Combining Rules and Semantics in Drools
A Preliminary Study

Davide Sottara [dsotty@gmail.com]

\(^a\)Department of Computer Science, Electronics and Systems
Faculty of Engineering, University of Bologna
Viale Risorgimento 2, 40100 Bologna (BO) Italy

Bologna/San Diego - April 19-23th, 2010
Outline

1 Introduction
   - Motivations
   - Modelling Knowledge

2 Integrated “Semantic Reasoning”
   - RDF
   - RDFS
   - Towards Description Logics
     - OWL-like Axioms
     - OWL-like Constructors

3 Embedding Semantics in Rules

4 Conclusions
Ontologies

Ontology: *A formal specification of the terms in a domain*

- Capture knowledge about some **domain** of interest
- Describe the **concepts** in the domain
- State the **relationships** that hold between them
- List the **individuals** and their features
Outline

1. Introduction
   - Motivations
     - Modelling Knowledge

2. Integrated “Semantic Reasoning”
   - RDF
   - RDFS
   - Towards Description Logics
     - OWL-like Axioms
     - OWL-like Constructors

3. Embedding Semantics in Rules

4. Conclusions
Semantic Descriptions: Motivations

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To separate domain knowledge from operational knowledge
- To analyse domain knowledge
Objects naturally fall into categories, possibly more than one...
Categories (simple or complex) can be more general or specific than others...
Objects have parts and relationships among them...
Semantic Reasoning: Motivations

- Objects naturally fall into categories, possibly more than one...
- Categories (simple or complex) can be more general or specific than others...
- Objects have parts and relationships among them...

So we would like...

- to define generalization relations
- to automatically infer generalization hierarchies from the provided descriptions
- to represent complex concepts by “composition” of simpler concepts
- to know if an individual belongs to some category or not
Semantic Rule-Based Reasoning: Motivations

- Descriptions still have some limitations
  - Capturing complex relations between properties
  - Capturing complex relations between individuals
- Adding **Operative** behaviour
  - We know that an individual belongs to some class: now what?
Many languages in different contexts, including:

**Semantic Web**
- Relational
  - RDF
  - RDF-S
- (Description) Logic-Based
  - OWL (Lite, DL, Full)
- Rule Integrations
  - SWRL
Challenges

- Languages need Reasoners to be useful
  - Complete
  - Correct
  - Efficient
  - Efficacious
Challenges

- Languages need Reasoners to be useful
  - Complete
  - Correct
  - Efficient
  - Efficacious

Drools and Semantics

Where does Drools stand?

- No support for "semantics" yet
- Some (relevant) limitations

What we want:

- Homogeneous Integration? (tightly coupled)
- Hybrid is also possible (loosely coupled)
Tight vs Loose Coupling

Hybrid:
- Separated Rule and Semantic Engines
- Different languages with common points
- Rule Engine delegates the evaluation

Hybrid: Drools Example
- Custom Evaluator wrapper

Homogeneous:
- Single Rule/Semantic Engine
- Unique language with sufficient expressiveness
- Engine supports both types of reasoning

Homogeneous: Drools Example
- Native evaluation

Person( this isA Patient.class )
**Tight vs Loose Coupling**

**Hybrid**
- Separated Rule and Semantic Engines
- Different languages with common points
- Rule Engine delegates the evaluation

**Homogeneous**
- Single Rule/Semantic Engine
- Unique language with sufficient expressiveness
- Engine supports both types of reasoning

**Pros and Cons**
- **Hybrid**
  - “Full” Expressiveness
  - Efficiency
  - Interfacing
  - KB alignment
- **Homogeneous**
  - Single component
  - Unified model
  - Limited expressiveness ??
  - Efficiency ??
Tight vs Loose Coupling

Hybrid

Use the Wrapper Pattern (see my other talk...)

Homogeneous

- Many Potentialities
- Currently many Open Issues !!
Tight vs Loose Coupling

Hybrid
Use the Wrapper Pattern (see my other talk...)

Homogeneous
- Many Potentialities
- Currently many Open Issues !!
- We’ll see what can be (easily) done...
Tight vs Loose Coupling

Hybrid
- Use the Wrapper Pattern (see my other talk...)

