UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA – INE
CIÊNCIA DA COMPUTAÇÃO

REENGENHARIA DO *ONTOCOVER*: UMA BIBLIOTECA JAVA PARA MANIPULAR ONTOLOGIAS EM APLICAÇÕES DA WEB SEMÂNTICA.

Florianópolis, 2007.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA – INE
CIÊNCIA DA COMPUTAÇÃO

REENGENHARIA DO *ONTOCOVER*: UMA BIBLIOTECA JAVA PARA MANIPULAR ONTOLOGIAS EM APLICAÇÕES DA WEB SEMÂNTICA.

Trabalho de Conclusão do Curso apresentado como um dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Autor: Marcelo Oliveira de Moraes
Orientação: Professor Renato Fileto

Florianópolis, 2007.

Marcelo Oliveira de Moraes

REENGENHARIA DO *ONTOCOVER*: UMA BIBLIOTECA JAVA PARA MANIPULAR ONTOLOGIAS EM APLICAÇÕES DA WEB SEMÂNTICA.

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Renato Fileto

Banca examinadora:
Fernando Ostuni Gauthier
Ronaldo dos Santos Mello

Florianópolis, 2007.

ÍNDICE

Lista de Figuras

Resumo

Este trabalho relata alguns estudos em Web Semântica, principalmente ferramentas, e a utilização dos conhecimentos obtidos na manutenção e documentação da biblioteca Java, denominada *OntoCover*, para manipulação de ontologias em aplicações baseadas em conhecimento.

Palavras-chave: Web semântica, ontologias, visões de ontologias, coberturas ontológicas, *OntoCover*.

Proceedings of the XII SIBGRAPI (October 1999) 101-104

Abstract

This work approaches some studies about Semantic Web, mainly the tools, and the use of knowledge gotten from maintenance and documentation of the Java library, named OntoCover, to handle the ontologies into knowledge based applications.

Keywords: semantic web, ontology, ontology views, ontological coverages, *OntoCover*.

1. INTRODUÇÃO

A Web Semântica [2, 3] é uma área de pesquisa em expansão que visa estender o papel dos computadores no suporte a diversas atividades humanas. Ela propõe a formalização do conhecimento na forma de ontologias de modo a permitir a agentes e outros programas catalogar, recuperar, selecionar e compor recursos (dados e serviços) publicados na Web, de maneira mais inteligente e precisa.

O OntoCover é uma biblioteca para manipulação de ontologias, com funcionalidades para extrair visões de ontologias, visualizar e navegar nessas visões, além de especificar e comparar eficientemente anotações semânticas baseadas em alguma dessas visões de uma ontologia. Ele constitui uma contribuição para o desenvolvimento de aplicações sobre a Web semântica.

O OntoCover foi idealizado e desenvolvido no Instituto de Computação da Unicamp em 2003, pelo professor Renato Fileto com o auxílio do bolsista Lauro Ramos Venâncio na implementação. Com o passar do tempo foram surgindo novas versões de pacotes de software utilizados pelo OntoCover, particularmente a linguagem Java e a biblioteca Jena [8]. O OntoCover também não apresentava documentação adequada, fato esse que dificultava o seu uso e a implementação de melhorias.

Este trabalho relata as atualizações e ajustes efetuados no OntoCover, incluindo manutenções, extensões de funcionalidade e documentação. A atualização das versões do Java e do Jena envolveu várias manutenções do código fonte e propiciou melhorias de robustez e desempenho. A documentação gerada inclui o diagrama de classes do projeto, documentação de cada classe na forma de JavaDocs e manual do usuário. Isto promove o reuso do código da biblioteca, facilita a incorporação de melhorias e indica a maneira que o usuário deve manipulá-la para a criação de aplicações na Web Semântica.

Uma outra contribuição deste trabalho foi a implantação de uma nova funcionalidade no OntoCover: suporte a buscas na ontologia através da linguagem SPARQL, padrão da W3C para manipular RDF [9]. Com isso é possível manipular informação específica extraída da ontologia. Por exemplo, fica mais fácil identificar se o valor de uma propriedade de um recurso (e.g. seu nome) aparece em outros pontos da ontologia. Se isso acontecer, é caracterizada uma ambigüidade para aquele nome. Uma forma de resolver tais ambigüidades é associar o recurso ao conceito que melhor expresse a intenção do usuário. A identificação dos possíveis conceitos (e.g. país, estado, time de futebol) que podem ser associados a um recurso com um certo valor na propriedade nome (e.g., São Paulo) também pode ser feita através de expressões de consulta em SPARQL.

O trabalho está organizado em 7 capítulos. O capítulo 2 apresenta o contexto no qual se inclui a biblioteca OntoCover, discutindo diversos conceitos e aplicações da Web semântica. O capítulo 3 ilustra as visões e a exibição da ontologia pelo OntoCover e descreve as interações entre os módulos presente na arquitetura. O capítulo 4 apresenta o estado da implementação encontrada antes de iniciar o trabalho e o que foi feito para sanar os problemas relatados. O capítulo 5 apresenta o resultado das alterações. No capitulo 6 apresenta a pesquisa de trabalhos relacionados e também trabalhos que auxiliem o OntoCover. Finalmente o capítulo 7 apresenta a conclusão do trabalho e sugestões para trabalhos futuros.

2. FUNDAMENTOS

2.1. WEB SEMÂNTICA

Tim Berners-Lee [2] criador da linguagem HTML e líder na criação do consórcio mundial W3C (*Word Wide Web Consortium*) no *Massachusetts Institute of Technology* (MIT), foi o idealizador da Web Semântica. Ela tem como objetivo estender a Web atual possibilitando a compreensão da informação pelas máquinas. Para isso ocorrer é necessário: o desenvolvimento de linguagens que atribuam significado aos dados e tecnologias que permitam o processamento baseado nesse significado.

A Web Semântica sugere que a informação na Web seja descrita através de anotações semânticas, que indiquem o significado da informação. Mecanismos baseados em regras e inferência sobre as ontologias e essas anotações devem viabilizar a construção de sistemas e agentes mais inteligentes para automatizar a execução de sofisticadas tarefas, tais como planejar uma viagem, buscando as melhores opções em passagens aéreas, translados e hotéis, segundo preferências do usuário e restrições de tempo e orçamento.

Dentre as técnicas e fundamentos necessários para concretizar os objetivos da Web semântica ontologias têm um papel fundamental. Ontologias atuam no cerne do propósito da Web Semântica, uma vez que a permitem descrever e atribuir significado à informação. A sua utilização favorece o compartilhamento e o reuso da informação.

2.2. ONTOLOGIAS

Uma ontologia é uma conceitualização explícita, formal e compartilhada de uma área de conhecimento [1]. Em outras palavras, uma ontologia é um modelo que possui conceitos, propriedades, relações, funções e axiomas, explicitamente definidos de uma forma que permitam a manipulação pelo computador.

O termo ontologia é oriundo da filosofia, onde designa o estudo das teorias sobre a natureza da existência. Pesquisadores dos ramos da inteligência artificial e Web adotaram o termo para designar uma estrutura que define formalmente relações de significado entre termos [2].

Entre os benefícios de se utilizar ontologias estão: [4]

- **Compartilhamento:** uma ontologia promove uma compreensão comum de um domínio de conhecimento.
- **Reuso**: o uso de definições explícitas e formais facilita a manutenção do conhecimento, permitindo o fácil entendimento por parte dos usuários e facilitando a reutilização da ontologia, ou de parte dela.
- **Estruturação da informação**: permite a captura da semântica dos dados e seu processamento automático, gerando conhecimento para os humanos.
- **Interoperabilidade**: permite que diferentes sistemas computacionais compartilhem dados e informações.
- **Confiabilidade**: uma representação formal torna possível uma automatização consistente e mais confiável.

A figura 1 ilustra uma ontologia na forma de um grafo, onde os vértices representam recursos e as arestas representam propriedades desses recursos. Os valores das propriedades podem ser outros recursos (apontados pelas arestas) ou literais. Na figura 1, os recursos representados por elipses referem-se a conceitos e os recursos representados por retângulos referem-se a instâncias de conceitos. As instâncias denominadas “*Strawberry*” e “*Potato*” têm propriedades com valores literais, tais como: “*Color*”, “*Seedless*”, “*Flavor*” e “*Climate*”.

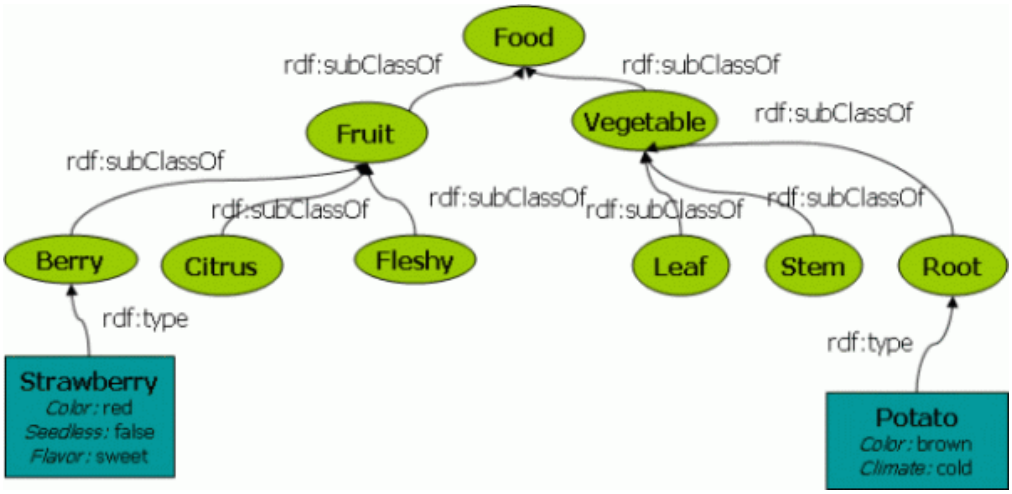


Figura 1 – Exemplo de Ontologia

fonte: http://www.sei.cmu.edu/isis/guide/gifs/fruit-ontology.gif

Através da navegação nesse grafo é possível extrair conhecimento. Por exemplo: *Strawberry* é do tipo "*Berry*" que é uma subclasse de "***Fruit***" que por sua vez é subclasse de "***Food***" (comida). Caso houvesse outro recurso com o nome "***Strawberry***" (e.g., um peixe vermelho com pintinhas pretas) relacionado a outro recurso que não "***Berry***" (e.g., personagem de desenho animado) seria possível distinguir os dois recursos com o nome "***Strawberry***", através de inferências na ontologia.

2.3 REPRESENTAÇÃO DA ONTOLOGIA.

Uma ontologia usualmente é representa pelos padrões RDF [9] e OWL [20]. As próximas seções tratam esses dois padrões.

2.3.1 RDF/ RDF SCHEMA.

A linguagem RDF [9] (Resource Description Framework) foi criada pelo consórcio W3C *(World Wide Web Consortium)* [5] para a representação de conhecimento, através da associação de recursos com suas propriedades e valores dessas propriedades. Um recurso é identificado através do seu URI[6]. A definição de uma propriedade também pode ser referenciada por um URI, assim como o valor de uma propriedade, que também pode ser um valor literal, como uma string ou número..

