Jena – Um *Framework Web* Semântico em *Java*

Angelo Augusto Frozza, Rodrigo Gonçalves

{frozza, rodrigog}@inf.ufsc.br
Universidade Federal de Santa Catarina – UFSC
Florianópolis – Santa Catarina

Introdução

O *Jena* é um projeto que se originou dentro do núcleo de pesquisa em *Web Semântica* da HP. Seu objetivo é proporcionar um *framework* na linguagem *Java* que dê suporte à utilização da *Web Semântica* por qualquer aplicativo capaz de utilizá-lo. Este suporte inclui recursos para manipulação de RDF, RDFS, OWL e DAML+OIL.

É um projeto de código aberto (*open source*), gratuito e disponível na *Web* no endereço `http://jena.sourceforge.net`. No *site* está disponível uma ampla documentação, assim como também existe uma lista de discussão (cujo endereço está disponível no *site*) para aqueles que necessitarem maiores informações.

O objetivo deste estudo de caso é conhecer o *Jena*, suas características principais e como ele pode ser utilizado para o desenvolvimento de aplicações que usem ontologias representadas na linguagem OWL.

Neste estudo, a motivação surge a partir da possibilidade de criar, carregar, manipular e salvar ontologias em diversos formatos, entre eles OWL, por meio do *Jena*. As ontologias podem estar disponibilizadas localmente ou em um endereço na *Web*. Ontologias criadas podem ser salvas tanto em arquivos em disco como em bancos de dados, facilitando a persistência de ontologias extensas. Recursos para recuperação de ontologias embutidas em uma ontologia também estão disponíveis.

A versão 1 do *Jena* dispunha suporte apenas para DAML+OIL e RDF, pois na época o OWL não havia sido definido. Esta versão também dispunha de recursos muito limitados para o uso de *reasoners* e inferências. Com a versão 2 estes recursos foram expandidos, assim como o suporte a OWL incluído.

O *Jena* trabalha de forma transparente em relação à linguagem adotada para representar a ontologia. De acordo com a linguagem, determinados recursos são habilitados ou não, porém, a forma de manipular e criar ontologias é a mesma para qualquer linguagem de representação.

Criação de Ontologias

A criação de uma ontologia é realizada por um gerenciador denominado `ModelFactory`. Em sua forma mais simples, criar uma ontologia no *Jena* resume-se a:

```
OntModel ontologia = ModelFactory.createOntologyModel();
```

Esta forma cria uma ontologia sem associação a uma linguagem específica para representação. Porém, em geral, criam-se as ontologias já segundo um modelo de representação, para que se possa compartilhar as mesmas posteriormente. Neste caso, basta especificar a URI da ontologia ao criar a mesma, da seguinte forma:

```
OntModel ontologia = ModelFactory.createOntologyModel (URIdaOntologia);
```

Para facilitar, uma classe denominada `ProfileRegistry` contém as URIs das representações de ontologias suportadas pelo *Jena*, ficando a criação de uma ontologia dessa forma:

```
OntModel ontologia = ModelFactory.createOntologyModel (ProfileRegistry.
OWL_LITE_LANG);
```

As seguintes constantes estão definidas em `ProfileRegistry`:

Constante	Representação
DAML_LANG	DAML+OIL
OWL_DL_LANG	OWL-DL
OWL_LANG	OWL-*Full*
OWL_LITE_LANG	OWL-*Lite*
RDFS_LANG	RDFS

Além da linguagem de representação, outras configurações em relação a uma ontologia criada podem ser feitas pela classe `OntModelSpec`. Cada ontologia associa-se a um representante desta classe e este permite manipular aspectos como o *reasoner* adotado e o tratamento de documentos complexos (documentos que referenciam outros documentos).

Carga de ontologias

O *Jena* disponibiliza várias formas de carregar uma ontologia (a partir de um arquivo, de um endereço na *Web* ou de um *stream* de dados). Para carregar uma ontologia, basta instanciar um objeto `OntModel` e requisitar ao mesmo que recupere a ontologia utilizando o método `read` do mesmo. O *Jena* disponibiliza as seguintes versões deste método, conforme a origem dos dados da ontologia:

```
read( String url )
```

```
read( Reader reader, String base )
read( InputStream reader, String base )
read( String url, String lang )
read( Reader reader, String base, String Lang )
read( InputStream reader, String base, String Lang )
```

Sobre a carga de ontologias, um aspecto que merece destaque é o tratamento de referências a outras ontologias contidas na ontologia sendo carregada. Para definir se as ontologias associadas devem ou não ser carregadas, pode-se utilizar um `DocumentManager`, que é uma classe responsável pela manipulação de documentos representando ontologias.

