

Prova – A Distributed Semantic Web Rule Engine

2007-10-15

Biotec / Technical University Dresden
+49 351 463 40074
<http://biotec.tu-dresden.de>



Prova – A Distributed Semantic Web Rule Engine

Agenda

- Prova – A Semantic Web Rule Engine
- Distributed Prova Inference Services on the Web
- Summary and Future Projects

Adrian Paschke

BioTec TU Dresden, Germany

W3C HCLS Telecon, October, 15th, 2007

© Chair of Bioinformatics
Dept. of Artificial Intelligence, TU Dresden



Prova

- Open-Source Project hosted at Sourceforge (<http://www.prova.ws/>)
- Distributed Semantic Web Rule Engine and Agent/Service Oriented Architecture written in Java focusing among other application domains on bioinformatics applications
- Principles: Separation of *logic* (declarative), *data access* (procedural attachments and query languages), and *computation* (inference and external procedural computation)
- Combines the benefits of declarative (rule-based), Semantic Web meta data and ontology, procedural/object oriented (e.g., Java), and workflow languages (e.g., BPEL)
- Standard ISO Prolog scripting syntax + tight integration of Java and “webized” Semantic Web languages as type systems (vocabularies)
- “*Distributed Modular Meta-Data Annotated Labeled Order-Sorted Typed Extended Logic Programs*”

Syntax – Selected Examples (1)

■ Variables (start with upper case letter), Constants (start with lower case letter)

■ Fact: `availability(s1,99%).`

■ Rule: `qos(S,high):- availability(S,99%).`

■ Query: `:-solve(not(qos(S,high))).`

■ Module Imports: `:-eval(consult("ContractLog/list.prova")).`

```
:-eval(consult("http://ibis.in.tum.de/projects/
rbsla/ContractLog/list.prova")).
```

■ Meta Data Annotation:

```
metadata(label(r1),src("www.prova.ws"),dc_author("AP")) ::
    qos(S,medium) :- availability(S,98%).
```

■ Scoping:

```
p(Out):-
    partial(append([],
    [neg,[operonNeighbors,A,B,Direction,Gap]],Out),
    "ContractLog/list.prova").
```

```
(→Out=neg(operonNeighbors(A,B,Direction,Gap)))
```

Syntax – Selected Examples (2)

■ Procedural Attachments (Java Integration):

```
test_collections(Map) :-  
  Map=java.util.HashMap(),% Constructor of a Java HashMap  
  Map.put(0,"false"),      % Instance method calls  
  Map.put(1,"true"),  
  println(["Map=",Map]).  % Built-in predicate for printing  
  
:-eval(test_collections(java.util.HashMap.Map)).
```

■ Typing:

```
p(X:gene1_Gene) :- q(X:gene1_Gene).
```

■ Updates:

```
a(X,Y,Z):-add(id1,"b(_0,_1,_2). c(_1,_0).",[X,Y,Z]).
```

■ Integrity Constraints: (and, or, not, xor):

```
integrity(not(p1(x),p1(a))).
```

```
integrity(xor(p1(a),z(),p1(x))).
```

Prova – Selected Expressive Features

■ External Data and Object Integration

■ Java Integration

```
date(Calendar):- Calendar= java.util.Calendar.getInstance().
```

■ XML Integration

```
document(DomTree,DocumentReader) :- XML(DocumentReader).
```

■ SQL Integration

```
domain(Domain):- ...,sql_select(DB,cla,[pdb_id,"1alx"],[px,Domain]).
```

■ RDF Integration

```
rdf(http://..., "rdfs", Subject, "rdf_type", "gene1_Gene").
```

■ External Type Systems

■ Java Class Hierarchies

```
add(java.lang.Double.Out, java.lang.Integer.In1, java.lang.Integer.In2) :-
```

```
add(java.lang.String.Out, java.lang.String.In1, java.lang.String.In2):-
```

■ Semantic Web Ontologies

```
sameTranscriptionDirection(A:gene_Operon, B:gene_Operon) :- ...
```

■ Input/Output Mode Declarations

```
add(-,+,+):: add(Out, In1, In2) :- free(Out), bound(In1), bound(In2), ... .
```

■ Meta Data Label and Scopes (constructive views)

```
metadata(topic("mutagenesis")) :: <rule>.
```

```
<head> :- scope(<literal>, topic("mutagenesis")).
```



OWL2Prova Homogeneous Integration

1. Different reasoners

- *Transitive, RDFS, OWL, OWL Mini*
- Precompile RDF/RDFS/OWL model into inferred triple statement model

2. Converters are used to translate triple statements into arbitrary LP facts / rules

- SimpleConverter translates to arbitrary outputs according to user-defined patterns, e.g:

`["predicate","subject","object"] => predicate(subject,object)`

`["rdf","subject","object"] => rdf(subject,object)`

`["rdfTriple","subject","predicate","object"] => rdfTriple(subject,predicate,object)`

- DLPCConverter translates to DLPs:

$C(X) \rightarrow D(x)$ class C is subclassOf class D

$P(x,y) \rightarrow Q(x,y)$ property P is a subproperty of property Q

$P(x,y) \rightarrow C(y)$ range of property P is class C

$C(X) \rightarrow D(X)$

$D(X) \rightarrow C(X)$ Class C and Class D are equivalent

...

