# JENA API

José Nelson A. Alves

Nº 2008299

josenelson@netmadeira.com

## 1. Usando a API

Para começar a utilizar a API do Jena é necessário fazer o download do Jena em jena.sourceforge.net. Depois do download e da descompressão do ficheiro, as bibliotecas para poder usar o Jena estão localizadas no subdirectório /lib. Lá está o jar com as bibliotecas do Jena e todas as outras bibliotecas necessárias para poder usar o Jena.

### Ler uma Ontologia

O modelo ontológico fornece uma gama de classes para representar as ontologias em memória. A classe base para representar uma ontologia em memória é a *OntModel*.

- **Exemplo:** ler um ficheiro OWL

```
OntModel model;

model=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

model.read("http://max.uma.pt/~a2018601/RUD.owl");
```

No caso anterior o Jena faz o download da ontologia da Internet, esta aproximação poder trazer problemas de performance. No entanto existe a possibilidade de fazer caching de modelos (permitir usar ficheiros locais) sem ter que alterar o ficheiro OWL. Para isso temos que usar uma helper class que faz a gestão de documentos. Isto faz com que leituras subsequentes dos modelos OWL sejam feitos no disco.

- **Exemplo**: Ler um ficheiro OWL, mantendo uma cache em disco

```
OntModel m = ModelFactory.createOntologyModel();
OntDocumentManager dm = m.getDocumentManager();
dm.addAltEntry( "http://max.uma.pt/~a2018601/RUD.owl",
                "file:///c:/RUD.OWL ");
m.read("http://max.uma.pt/~a2018601/RUD.owl ");
```

Também é possível ler directamente as ontologias do disco sem usar a rede. No entanto esta aproximação traz a desvantagem de ser necessário actualizar as ontologias no disco manualmente.
A classe OntModel também permite salvar uma ontologia através do método write().

**Classes**

As classes OWL são descritas na classe OntClass. Para obter uma classes podemos usar simplesmente o método getClass(URI) do OntModel, ou podemos usar o método listClasses() para obter uma lista de todas as classes da ontologia.

A classe OntClass permite também obter todas as subclasses dessa classe através do método listSubClasses().

Exemplo: Listar subclasses da classe “Person” do RUD

```
String baseURI="http://dme.uma.pt/RUD.owl#";
OntModel model;
model=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
model.read("file:///c:/RUD.OWL");
OntClass person=model.getOntClass(baseURI+"Person");
for(Iterator i=person.listSubClasses();i.hasNext();)
{
     OntClass Class=(OntClass)i.next();
      System.out.println(Class.getURI());
}
```

**Instancias**

No Jena as instancias (ou individuals/indivíduos) das classes são representados pela classe Instance. A partir de uma classe (OntClass) é possível listar todas as suas instancias a partir do método listInstances(). Existe um método parecido na Classe OntModel mas com o nome listIndividuals().

- **Exemplo**: Listar todos os Indivíduos da ontologia

```
OntModel
schema=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
```

```
schema.read("http://max.uma.pt/~a2018601/RUDTest.owl");

for(Iterator i=schema.listIndividuals();i.hasNext();)

{

       System.out.println(((Individual)i.next()).toString());

}
```

- **Exemplo**: Listar todos os Indivíduos da classe Student:

```
OntModel schema=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

schema.read("http://max.uma.pt/~a2018601/RUDTest.owl");

OntClass Student;

Student=schema.getOntClass("http://max.uma.pt/~a2018601/RUDTest.owl#Student");

For(Iterator i= Student.listInstances();i.hasNext();)

{

       System.out.println(((Individual)i.next()).toString());

}
```

- **Exemplo**: Listar todas as classes e todas as suas instâncias:

```
OntModel
schema=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

schema.read("http://max.uma.pt/~a2018601/RUDTest.owl");

for(Iterator i=schema.listClasses();i.hasNext();)

{

     OntClass cls=(OntClass)i.next();

     System.out.println("Instances for class:"+cls.getURI());

     for(Iterator j=cls.listInstances();j.hasNext();)

    {
```

```
     System.out.println("\t"+((Individual)j.next()).getURI());
     }
}
```

## Propriedades

No Jena as propriedades são representadas pela classe OntProperty. Existem 2 tipos básicos de propriedade:

- Datatype Properties: referem-se em atributos das classes (ex: idade, nota) que são descritas sobre valores primitivos.
- Object Properties: referem-se à relação com outras classes da ontologia (ex: Teacher, Grade), basicamente são atributos que se referem a instâncias de outras classes.