A construção básica do RDF é a tripla da forma (Sujeito, Predicado, Objeto), entendida como "Um **sujeito** (**recurso**) possui um **predicado** (**propriedade**) com um determinado valor definido por um **objeto (outro recurso ou um literal).**" A figura 2 ilustra um conjunto de tais triplas interligadas, como por exemplo:

- **Sujeito**: "http://www.w3c.org/People/EM/contact#me"
- **Predicado**: "http://www.w3c.org/2000/10/swap/pim/contact#personalTitle"
- **Objeto**: "Dr."

Figura 2 – Exemplo gráfico de RDF,

fonte: http://www.w3.org/TR/2003/WD-rdf-primer-20030123/

Note que na figura 2 há objetos que são literais, como a string “Dr.” representando a propriedade título no exemplo acima, assim como objetos detonados por URIs, tais como:

- “mailto:em@w3.org”
- “http://www.w3c.org/2000/10/swap/pim/contact#Person”

A ontologia ilustrada na figura 2 é representada na notação formal do RDF com sintaxe XML como iespecificado na figura 3.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>

</rdf:RDF>
```

Figura 3 – Arquivo RDF.

fonte: http://www.w3.org/TR/2003/WD-rdf-primer-20030123/

A linguagem RDF Schema é utilizada para definir as classes, hierarquias de classes e as propriedades. Este padrão foi definido seguindo as recomendações da W3C, ele define a estrutura válida para os arquivos RDF.

2.3.2 OWL.

A OWL (*Ontology Web Language*) [18] foi desenvolvida pelo W3C (*Web Ontology Working Group*) para suprir as limitações dos padrões RDF e RDF Schema. A sua elaboração ocorreu a partir desses dois padrões e também do DAML+OIL [19].

Dentre as limitações existentes no RDF e RDF Schema que foram supridas pelo OWL estão [17]:

- Escopo local das propriedades: a restrição rdfs:range, utilizada pelo RDF Schema para indicar que os valores de uma determinada propriedade são instâncias de uma ou mais classes, limita os valores que podem ser aplicados a uma determinada propriedade. Não é possível modelar, por exemplo, uma classe do tipo animal que apresenta o valor “planta” para a propriedade “se alimenta de” enquanto outros animais podem se alimentar também de carnes.
- Classes Disjuntas: No RDF é possível definir, por exemplo, uma classe *mulher* sendo uma subclasse da classe *pessoa*. Porém não é possível indicar que as classes *mulher* e *homem*, apesar de serem derivadas da classe pessoa, são classes disjuntas.
- Combinação booleana de classes: A criação de classes a partir de outras através das operações união, intersecção e complemento não são suportados pelo padrão RDF. OWL propõe extensões de vocabulário padronizadas para essas operações.
- Restrições de Cardinalidade: O RDF não fornece suporte para a definição da quantidade de valores que uma propriedade pode ter. Por exemplo: uma classe carro contém somente o valor 4 para o atributo rodas.
- Caracterização das propriedades: O RDF não fornece suporte para rotular as propriedades como transitiva, única ou inversa de outra propriedade. Com isso não é possível aplicar inferência sobre as classes de acordo com as suas propriedades.

Além dessas limitações, outras limitações são tratadas pelo OWL. A sua divisão é de acordo com a complexidade das extensões da linguagem incorporada. São tipos de OWL [17,20]:

OWL-Lite: é a versão mais simplificada que oferece hierarquias de classificação e restrições simples. A sua vantagem que é mais simples de ser implementada e tem o melhor desempenho de execução, sua desvantagem é a pouca expressividade da linguagem.

OWL-DL: apresenta todos os recursos da linguagem OWL, mas impõe restrições quanto à utilização dos recursos. O OWL-DL aumenta a expressividade da linguagem e mantém a decidibilidade.

OWL-Full: possui todos os recursos da linguagem OWL. Esta sublinguagem não impõe restrição sintática. A sua grande vantagem é a compatibilidade, pois qualquer documento RDF válido é um documento OWL-Full válido. A sua desvantagem é que a decidibilidade não é mais garantida.

2.4 CONSULTAS EM UMA ONTOLOGIA: SPARQL

O SPARQL[21] é uma linguagem de consulta declarativa que extrai informações da ontologia gravada no padrão RDF. O SPARQL permite resgatar campos na ontologia que estão na forma de URIs, bNodes[1] e valores. Outra funcionalidade do SPARQL é a extração de sub-grafos RDF da ontologia e a construção de novos grafos RDF através das queries SPARQL informadas.

O Jena na versão 2.5.3 passou a fornecer suporte a linguagem SPARQL, a seguir uma simulação do uso de queries SPARQL, executadas com o auxílio da biblioteca Jena.

Pesquisa simples, procurando todos os recursos e valores que tenha a propriedade "<http://www.w3.org/2001/vcard-rdf/3.0#FN>":

```
SELECT ?recurso  ?valor
WHERE { ?recurso  <http://www.w3.org/2001/vcard-rdf/3.0#FN>  ?valor }
```

Resultado após a execução da Query.