(Note: **Needs loop checker** – OWL2Prova adds a kind of memoization of critical subgoals in order to prevent loops)

- DefeasibleDLPCConverter translates into defeasible DLPs
- Further user-defined converters can be easily added, extending the OWL2Prova Converter interface

3. The precompiled and translated models can be added to an existing KB and the rule engine is used for querying

OWL2Prova Converter Example

Ontology

```
...  
<owl:Class rdf:ID="Red_Wine">  
  <rdfs:subClassOf rdf:resource="#Wine"/>  
</owl:Class>  
...
```

Homogeneous Integration

```
:-eval(consult('ContractLog/owl.prova')).  
% translate and include OWL Facts  
:-eval(assertOWL(  
  "", % no reasoner  
  ["triple","subject","predicate","object"],  
  "./WineProjectOWL.owl")). % input file  
  
% query all triples  
:-solve(triple(Subject,Predicate,Object)).
```

Translated facts

```
... triple(wine_Red_Wine, rdfs_subClassOf, wine_Wine). ...
```



OWL2Prova Hybrid Integration

- RDF Triple Query Syntax
- Uses external reasoner to answer queries
- Uses efficient cache to memorize previous queries

```
vCardURL("http://ibis.in.tum.de/projects/paw/use_case4/vCard_Adrian.rdf").  
person(Name,Role, Title, EMail, Telephone):-  
    vCardURL(URL),  
    rdf(URL,"default",_, "vCard_FN", Name),  
    rdf(URL,"default",_, "vCard_ROLE", Role),  
    rdf(URL,"default",_, "vCard_TITLE", Title),  
    rdf(URL,"default",_, "vCard_TEL", TelNode),  
    rdf(URL,"default",TelNode,"rdf_value",Telephone),  
    rdf(URL,"default",_, "vCard_EMAIL",EMailNode),  
    rdf(URL,"default",EMailNode,"rdf_value",EMail).
```

OWL2Prova DL-Typed Hybrid Description Logic Programs

■ Hybrid integration

- of DL ontologies serialized in OWL Lite / DL (or RDFS) into Rules as types of logical terms

■ Prescriptive Typing Approach: “*term:type*”

- Fresh individuals are allowed in rule heads
- Free DL-typed variables are allowed in facts → Query on *known* individuals in A-box

■ Dynamic Type Checking

- Via calling external DL reasoner for e.g. *subsumption inference*, *instantiate inference*, *equivalence mapping* etc.
- Caches inferred models for faster access on DL knowledge base

■ Polymorphic Order-Sorted Unification

- Ad-hoc polymorphism: Variables might change their types during unification
- Type-casting in the sense of coercion is supported by typed unification

■ Supports different Semantic Web Type Systems:

- RDFS, OWL-Lite, OWL-DL , DL ontol.



Advantages

- Static and dynamic type checking
- Supports SE principles such as data abstraction or modularization for large distributed unitized rule bases
- Reduces search space of queries on typed rule sets
- Extends expressive power of rules with Description Logics KR
- Exploits external optimized DL
- Caches inferred models for faster access on DL knowledge base
- Allows ad-hoc polymorphism with overloading and coercion
- Integration of domain-specific Semantic Web ontologies (vocabularies) into domain-independent rule specifications and executions
 - Facilitates Rule Interchange
 - Facilitates Collaboration
 - Facilitates Distributed Management



Typed Unification Rules – Operational Semantics

■ Untyped Unification

- Ordinary untyped unification without type checking

■ Untyped-Typed Unification

- The untyped query variable assumes the type of the typed target

■ Variable-Variable Unification:

- If the query variable is of the same type as the target variable or belongs to a subtype of the target variable, the query variable retains its type, i.e. the target variable is replaced by the query variable.
- If the query variable belongs to a super type of the target variable, the query variable assumes the type of the target variable, i.e. the query variable is replaced by the target variable.
- If the query and the target variable are not assignable the unification fails

■ Variable-Constant Term Unification:

- If a variable is unified with a constant of its super-type, the unification fails.
- If the type of the constant is the same or a sub-type of the variable, it succeeds and the variable becomes instantiated.

■ Constant-Constant Term Unification:

- Both constants are equal and the type of the query constant is equal to the type of the target constant.

■ Complex terms such as lists are untyped by default and hence are only allowed to be unified with untyped variables resp. variables of type "Resource".

Typing, Import and Java Intergration Examples

```
import("http://../dl_typing/businessVocabulary1.owl").
import("http://../dl_typing/mathVocabulary.owl").
import("http://../dl_typing/currencyVocabulary.owl").

reasoner("dl"). % configure reasoner (OWL-DL=Pellet)

% Rule-based Discount Policy
discount(X:businessVoc1_Customer, math_Percentage:10) :-
    gold(X:businessVoc1_Customer).
discount(X:businessVoc2_Client, math_Percentage:5) :-
    silver(X:businessVoc1_Client).
discount(X:businessVoc2_Customer, math_Percentage:2) :-
    bronze(X:businessVoc1_Customer).
discount(X, 0) :-
    not(spending(X,lastYear)).    ...
```

```
% module imports
:-eval(consult(
    './ContractLog/math.prova')).

:-eval(consult(
    'http://ibis.in.tum.de/
    projects/rbsla/test.prova')).

% Java Integration

readfiles(Files) :-
    Dir=java.io.File("."),
    Files=Dir.list().

% Goal
:- eval(readfiles(Files)).
```

```
:-solve(discount(X:businessVoc2_Customer,Y:math_Percentage)).
:-solve(discount(X:businessVoc1_Customer, math_Percentage:2)).
:-solve(discount(X:rdfs_Resource, 5)).
:-solve(discount(X,Y)).
:-solve(readfiles(Files)).
```

SPARQL Built-In

■ `sparql_select` built-in:

```
% The most straightforward way of running SPARQL queries that
% indicates the required data right inside
% the FROM clause
ex059() :-
    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 foaf:weblog ?url .
            ?contributor rdf:type ?type .
        }
    ' ,
    sparql_select(QueryString,url(URL),type(Type)|X) ,
    println([[url,URL],[type,Type]|X],",").

% Prints:
% ["url",http://www.snee.com/bobdc.blog/],
% ["type",http://xmlns.com/foaf/0.1/Agent],
% ["contributor",72260fc5:10c1027fc9f:-7fef]

% ["url",http://www.snee.com/bobdc.blog/],
% ["type",http://xmlns.com/foaf/0.1/Person],
% ["contributor",72260fc5:10c1027fc9f:-7fef]
```