Homogeneous
- Many Potentialities
- Currently many Open Issues !!
- We’ll see what can be (easily) done...
- And what can’t be done (yet?)
Outline

1. Introduction
   - Motivations
   - Modelling Knowledge

2. Integrated “Semantic Reasoning”
   - RDF
   - RDFS
   - Towards Description Logics
     - OWL-like Axioms
     - OWL-like Constructors

3. Embedding Semantics in Rules

4. Conclusions
Knowledge is encoded using “triples”

\[ P(S,O) \]

reads (e.g.) “\( S \) has property \( P \) with respect to \( O \)”
Also graphical notation

RDF Triples\(^a\)

\(^a\)Prolog-like, namespaces omitted

type(sanGiovanni,pediatricHospital)
hasPatient(sanGiovanni,p)
hasName(p,"mario")
type(p,child)
RDF vs Drools

Mapping triples on (dynamic) beans

- generics?
- automatic translation?

```java
declare Property
@role(property)
@namespace(...)
  subject : Resource // Object
  object : Resource // Object
end

declare PropertyValue
  pred : Class<? extends Property>
  subject : Resource
  object : Resource
end
```
RDF vs Drools

Equivalent representation:

```java
rule "Triple 2 PropVal"
when
    $t : Property( $s: subject, $o : object)
then
    insert( new PropertyValue($t.class, $s, $o) );
end

rule "PropVal 2 Triple"
when
    PropertyValue( $p : pred, $s: subject, $o : object)
then
    insert( $p.newInstance($s, $o) );
end
```
Triples could be used in rules explicitly, possibly mixed with “usual” beans

**rule** "Visiting Parents"

**when**

$c : \text{Person( ) HasName}(c,"mario")$

$h : \text{Hospital( ) HasType}(h, \text{PediatricHospital.class})$

$p : \text{Person( ) HasChild}(p,c)$

$r : \text{HasPatient}(h,p)$

**then**

`insert( new Visits(p,h) );`

**end**

But this is just the beginning...
Outline

1. Introduction
   - Motivations
   - Modelling Knowledge

2. Integrated “Semantic Reasoning”
   - RDF
   - RDFS
   - Towards Description Logics
     - OWL-like Axioms
     - OWL-like Constructors

3. Embedding Semantics in Rules

4. Conclusions
**RDF Schema**

**RDF-S**

- Adds Schema information
  - Entity/Class Relations
  - Class/Class Relations

- Reason over and with *types*
### RDF Schema

**RDF-S**

- Adds Schema information
  - Entity/Class Relations
  - Class/Class Relations

- Reason *over* and *with* *types*
- Overcomes the *extends/instanceof* limitations
RDF Schema

RDF-S

Adds Schema information
- Entity/Class Relations
- Class/Class Relations

- Reason over and with types
- Overcomes the extends/instanceof limitations

Even in Drools:

```java
// static type
when Patient( ... )

// dynamic type
when $p : Person()
    Type($r, Patient.class)
```
Provided a few relations are defined:

<table>
<thead>
<tr>
<th>Schema Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Resource $\times$ Class</td>
</tr>
<tr>
<td>Subject</td>
<td>Property $\times$ Resource</td>
</tr>
<tr>
<td>Object</td>
<td>Property $\times$ Resource</td>
</tr>
<tr>
<td>Predicate</td>
<td>Property $\times$ Class</td>
</tr>
<tr>
<td>Value</td>
<td>Resource $\times$ Resource</td>
</tr>
<tr>
<td>Domain</td>
<td>Class$_{Property} \times$ Class</td>
</tr>
<tr>
<td>Range</td>
<td>Class$_{Property} \times$ Class</td>
</tr>
<tr>
<td>SubClassOf</td>
<td>Class $\times$ Class</td>
</tr>
<tr>
<td>SubPropertyOf</td>
<td>Class$<em>{Property} \times$ Class$</em>{Property}$</td>
</tr>
</tbody>
</table>