```
-----------------------------------------------------
| recurso                          | valor          |
=====================================================
| <http://somewhere/MattJones/>    | "Matt Jones"   |
| <http://somewhere/SarahJones/>   | "Sarah Jones"  |
| <http://somewhere/RebeccaSmith/> | "Becky Smith"  |
| <http://somewhere/JohnSmith/>    | "John Smith"   |
-----------------------------------------------------
```

[1] Blank Nodes(bNodes) são nodos no grafo RDF que não contem uma URI. Ver http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/#BlankNodes

2.5 ESPECIFICAÇÃO DE VISÕES DE ONTOLOGIAS

Muitas ontologias vêm sendo desenvolvidas com o intuito de ser um padrão. Elas chegam a definir satisfatoriamente um ou mais domínio com os seus conceitos, acabam apresentando um grande porte. Porém os usuários, nem sempre utilizam todos os conceitos fornecidos [16].

As visões na ontologia, assim como nas tabelas dos bancos de dados, é uma representação personalizada do domínio que permite reduzir o universo de dados.

Especificar várias visões da ontologia ajuda a compartilhar o conhecimento e permite que aplicações com o escopo distinto utilizem a ontologia sem precisar carregar conceitos que não serão utilizados [17],

Isso é um dos desafios da Web Semântica ao qual as ontologias podem contribuir, permitindo elas o reuso da informação através de agentes.

3. AS FUNCIONALIDADES E A ARQUITETURA DO ONTOCOVER.

O OntoCover foi idealizado e desenvolvido no Instituto de Computação da Unicamp em 2003, pelo professor Renato Fileto com o auxílio do bolsista Lauro Ramos Venâncio na implementação. Com esta biblioteca é possível:

- Carregar ontologias
- Extrair visões
- Visualizar e navegar.
- Escolher e comparar Coberturas Ontológicas

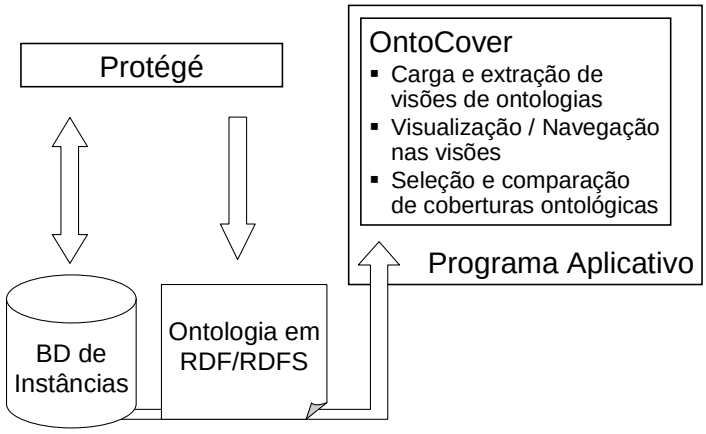


Figura 4 - Componentes relacionados com a biblioteca OntoCover.

A figura 4 ilustra o funcionamento da biblioteca com as partes envolvidas. O Protégé fica encarregado da construção da ontologia, esta será armazenada em arquivos do tipo RDF/ RDFS. As instâncias referentes aos conceitos da ontologia podem ser armazenadas numa base de dados (BD de Instâncias) e carregadas posteriormente. O OntoCover, por ser um biblioteca, têm a necessidade de estar contido em um Programa Aplicativo que forneça a ontologia e extraia da biblioteca as funcionalidades necessárias para a aplicação.

3.1 VISÕES DE ONTOLOGIAS NO ONTOCOVER

O Ontocover carrega as visões de acordo com as anotações inseridas no aquivo RDF Schema da Ontologia. Para inserir estas anotações é aconselhável utilizar o Protegé. Com ele deve criar as metaclasses tendo a seguinte configuração:

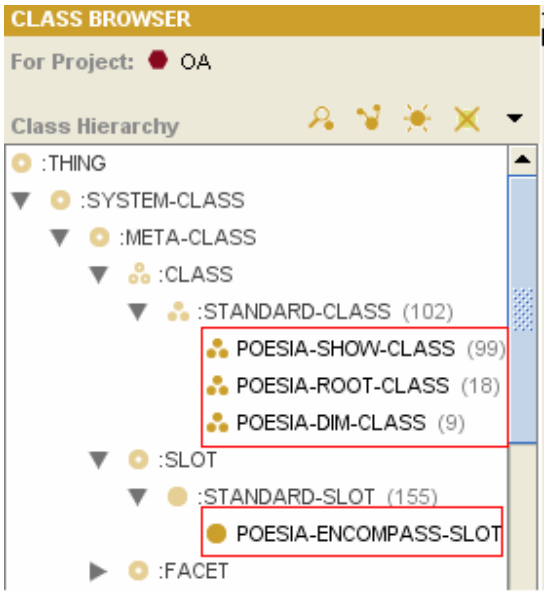


Figura 5 - Classes para a marcação da visão

- ***Poesia-dim-class****:* classe pertencente ao nodo raiz da Ontologia, onde os conceitos derivam dela.
- ***Poesia-root-class*****:** esta classe relaciona as classes Poesia-dim-class com Poesia-show-class sendo uma propriedade entre elas.
- *Poesia-show-class:* essa classe contém as instâncias que serão resgatadas pela visão.
- *Poesia-encompass-slot*: para que as classes contenham atributos, o Protégé disponibiliza a criação de s*lots* confome ilustra a figura abaixo. O OntoCover identifica-os através da classe *Poesia-encompass-slot*, com isso os atributos são adicionados aos conceitos da ontologia.

A figura abaixo ilustra uma ontologia no Protégé:

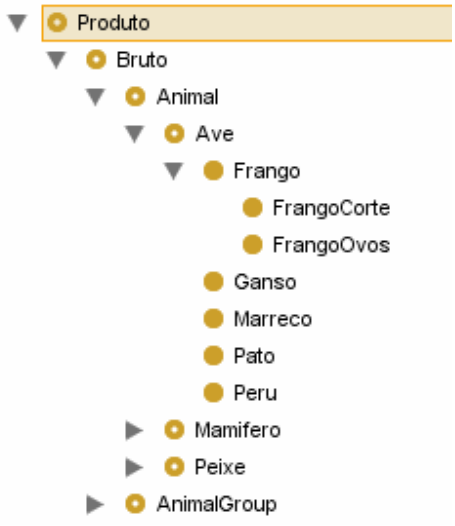


Figura 6 – Ontologia no Protégé

Na figura 6 temos as classes e respectivamente as meta-classes:

Produto:	poesia-dim-class
Bruto:	standard-class
Animal:	poesia-root- class
Ave:	poesia-show- class
Frango:	poesia-show- class
FrangoCorte:	poesia-show- class

Uma classe pertencente a *standard-class,* como a classe Bruto na figura 2.4.2, não terá o seu conceito relacionado na visão com isso o Ontocover não irá buscar a(s) instância(s) dessa classe.

3.2 REPRESENTAÇÃO DA ONTOLOGIA POR ÁRVORE HIPERBÓLICA.

A Árvore Hiperbólica [22] é uma técnica criada para visualizar e manipular grandes hierarquias sem que o foco e o contexto perca destaque. Nessa técnica uma estrutura plana é disposta em um plano circular, com isso elementos próximos ao foco aparecem com tamanho maior do que os elementos mais periféricos do foco. Vale ressaltar que o foco não precisa ser obrigatoriamente o centro da imagem.

Ao utilizar Árvore Hiperbólica para representar ontologias, o nodo posicionado pelo usuário no foco aparece maior. Na navegação, o aparecimento de nodos periféricos cresce exponencialmente à medida que a ontologia é deslocada. Abaixo uma tela do OntoCover utilizando a representação de Árvore Hiperbólica com o foco no centro da imagem.

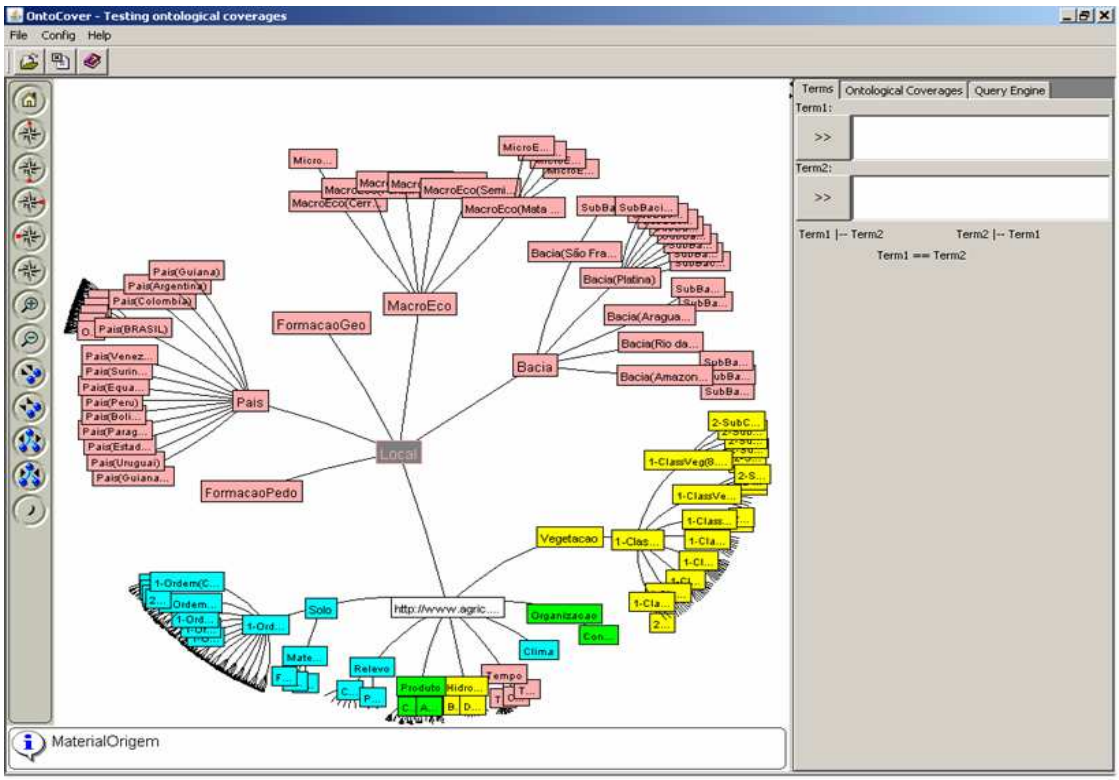


Figura 7 - Visualização de Ontologia pelo OntoCover por Árvore Hiperbólica

3.3 A ARQUITETURA DO ONTOCOVER:

A figura 4 ilustrou uma visão geral do OntoCover. Nesse tópico serão ilustrados os módulos na arquitetura (figura 8), responsáveis pelas funcionalidades, assim como as interações existentes entre os módulos. As classes que serão mencionadas, a seguir, que tenham as iniciais “OC” são encontradas na biblioteca OntoCover.

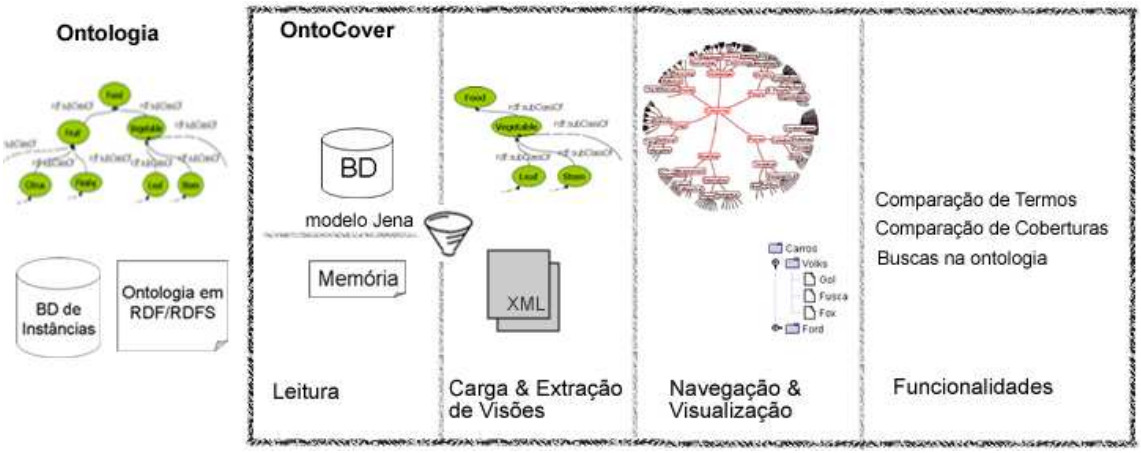


Figura 8 – Módulos da arquitetura do OntoCover

Leitura da ontologia:

Nesse módulo é utilizado basicamente o Jena para ler a ontologia de um arquivo no padrão RDF ou em um banco de dados relacional. Caso a ontologia esteja presente no banco de dados, é realizada uma conexão ODBC para trazer instâncias de algumas classes.

A ontologia será lida pela classe OCModel do OntoCover na memória ou no banco de dados, ficando a cargo do usuário decidir. A classe OCModel contém o objeto Model do pacote Jena que é responsável por essa tarefa.

O próximo módulo irá trabalhar com a carga e extração dos conceitos da ontologia que foi lida nesse módulo.

Carga e Extração da Visão:

O módulo anterior percorreu toda a ontologia e criou um modelo, objeto do tipo Model do pacote Jena, na memória ou no banco de dados. O módulo atual irá gerar uma visão da ontologia de acordo com as anotações semânticas presente no esquema.

As anotações são as mesmas descritas na seção 3.1 Visões de Ontologias no Ontocover. À medida que são realizadas as comparações, da ontologia lida no passo anterior com as anotações, é gerado um arquivo e atribuído valores no objeto do tipo OCNode da classe OCModel.

O arquivo gerado possui o formato xml e contém a representação da visão aplicada na ontologia. O objeto OCNode irá conter os nodos da ontologia que satisfazem o processo de comparação.

O próximo módulo irá apresentar visualmente o resultado da visão aplicada na ontologia. Esse resultado foi inserido nesse módulo no objeto OCNode e no arquivo no formato xml.

Navegação e visualização

Nesse módulo o objeto OCNode produzido no módulo anterior pela classe OCModel, será utilizado para gerar a representação visual da visão da ontologia. Esse módulo permite que o usuário navegue e manipule a visão.

O OntoCover disponibiliza a navegação na forma de Árvore Hiperbólica e pastas, respectivamente dos pacotes Treebolic e AWT. Essas duas classes são implementações da classe OCTree pertencente ao OntoCover. A decisão cabe ao usuário.

O objeto OCTree permite a navegação e manipulação da visão da ontologia. Ele é fabricado pela classe OCModel a partir do objeto OCNode, fabricado no módulo anterior.

Este módulo disponibiliza o objeto que contém a representação da visão e que pode ser usado na interface gráfica.

Seleção e comparação de termos

O módulo anterior se encarregou de gerar a representação visual da ontologia de acordo com as anotações semânticas contidas no esquema. O módulo atual compara termos da ontologia que foram selecionados pelo usuário durante a navegação na ontologia.

O usuário pode seleciona e fazer comparações de termos ou tuplas. A interface OCTree tem um método abstrato para retornar os termos e tuplas que foram selecionados pelo usuário, com isso tanto a implementação dessa classe pela Treebolic ou AWT irá implementar esse método.

As classes que representam os termos e tuplas são respectivamente OCTerm e OCTuple. Ambas as classe tem o método (treeEncompass) para comparar dois termos (no caso de OCTerm) ou duas tuplas (no caso de OCTuple) na ontologia.

Esse módulo encerra o fluxo de interações entre os módulos ilustrados até agora.

Buscas na ontologia

Esse módulo executa buscas na ontologia utilizando a linguagem SPARQL. Para seu funcionamento é importante que o módulo de “Carga e extração da visão” tenha sido executado com êxito.

A busca é realizada passando o comando (query) para o método (runQuery) da classe OCModel, que utiliza as funcionalidades da biblioteca Jena para fazer buscas na ontologia. O resultado devolvido pelo Jena é formatado podendo ser listado, salvo em arquivo ou exibido em componentes visuais.

4. ABORDAGEM

4.1 ANÁLISE DA IMPLEMENTAÇÃO ENCONTRADA.

O OntoCover foi inicialmente implementado sob a plataforma Java Development Kit 1.4.

O projeto contava com as bibliotecas:

- **Jena na versão 2.0**: utilizada para extrair os conceitos da ontologia no arquivo RDF de acordo com as marcações inseridas para obter a visão.
- **Treebolic-poesia.jar**: implementação da árvore hiperbólica utilizada para visualizar e navegar na ontologia. A biblioteca foi modificada para permitir capturar na navegação eventos do mouse e seleção de mais de um nodo.
- **Driver JBDC do PostgreSQL 7.3**: o JBDC (Java Database Connectivity) permite que as aplicações Java enviem comandos SQL para qualquer Banco de Dados Relacional. Este driver permitiu, ao OntoCover, carregar as instâncias RDF do Banco de Dados PostgreSQL

Por ter sido desenvolvido em pouco tempo, a elaboração da documentação e organização de código não foi devidamente feita.

As bibliotecas ficaram obsoletas, o que pode implicar em deteriorização do desempenho. O fato de não ter documentação dificulta na elaboração de futuras melhorias e manutenções.

4.2 ALTERAÇÕES REALIZADAS.

4.2.1 ATUALIZAÇÃO DAS BIBLIOTECAS:

- Jena na versão 2.0 (ago. /2003) foi atualizada para a versão 2.5.3 (jan. /2007). Todos os recursos (arquivos jar) da distribuição antiga do Jena tinham sido incorporados ao projeto - mesmo usando parcialmente a biblioteca. Na atualização para a nova biblioteca, só foram adicionados os recursos requisitados pela aplicação.
- Driver JBDC do PostgreSQL da versão 7.3 para a 8.3 (jul. /2007)

Não foi possível atualizar a biblioteca utilizada(treebolic-poesia.jar) para Árvore Hiperbólica, pois ela foi modificada e não foi documentada a origem da implementação. Mesmo após pesquisar, não foi possível encontrar versões recentes da implementação da Árvore Hiperbólica que apresente compatibilidade de código com esta.

Algumas classes pertencentes a essas versões antigas ficaram com alguns métodos obsoletos, com isso foi necessário consultar a documentação para realizar as alterações de modo a não prejudicar as funcionalidades presentes no OntoCover.

4.2.2 ALTERAÇÕES NO CÓDIGO:

O código precisou sofrer ajustes uma vez que foi projetada para a versão 1.4 do Java Development Kit (JDK). Para se adaptar a versão 6 do JDK e usufruir do ganho de performance dessa versão, foram necessárias as adaptações:

- Instanciação das Coleções Java no formato parametrizado.
- Alguns objetos tinham um nome que passou a ser palavra reservada na versão 6 do Java, foi necessário mudar o nome do objeto.

4.2.3 ARQUIVO DE CONFIGURAÇÃO.

Foi criado arquivo de configuração no formato XML (oc-config.xml) para permitir ao usuário definir:

- A ontologia a ser carregada ao iniciar o aplicativo.
- O nome do arquivo gerado pela extração da visão que será utilizado para exibir a Árvore Hiperbólica.
- Informações para conectar ao Banco de Dados como: nome do banco, driver a ser usado, nome das tabelas para o esquema RDF e a instância RDF, nome do usuário e senha.

Na versão anterior o nome do arquivo da instancia deveria possuir o mesmo nome do seu esquema. Por exemplo: para o esquema com o nome *OA.rdfs,* a instância deveria obrigatoriamente se chamar *OA.rdf*. Isso prejudicava o trabalho de um esquema com várias instâncias.

4.2.4 DOCUMENTAÇÃO DO CÓDIGO

Foi utilizado o sistema de documentação *Javadoc* [6], desenvolvido pela Sun Microsystems. A ferramenta gera no formato HTML uma representação de cada classe ilustrando: métodos, atributos e comentários feitos no código para descrevê-los.

Esse tem sido o padrão para expor os serviços de cada entidade dos aplicativos codificados na linguagem Java. O resultado pode ser consultado em http://www.inf.ufsc.br/~celoom/ontocover/javadoc/.

No Anexo I encontra-se disponível o diagrama de classes simplificado e no Anexo II o manual do usuário.

4.2.5 IMPLEMENTAÇÃO DA LINGUAGEM DE CONSULTA: SPARQL.

A biblioteca Jena na versão 2.5.3 passou a suportar buscas SPARQL na ontologia no padrão RDF. Abaixo temos um trecho de código que ilustra a busca feita através da programação em Java com o Jena.