Inductive Logic Programming Examples in Prova

```
% substitute a clause p(X):-q(X) with X/h(a) → [p(h(a)), q(h(a))]  
:-solve(substitute( [p(X), q(X)], Instance, [[X, h(a)]] )).
```

```
% lgg of two clauses; LGG = p(X) :- q(X)  
:-solve(lgg( [p(a), q(a)], [p(b), q(b)], LGG )).
```

```
% Generalize a LP with the rules p(a):-q(a). p(a):-q(a),r(a). ... and the facts r(a). q(a). ...  
% and return the generalized LP (set of general rules)
```

```
:-solve(generalize( [  
    [p(a), q(a)],  
    [p(a), q(a), r(a)],  
    [p(b), q(b)],  
    [p(c), q(c)],  
    [r(a)], [q(a)], [q(b)], [q(c)] ],  
    Generalization )).
```

```
% Return the covered clause with respect to the background knowledge  
:-solve(cover( LP, B, CoveredClause )).
```

```
% Computes the coverage level for a given test (goal) and a LP.
```

```
:-solve(coverage( G, LP, CoveredClauses, NotCoveredClauses, CoverageLevel ) )
```

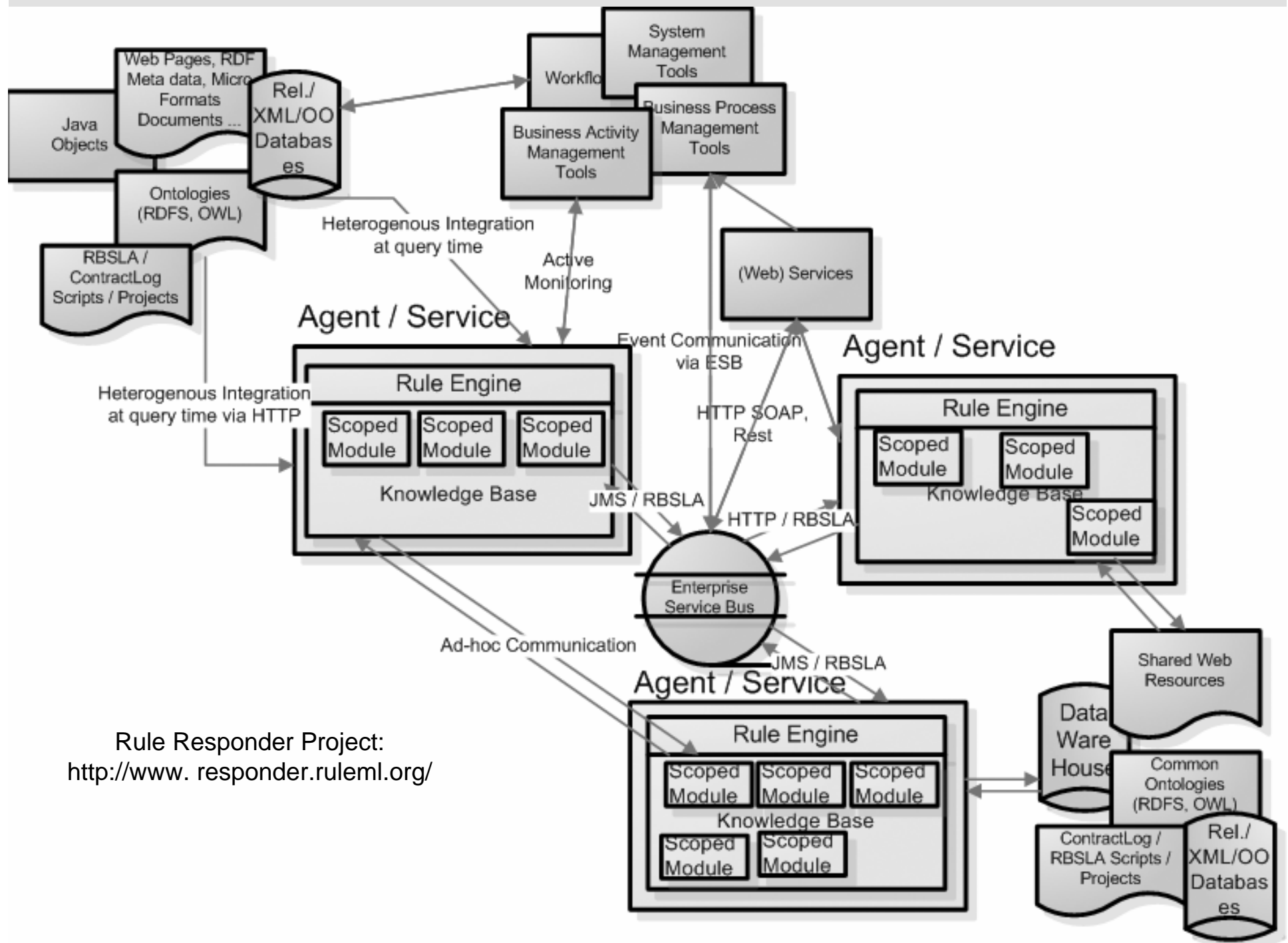


ContractLog KR – Framework of Logic Formalisms

Logic	Application
Extended Logic Programs + Extensions (Hybrid LPs)	Derivation rules, negation, integration of object-oriented code (Java), external databases (SQL)
Event-Condition-Action rules (ECA)	Reaction rules, complex event/action processing
Event Calculus	Temporal reasoning about effects of events / actions
Defeasible logic and Integrity Constraints	Integrity constraints, default rules, exceptions, priorities between rules and rule sets
Deontic logic	Rights and obligations, contract norms
Description logic	Integration of Semantic Web contract vocabularies (RDFS, OWL) and domain-specific meta data
Metadata Annotated Ordered Logic Programs	Metadata annotation of rules, modularization of rule sets, scoped reasoning
Inductive Logic	Meta Inference Engine (Substitution, Specialization, Generalization, Least General Generalization)
Libraries	Math, Date, Time, Interval, Lists ...



