PSOA RuleML API : An Open-source JAVA Based Tool for Rendering PSOA RuleML/XML to Presentation Syntax

Sadnan Al Manir, Harold Boley, Alexander Riazanov

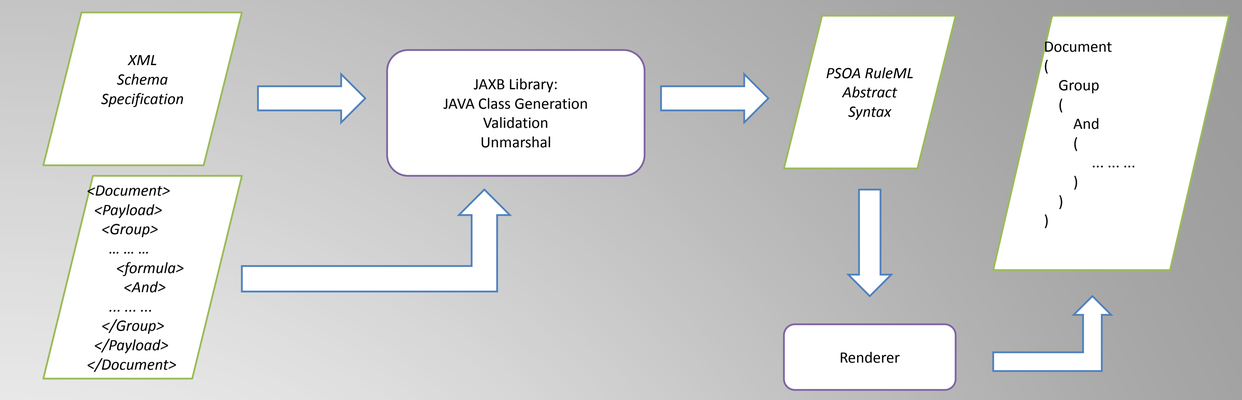
What it is

- A Syntactic parser
- Open-source tool
- Takes PSOA RuleML/XML inputs
- Output in human readable form
- Complementary translator to TPTP enables using Reasoner e.g., VampirePrime

Motivation

- A tool to write web rules and their interoperation
- Existing standards e.g., RIF, F-logic handle various aspects of terms separately
- Association of objects with functions and predicates as positional and slotted form is necessary
- Using PSOA results in the reduction of the number of terms
- Linked objects and Rules can be treated as generalized linked data

Workflow Diagram



Simulation

PSOA RuleML/XML Input

```
<!DOCTYPE Document [
   <!ENTITY ex http://example.com/>
   <!ENTITY psoa "http://www.w3.org/2011/psoa#">
   <!ENTITY xs "http://www.w3.org/2001/XMLSchema#">
]>
<Document xmlns="&psoa;">
<payload>
  <Group>
   <groupelement>
    <Subclass xmlns="&psoa;">
     <sub>
      <Const type="&psoa;iri”>ex:Student</Const>
     </sub>
     <super>
      <Const type="&psoa;iri”>ex:Person</Const>
     </super>
    </Subclass>
   </groupelement>
  </Group>
 </payload>
</Document>
```

Yes. We will make PSOA RuleML/XML Rules readable to Human!!

Output in Human Readable Presentation Syntax

```
Document (

  Prefix(  ex   <http://example.com/> )

  Group
   (

    ex:Student  ##  ex:Person

   )
  )
```

Rendered According to Grammar

PSOA RuleML Presentation Syntax Grammar

```
Document      ::= ' Document ' ' ( ' Base? Prefix* Import* Group? ' ) '
Base          ::= ' Base ' ' ( ' ANGLEBRACKIRI ' ) '
Prefix        ::= ' Prefix ' ' ( ' Name ANGLEBRACKIRI ' ) '
Import        ::= ' Import ' ' ( ' ANGLEBRACKIRI PROFILE? ' ) '
Group         ::= ' Group ' ' ( ' (RULE | Group)* ' ) '
RULE          ::= ( ' Forall ' Var+ ' ( ' CLAUSE ' ) ') | CLAUSE
CLAUSE        ::= Implies | ATOMIC
Implies       ::= (HEAD | ' And ' ' ( ' HEAD*  ' ) ') ' :- ' FORMULA
HEAD          ::= ATOMIC | ' Exists ' Var+ ' ( ' ATOMIC ' ) '
PROFILE       ::= ANGLEBRACKIRI

FORMULA       ::=  ' And ' ' ( ' FORMULA* ' ) ' |
                   ' Or ' ' ( ' FORMULA* ' ) ' |
                   ' Exists ' Var+ ' ( ' FORMULA ' ) ' |
                   ATOMIC |
                   ' External ' ' ( ' Atom ' ) '
ATOMIC        ::=  Atom | Equal | Subclass
Atom          ::=  PSOA
PSOA          ::=  TERM ' # ' TERM ' ( ' (TUPLE*  (TERM ' -> ' TERM)*) ' ) '
Equal         ::=  TERM ' = ' TERM
Subclass      ::=  TERM ' ## ' TERM
TUPLE         ::=  ' [ ' TERM ' ] '
TERM          ::=  Const | Var | Expr | ' External ' ' ( ' Expr ' ) '
Expr          ::=  PSOA
Const         ::=  ' " ' UNICODESTRING ' "^^ ' SYMSPACE | CONSTSHORT
Var           ::=  ' ? ' UNICODESTRING?
Name          ::=  NCName | ' " ' UNICODESTRING ' " '
SYMSPACE      ::=  ANGLEBRACKIRI | CURIE
```