As object properties subdividem-se ainda em outras como Function Property e Transitive. A classe OntProperty dispõe de métodos para poder avaliar o tipo de propriedade, como por exemplo isTransitiveProperty.

- **Exemplo**: Listar todas as classes e suas propriedades:

```
OntModel schema;
schema=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
schema.read("http://max.uma.pt/~a2018601/RUDTest.owl");
for(Iterator i=schema.listClasses();i.hasNext();)
{
      OntClass cls=(OntClass)i.next();

      System.out.println("Properties class:"+cls.getURI());
      for(Iterator j=cls.listDeclaredProperties();j.hasNext();)
      {
           System.out.println("\t"+j.next());
       }
```

```
}
```

**Criação Dinâmica de Classes e Instancia**

Também o possível criar novas classes dinamicamente e atribuir propriedades a elas. A classe OntModel fornece método createXXX() para criar , entre outros, classes e propriedades.

- **Exemplo:** Criar uma classe Project que tem uma ObjectProperty ProjectOwner que é do tipo Person (assume-se que já existe um método que lista todas as instancias e propriedades de um OntModel)

```
OntModel model;

String BaseUri=”http://max.uma.pt/~a2018601/RUDTest.owl”;

model=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

model.read(BaseUri);

OntClass cls=model.createClass(BaseUri +"#Project");

ObjectProperty p=model.createObjectProperty(BaseUri+"#ProjectOwner");

p.setRange(model.getResource(BaseUri+"#Person"));

p.setDomain(cls);

printClassInfo(model);
```

Podias também fazer um método para listar as instâncias e valores das suas propriedades

- **Exemplo**: Listar instâncias e valores das suas propriedades

```
  private void printInstancesInfo(OntModel model) {

              for(Iterator i=model.listClasses();i.hasNext();)
```

```
            {
              OntClass c=(OntClass)i.next();
              if(c.isAnon()) continue;
              System.out.println("Instances for Class:
"+c.getURI());
              for(Iterator j=c.listInstances();j.hasNext();)
              {
                Individual next=(Individual) j.next();
                System.out.println("\t"+next);
                for(Iterator
t=c.listDeclaredProperties();t.hasNext();)
                {
                  OntProperty p=(OntProperty)t.next();
                  System.out.print("\t\t"+p.getURI()+": ");
                  System.out.print(next.getPropertyValue(p)+"\n");
                }
              }
            }
          }
```

Agora podemos então criar instâncias dinamicamente.

- **Exemplo:** criar uma instância JonhDoe do tipo person, criar um Project SW e associar o Project owner como sendo o JonhDoe.

```
String baseURI="http://max.uma.pt/~a2018601/RUDTest.owl";
OntModel model;
model=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
model.read(baseURI);
```

```
OntClass cls=model.createClass(baseURI+"#Project");

ObjectProperty
p=model.createObjectProperty(baseURI+"#ProjectOwner");

Resource person=model.getResource(baseURI+"#Person");

p.setRange(person);

p.setDomain(cls);

Individual
jonhdoe=model.createIndividual(baseURI+"#JonhDoe",person);

Individual
SWProject=model.createIndividual(baseURI+"#SemWeb",cls);

SWProject.setPropertyValue(p,jonhdoe);



printClassInfo(model);

System.out.println("---------------------------------");

printInstancesInfo(model);
```

Criar propriedades DataTypeProperty ainda é mais simples.

- **Exemplo:** Seguindo o exemplo anterior adicionar a propriedade Project Released Date à classe projecto, seria preciso acrescentar as seguintes linhas:

```
…

DatatypeProperty pdate;

pdate=model.createDatatypeProperty(baseURI+"ProjectReleaseDate");

//Indicar a que classe pretence essa propriedade

pdate.setRange(cls);

…

SWProject.addProperty(pdate,"4/10/05");
```

```
…
```

**Inferência**

Como já descrito anteriormente a classe central à inferência é o Reasoner. È a partir deste que se pode extrair conhecimento da ontologia.