Distributed Prova Inference Services



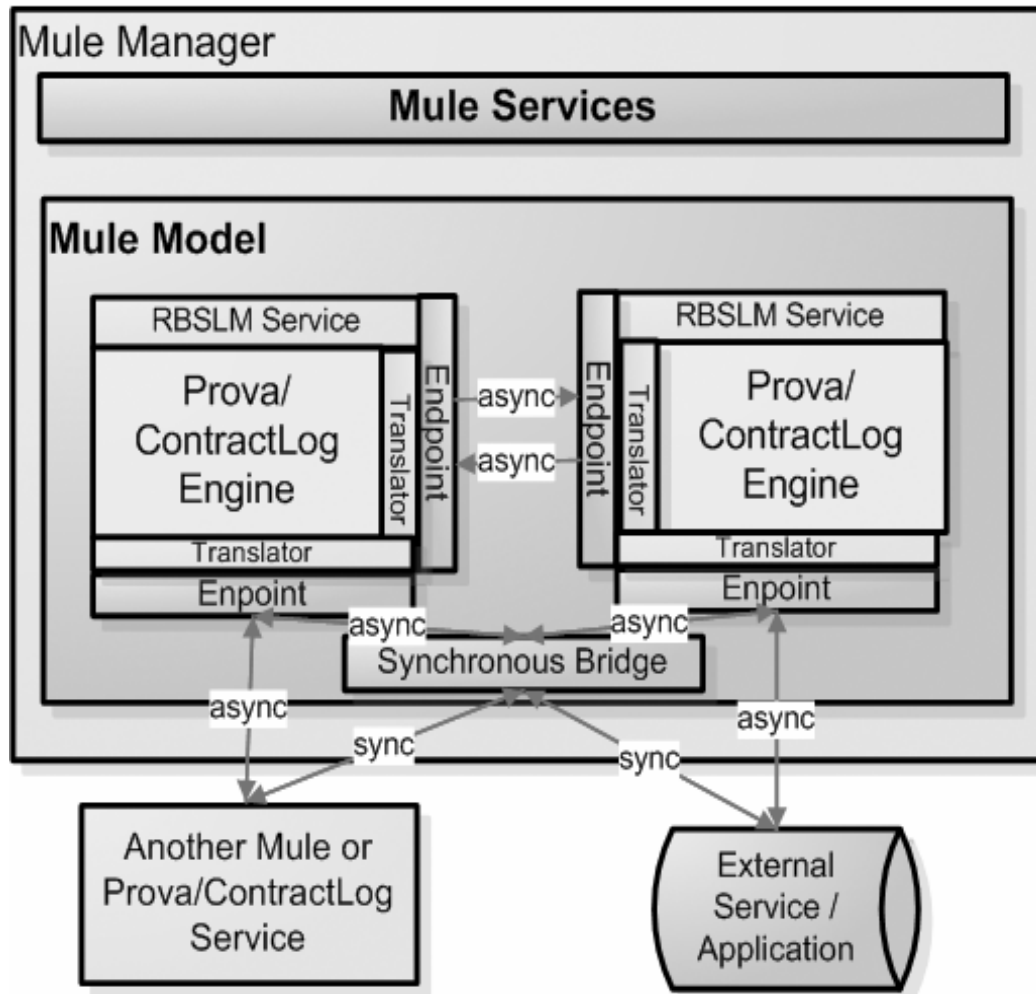
Rule Responder Project:
<http://www.responder.ruleml.org/>

Rule Responder Middleware

1. Computational independent model (CIM) with rules in a natural or visual language
 - e.g. Reaction RuleML Editor
2. Platform independent model (PIM) which represents the rules in a common (standardized) interchange format (e.g. a markup language)
 - RuleML / Reaction RuleML
 - Enterprise Service Bus (ESB); >30 transport protocols (e.g. JMS, HTTP)
3. Platform specific model (PSM) which encodes the rule statements in the language of a specific execution environment
 - e.g. Prova scripting language