Para evitar a carga das ontologias associadas, basta chamar o método `setProcessImports` do `DocumentManager` da ontologia em questão, passando como parâmetro uma constante *false*.

```
OntModel m = ModelFactory.createOntologyModel();
OntDocumentManager dm = m.getDocumentManager();
dm.setProcessImports(false);
```

Pode-se também evitar apenas a carga de determinadas ontologias associadas, bastando para tal passar suas URIs ao método `addIgnoreImports()`.

Manipulando classes de uma ontologia

Todas as entidades de uma ontologia que representam valores em uma ontologia têm como classe base `OntResource`. Cada recurso de uma ontologia tem um conjunto de atributos básicos que são:

Atributo	Descrição
`versionInfo`	A versão do recurso
`comment`	Um comentário sobre o recurso
`label`	Uma identificação do recurso
`seeAlso`	Onde procurar informações adicionais
`isDefinedBy`	Um local na *Web* que possui uma definição do recurso
`sameAs`	Indica um recurso que é equivalente
`differentFrom`	Indica um recurso que é distinto

Para cada um destes atributos, métodos no formato `add<atributo>`, `set<atributo>`, `get<atributo>`, `has<atributo>` e `remove <atributo>` permitem a manipulação destes.

As classes de uma ontologia são representadas por objetos da classe `OntClass`. Para recuperar uma classe, informa-se ao *Jena* a URI da mesma. Existem duas maneiras no *Jena* de recuperar uma classe:

```
String camNS = "http://www.xfront.com/owl/ontologies/camera/#";
Resource r = m.getResource( camNS + "Camera" );
OntClass camera = (OntClass) r.as( OntClass.class );
```

Ou:

```
OntClass camera = m.getOntClass( camNS + "Camera" );
```

No caso de não encontrar a classe informada, o *Jena* retorna a constante *null*.

Para criar classes, utiliza-se o método `createClass()`, informando ou não ao método a URI da classe. Caso uma URI seja informada, o *Jena* verifica se a classe já existe ou não. Em caso positivo, apenas retorna a mesma, sem criar uma nova classe. Se não é informada uma URI o *Jena* cria uma classe anônima (útil em documentos complexos).

```
OntClass pinCamera = m.createClass( camNS + "PinholeCamera" );
```

Da mesma forma que os recursos, as classes no *Jena* possuem alguns atributos básicos que podem ser manipulados de uma forma padrão, através de métodos `add<atributo>`, `get<atributo>`, `list<atributo>` e `remove <atributo>`. Alguns dos atributos básicos das classes são:

Atributo	Descrição
`subClass`	Indica uma classe subclasse da classe
`superClass`	Indica uma superclasse da classe
`equivalentClass`	Indica uma classe equivalente
`disjointWith`	Indica uma classe com a qual a classe não tem instâncias comuns.

Para listar as subclasses de uma classe, fazemos:

```
OntClass camera = m.getOntClass( camNS + "Camera" );
for (Iterator i = camera.listSubClasses(); i.hasNext(); ) {
  OntClass c = (OntClass) i.next();
  System.out.print( c.getLocalName() + " " );
}
```

Manipulando atributos de uma ontologia:

No *Jena*, as propriedades de uma ontologia são representadas pela classe `OntProperty`. Da mesma forma que para os recursos, as propriedades também possuem um conjunto de atributos básicos, acessíveis por métodos padrões de `get, set, list, remove` etc.

Atributo	Descrição
`subProperty`	Uma subpropriedade da propriedade
`superProperty`	Uma superpropriedade da propriedade
`domain`	O domínio desta propriedade
`range`	Os valores possíveis desta propriedade
`equivalentProperty`	Uma propriedade equivalente
`Inverse`	A propriedade inversa a esta

Com o método `listDeclaredProperties()`, pode-se obter as propriedades de uma dada classe (`OntClass`). Nesta listagem, pode-se incluir propriedades das superclasses ou não, conforme um parâmetro passado.

Um exemplo de manipulação de propriedades em *Jena*:

```
OntModel newM = ModelFactory.createOntologyModel();
OntClass Camera = newM.createClass( camNS + "Camera" );
OntClass Body = newM.createClass( camNS + "Body" );

ObjectProperty part = newM.createObjectProperty( camNS + "part" );
ObjectProperty body = newM.createObjectProperty( camNS + "body" );

body.addSuperProperty( part );
body.addDomain( Camera );
body.addRange( Body );
```

Como suporta OWL, *Jena* define dois tipos básicos de propriedades: `ObjectProperty` e `DatatypeProperty`. O primeiro tipo suporta objetos em seu *range*, enquanto que o segundo tipo suporta apenas literais (*strings*, números etc.).

