Atom>
RuleML 0.91 – Implicational Derivation Rules

- (Derivation) Rules are represented with the `<Implies>` tag
- First child element is the body of the rule, which can be either a single `<Atom>` or a conjunction of `<Atom>`s in an `<And>` tag.
- Second child element is the head of the rule, which must be an atomic formula (Atom).
"The **discount** for a *customer* buying a *product* is **5.0 percent** if the *customer* is **premium** and the *product* is **regular**."
A RuleML knowledge base is a conjunction of clauses that are asserted to be true

```xml
<Assert>
  <And innerclose="universal">
    <!-- Clauses (Facts and Rules) here -->
  </And>
</Assert>
```
RuleML 0.91 – Queries

- Queries are represented with `<Query>` tag
- Contains one child to represent the body of the query - this is an `<Atom>` or a conjunction of `<Atom>`s in an `<And>`
- Example: `premium(Who)?`

```xml
<Query>
  <Atom>
    <Rel>premium</Rel>
    <Var>who</Var>
  </Atom>
</Query>
```
Slotted Un-positional Object Oriented Representation

- **Position independent** user-defined role -> filler pairs:

```xml
<Implies>
  <Atom>
    <Rel>spending</Rel>
    <slot><Ind>spender</Ind><Var>customer</Var></slot>
    <slot><Ind>amount</Ind><Ind>min 5000 euro</Ind></slot>
    <slot><Ind>period</Ind><Ind>previous year</Ind></slot>
  </Atom>
  <Atom>
    <Rel>premium</Rel>
    <slot><Ind>client</Ind><Var>customer</Var></slot>
  </Atom>
</Implies>
```
Reaction RuleML
Scope of Reaction RuleML

- **Active Databases**
  - Transient Events
  - ECA Paradigm
  - Global Active Rules
  - Trigger (EA Rules)
  - Complex Event Algebra

- **Production Rule Systems**
  - Implicit Sequence of Knowledge Updates
  - CA Rules

- **Rule-Based Event Notification Systems / Distributed Complex Event Processing**
  - Event / Action Messages
    - Inbound / Outbound
    - Enterprise Service Bus
  - (Agent) Conversation
    - Protocols
    - Performatives (e.g. FIPA ACL)

- **KR Event / Action / Transition / Process Logic Systems**
  - Event / Action Axioms
  - Reasoning on Effects / Transitions
    - fluents / states / processes
    - akin to e.g. state machines, petri-nets or pi-calculus
Reaction Rules on the Web (1)

1. (Temporal) KR event/action logics
   - Members e.g. Event, Situation, Fluent Calculus, TAL
   - Actions with effects on changeable properties / states / fluents, i.e. actions ~ events
   - Focus: reasoning on effects of events/actions on knowledge states and properties

2. KR evolutionary transaction, update, state transition logics
   - Members e.g. transaction logics, dynamic LPs, LP update logics, transition logics,
   - Knowledge self-updates of extensional KB (facts / data) and intensional KB (rules)
   - Transactional updates possibly safeguarded by post-conditional integrity constraints / tests
   - Complex actions (sequences of actions)
   - Focus: declarative semantics for internal transactional knowledge self-update sequences (dynamic programs)

3. Condition-Action / Production rules
   - Members, e.g. OPS5, Clips, Jess, JBoss Rules/Drools, Fair Isaac Blaze Advisor, ILog Rules,
     CA Aion, Haley, ESI Logist, Reaction RuleML
   - Mostly forward-directed non-deterministic operational semantics for Condition-Action rules
   - Focus: 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
4. Active Database ECA rules
- Members, e.g. ACCOOD, Chimera, ADL, COMPOSE, NAOS, HiPac, Reaction RuleML, Prova, XChange
- ECA paradigm: “on Event when Condition do Action”; mostly operational semantics
- Instantaneous, transient events and actions detected according to their detection time
- Focus: Complex events: event algebra (e.g. Snoop, SAMOS, COMPOSE) and active rules (sequences of self-triggering ECA rules)

5. Process Calculi, Event Messaging and distributed rule-based Complex Event Processing
- Members, e.g. process calculi (CSP, CSS, pi-calculus, join-calculus), event/action messaging reaction rules (inbound / outbound messages), rule-based intelligent CEP with rule-based Event Processing Languages (EPLs, e.g. Prova, Reaction RuleML, AMIT, Rule Core)
- Focus: process calculi focus on the actions, event messaging and CEP on the detection of complex events; often follow some workflow pattern, protocol (negotiation and coordination protocols) or CEP pattern
Reaction RuleML