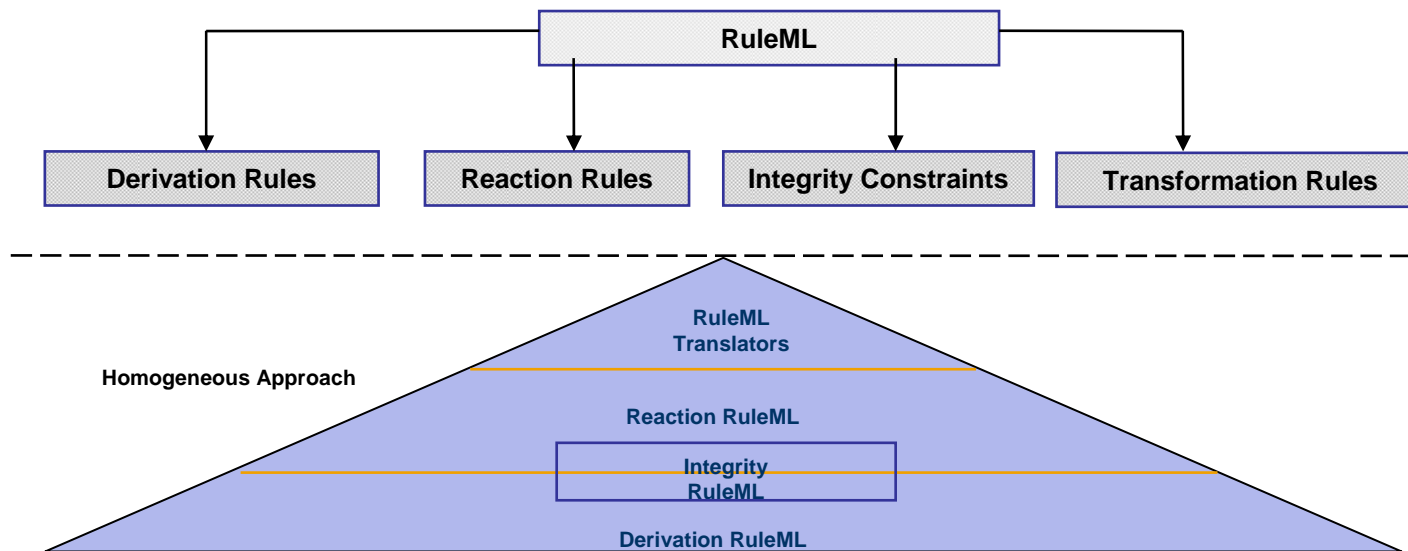
Mule Enterprise Service Bus



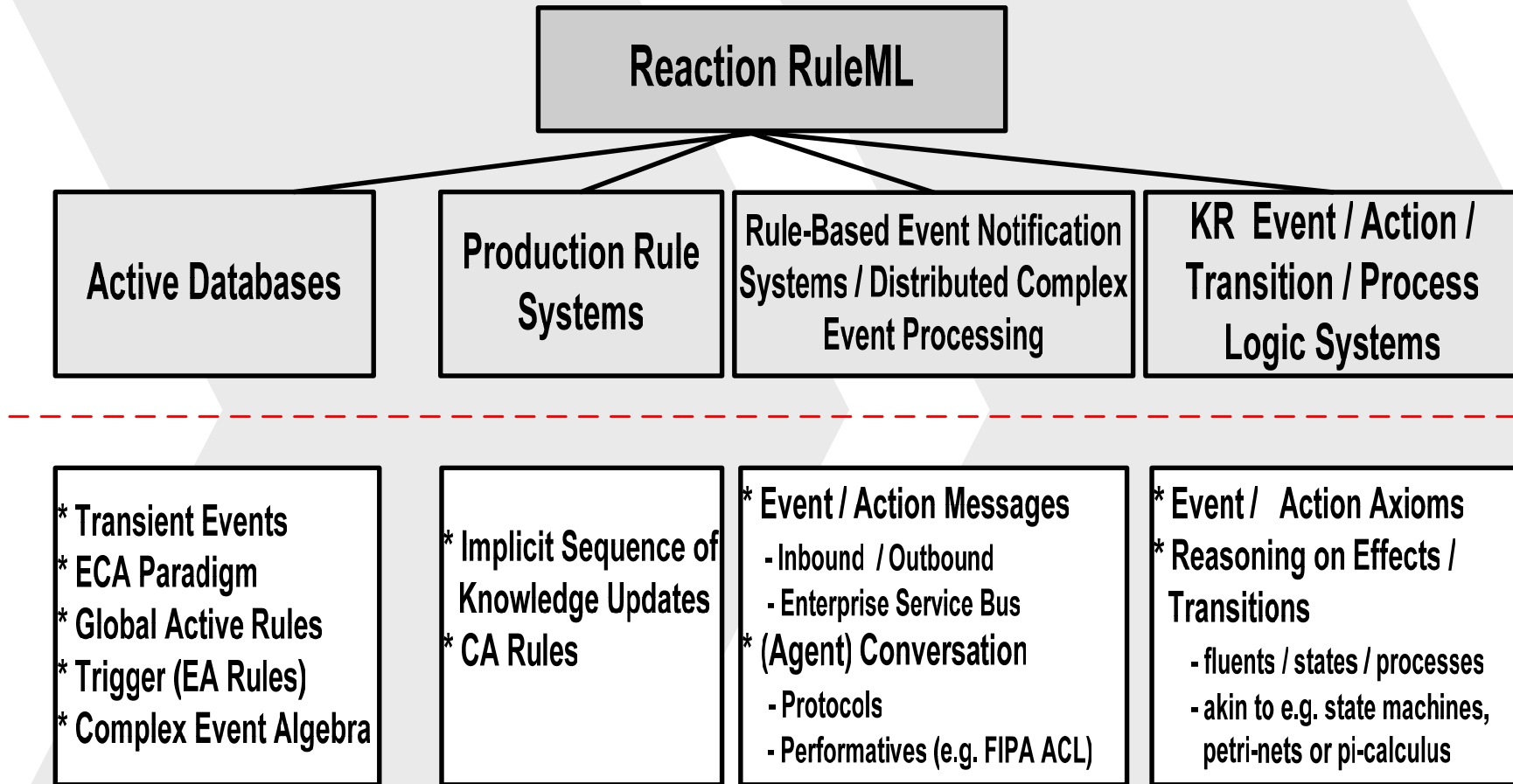
- Mule ESB Open Source
- Message Platform and distributed Object Broker
- Staged Event Driven Architecture (SEDA)
- > 30 Protocols (JMS, HTTP, SOAP ...)
- Synchronous and Asynchronous Communication
- Complex Message-driven Event Processing (CEP)