- **Exemplo**: Inferência sobre um modelo (*model*) e uma instancia (*data*)

```
OntModel
model=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

OntModel
data=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

model.read("http://max.uma.pt/~a2018601/RUD.owl");

data.read("http://max.uma.pt/~a2018601/RUDI.owl");

Reasoner reasoner = ReasonerRegistry.getOWLReasoner();

reasoner=reasoner.bindSchema(model);

InfModel infmodel=ModelFactory.createInfModel(reasoner,data);
```

Agora podemos obter informações acerca da ontologia como por exemplo obter todos os recursos relativos à cadeira de Semantic Web, definida com o ID SW.

- **Exemplo**:

```
Resource res;

String URN=” http://max.uma.pt/~a2018601/RUDI.owl”;

res= infmodel.getResource(URN+"#SW");

System.out.println("Semantic Web *:");

printStatements(infmodel, res, null, null);


public void printStatements(Model m, Resource s,
```

```
Property p, Resource o) {

           for (StmtIterator i =
m.listStatements(s,p,o); i.hasNext(); ) {

             Statement stmt = i.nextStatement();

             System.out.println(" - " +
PrintUtil.print(stmt));

           }

}
```

No entanto não é necessário que tenhamos o schema e os indivíduos em ficheiros separados, simplesmente deveríamos retirar a parte do binding do schema. No entanto por razões de performance é aconselhado ter as ontologias com os dados e o esquema separados.

- **Exemplo**: Validar uma Ontologia

```
String BaseURI="http://max.uma.pt/~a2018601/RUDTest.owl";

OntModel model;

model=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

model.read(BaseURI);



Reasoner reasoner = ReasonerRegistry.getOWLReasoner();

InfModel modelInf =
ModelFactory.createInfModel(reasoner,model);

ValidityReport vrp1 = modelInf.validate();

if (vrp1.isValid())

{

             System.out.println("Valid OWL");

}
```

```
else
{
    System.out.println("Not valid OWL");
    for (Iterator i = vrp1.getReports();i.hasNext();)
    {
          System.out.println(" - " + i.next());
    }
}
```

O Reasoner do Jena não se limita a obter informação relacionada com os recursos, também é possível obter informação baseada em regras definidas pelo utilizador. Essas regras têm que ser definidas em linguagem de 1ª ordem, tipo prolog. O Jena vem com um reasoner genérico que depois pode ser implementado por terceiros. Esse reasoner (generic rule reasoner) tem 2 motores de inferência:

- Forward Chaining Engine: verifica cada regra para avaliar se cada uma é válida, se sim executa a regra obtendo novos dados e continua aplicando as regras tendo em conta os novos dados.
- Backward Chaining Engine: verifica o objectivo e verifica posteriormente que regras satisfazem o objectivo (similar ao Prolog).

- **Exemplo: Para o objectivo: "A is C" , como os dados "A is B", "B is C"**

| Regra | Forward Chaining Engine | Backward Chaining Engine |
|---|---|---|
| (?A p ?B) → (?A is ?C) | Verifica a regra (?A p ?B) e não chega ao objectivo | Obtém todas as regras que satisfazem (?A is ?C) e verifica que a condição não é verificada |
| (?A is ?B) (?B is ?C) → (?A is ?C) | Verifica ambas as condições e obtém o objectvo. | Verifica que as condições fazem parte das regras que satisfazem o objectivo. |

È também possível adicionar outros reasoners ao Jena através a interface DIG (Discription Logic Reasoner Interface). A seguinte imagem mostra como essa interface funciona.