```
      String ocQuery =  "SELECT * " +
                        "WHERE " +
                        "{?recurso  ?propriedade  ?valor } LIMIT 5";

      Model ocSchema =
FileManager.get().loadModel("file:Ontologias/OA.rdfs");


      QueryExecution ocQe = QueryExecutionFactory.create(ocQuery,
ocSchema);
      ResultSet ocResults = ocQe.execSelect();
      ResultSetFormatter.out(ocResults);
      ocQe.close();
```

No código acima é executada a query “ocQuery” que busca as 5 primeiras triplas RDF na ontologia “OA.rdfs”. O resultado da execução da query é armazenada no objeto do tipo ResultSet.

A manipulação do resultado através desse Objeto apresentou problemas, pois este objeto não apresenta uma estrutura de dados de fácil manipulação.

A solução então foi converter essa estrutura de dados para as estruturas de dados mais simples em Java como as listas do pacote java.util. e incorporar no OntoCover esse mecanismo de busca SPARQL em uma forma mais fácil de usar, que o usuário repasse apenas a query.

O procedimento implementado na biblioteca OntoCover para essa tarefa é bem simples. Basta seguir os passos abaixo:

1. Carregar a ontologia:

```
OCModel ocModel = new OCModel();
ocModel.loadRDF( “nome do arquivo .rdf “, false, false);
```

2. Fornecer somente a Query para o método runQuery.