...
rule "DomainRange"
when
    $prop : SomeProperty( $subj, $obj )
    Domain( $prop.class, $dom )
    Range( $prop.class, $range )
then
    // from $prop definition:
    insert( new Type($subj,$dom) );
    insert( new Type($obj, $range) );
end
rule "SubClassOf"
when
  Type( $x, $klass )
  SubClassOf( $klass, $super )
then
  insert( new Type($x, $super) );
end

Type(X, Patient), SubClassOf(Patient, Person) ⇒
Type(X, Person)
RDF Schema - Axioms

**Rule**  "SubPropertyOf"

**When**

$p : Property( s, o )$

$SubPropertyOf( p.class, super )$

**Then**

$insert( super.newInstance( s, o ) )$

**End**
RDF(S) : Considerations

RDFS just makes implicit type declarations explicit
- Expressiveness is limited
- So is inference
+ Simple: Drools supports it easily
Outline

1. Introduction
   - Motivations
   - Modelling Knowledge

2. Integrated “Semantic Reasoning”
   - RDF
   - RDFS
   - Towards Description Logics
     - OWL-like Axioms
     - OWL-like Constructors

3. Embedding Semantics in Rules

4. Conclusions
Description Logics

- Several logic(s) with different expressive power
  - Different features: F,E,U,C,S,H,R,O,I,N,Q, ...
  - Different languages to encode them
    - OWL, KIF, ...

OWL-DL will be considered for reference
Uses of DL - Objectives

Define (complex) concepts - aka classes
Uses of DL - Objectives

Define (complex) concepts - aka classes in terms of other classes and properties
Uses of DL - Queries

**Subsumption**

$C \subseteq D$?
- Is $D$ a more general concept than $C$?

**Satisfiability**

$\exists x : x \in C$?
- Does $C$ allow members?

**Consistency**

$\{...\} \models \bot$?
- Does a set of facts lead to contradiction?

**Instantiation**

$\{...\} \models x \in C$?
- Is $x$ member of $C$ given the available knowledge?
Description Logics

OWL defines **axioms** and class **constructors**:

**Axioms**
- subClassOf
- equivalentClass
- subPropertyOf
- equivalentProperty
- disjointWith
- sameAs
- differentFrom
- transitiveProperty
- inversefunctionalProperty
- symmetricProperty
- inverseOf

**Constructors**
- intersectionOf
- unionOf
- complementOf
- oneOf
- allValuesFrom
- someValuesFrom
- hasValue
- minCardinality
- maxCardinality
Drools Integration

Drools works with instances.
Drools works with instances.

- \textit{Instantiation} is (almost) immediate
Drools Integration

Drools works with instances.

- *Instantiation* is (almost) immediate

- *Subsumption* can be reduced to *Satisfiability*
  - *Satisfiability* is still an open issue
Drools Integration

Drools works with instances.

- **Instantiation** is (almost) immediate

- **Subsumption** can be reduced to **Satisfiability**
  - **Satisfiability** is still an open issue

**Preliminary analysis:**
- Tableau algorithms seem the most likely candidates
  - generative
- Still need some features (e.g. backtracking, false relations)
- We’ll start from what can be done already
Outline

1. Introduction
   - Motivations
   - Modelling Knowledge

2. Integrated “Semantic Reasoning”
   - RDF
   - RDFS
   - Towards Description Logics
     - OWL-like Axioms
     - OWL-like Constructors

3. Embedding Semantics in Rules

4. Conclusions
Introduction

Towards Description Logics

Axioms: General Principles

Most axioms define features of Properties

- Meta-data specified using attributes
- Engine automatically inserts meta-facts
- Rule Bases automatically include meta-rules
Same as in RDFS, but...

```plaintext
declare Patient
@role(entity)
@subclass(Person)
@subclass(...)
end
```

Attribute @subclass inserts

`SubclassOf(Patient.class, Person.class)`
Same as in RDFS, but...

deflate  Patient
@role(entity)
@subclass(Person)
@subclass(…)
end

Attribute @subclass inserts
SubclassOf(Patient.class,Person.class)
Here hierarchy is declared, but not inferred
As for SubClassOf:

```xml
declare HasSon
@role(property)
@subproperty(HasChild)
end
```