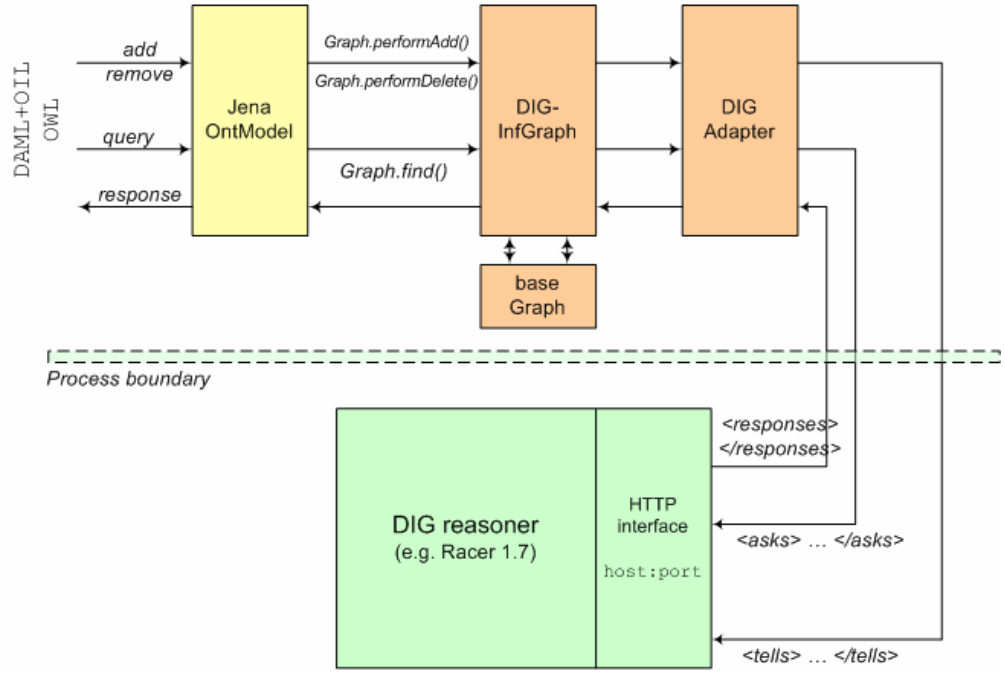


**Fig. 1.** Arquitectura dos DIG reasoners.

Actualmente existem 3 reasoners que podem ser usados externamente com o Jena através a interface DIG: o Pellet, o Racer e o FaCT.

O Generic Rule Reasoner é o reasoner mais complexo que vem com o Jena. Este permite fazer reasoning através das regras que foram descritas anteriormente. Essas regras funcionam com base em triplos (Sujeito Propriedade Valor). As opções que podem ser usadas nessas regras são:

| Construtor | Usado em |
|---|---|
| rdfs:subClassOf, rdfs:subPropertyOf, rdf:type | all |
| rdfs:domain, rdfs:range | all |
| owl:intersectionOf | all |
| owl:unionOf | all |
| owl:equivalentClass | all |
| owl:disjointWith | full, mini |
| owl:sameAs, owl:differentFrom, owl:distinctMembers | full, mini |
| Owl:Thing | all |
| owl:equivalentProperty, owl:inverseOf | all |

| | |
|---|---|
| owl:FunctionalProperty, owl:InverseFunctionalProperty | all |
| owl:SymmeticProperty, owl:TransitiveProperty | all |
| owl:someValuesFrom | full, (mini) |
| owl:allValuesFrom | full, mini |
| owl:minCardinality, owl:maxCardinality, owl:cardinality | full, (mini) |
| owl:hasValue | all |

**Persistência**

Além de poder usar ficheiros para armazenar as ontologias o Jena permite armazenar as ontologias em bases de dados relacionais. Actualmente o Jena apenas suporta as bases de dados: MySQL, Oracle e PostgreSQL.

Para criar um modelo persistente em bases de dados usamos a classe model factory:

```
ModelFactory.createModelRDBMaker(conn).createModel()
```

Em que "conn" representa a ligação à base de dados.

- **Exemplo**: Salvar um modelo existente no disco numa base de dados

```
Class.forName("com.mysql.jdbc.Driver");

String BaseURI="http://max.uma.pt/~a2018601/RUDTest.owl";

DBConnection conn;

Conn=new
DBConnection("jdbc:mysql://localhost/RUDTest","semweb","1234
65","MySQL");

ModelMaker maker=ModelFactory.createModelRDBMaker(conn);

Model base=maker.createModel(BaseURI,false);

base.begin();

base.read(BaseURI);

base.commit();
```

- **Exemplo**: Lêr um modelo a partir de uma base de dados:

```
Class.forName("com.mysql.jdbc.Driver");

String BaseURI="http://max.uma.pt/~a2018601/RUDTest.owl";

DBConnection conn=new
DBConnection("jdbc:mysql://localhost/RUDTest","semweb","1234
65","MySQL");

ModelMaker maker=ModelFactory.createModelRDBMaker(conn);

Model base=maker.createModel(BaseURI,false);

OntModel model;

model=ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM,
base);
```