Além destes tipos básicos de propriedades, *Jena* define outros tipos mais avançados como `TransitiveProperty`, `FunctionalProperty`, `SymmetricProperty` e `InverseFunctionalProperty`. Uma propriedade simples (`ObjectProperty`) pode ser acessada como qualquer um destes tipos através de uma simples chamada a um método:

```
public TransitiveProperty asTransitiveProperty();
public FunctionalProperty asFunctionalProperty();
public SymmetricProperty asSymmetricPropery();
public InverseFunctionalProperty asInverseFunctionalProperty();
```

Além disto, pode-se converter o tipo de uma propriedade, também através de uma simples chamada a um método:

```
public TransitiveProperty convertToTransitiveProperty();
public FunctionalProperty convertToFunctionalProperty();
public SymmetricProperty convertToSymmetricPropery();
public InverseFunctionalProperty convertToInverseFunctionalProperty();
```

Restrições em propriedades

Além das restrições já apresentadas, existem outros tipos de restrições associadas às propriedades suportadas pelo *Jena*:

Restrição	Descrição
`has value`	A propriedade tem um valor exato
`All values from`	Classe da qual os valores da propriedade devem ser
`Some values from`	Pelo menos um valor da propriedade deve ser de uma certa classe
`Cardinality`	A propriedade tem exatamente *n* valores
`Min Cardinality`	A propriedade tem pelo menos *n* valores
`Max Cardinality`	A propriedade tem no máximo *n* valores

No *Jena*, pode-se obter uma restrição existente através de sua URI, assim como definir uma:

```
Restriction r = m.getRestriction( rURI );
```

Ou:

```
OntProperty p = m.createOntProperty( ns + "p" );
Restriction anonR = m.createRestriction( p );
```

A restrição criada aqui é anônima, ou seja, não tem uma característica de restrição definida. Para estabelecer uma restrição, utilizam-se os métodos `asTIPODARESTRICAO` (que retorna uma nova restrição) ou `convertTIPODARESTICAO` (que transforma a restrição no tipo desejado):

```
OntClass c = m.createClass( Ns + "C" );
AllValuesFromRestriction avf = anonR.convertToAllValuesFromRestriction(
c );
```

Pode-se, também, criar uma restrição já com o tipo definido:

```
OntClass c = m.createClass( Ns + "C" );
ObjectProperty p = m.createObjectProperty( Ns + "p" );
```

```
AllValuesFromRestriction rst = m.createAllValuesFromRestriction( null,
p, c );
```

Intersecção, união e complemento de classes

Para determinadas classes, as mesmas podem ser tanto de um tipo *A* ou de um tipo *B*, ou ser de um tipo *A* e de um tipo *B* ou ainda não ser de um tipo *A*. Estes requisitos podem ser interpretados como operações de conjunto (união de *A* e *B*, intersecção de *A* e *B* e complemento de *A* respectivamente). O *Jena* disponibiliza recursos para definir estas características:

```
OntClass Window = m.createClass( camNS + "Window" );
Individual throughTheLens = m.createIndividual( camNS +
"ThroughTheLens", Window );

ObjectProperty viewfinder = m.createObjectProperty( camNS +
"viewfinder" );

HasValueRestriction viewThroughLens =
m.createHasValueRestriction( null, viewfinder, throughTheLens );

OntClass Camera = m.createClass( camNS + "Camera" );

IntersectionClass SLR = m.createIntersectionClass( camNS + "SLR",
m.createList( new RDFNode[] {viewThroughLens, Camera} ) );
```

Manipulação de indivíduos

Para criar indivíduos em uma ontologia, basta utilizar o método `createIndividual()` da classe à qual ele será integrado. Na criação, o indivíduo pode ou não ser identificado por um nome.

Meta-dados de uma ontologia

Um documento que representa uma ontologia pode ser manipulado no *Jena*, permitindo definir atributos referentes a sua versão, estrutura e requisitos. As propriedades disponibilizadas são:

Propriedade	Descrição
`backwardCompatibleWith`	Compatibilidade com versões anteriores da ontologia
`incompatibleWith`	Incompatibilidade com versões anteriores da ontologia
`priorVersion`	Uma versão anterior desta ontologia
`Imports`	Ontologias importadas

Inferência

Através do uso de *reasoners* é possível fazer inferências sobre uma ontologia no *Jena*. Um *reasoner* nada mais é que uma entidade dentro do *framework* do *Jena* capaz de responder a determinados questionamentos e afirmações feitos em relação a uma Ontologia.