RuleML as Common Rule Interchange Format

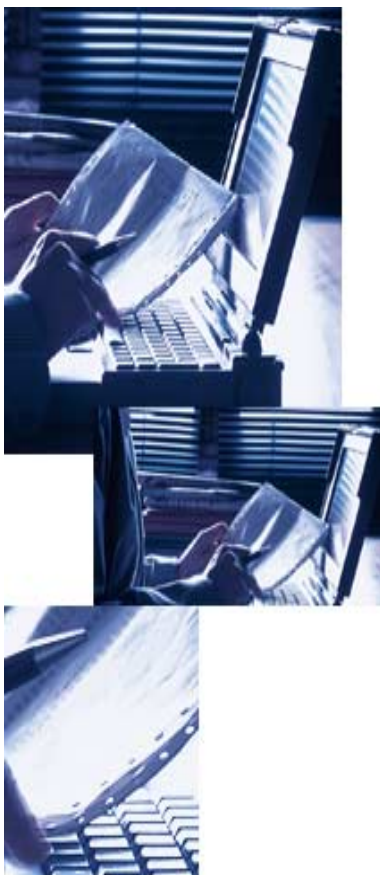
- Rule Markup and Modeling Initiative (RuleML) (www.ruleml.org)
 - Standardization effort for a rule markup and modelling language, tools and applications
- Reaction RuleML (<http://ibis.in.tum.de/research/ReactionRuleML/>)
 - Language family for reaction rules and complex event messaging / processing
- RuleML is the de facto open language standard for rule interchange/ rule markup



Scope of Reaction RuleML



General Syntax for Reaction Rules (Reaction RuleML 0.2)



```
<Rule style="active" evaluation="strong">  
  <label> <!-- metadata --> </label>  
  <scope> <!-- scope --> </scope>  
  <qualification> <!-- qualifications --> </qualification>  
  <oid> <!-- object identifier --> </oid>  
  
  <on> <!-- event --> </on>  
  <if> <!-- condition --> </if>  
  <then> <!-- conclusion --> </then>  
  <do> <!-- action --> </do>  
  <after> <!-- postcondition --> </after>  
  <else> <!-- else conclusion --> </else>  
  <elseDo> <!-- else/alternative action --> </elseDo>  
  <elseAfter> <!-- else postcondition --> </elseAfter>  
  
</Rule>
```

Messages in Reaction RuleML

```
<Message mode="outbound" directive="ACL:inform" >
  <oid> <!-- conversation ID--> </oid>
  <protocol> <!-- transport protocol --> </protocol>
  <sender> <!-- sender agent/service --> </sender>
  <content> <!-- message payload --> </content>
</Message>
```

- **@mode** = inbound|outbound – attribute defining the type of a message
- **@directive** – attribute defining the pragmatic context of the message, e.g. a FIPA ACL performative
- **< oid >** – the conversation id used to distinguish multiple conversations and conversation states
- **< protocol >** – a transport protocol such as HTTP, JMS, SOAP, Jade, Enterprise Service Bus (ESB) ...
- **< sender >< receiver >** – the sender/receiver agent/service of the message
- **< content >** – message payload transporting a RuleML / Reaction RuleML query, answer or rule base

Prova Agent / Service Architecture

Prova-AA offers

- ✓ message-oriented context-dependend reaction rules;
- ✓ message sending and receiving for local and remote communication actions;
- ✓ uniform handling of communication protocols (JMS, JADE, plus any of the more than 30 protocol supported by the Enterprise Service Bus).
- ✓ message payload as complex terms containing typed or untyped Java variables
and serializable Java objects;
- ✓ state machine, Petri nets, or pi-calculus based conversation protocols;
- ✓ context-dependent inline reactions for asynchronous message exchange;
- ✓ ability to distribute mobile rulebases to remote agents;
- ✓ *Communicator* class for simplified embedding of Prova agents in Java and Web applications;
- ✓ Prova Universal Message Object gateway for reactive agents on ESB.
- ✓ Multi-threaded Swing programming with Prova Java calls and reaction rules.



Messaging Reaction Rules

- Send a message

sendMsg(XID, Protocol, Agent, Performative, [Predicate|Args]|Context)

- Receive a message

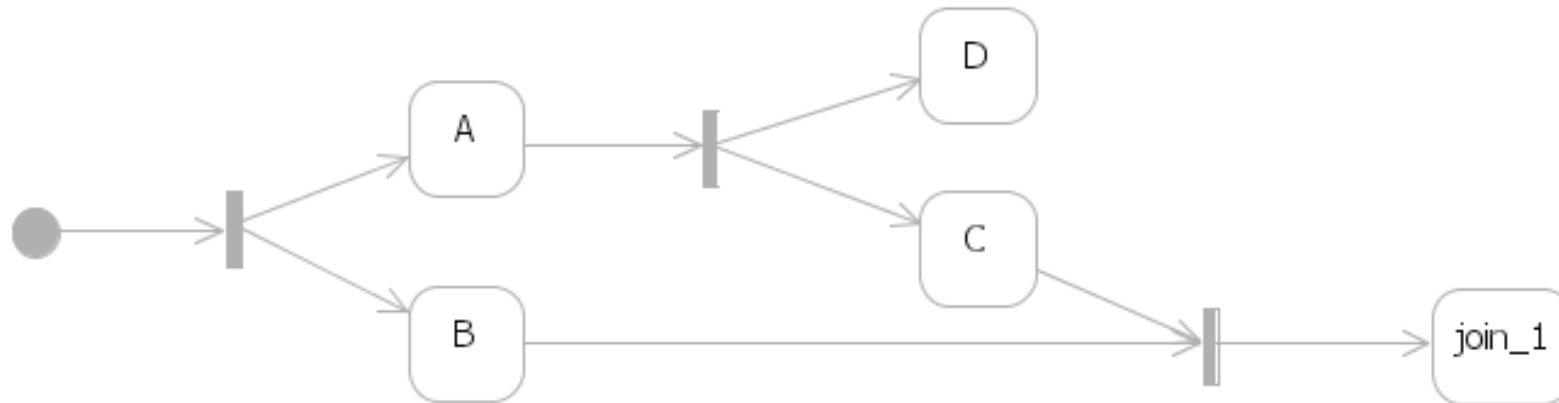
rcvMsg(XID, Protocol, Agent, Performative, [Predicate|Args]|Context)

- 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(Arg₁,...,Arg_n): Message payload*