- Reaction RuleML
- Quasi-Standard for Reactive Web Rules
  - Production rules, ECA rules and variants such as Trigger (EA), intelligent rule-based CEP, KR Event/Action Logics, Process Algebras, …
- Application Domains:
  - 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
  - Web-based Workflow Systems
  - …
General Concepts

- General (reaction) rule form that can be specialized as needed

- Three general execution styles:
  - **Active**: 'actively' polls/detects occurred events in global ECA style, e.g. by a ping on a service/system or a query on an internal or external event database
  - **Messaging**: Waits for incoming complex 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 Syntax for Reaction Rules

```xml
<Rule style="active" eval="strong">
  <on>
    <!-- event -->
  </on>

  <if>
    <!-- condition -->
  </if>

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

  <ifPost>
    <!-- postcondition -->
  </ifPost>

  <doAlternative>
    <!-- alternative/else action -->
  </doAlternative>
</Rule>
```
Reaction RuleML – Rule Type Examples

- Derivation Rule: 
  `<Rule style="reasoning"> 
      <if>...</if> 
      <then>...</then> 
  </Rule>`

- Production Rule: 
  `<Rule style="active"> 
      <if>...</if> 
      <do>...</do> 
  </Rule>`

- ECA Rule: 
  `<Rule style="active"> 
      <on>...</on> 
      <if>...</if> 
      <do>...</do> 
  </Rule>`
Complex Event / Action Algebra Operators

- **Action Algebra:**
  - *Succession* (Ordered Succession of Actions), *Choice* (Non-Deterministic Choice), *Flow* (Parallel Flow), *Loop* (Loops)

- **Event Algebra:**
  - *Sequence* (Ordered), *Disjunction* (Or), *Xor* (Mutual Exclusive), *Conjunction* (And), *Concurrent*, *Not*, *Any*, *Aperiodic*, *Periodic*

```xml
<event>
  <Sequence>
    <Concurrent>
      <Ind>a</Ind>  <Ind>b</Ind>
    </Concurrent>
    <Ind>c</Ind>
  </Sequence>
</event>
```
Example: Request / Query

...<Message mode="outbound" directive="ACL:query-ref">
  <oid> <Ind>RuleML-2008</Ind> </oid>
  <protocol> <Ind>esb</Ind> </protocol>
  <sender> <Ind>User</Ind> </sender>
  <content>
    <Atom>
      <Rel>getContact</Rel>
      <Ind>Sponsoring</Ind>
      <Var>Contact</Var>
    </Atom>
  </content>
</Message>
...

- Event Message is local to the conversation state (oid) and pragmatic context (directive)
Prova
Prova: Declarative Rules for Java

- Combine the benefits of declarative and object-oriented programming and workflow languages;
- Interpreted scripting syntax combines those of ISO Prolog and Java + Ontologies (as type system);
- Access data sources via wrappers written in Java, query language built-ins (e.g. SQL, SPARQL, RDF Triples, XQuery) or message-driven (Web) service interfaces;
- Make all Java API (EJBs) from available packages directly accessible from rules;
- Be compatible with modern enterprise service and agent-based software architectures.
Syntax design features of Prova

- The syntax combines ISO Prolog and Java but the key is simplicity:

```
% Prolog
N2 is N + 1,
----------------------------------------------------------------
// Java
List l = new java.util.ArrayList();
```

- Low-cost creation of distributed integration and computation workflows.
- Prova separates logic, data, and computation.
- Low-cost integration of rule-base scripted agents inside Java and Web applications.
- For more information check the User`s Guide in the prova web page.
Using colours for language elements

The colours below are used to distinguish the language elements:

% Comments are in green
% Built-in predicates are brown
tokenize_list(Line, "\t", [T|Ts])

% User-defined predicates are blue
member(X, [X|Xs])

% Java calls and constructors are red
Text1=P1.toString()

% Table names are pink
sql_select(DB, cla, px(PXA), pdb_id(PDB))

% Message performatives (speech acts) are navy blue
rcvMsg(Protocol, From, query_ref, [X|Xs]) :-
Prova Syntax