O *reasoner* associado a uma ontologia é definido através do método `setReasoner` da mesma. Os *reasoners* são criados pela classe `ReasonerFactory`, e na classe `ReasonerRegistry` estão definidos alguns *reasoners* padrões disponibilizados.

No *Jena* já estão disponibilizados os seguintes *reasoners*:

Reasoner	**Descrição**
`Transitive`	Implementa a transitividade de propriedades simétricas como `subClassOf`
`RDFS`	Implementa um subconjunto das implicações do RDFS
`OWL, OWL Mini, OWL Micro Reasoners`	Implementa um subconjunto (*incomplete*) das inferências da OWL/*Lite*
`Generic rule reasoner`	Um *reasoner* que aceita regras personalizadas, para uso geral.

Referente ao *reasoner* para OWL, existem três versões do mesmo: *full, mini* e *micro*. O *full* apresenta a maior quantidade de recursos, porém menor velocidade, já o *micro* apresenta a maior velocidade, porém menos recursos.

Os construtores da OWL suportados pelos *reasoners* são:

Construtor	***Reasoners***
`rdfs:subClassOf, rdfs:subPropertyOf, rdf:type`	Todos
`rdfs:domain, rdfs:range`	Todos
`owl:intersectionOf`	Todos
`owl:unionOf`	Todos
`owl:equivalentClass`	Todos
`owl:disjointWith`	*Full, mini*
`owl:sameAs,`	*Full, mini*

owl:differentFrom, owl:distinctMembers	
Owl:Thing	Todos
owl:equivalentProperty, owl:inverseOf	Todos
owl:FunctionalProperty, owl:InverseFunctionalProperty	Todos
owl:SymmeticProperty, owl:TransitiveProperty	Todos
owl:someValuesFrom	*Full, mini*
owl:allValuesFrom	*Full, mini*
owl:minCardinality, owl:maxCardinality, owl:cardinality	*Full, mini*
owl:hasValue	Todos

A utilização de um *reasoner* baseia-se em instanciar um *modelo de informações* sobre uma determinada ontologia e então requisitar um conjunto de afirmações sobre um determinado recurso e um determinado construtor:

```
OntModel ontologia = carregarOntologia();
InfModel modelo = ModelFactory.createInfModel(ontologia.getReasoner(),
ontologia);
Resource ie = modelo.getResource(prefix + "InstituicaoDeEnsino");
Resource ufsc = modelo.getResource(prefix + "UFSC");

if (modelo.contains(ufsc, RDF.type, ie)) {
      System.out.println("UFSC é uma instituição de ensino");
}
```

No exemplo, faz-se uma consulta à ontologia pelo *reasoner*, perguntando se a entidade UFSC é uma instituição de ensino.

Considerações finais

O *Jena* mostrou-se uma ferramenta bastante poderosa para o suporte a ontologias. Acrescenta-se ainda o fato de ser desenvolvida na linguagem *Java*, que é uma das linguagens de programação mais usadas comercialmente, com um grande número de recursos.

O *Jena* possui uma documentação bastante farta. Em grande parte isto deve-se ao fato de sua origem ser nos laboratórios da HP, que até hoje contribuem no desenvolvimento e na geração de documentação.

Por outro lado, o *framework* apresenta-se como uma ferramenta complexa de ser usada por quem ainda não tem o domínio dos conceitos relacionados à ontologias, nem experiência no desenvolvimento das mesmas.

Uma pequena aplicação foi desenvolvida junto com este estudo de caso, a fim de ajudar no melhor entendimento do *Jena*. Esta aplicação introduz cinco operações básicas com o Jena:

- Carga de ontologias do disco local
- Busca por entidades que possuam determinado atributo (*slot* no jargão do Protegé)
- Listagem de uma determinada propriedade em todos os objetos da ontologia
- Criação de uma ontologia simples com herança de classes
- Uso de reasoners para recuperar afirmações sobre uma ontologia

No desenvolvimento desta aplicação percebeu-se o quanto um bom entendimento sobre ontologias e seus conceitos é fundamental para o uso adequado do Jena. Sua documentação não tem em mente um usuário leigo, mas sim um usuário com bons conhecimentos sobre ontologias.

Referente à documentação, convém ainda destacar que existem alguns tutoriais que facilitam um entendimento inicial de como funciona o *framework*.

Bibliografia

Esta introdução ao *Jena* e OWL foi baseada nos tutoriais *Jena 2 Inference support* e *Jena 2 Ontology API* disponíveis na documentação do *Jena*.