```
List<OCLinha> tabela = ocModel.runQuery(query);
```

3. Extrair o resultado:

O resultado é uma lista do tipo OCLinha. Esse tipo foi criado para representar os atributos contidos numa linha de resultados.

O primeiro elemento da lista são os metadados que servirão para identificar os valores de retorno.

Os elementos seguintes da lista são os valores propriamente dito.

Abaixo temos uma representação da consulta SPARQL feita por um aplicativo utilizando a biblioteca OntoCover.

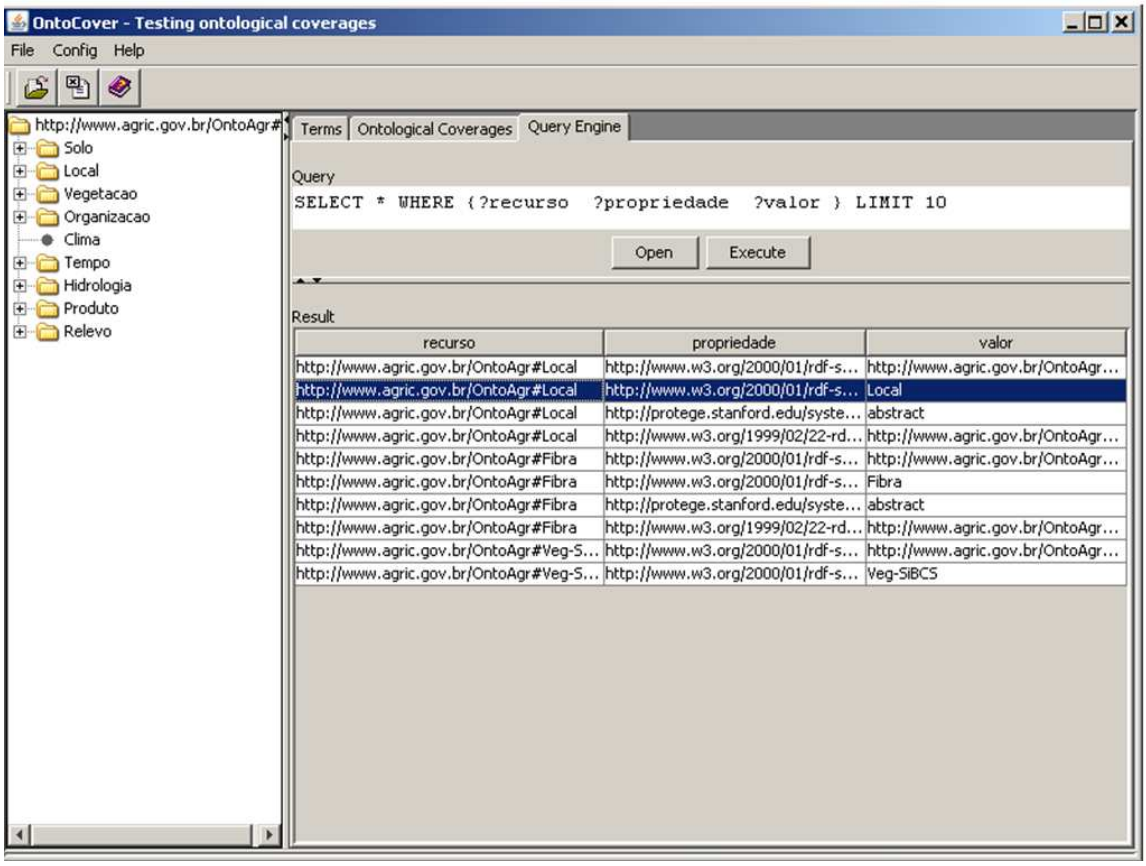


Figura 9 – Exemplo de busca SPARQL realizada pelo OntoCover

5. RESULTADOS DA IMPLEMENTAÇÃO

Os resultados estão organizados quanto aos itens abaixo:

Compatibilidade

A manutenção no código não afetou as funcionalidades e assinaturas de métodos públicos. Com isso a compatibilidade de código não foi prejudicada, sendo desnecessárias eventuais mudanças nos Programas Aplicativos que utilizavam a versão antiga e queiram migrar para a nova.

Desempenho

Foi obtido ganho de performance, para a aplicação e para a biblioteca, com a atualização na biblioteca Jena e no código para JDK 6.

Usabilidade

Para o usuário a atualização das bibliotecas ficou transparente, pois não foi necessário modificar a interface gráfica do aplicativo e a assinatura dos métodos públicos das classes.

A criação do arquivo de configuração permitiu portabilidade. O OntoCover agora pode ser configurado de acordo com o ambiente de trabalho.

Documentação

A documentação do código facilita a contextualização do OntoCover, ajudando em futuras manutenções e na utilização pelos usuários.

Funcionalidades

Infelizmente nem todas as funcionalidades puderam ser resgatadas, fato que aconteceu com o mecanismo de banco de dados. Com isso, essa nova versão não permite ler uma ontologia armazenada no banco de dados ou carregar ontologias sem ser na memória.

Apesar de ter uma funcionalidade que não pode ser resgatada, neste presente trabalho foi inserida uma nova funcionalidade na biblioteca que é a busca SPARQL na ontologia. Agora é possível identificar se um valor está associado a mais de um recurso, o que ajuda a evitar ambigüidades.

6. TRABALHOS RELACIONADOS

Esta seção descreve ferramentas que apresentam as funcionalidades contidas no OntoCover para efeitos comparativos e também ferramentas que complementam o trabalho através da criação e edição de ontologias.

6.1 Protégé

É um editor de ontologias desenvolvido pela Universidade de Stanford sendo ele livre e de código aberto. Apresenta também suporte para classes e hierarquias de classes com herança múltipla, valores padrão, restrições de cardinalidade, hierarquias de *metaclasses* e *metaclass* e pode incorporar novas funcionalidades através da instalação de plugins.

A modelagem pode ser feita por Protégé-Frames e Protégé-OWL.

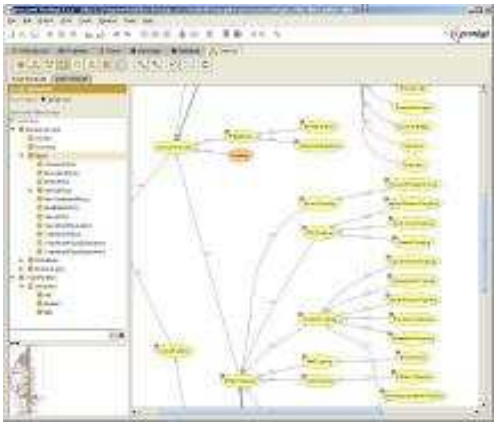


Figura 10 - Protégé-Frames

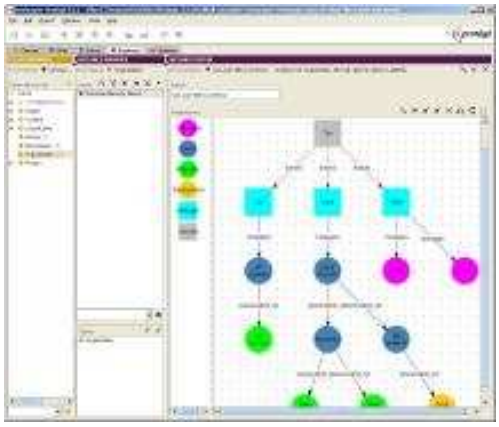


Figura 11 - Protégé-OWL

O Protégé exporta a ontologia para os formatos CLIPS, OWL, RDF Schema e HTML.

O OntoCover por não trabalhar com a criação e modificação da ontologia delega essa responsabilidade para os Editores.

6.2 IsaViz

O software IsaViz[13] foi desenvolvido por Emmanuel Pietriga através do W3C. Neste ambiente é possível navegar e criar modelos RDF representados por grafos direcionados. Dentre os recursos estão a aplicação de stylesheets aos modelos, salvamento do modelo em SVG[14], RDF/XML, esquema de zoom e visão com a tecnologia ZVTM[15]. Esse tipo de visão permite o usuário navegar em todas as regiões da ontologia inclusive as que não foram mostradas na tela.

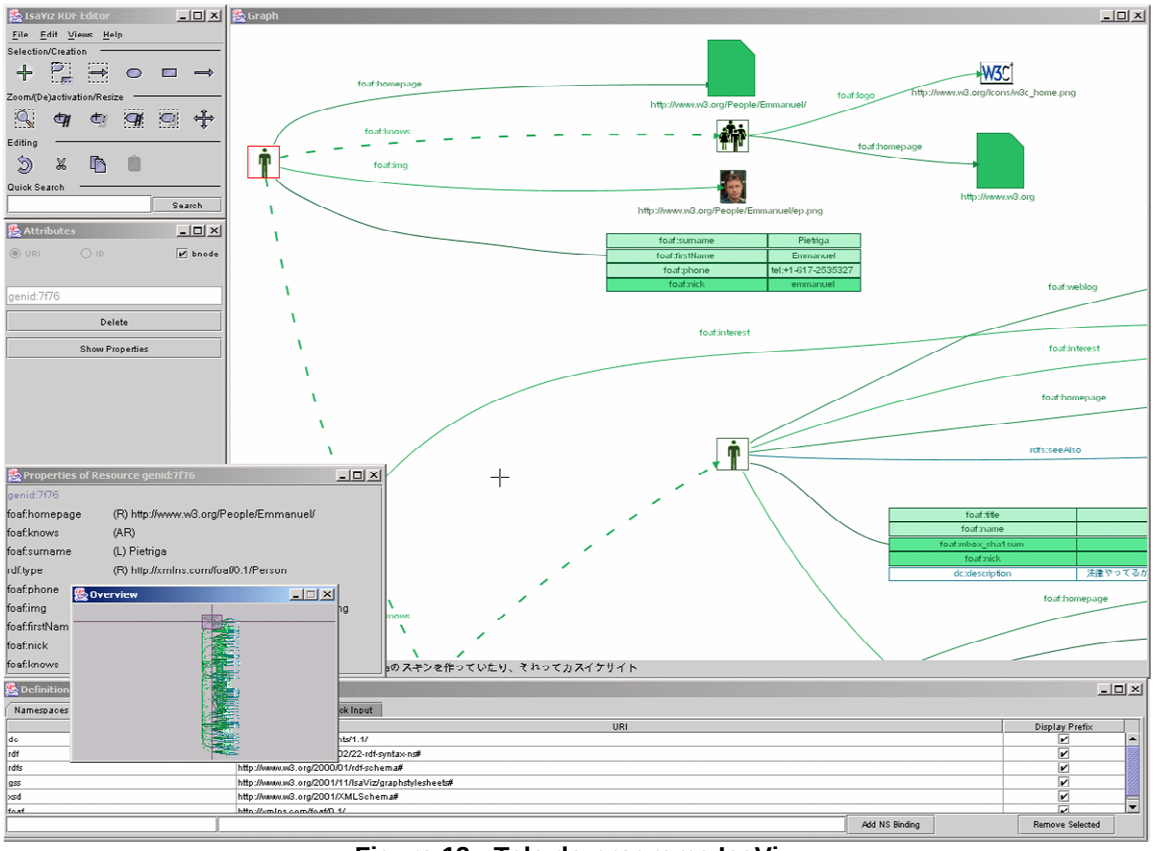


Figura 12 - Tela do programa IsaViz.

6.3 RDF Gravitz(RDF Graph Visualization Tool)

Ferramenta livre para visualização em grafo para ontologias dos tipos RDF e OWL[7]. A ferramenta apresenta na sua implementação as bibliotecas JUNG Graph para visualização na forma de grafo e o Jena[8] para extração da ontologia. Abaixo uma tela do programa:

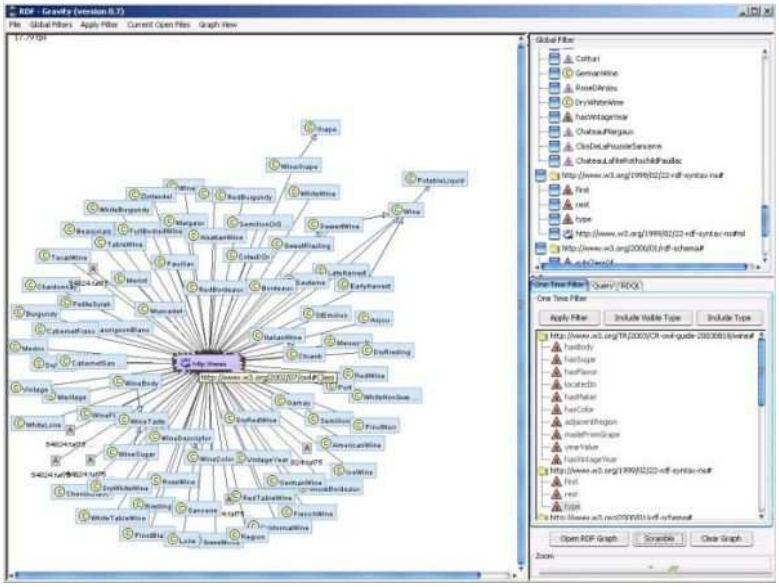


Figura 13 - Tela do programa RDF Gravitz.

Funcionalidades Apresentadas:

- Visualização da ontologia através de Grafo.
- Filtros que permitem visões específicas, removendo arestas e vértices.
- Busca textual.
- Geração de visões através de *queries* RDQL, ver figura 5.2
- Visualização de múltiplos arquivos RDF/OWL.

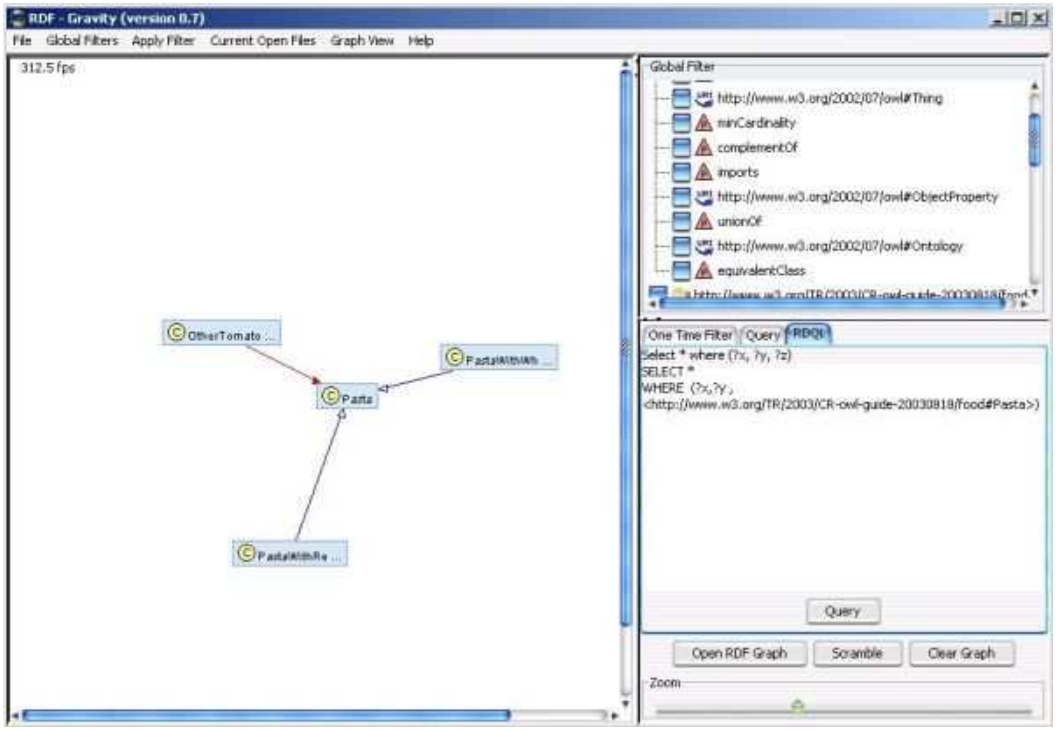


Figura 14 - RDF Gravitz após executar uma query RDQL.

A visualização através de grafos perde a clareza à medida que surgem mais arestas e vértices. O usuário pode contornar isso através dos filtros, visões geradas pelas consultas RDQL e também pela compressão dos demais nodos.

Um outro problema é que está ferramenta não permite salvar a visão gerada e depois carregá-la. O desempenho da aplicação é deteriorado quando só se quer trabalhar com um conjunto de nodos, pois a ontologia é carregada por completa ao invés de somente a visão.

Por outro lado, apresenta uma boa documentação e uma interface com o usuário intuitiva e a opção de fazer buscas através de texto ou *query* RDQL. Estas duas alternativas permitem que usuários de níveis de conhecimentos distintos tirem proveito da ferramenta.

7. Conclusões

Este trabalho apresenta as alterações ocorridas no processo de reengenharia da biblioteca OntoCover e as melhorias inseridas nessa nova versão. Com o OntoCover uma aplicação na Web Semântica tem os recursos necessários para manipular ontologia, podendo a aplicação: carregar, extrair visões, navegar na ontologia, comparar descritores semânticos e nessa nova versão procurar informação na ontologia através da linguagem SPARQL.

As principais contribuições desse trabalho são: (1) manutenção no OntoCover, os pacotes de softwares utilizados pelo OntoCover e o código foi portado para versões mais recentes para reduzir a probabilidade de extinção das funcionalidades a cada lançamento de novos pacotes de softwares e também para o OntoCover evoluir junto com os seus artefatos de software; (2) documentação do código, a documentação facilita futuras manutenções e ajuda o manuseio pelos usuários; (3) extensão de funcionalidade através da implementação SPARQL, com essa nova funcionalidade fica possível encontrar recursos, propriedades e valores através de buscas e saber a quantidade de vez que aparece;

Outras contribuições podem ser dadas ao OntoCover, uma contribuição interessante seria a incorporação de uma linguagem declarativa para a especificação das visões de ontologias, de modo a oferecer um mecanismo mais adequado que os recursos de reflexão computacional utilizados atualmente para esta finalidade. Isto pode ser realizado com o estudo da linguagem SPARQL uma vez que ela tem um mecanismo para extrair sub-grafos da ontologia.