PSOA RuleML API

JAXB Validation: Input – Schema Conformance

Generation of Classes from Schema by JAXB

```
<xs:element name="Subclass">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sub"/>
      <xs:element ref="super"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="sub">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="super">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="TERM">
    <xs:choice>
      <xs:element ref="Const"/>
        … … …
    </xs:choice>
</xs:group>
```

Schema

```
public class Subclass {

   protected Sub sub;
   protected Super _super;

   public Sub getSub() {
      return sub;
   }
   public void setSub(Sub value) {
      this.sub = value;
   }
   public Super getSuper() {
      return _super;
   }
   public void setSuper(Super value) {
      this._super = value;
   }
```

Generated Class

```
public class Sub {
                    ……
}
```

Generated Class

```
public class Super {
                    ……
}
```

Generated Class

```
private Subclass convert(
                    psoa.ruleml.parser.jaxb.Subclass subclass,
                    AbstractSyntax absSynFactory) {

                    Sub sub = subclass.getSub();

                    AbstractSyntax.Term subcls;

                    if (sub.getVar() != null)
                    {
                    subcls = convert(sub.getVar(),absSynFactory);
                    }
                    else if (sub.getConst() != null){
                    subcls = convert(sub.getConst(),absSynFactory);
                    }
                    else if (sub.getExpr() != null){
                    subcls = convert(sub.getExpr(),absSynFactory);
                    }
                    else if (sub.getExternal() != null){
                    subcls = convert(sub.getExternal(),absSynFactory);
                    }
                    else
                    throw new Error("Bad instance of Sub");

                    Super sup = subclass.getSuper();
                                … … …
                    return absSynFactory.createSubclass(subcls, supcls);
}
```

Rendering Native PSOA RuleML to Presentation Syntax

```
public interface Subclass extends Atomic {

      public Term getSubclass();

      public Term getSuperclass();

      public Set<Var> variables();

   }
```

```
Public AbstractSyntax.Subclass
      createSubclass(AbstractSyntax.Term lhs,
                     AbstractSyntax.Term rhs)
      {
         return new Subclass(lhs, rhs);
      }
```

```
public static class Subclass extends Atomic implements
      AbstractSyntax.Subclass {

   public Subclass(AbstractSyntax.Term lhs,
                   AbstractSyntax.Term rhs) {
     _lhs = lhs;
     _rhs = rhs;
   }
   public AbstractSyntax.Term getSubclass() {
     return _lhs;
   }
   public AbstractSyntax.Term getSuperclass() {
     return _rhs;
   }
   public Set<AbstractSyntax.Var> variables() {
     Set<AbstractSyntax.Var> result = _lhs.variables();
     result.addAll(_rhs.variables());
     return result;
   }
   public String toString() {
     return toString("");
   }
   public String toString(String indent) {
     String result = indent + _lhs.toString("") + " ## "
          + _rhs.toString("");
     return result;
   }
   private AbstractSyntax.Term _lhs;
   private AbstractSyntax.Term _rhs;
}
```

Conclusion

- This is the only API for PSOA RuleML
- Schema is very close to RIF, hence interoperability is possible
- Off the shelf complementary translator PSOA2TPTP allows reasoning

References

- Boley H., A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules, RuleML Europe 2011, 194-211
- The TPTP Problem Library for Automated Theorem Proving , http://www.cs.miami.edu/~tptp/
- PSOA2TPTP http://psoa2tptp.googlecode.com