Attribute @subproperty inserts
SubPropertyOf(HasSon.class,HasChild.class)
same as before - but DL do not entail subproperty relations!
Class/Property Equivalence

Two more Attributes:

- @equivalentClass()
- @equivalentProperty()

Syntactic sugar: $C \equiv D \iff (C \rightarrow D \land D \rightarrow C)$
Class/Property Equivalence

Two more Attributes:
- @equivalentClass()
- @equivalentProperty()

Syntactic sugar: $C \equiv D \iff (C \rightarrow D \land D \rightarrow C)$

... but also $(C \rightarrow D \land \neg C \rightarrow \neg D)$

remember/see the imperfect case?
Towards Description Logics

Disjoint

**declare** Male
@role(entity)
@disjointWith(Female)
end

The attribute controls the insertion of an instance of the relation:

**declare** DisjointWith
@role(property)
@symmetric
subject : Class
object : Class
end
"Disjoint" // not in standard Drools...

```
rule "Disjoint" 
when
  Type($x, $klass)
  DisjointWith($klass, $anotherKlass)
then
  insert( new Type($x,$anotherKlass, FALSE));
end
```

*Type*(X, Male) ⊨ ¬*Type*(X, Female)
(In)Equalities

Two relations, to be specified on an individual basis

```
declare Equals
  @role(property)
  @symmetric
  @transitive
  subject : Resource
  object : Resource
end
```

```
declare DifferentFrom
  @role(property)
  @symmetric
  subject : Resource
  object : Resource
end
```
Transitivity

The relation attribute @transitive allows to compute closures:

```java
declare Transitive
@role(property)
    subject : Class <? extends Property >
    object : boolean
end

rule "Closure"
when
    PropertyValue( $p, $x, $y )
    PropertyValue( $p, $y, $z )
    Transitive($p, true)
then
    insert( $p.newInstance($x,$z) );
end
```

```
Relative(X, Y), Relative(Y, Z) ⇒ Relative(X, Z)
```
Symmetry

The relation attribute `@symmetric` inverts roles:

```
declare Symmetric
@role(property)
  subject : Class < property extends Property >
  object : boolean
end

rule "Symmetry"
when
  $prop : PropertyValue( $p, $x, $y )
  Symmetric($p, true)
then
  insert( $p.newInstance($y,$x) );
end
```

Relative(X, Y) ⇒ Relative(Y, X)
Functionality

Functionality (resp. inverse-functional) properties are decorated using the attributes @functional and @invFunctional

```plaintext
rule "Functionality" //resp. inverse
when
    PropertyValue( $p, $x, $y )
    // as per object identity
    PropertyValue( $p, $x, $z != $y )
    Functional($p, true)
then
    insert( new SameAs($y,$z) );
end
```

HasFather(X, "john"), HasFather(X, "mrWhite") ⇒
SameAs("john", "mrWhite")
The relation attribute @inverse allows:

```%
% declare Inverse
% @role(property)
% @symmetric
% subject : Class<? extends Property>
% object : Class<? extends Property>
%
% rule "Inverse"
% when
%   PropertyValue( $p, $x, $y )
%   Inverse($p, $q)
% then
%   insert( $q.newInstance($y,$x) )
%
HasFather(X, Y) ⇒ FatherOf(Y, X)
```
Outline

1 Introduction
   • Motivations
   • Modelling Knowledge

2 Integrated “Semantic Reasoning”
   • RDF
   • RDFS
   • Towards Description Logics
     • OWL-like Axioms
     • OWL-like Constructors

3 Embedding Semantics in Rules

4 Conclusions
Constructors become specialized rule-like patterns

```
declare Klass
@restriction($x)( // target variable
    // Patterns here
    $x : Resource(... ) // binding
    ...
    )
end
```

Automatically inserts Type(x,Klass.class)
Towards Description Logics

Intersection

\[ C_1 \land \cdots \land C_n \rightarrow K \]

```
rule "Intersect"
when
  $x : \text{Resource}()
  \text{Type}($x, C_1.\text{class})
  \ldots
  \text{Type}($x, C_n.\text{class})
then
  \text{insert} (\text{new} \ \text{Type}($x, K.\text{class}) )
end
```
Union

\[ C_1 \lor \cdots \lor C_n \rightarrow K \]