- Variables (upper case), Constants (lower case)
- Fact: `availability(s1,99%)`.
- Rule: `qos(S,high):- availability(S,99%)`.
- Query 1: `:- solve (not(qos(S,high)))`.
- Query 2: `:- eval (not(qos(S,high)))`.
- Derive: `derive([X|Args])`, ...
- Scoping: `@label("www.prova.ws") p(X), ...`
- Memoization: `cache(p(X)), ...`
- Lists: `[Head|Tail] = [Head,Arg2,...,ArgN] = Head(Arg2,...,ArgN)`
- Module Imports:
  `:- eval(consult("ContractLog/list.prova"))`.
  `:- eval(consult("http://rule.org/list.prova"))`.
- Meta data annotation:
  `@label(r1), @src("www.prova.ws"), @dc_author("AP")`
  `qos(S,medium) :- availability(S,98%)`. 
Example: General language constructs

Prova extends ISO Prolog syntax:

% Facts
i_am("mediator").
portfolio("balanced",P).
reachable(X,X).
is_a("anticoagulant","molecular_function").

% Clauses (head true if conditions true in the body)
parent(X,Y) :-
    is_a(Y,X).
parent(X,Y) :-
    has_a(X,Y).

% Goals (note there is no head)
:- solve(parent(Parent,"anticoagulant")). % Print solutions
:- eval(parent(Parent,"anticoagulant")). % Just run exhaustive search

Format of the output for the goal solve above:
    Parent="molecular_function"
Pervasive use of the Java type system

- Java typed and untyped variables;
- Natural rules for subclasses unification;
- Java variables prefixed by full package prefix with `java.lang` being default;

```prolog
:- solve(member(X,[1,Double.D,"3"])).
:- solve(member(Integer.X,[1,Double.D,"3"])).

% Standard type-less rules for the standard member predicate
member(X,[X|Xs]).  % X is a member of a list if it is the first element
member(X,[_|Xs]) :- % X is a member of a list if it is in the list tail
       member(X,Xs).
```

> X=1
> X=java.lang.Double.D
> X=3
> java.lang.Integer.X=1
Description Logic (DL) type system

- DL-typed and untyped variables;
- Uses Semantic Web ontologies as type systems
- Uses external DL reasoner (e.g. Pellet) for dynamic type checking
- Syntax: `[Variable]^[nameSpace]_[[ClassType]]
  [individualConstant]^[nameSpace]_[[ClassType]]

```prolog
:- eval(consult('ContractLog/owl.prova')). % needed
% import external type system (T-Box model) and individuals (A-Box)
import("http://example.org/WineProjectOWL.owl").
% use OWL-DL reasoner; for a list of available predefined reasoners see OWL2PROVA.java
reasoner ("dl").
% typed rule
serve (X^[default:Wine]) :- recommended(X^[default:Wine]).
% ground fact; defines an instance of class
recommended("Chardonnay^[default:Wine]").
% non ground DL facts are interpreted as queries on external ontology
recommended(X^[default:White_Wine]).
recommended(X^[default:Red_Wine]).
:-solve(recommended(X^[default:Wine])).
:-solve (recommended(X^[default:White_Wine])).
```
Java Method Calls

- Constructors, instance and static methods, and public field access;
- Ability to embed Java calls makes the Prolog-like programming style more suitable for integration and computation workflows.

```prolog
hello(Name):-
  S = java.lang.String("Hello").
  S.append (Name),
  java.lang.System.out.println (S).
```
Example: Java-based XML Processing (DOM)

- A small wrapper for XML DOM in Java is needed because of bugs in reflection and to improve access to elements.
- Non-deterministic iteration over elements and attributes node collections with `nodes` method.

```java
:- eval(test_xml()).

test_xml() :-
  % This extends the standard DOM API: create a Document based on File name
  Document = XML("blast.xml"),
  Root = Document.getDocumentElement(),
  Elements = Root.getElementsByTagName("Hit"),
  % This extends the standard DOM API: enumerate the nodes
  Elements.nodes(Element),
  SubElements = Element.getElementsByTagName("Hsp"),
  SubElements.nodes(SubElement),
  ChildNodes = SubElement.getChildNodes(),
  ChildNodes.nodes(ChildNode),
  ChildNodeName = ChildNode.getNodeName(),
  DataName = ChildNode.getFirstChild(),
  StringName = DataName.getNodeValue(),
  println(['Child name: ',ChildNodeName]),
  println(['Child value: ',StringName]).
```
Exception Handling

- Exception handling that results in a failure and backtracking
- Compensation handling in which the control flow continues

```prolog
:- eval(raise_test()).

handle_exceptions (Msg) :-
    exception(Ex),
    println([Ex]),
    println([Msg]),
    fail(). % fail

raise_test () :-
    on_exception(java.lang.Exception,
      handle_exceptions(" in raise_test()"),
    Ex = java.lang.Exception("A Prova exception"),
    raise(Ex). % throw exception
```
Built-Ins