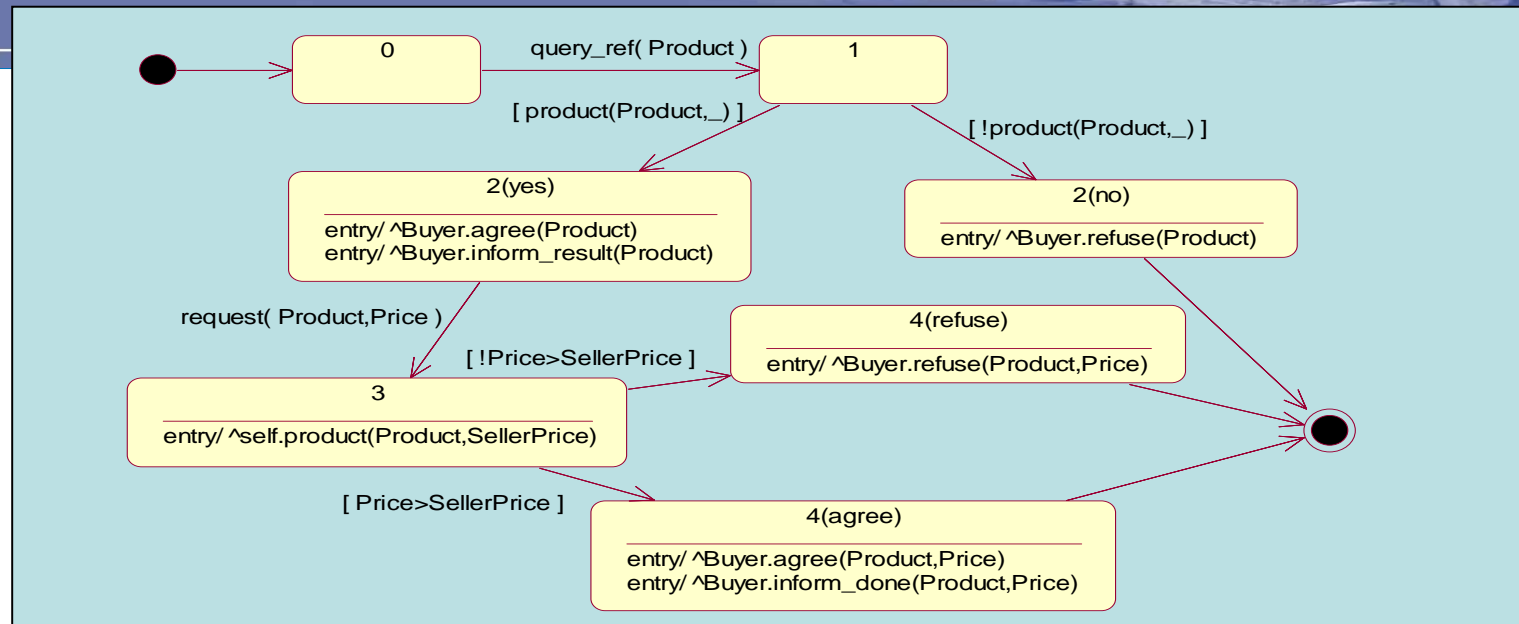
```

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]]
  
```

State machines based conversations



```

directbuy_seller_1(XID,Protocol,From,Product) :-
    product(Product|_),
    !,
    sendMsg(XID,Protocol,From,agree,Product,seller),
    sendMsg(XID,Protocol,From,inform_result,Product,seller),
    directbuy_seller_2(yes,XID,Protocol,From,Product).
directbuy_seller_1(XID,Protocol,From,Product) :-
    sendMsg(XID,Protocol,From,refuse,Product,seller),
    directbuy_seller_2(no,XID,Protocol,From,Product).
directbuy_seller_2(yes,XID,Protocol,From,Product) :-
    !,
    rcvMsg(XID,Protocol,From,request,[Product,Price],buyer),
    product(Product,SellerPrice),
    directbuy_seller_3(XID,Protocol,From,Product,Price,SellerPrice).

directbuy_seller_2(no,XID,Protocol,From,Product).
    
```

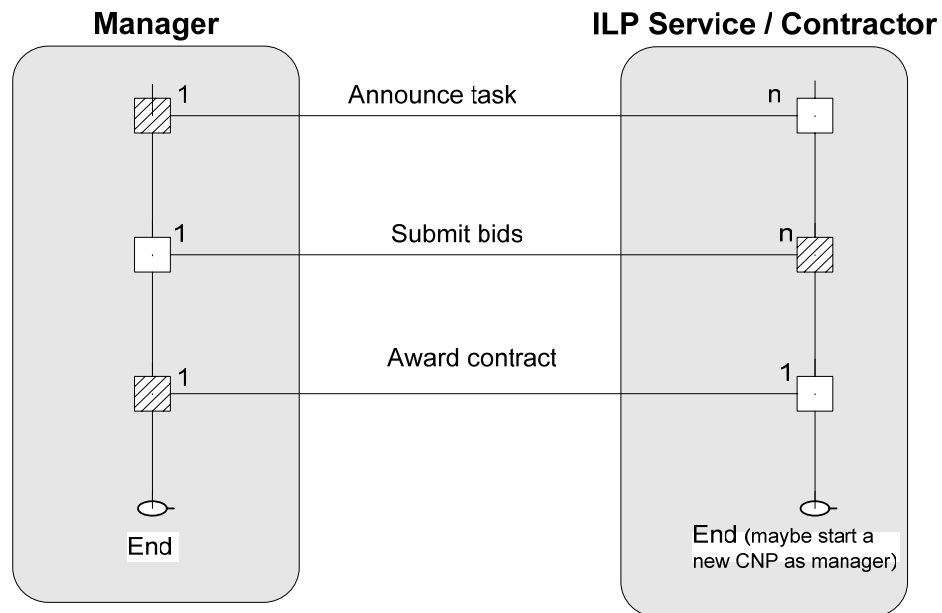
Job/Task Scheduling

```
% Manager
```

```
upload_mobile_code(Remote,File) :  
  Writer = java.io.StringWriter(), % Opening a file fopen(File,Reader),  
  copy(Reader,Writer),  
  Text = Writer.toString(),  
  SB = StringBuffer(Text),  
  sendMsg(XID,esb,Remote,eval,consult(SB)).
```

```
% ILP Service (Contractor)
```