**rule** "Union"

**when**

\$x : \text{Resource}(\)\]

\text{Type}(\$x, C_1.\text{class}) \\
\ldots \\
\text{or} \ \text{Type}(\$x, C_n.\text{class})

**then**

\text{insert}( \text{new} \ \text{Type}(\$x, K.\text{class}) )

**end**
Complement

$C \rightarrow \neg K$

See \texttt{disjointWith}

\begin{verbatim}
rule "Complement"
when
  $x : \text{Resource}()$
  \text{Type}($x, \ c : \ C.\ class$)
then
  \text{insert}( \ \text{new} \ \text{Type}($x, \ K.\ class, \ FALSE$) \ );
end
\end{verbatim}
Towards Description Logics

OneOf

\{e_1, \ldots, e_n\} \subseteq K

*rule* "One of Many"

*when* //one rule for each individual

\$x : \text{Resource}(\ldots)\ // e_x

*then*

*insert*( new \text{Type}($x$, K.class) );

*end*
All Values from

\[ \forall (P(X, Y) \land C(Y)) \rightarrow K(X) \]

**rule** "AllValues"

**when**

\$x : \text{Resource()}

\$k : \text{Class(...) } // k \text{ may be a "literal"}

**forall** ( SomeProperty($x$, $y$)

Type($y$, $k$) )

**then**

**insert** ( new Type($x$, K.class) )

**end**
Towards Description Logics

Some Values from

\[ \exists (P(X, Y) \land C(Y)) \rightarrow K(X) \]

**rule** "SomeValues"

**when**

\$x : \text{Resource}() \\
\$k : \text{Class(...)} // \$k \text{ may be a "literal"}

**exists** ( SomeProperty($x, y) \\
 Type($y, k) )

**then**

**insert** ( new Type($x, K.class) )

**end**
Cardinality of Values

\(|P(X, Y)| \leq n) \rightarrow K(X)\)

**Rule** "Cardinality"

**When**

$x : Resource()$

$Collection(size = N) // also > or <$

from collect ( SomeProperty($x$, $y$) )

**Then**

insert( new Type($x$, K.class) );

**End**
Towards Description Logics

On Constructors

- So far, more like class constraints
  - Still useful in practice!
- Not quite like DL reasoners
- Necessary (but not sufficient) : reverse constructors
Towards Description Logics

Intersection (reverse)

\[ C_1 \land \cdots \land C_n \leftarrow K \]

```
rule "IntersectRev"
when
  $t : \text{Type}(\$x, K.\text{class})
  \text{not} \ (\text{Type}(\$x, C_1.\text{class})
                       \text{Type}(\$x, C_2.\text{class}))
then
  \text{insert}( \text{new} \ \text{Type}(\$x, C_1.\text{class}) );
  \ldots
  \text{insert}( \text{new} \ \text{Type}(\$x, C_n.\text{class}) );
end
```
Non-deterministic: requires new features!

\[
C_1 \lor \cdots \lor C_n \leftarrow K
\]

```
rule "UnionRev"
when
  $t : Type($x,K.class )
  not (Type($x,C1.class ))
  not (Type($x,C2.class ))
then
  insertBackTrack(
    new Type($x,C1.class ),
    new Type($x,Cn.class )
  );
end
```
All Values from (reverse)

\[ \forall (P(X, Y) \land C(Y)) \leftarrow K(X) \]

rule "AllValues"
when
    Type($x,K.class$)
    $p : \text{SomeProperty($x,y$)}$
    not ( Type($y,C.class$) )
then
    insert( new Type($y,C.class$) );
end
Some Values from (reverse)

\[ \exists (P(X, Y) \land C(Y)) \leftarrow K(X) \]

```plaintext
rule "SomeValues"
when
  $t : Type($x,K.class)
  not ( SomeProperty($x,$y)
    Type($y,C.class) )
then
  Resource o = new Blank();
  insert ( new SomeProperty($x,o) );
  insert ( new Type(o,C.class) );
end
```
Towards Description Logics