Outro tema para trabalho futuro é a associação da ontologia, carregada pelo OntoCover, com um documento. A cada termo selecionado na ontologia é destacado no documento a(s) palavra(s) referente ao conceito da ontologia. Isto poderia ser feito indexando o documento de acordo com cada termo presente na ontologia. Esta tarefa poderia contar com o auxílio de uma ferramenta de indexação de documentos, como por exemplo, o Lucene desenvolvido pela Apache Software Foundation.

8. Referências

[1] M.Uschold and M. Grüninger. *Ontologies: Principles, Methods and Applications.* Knowledge Engineering Review, 11(2): 93-155, 1996.

[2] T. Berners-Lee, J. Hendler, O. Lassila. *The Semantic Web*. Scientific American, May, 2001.

[3] Davies, J., Studer, R., Warren, P. (Eds.) Semantic Web Technologies: trends and research in ontology-based Systems, John Wiley & Sons, 2006

[4] T. Menzies. Cost benefits of ontologies. *Intelligence* ,10(3):27-32, 1999.

[5] The World Wide Web Consortium (W3C). RDF-Primer. Disponível em: <http://www.w3.org/TR/2003/WD-rdf-primer-20030123/>. Acesso em: 13 ago. 2007

[6] JavaDoc. JavaDoc Tool. Disponível em <http://java.sun.com/j2se/javadoc/>. Acesso em 31 de agosto de 2007.

[7]RDF Gravity. RDF Gravity (RDF Graph VisualizationTool) Disponível em : <http://semweb.salzburgresearch.at/apps/rdf-gravity/user_doc.html>. Acesso em: 22 ago. 2007.

[8] Jena – A Semantic Web Framework for Java. Disponível em <http://jena.sourceforge.net/>. Acesso em 25 ago. 2007.

[9] The World Wide Web Consortium (W3C). Resource Description Framework (RDF). Disponível em: <http://www.w3c.org/RDF>. Acesso em: 04 set. 2007

[10] Venâncio, L. R., Fileto, R. Medeiros, C. B. Aplicando Ontologias de Objetos Geográficos para Facilitar Navegação em GIS. GeoInfo, 2003.

Disponível em: <http://www.geoinfo.info/geoinfo2003/papers/geoinfo2003-45.pdf>. Acesso em: 04 set. 2007

[11] The World Wide Web Consortium (W3C). Semantic Web. Disponível em: <http://www.w3c.org/2001/sw>. Acesso em: 04 set. 2007

[12] Eduardo de Castro, Laura Mastella, Mara Abel, Luiz Fernando De Ros. PetroQuery: Uma ferramenta para consulta e navegação sobre ontologias. Escola Regional de Banco de Dados(ERDB). Rio Grande do Sul, abr 2005.

[13] The World Wide Web Consortium (W3C). IsaViz: A Visual Authoring Tool for RDF. Disponível em: <http://www.w3.org/2001/11/IsaViz/>. Acesso em: 04 set. 2007

[14] The World Wide Web Cons. (W3C). Scalable Vector Graphics (SVG). Disponível em: <http://www.w3c.org/Graphics/SVG>. Acesso em: 04 set. 2007

[15] ZVTM - Zoomable Visual Transformation Machine. Disponível em <http://zvtm.sourceforge.net/> Acesso em : 04 set. 2007.

[16] Noy, N; Musen, M. (2004). Specifying Ontology Views by Traversal. Submission to ISWC 2004, Hiroshima, Japan.

[17] Grigoris Antoniou and Frank van Harmelen A Semantic Web Primer, The MIT Press, Cambridge, MA, USA, 2004.

[18] Uceda-Sosa, R., C. X. Chen et al. (2004) CLOVE: A Framework to Design Ontology Views. 23th International Conference on Conceptual Modeling (ER ’04), Shanghai, China, Springer-Verlag.

[19] The World Wide Web Consortium (W3C). DAML+OIL. Disponível em: < http://www.w3.org/TR/daml+oil-reference/>. Acesso em: 14 set. 2007

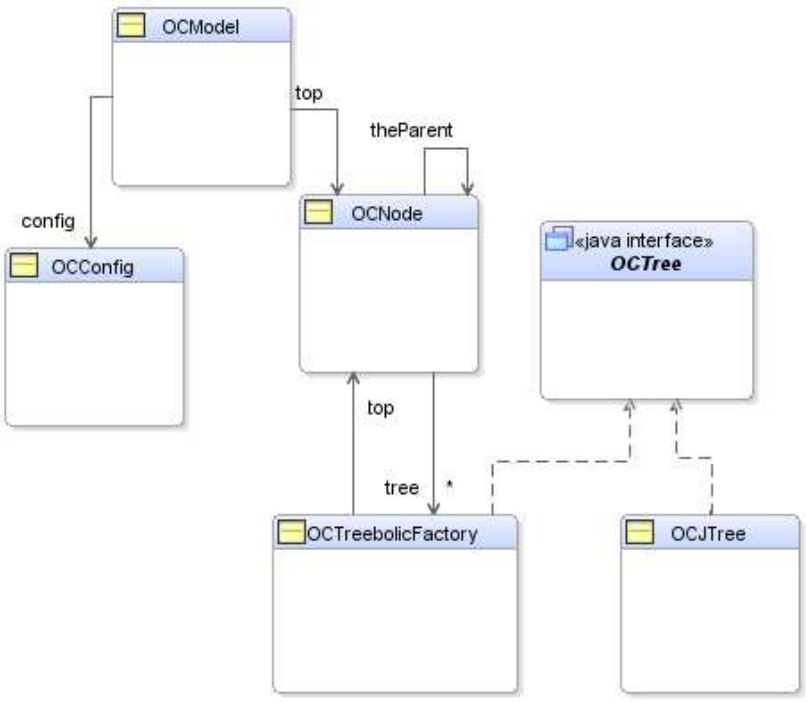
[20] The World Wide Web Consortium (W3C). Ontology Web Language (OWL). Disponível em: < http://www.w3.org/TR/owl-guide/>. Acesso em: 14 set. 2007

[21] World Wide Web Consortium (W3C). SPARQL Query Language for RDF, Disponível em: <http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217>. Acesso em: 15 set. 2007.

[22] J. Lamping, R. Rao, e P. Pirolli. A Focus+Context Techinique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. Proc. ACM SIGCHI Conf. on Human Factor in Computing System, 1995.

ANEXO I - DIAGRAMA DE CLASSES

Abaixo temos um diagrama simplificado da biblioteca ilustrando as principais classes:



A classe **OCModel** fornece acesso a todas as funcionalidades existentes no OntoCover através dos métodos ilustrados no ANEXO I, portando essa é a classe que o aplicativo deve referenciar para poder manipular a ontologia. A figura ilustra também a composição da classe OCModel pela classe **OCConfig** através do atributo "*config*" e pela classe **OCNode** através do atributo "*top"*.

A classe **OCConfig** foi criada nessa nova versão para armazenar atributos de configuração como: localização dos arquivos RDF e RDF Schema; nome do arquivo que será gravado o resultado da visão aplicada as instâncias da ontologia; parâmetros de conexão com o Banco de dados. Esses atributos são utilizados pela classe OCModel.

Já a classe **OCNode** fica responsável por ir armazenando os conceitos extraídos da ontologia, com o auxílio da biblioteca Jena, em nodos para ser utilizado na geração da árvore Hiperbólica. Esses nodos são estruturados de modo a formar uma árvore, isso é conseguido através dos seguintes atributos privados contidos nessa classe:

- theParent: esse atributo é do tipo OCNode, utilizado para indicar o nodo pai do nodo em questão. Um nodo é raiz quando esse atributo é nulo.
- theChildren.: esse atributo é do tipo vetor podendo contendo uma coleção de nodos filhos. Um valor nulo para esse atributo indica que o nodo é folha, isto é, não possui descendentes.

A classe **TreebolicFactory** contém o applet da Árvore hiperbólica. A implementação original da biblioteca da Árvore Hiperbólica foi modificada para permitir selecionar mais de um nodo. Foi inserida nessa modificação uma regra de negócio para evitar que selecione nodos que apresente parentescos do tipo pai e filho. A classe TreebolicFactory implementa a interface OCTree.

A interface OCTree existe para garantir as funcionalidades que as implementações de navegação na ontologia deverão ter para que seja possível executar as funcionalidades presentes no Ontocover. A interface OCTree além de ser implementada pela classeTreebolicFactory, é implementada também pela classe OCJTree.

A classe OCJTree implementa a navegação em pastas e extende a classe JTree presente no pacote swing do Java.

ANEXO II – INSTRUÇÕES BÁSICAS DE UTILIZAÇÃO

Requisitos mínimos:

- Java Development Kit 1.5 O OntoCover foi escrito em Java e, portanto, precisa do JDK para ser compilado.

Utilização:

A seguir será ilustrado os procedimentos para a invocação das funcionalidades.

Carregando a ontologia:

```
1.  OCModel ocModel = new OCModel();
2.  ocModel.loadRDF(<arquivoRDF.rdf>, false, false);
3.  ocModel.loadRDFs(<arquivoRDFs.rdfs>, false, false);
```

Nas linhas 2 e 3 o segundo parâmetro com o valor “false” indica que será carregada na memória, o terceiro parâmetro com o valor “false” indica que será lida do arquivo no disco.

Navegando na ontologia:

```
4.  OCTree octree = ocModel.getOCTree(true);
5.  Component component = octree.getVisualTree();
```

Na linha 4, o parâmetro “true” faz com que o método retorne uma representação hiperbólica da ontologia, caso fosse “false” retornaria uma representação através de pastas.

Na linha 5, é obtido o componente que poderá ser adicionado na interface gráfica da aplicação que dependendo da linha 4 irá exibir a navegação por pastas ou árvore hiperbólica.

Buscas na ontologia e extração dos resultados:

```
6.  List<OCLinha> queryResult = ocModel.runQuery(<query>);
```

Deverá ser passado o comando SPARQL no parâmetro query. O retorno será um List do pacote java util do tipo OCLinha. O primeiro elemento da Lista será os metadados e as linhas serão os valores que satisfazem a query.

Fechando a ontologia:

```
7.  ocModel.closeOntology();
```

Após esse comando, serão liberados os recursos computacionais que foram utilizados para carregar a ontologia. Caso seja necessário manipular a ontologia novamente, os procedimentos para carregar a ontologia terão que ser repetidos.

Reengenharia do Ontocover: uma biblioteca Java para manipular ontologias em aplicações da Web Semântica.

Marcelo Oliveira de Moraes

Departamento de Informática e Estatística –

Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

celoom@inf.ufsc.br

Abstract. *This work approaches some studies about Semantic Web, mainly the tools, and the use of knowledge gotten from maintenance and documentation of the Java library, named OntoCover, to handle the ontologies into knowledge based applications.*

Resumo. *Este trabalho relata alguns estudos em Web Semântica, principalmente ferramentas, e a utilização dos conhecimentos obtidos na manutenção e documentação da biblioteca Java, denominada OntoCover, para manipulação de ontologias em aplicações baseadas em conhecimento.*

1. Introdução

A Web Semântica [2, 3] é uma área de pesquisa em expansão que visa estender o papel dos computadores no suporte a diversas atividades humanas. Ela propõe a formalização do conhecimento na forma de ontologias de modo a permitir a agentes e outros programas catalogar, recuperar, selecionar e compor recursos (dados e serviços) publicados na Web, de maneira mais inteligente e precisa.

O OntoCover é uma biblioteca para manipulação de ontologias, com funcionalidades para extrair visões de ontologias, visualizar e navegar nessas visões, além de especificar e comparar eficientemente anotações semânticas baseadas em alguma dessas visões de uma ontologia. Ele constitui uma contribuição para o desenvolvimento de aplicações sobre a Web semântica.

O OntoCover foi idealizado e desenvolvido no Instituto de Computação da Unicamp em 2003, pelo professor Renato Fileto com o auxílio do bolsista Lauro Ramos Venâncio na implementação. Com o passar do tempo foram surgindo novas versões de pacotes de software utilizados pelo OntoCover, particularmente a linguagem Java e a biblioteca Jena [8]. O OntoCover também não apresentava documentação adequada, fato esse que dificultava o seu uso e a implementação de melhorias.

Este trabalho relata as atualizações e ajustes efetuados no OntoCover, incluindo manutenções, extensões de funcionalidade e documentação. A atualização das versões do Java e do Jena envolveu várias manutenções do código fonte e propiciou melhorias de robustez e desempenho. A documentação gerada inclui o diagrama de classes do projeto, documentação de cada classe na forma de JavaDocs e manual do usuário. Isto promove o reuso do código da biblioteca, facilita a incorporação de melhorias e indica a maneira que o usuário deve manipulá-la para a criação de aplicações na Web Semântica.

Uma outra contribuição deste trabalho foi a implantação de uma nova funcionalidade no OntoCover: suporte a buscas na ontologia através da linguagem SPARQL, padrão da W3C para manipular RDF [9]. Com isso é possível manipular informação específica extraída da ontologia. Por exemplo, fica mais fácil identificar se o valor de uma propriedade de um recurso (e.g. seu nome) aparece em outros pontos da ontologia. Se isso acontecer, é caracterizada uma ambigüidade para aquele nome. Uma forma de resolver tais ambigüidades é associar o recurso ao conceito que melhor expresse a intenção do usuário. A identificação dos possíveis conceitos (e.g. país, estado, time de futebol) que podem ser associados a um recurso com um certo valor na propriedade nome (e.g., São Paulo) também pode ser feita através de expressões de consulta em SPARQL.

2. Web Semântica

Tim Berners-Lee [2] criador da linguagem HTML e líder na criação do consórcio mundial W3C (Word Wide Web Consortium) no Massachusetts Institute of Technology (MIT), foi o idealizador da Web Semântica. Ela tem como objetivo estender a Web atual possibilitando a compreensão da informação pelas máquinas. Para isso ocorrer é necessário: o desenvolvimento de linguagens que atribuam significado aos dados e tecnologias que permitam o processamento baseado nesse significado.

A Web Semântica sugere que a informação na Web seja descrita através de anotações semânticas, que indiquem o significado da informação. Mecanismos baseados em regras e inferência sobre as ontologias e essas anotações devem viabilizar a construção de sistemas e agentes mais inteligentes para automatizar a execução de sofisticadas tarefas, tais como planejar uma viagem, buscando as melhores opções em passagens aéreas, translados e hotéis, segundo preferências do usuário e restrições de tempo e orçamento.

Dentre as técnicas e fundamentos necessários para concretizar os objetivos da Web semântica ontologias têm um papel fundamental. Ontologias atuam no cerne do propósito da Web Semântica, uma vez que a permitem descrever e atribuir significado à informação. A sua utilização favorece o compartilhamento e o reuso da informação.

3. Ontologias

Uma ontologia é uma conceitualização explícita, formal e compartilhada de uma área de conhecimento [1]. Em outras palavras, uma ontologia é um modelo que possui conceitos, propriedades, relações, funções e axiomas, explicitamente definidos de uma forma que permitam a manipulação pelo computador.

O termo ontologia é oriundo da filosofia, onde designa o estudo das teorias sobre a natureza da existência. Pesquisadores dos ramos da inteligência artificial e Web adotaram o termo para designar uma estrutura que define formalmente relações de significado entre termos [2].

Entre os benefícios de se utilizar ontologias estão: [4]

- Compartilhamento: uma ontologia promove uma compreensão comum de um domínio de conhecimento.
- Reuso: o uso de definições explícitas e formais facilita a manutenção do conhecimento, permitindo o fácil entendimento por parte dos usuários e facilitando a reutilização da ontologia, ou de parte dela.
- Estruturação da informação: permite a captura da semântica dos dados e seu processamento automático, gerando conhecimento para os humanos.
- Interoperabilidade: permite que diferentes sistemas computacionais compartilhem dados e informações.
- Confiabilidade: uma representação formal torna possível uma automatização consistente e mais confiável.

A figura 1 ilustra uma ontologia na forma de um grafo, onde os vértices representam recursos e as arestas representam propriedades desses recursos. Os valores das propriedades podem ser outros recursos (apontados pelas arestas) ou literais. Na figura 1, os recursos representados por elipses referem-se a conceitos e os recursos representados por retângulos

referem-se a instâncias de conceitos. As instâncias denominadas “Strawberry” e “Potato” têm propriedades com valores literais, tais como: “Color”, “Seedless”, “Flavor” e “Climate”.

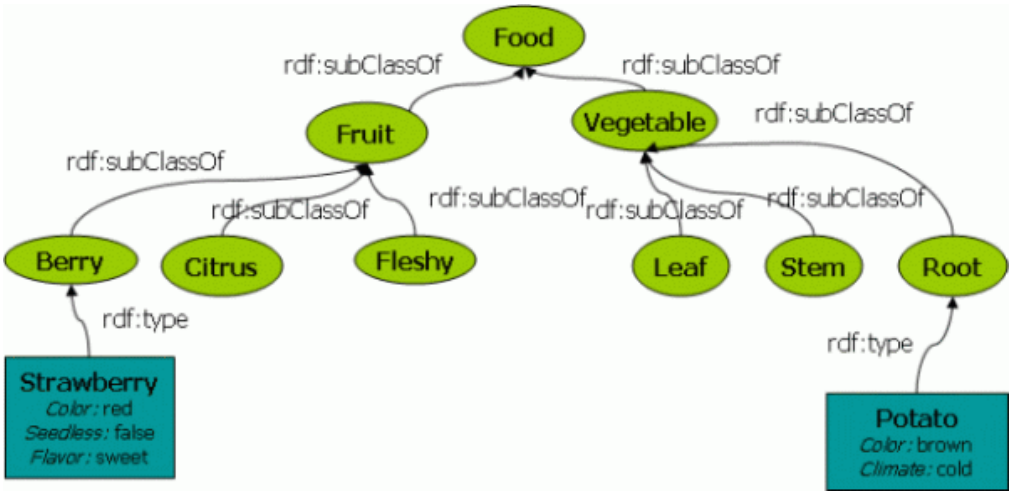


Figura 1 – Exemplo de Ontologia
fonte: http://www.sei.cmu.edu/isis/guide/gifs/fruit-ontology.gif

Através da navegação nesse grafo é possível extrair conhecimento. Por exemplo: Strawberry é do tipo “Berry” que é uma subclasse de “Fruit” que por sua vez é subclasse de “Food” (comida). Caso houvesse outro recurso com o nome ”Strawberry” (e.g., um peixe vermelho com pintinhas pretas) relacionado a outro recurso que não “Berry” (e.g., personagem de desenho animado) seria possível distinguir os dois recursos com o nome “Strawberry”, através de inferências na ontologia. Uma ontologia usualmente é representa pelos padrões RDF [9] e OWL [20]. As próximas seções tratam esses dois padrões.

4. O Ontocover.

O OntoCover foi idealizado e desenvolvido no Instituto de Computação da Unicamp em 2003, pelo professor Renato Fileto com o auxílio do bolsista Lauro Ramos Venâncio na implementação. As funcionalidades implementadas foram: (i) Carregar ontologias. (ii) Extração de visões. (iii) Visualização e navegação através de pastas ou via Árvore Hiperbólica. (iv) Escolher e comparar Coberturas Ontológicas. Abaixo uma imagem do OntoCover utilizando a Árvore Hiperbólica.