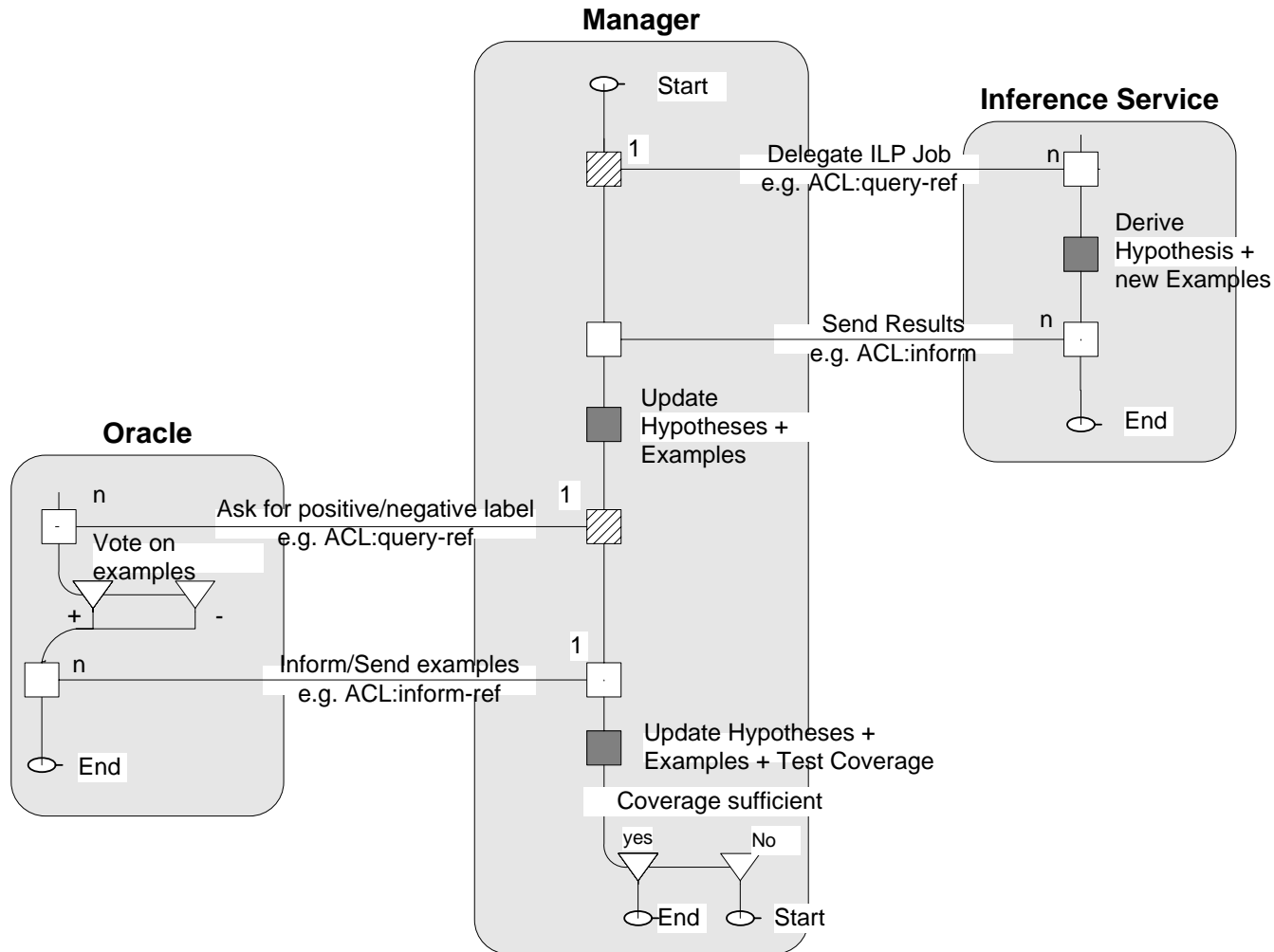
```
rcvMsg(XID,esb,Sender,eval,[Predicate|Args]):-derive([Predicate|Args]).
```



Contract Net Protocol



Interactive ILP with Conversations



**Summary
and
Future Steps**

Summary + Future Steps

■ Prova as distributed Semantic Web Inference Service Oriented

Architecture

- Combines declarative LP, Object-oriented Programming, rel. and semi-structured data access on external data sources
- Supports distributed Logic Programming via meta programming techniques and novel techniques from service-oriented computing, complex event processing as basis for inference service grids, resource sharing networks and parallel processing.
- External type systems / vocabularies
- Supports integration of **arbitrary inference and web services / tools**

■ Applications of Prova in different projects (GoPubMed, RBSLA, Rule Responder,...)

■ Future steps

- Integration of further rule engines into the middleware (Rule Responder project)
- Syntactical standardization issues (W3C Rule Interchange Format – Reaction RuleML)
- FungalWeb project based on GoWeb with chemical compounds (Chebi), enzymes (Mesh, EC classification), species; Chebi and enzyme classification (EC)



Links

- RuleML: <http://www.ruleml.org/>
- Reaction RuleML: <http://ibis.in.tum.de/research/ReactionRuleML/>
- Prova: <http://www.prova.ws/>
- Rule Responder: <http://responder.ruleml.org/>
- GoPubMed: <http://www.gopubmed.org/>



Thank you !!!!!!!

Questions ?