Cardinality (reverse)

\[(|P(X,Y)| \leq n) \leftarrow K(X)\]

**Rule**  "Prop Cardinality = N"

// assuming should be = N

**When**

$t : Type(x,K.class)$

$c : Collection( s : size < N )$

from collect ( SomeProperty($x,y$) )

**Then**

for (int j : 0..(N-$s$)) {
    Resource $y = new$ Blank();
    insert( new SomeProperty($x,y$) );
}

**End**
Conclusions (so far)

+ A subset of DL can be built on top of Drools natively
+ More features will be added
− Notation is still verbose
“On the fly” class declaration + rule

```plaintext
rule "No Fever"
when
    $p : Patient ()
    forall ( HasRecord ($p, $r)
        HasTemperature ($r, $t)
        LessOrEqual ($t, 37) // celsius )
then
    // ...
end
```
Goal:

```plaintext
rule "No Fever"
when
    Patient(hasRecord[], hasTemp all lessOrEqual 37)
then
    // ...
end
```
Tighter Integration - Proposals

Property role /1

- Properties as “virtual fields”

Patient( type Senior.class,
hasRecord[].hasTemp all lessOrEqual 37 )

\[ P(S, O) \iff S.P \ni O \]

- Query mode : \( \exists X : p(s, X) \)?
- “Fields” are set-valued unless properties are functional
Tighter Integration - Proposals

Property role /1

- Properties as “virtual fields”
- Properties can be navigated

Patient( type Senior.class,
hasRecord[].hasTemp all lessOrEqual 37 )

\[ P(S, O) \iff S.P \ni O \]

- Query mode : \( \exists X : p(s, X) \)?
- “Fields” are set-valued unless properties are functional
Tighter Integration - Proposals

- Properties as “virtual fields”
- Properties can be navigated
- “Fields” need not be declared at compile time

Patient( type Senior.class, hasRecord[].hasTemp all lessOrEqual 37 )

\[ P(S, O) \Leftrightarrow S.P \ni O \]

- Query mode: \( \exists X : p(s, X) \)?
- “Fields” are set-valued unless properties are functional
Tighter Integration - Proposals

Property role /2

- Properties as restrictions

Patient( type Senior.class,
hasRecord[].hasTemp all lessOrEqual 37 )

- Evaluation mode: $p(s, o)$?
  - iterates over all records
Tighter Integration - Proposals

Quantifier role

- Need quantifiers in constraints

Patient( type Senior.class, hasRecord[].hasTemp all lessOrEqual 37 )

Patterns:

- getProperty all evalProperty object
- getProperty only evalProperty object
  - implicit: maxCard=1, minCard=1
- getProperty some evalProperty object
  - implicit: minCard=1
- getProperty some @[max=”,”, min=””] evalProperty object
  - explicit maxCard and/or minCard
Tighter Integration - Logic Structure

\[ \forall x \leftarrow \text{hasRecord}[1].\text{hasTemp} \land \text{lessOrEqual} \land \left< (x, 37) \right> \land \text{LessOrEqual}(x, 37) \]
Tighter Integration - Logic Structure

In general:

- Left and right operands are accessed (recursively)
- Every possible pair is tested
  - Using a direct evaluator
  - Using asserted relations
- Behaviour is conditioned by quantifier
- Natural extension for uncertainty

```
Quantifier
\land
x \leftarrow \text{Access(Property)}
\land
y \leftarrow \text{Access(Property)}
\land
f(x,y)
\land
\text{Relation}(x,y)
```
On Implementation

Two main points:
- Dynamic fields
- Node behaviour

And questions (just to cite some):
- Field mapping
  - what if $P(S, O)$ is in the WM, but $S$ is not?
- Should triples always be kept explicitly in WM?
Conclusions

- Compact syntax is more Drools-like
- Comparable expressiveness with explicit triples
- Dynamic types and fields overcome the problem of static declarations
- Need improvements on language and engine
- Implementation and Efficiency to be tested
- Better architecture for uncertain reasoning...