- Simple arithmetic relations (+ - = ...)

```prolog
% Prova
N2 = N + 1,
```

- Negations (not, neg)

```prolog
% Default Negation
register(User) :- not(known(User)), ...
```

- Fact base updates

```prolog
% Update global facts
register(User) :- not(known(User)), assert(known(User)).
```

- Variable mode tests (free, bound, type)

- String manipulation predicates

```prolog
% Concat Strings
concat(["","In,"],Out), % prepend "{" and append "}"
```
External Data and Object Integration

- **File Input / Output**
  ```
  ..., fopen(File,Reader), ...
  ```

- **XML (DOM)**
  ```
  document(DomTree,DocumentReader) :-
      XML(DocumenReader),
  ```

- **SQL**
  ```
  ..., sql_select(DB,cla,[pdb_id,"lalx"],[px, Domain]).
  ```

- **RDF**
  ```
  ..., rdf(http://...,"rdfs",Subject,"rdf_type","genel_Gene"),
  ```

- **XQuery**
  ```
  ..., XQuery = ' for $name in StatisticsURL//Author[0]/@name/text() return $name',
  xquery_select(XQuery,name(ExpertName)),
  ```

- **SPARQL**
  ```
  ..., sparql_select(SparqlQuery,name(Name),class(Class),
                   definition(Def)),
  ```
Examples: File I/O and SPARQL

test_fopen() :-
    fopen(File,Reader),
    % Non-deterministically enumerate lines in the file
    read_enum(Reader,Line),
    println([Line]). % Print one line at a time

exampleSPARQLQuery(URL,Type|X) :-
    QueryString =
    ' PREFIX foaf: <http://xmlns.com/foaf/0.1/>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    SELECT ?contributor ?url ?type
    FROM <http://planetrdf.com/bloggers.rdf>
    WHERE {
        ?contributor foaf:name "Bob DuCharme" .
        ?contributor rdf:type ?type . }
    sparql_select(QueryString,url(URL),type(Type)|X),
    println([[url,URL],[type,Type]|X],"",").
Messaging Reaction Rules in Prova

- Send a message
  \[sendMsg(XID, Protocol, Agent, Performative, [Predicate|Args]|Context)\]
- Receive a message
  \[rcvMsg(XID, Protocol, Agent, Performative, [Predicate|Args]|Context)\]
- Receive multiple messages
  \[rcvMult(XID, Protocol, Agent, Performative, [Predicate|Args]|Context)\]

**Description:**
- **XID** is the conversation identifier
- **Protocol**: transport protocol e.g. *self*, *jade*, *jms*, *esb*
- **Agent**: denotes the target or sender of the message
- **Performative**: pragmatic context, e.g. FIPA ACL
- **[Predicate|Args]** or **Predicate(Arg\_1,..,Arg\_n)**: Message payload
Example: Remote Job/Task Scheduling

Manager

\[\text{upload\_mobile\_code}(\text{Remote}, \text{File}) : \]

\[
\begin{align*}
\text{Writer} &= \text{java.io.StringWriter}(), \quad \% \text{Opening a file} \\
\text{fopen} &=(\text{File}, \text{Reader}), \\
\text{copy} &=(\text{Reader}, \text{Writer}), \\
\text{Text} &= \text{Writer}.\text{toString}(), \\
\text{SB} &= \text{StringBuffer}(\text{Text}), \\
\text{sendMsg} &=(\text{XID}, \text{esb}, \text{Remote}, \text{query-ref}, \text{consult}(\text{SB})).
\end{align*}
\]

Service (Contractor)

\[\text{rcvMsg}(\text{XID}, \text{esb}, \text{Sender}, \text{eval}, [\text{Predicate}|\text{Args}]) : - \]

\[\text{derive}([\text{Predicate}|\text{Args}]).\]

Contract Net Protocol
Relations between Rule Standards

Mapping via RuleML / W3C RIF

BMI Semantics for Business Vocabularies & Rules (SBVR)
- Direct Mapping for OWL
- Formal Grounding (CL)

BMI Production Rule Representation (PRR)
- Vocabulary in ODM
- Rules in PRR

Ontology Definition Metamodell (ODM)

Information Management Metamodell (IMM)
- Mappings Planned for ER, Logical DB, XML Schema, ...

(in process)
- Questions?
- RuleML
  - http://ruleml.org/
- Reaction RuleML
  - http://reaction.ruleml.org
- Prova: http://www.prova.ws/