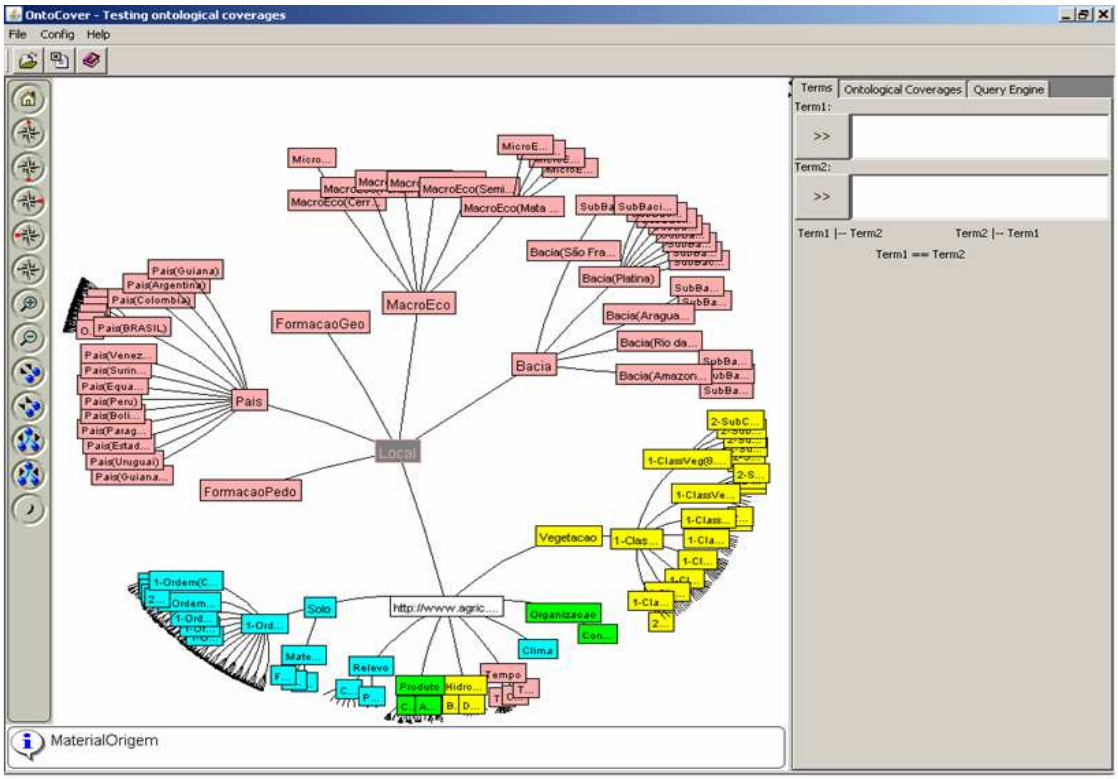


Figura 7 - Visualização de Ontologia pelo OntoCover por Árvore Hiperbólica

4.1. Análise da implementação encontrada.

O OntoCover foi inicialmente implementado sob a plataforma Java Development Kit 1.4. O projeto contava com as bibliotecas:

- Jena na versão 2.0: utilizada para extrair os conceitos da ontologia no arquivo RDF de acordo com as marcações inseridas para obter a visão.
- Treebolic-poesia.jar: implementação da árvore hiperbólica utilizada para visualizar e navegar na ontologia. A biblioteca foi modificada para permitir capturar na navegação eventos do mouse e seleção de mais de um nodo.
- Driver JBDC do PostgreSQL 7.3: o JBDC (Java Database Connectivity) permite que as aplicações Java enviem comandos SQL para qualquer Banco de Dados Relacional. Este driver permitiu, ao OntoCover, carregar as instâncias RDF do Banco de Dados PostgreSQL

5. Alterações realizadas.

5.1. Atualização das bibliotecas

- Jena na versão 2.0 (ago. /2003) foi atualizada para a versão 2.5.3 (jan. /2007).

Todos os recursos (arquivos jar) da distribuição antiga do Jena tinham sido incorporados ao projeto - mesmo usando parcialmente a biblioteca. Na atualização para a nova biblioteca, só foram adicionados os recursos requisitados pela aplicação.

- Driver JBDC do PostgreSQL da versão 7.3 para a 8.3 (jul. /2007)

Não foi possível atualizar a biblioteca utilizada (treebolic-poesia.jar) para Árvore Hiperbólica, pois ela foi modificada e não foi documentada a origem da implementação. Mesmo após pesquisar, não foi possível encontrar versões recentes da implementação da Árvore Hiperbólica que apresente compatibilidade de código com esta.

Algumas classes pertencentes a essas versões antigas ficaram com alguns métodos obsoletos, com isso foi necessário consultar a documentação para realizar as alterações de modo a não prejudicar as funcionalidades presentes no OntoCover.

5.2. Alterações no código fonte

O código precisou sofrer ajustes uma vez que foi projetada para a versão 1.4 do Java Development Kit (JDK). Para se adaptar a versão 6 do JDK e usufruir do ganho de performance dessa versão, foram necessárias as adaptações:

- Instanciação das Coleções Java no formato parametrizado.
- Alguns objetos tinham um nome que passou a ser palavra reservada na versão 6 do Java, foi necessário mudar o nome do objeto.

5.3. Documentação do código fonte

Foi utilizado o sistema de documentação Javadoc [6], desenvolvido pela Sun Microsystems. A ferramenta gera no formato HTML uma representação de cada classe ilustrando: métodos, atributos e comentários feitos no código para descrevê-los.

Esse tem sido o padrão para expor os serviços de cada entidade dos aplicativos codificados na linguagem Java. O resultado pode ser consultado em http://www.inf.ufsc.br/~celoom/ontocover/javadoc/.

Foi feito também o diagrama de classes simplificado e no Anexo II o manual do usuário.

5.4. Implementação da linguagem de consulta SPARQL.

O SPARQL[21] é uma linguagem de consulta declarativa que extrai informações da ontologia gravada no padrão RDF. O SPARQL permite resgatar campos na ontologia que estão na forma de URIs, bNodes1 e valores. Esta busca é aplicada em relação aos atributos da ontologia: recurso, propriedade e valor.

Outra funcionalidade do SPARQL é a extração de sub-grafos RDF da ontologia e a construção de novos grafos RDF através das queries SPARQL informadas.

O Jena na versão 2.5.3 passou a fornecer suporte a linguagem SPARQL, a seguir um exemplo de pesquisa SPARQL, feita pelo OntoCover:

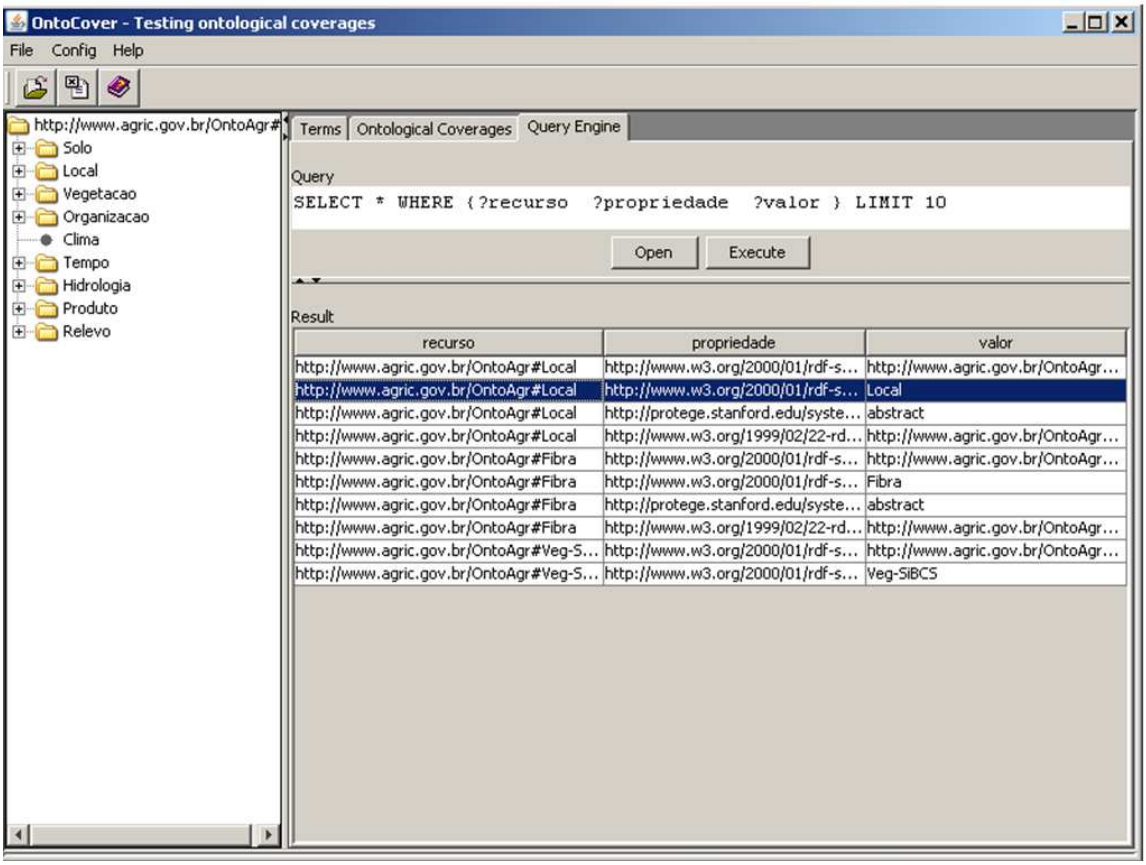


Figura 9 – Exemplo de busca SPARQL realizada pelo OntoCover

6. Resultados Obtidos

Os resultados estão organizados quanto aos itens abaixo:

Compatibilidade

A manutenção no código não afetou as funcionalidades e assinaturas de métodos públicos. Com isso a compatibilidade de código não foi prejudicada, sendo desnecessárias eventuais mudanças nos Programas Aplicativos que utilizavam a versão antiga e queiram migrar para a nova.

Desempenho

Foi obtido ganho de performance, para a aplicação e para a biblioteca, com a atualização na biblioteca Jena e no código para JDK 6.

Usabilidade

Para o usuário a atualização das bibliotecas ficou transparente, pois não foi necessário modificar a interface gráfica do aplicativo e a assinatura dos métodos públicos das classes.

A criação do arquivo de configuração permitiu portabilidade. O OntoCover agora pode ser configurado de acordo com o ambiente de trabalho.

Documentação

A documentação do código facilita a contextualização do OntoCover, ajudando em futuras manutenções e na utilização pelos usuários.

Funcionalidades

Infelizmente nem todas as funcionalidades puderam ser resgatadas, fato que aconteceu com o mecanismo de banco de dados.

Neste presente trabalho foi inserida uma nova funcionalidade na biblioteca, que é a busca SPARQL na ontologia. Agora é possível identificar se existe um valor associado a mais de um recurso, o que ajuda a evitar ambigüidades.

7. Trabalhos Relacionados

7.1 IsaViz

O software IsaViz[13] foi desenvolvido por Emmanuel Pietriga através do W3C. Neste ambiente é possível navegar e criar modelos RDF representados por grafos direcionados. Dentre os recursos estão a aplicação de stylesheets aos modelos, salvamento do modelo em SVG[14], RDF/XML, esquema de zoom e visão com a tecnologia ZVTM[15]. Esse tipo de visão permite o usuário navegar em todas as regiões da ontologia inclusive as que não foram mostradas na tela.

7.2 RDF Gravitz(RDF Graph Visualization Tool)

Ferramenta livre para visualização em grafo para ontologias dos tipos RDF e OWL[7]. A ferramenta apresenta na sua implementação as bibliotecas JUNG Graph para visualização na forma de grafo e o Jena[8] para extração da ontologia.

Funcionalidades Apresentadas: (i)Visualização da ontologia através de Grafo. (ii) Filtros que permitem visões específicas, removendo arestas e vértices. (iii) Busca textual. (iv) Geração de visões através de queries RDQL, (v) Visualização de múltiplos arquivos RDF/OWL.

A visualização através de grafos perde a clareza à medida que surgem mais arestas e vértices. O usuário pode contornar isso através dos filtros, visões geradas pelas consultas RDQL e também pela compressão dos demais nodos.

Um outro problema é que está ferramenta não permite salvar a visão gerada e depois carregá-la. O desempenho da aplicação é deteriorado quando só se quer trabalhar com um conjunto de nodos, pois a ontologia é carregada por completa ao invés de somente a visão.

Por outro lado, apresenta uma boa documentação e uma interface com o usuário intuitiva e a opção de fazer buscas através de texto ou query RDQL. Estas duas alternativas permitem que usuários de níveis de conhecimentos distintos tirem proveito da ferramenta.

8. Conclusões

Este trabalho apresenta as alterações ocorridas no processo de reengenharia da biblioteca OntoCover e as melhorias inseridas nessa nova versão. Com o OntoCover uma aplicação na Web Semântica tem os recursos necessários para manipular ontologia, podendo a aplicação: carregar, extrair visões, navegar na ontologia, comparar descritores semânticos e nessa nova versão procurar informação na ontologia através da linguagem SPARQL.

As principais contribuições desse trabalho são: (1) manutenção no OntoCover, os pacotes de softwares utilizados pelo OntoCover e o código foi portado para versões mais recentes para reduzir a probabilidade de extinção das funcionalidades a cada lançamento de novos pacotes de softwares e também para o OntoCover evoluir junto com os seus artefatos de software; (2) documentação do código, a documentação facilita futuras manutenções e ajuda o manuseio pelos usuários; (3) extensão de funcionalidade através da implementação SPARQL, com essa nova funcionalidade fica possível encontrar recursos, propriedades e valores através de buscas e saber a quantidade de vez que aparece;

Referências Bibliográficas

[1] M.Uschold and M. Grüninger. Ontologies: Principles, Methods and Applications. Knowledge Engineering Review, 11(2): 93-155, 1996.

[2] T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. Scientific American, May, 2001.

[3] Davies, J., Studer, R., Warren, P. (Eds.) Semantic Web Technologies: trends and research in ontology-based Systems, John Wiley & Sons, 2006

[4] T. Menzies. Cost benefits of ontologies. Intelligence ,10(3):27-32, 1999.

[5] The World Wide Web Consortium (W3C). RDF-Primer. Disponível em: <http://www.w3.org/TR/2003/WD-rdf-primer-20030123/>. Acesso em: 13 ago. 2007

[6] JavaDoc. JavaDoc Tool. Disponível em <http://java.sun.com/j2se/javadoc/>. Acesso em 31 de agosto de 2007.

[7]RDF Gravity. RDF Gravity (RDF Graph VisualizationTool) Disponível em : <http://semweb.salzburgresearch.at/apps/rdf-gravity/user_doc.html>. Acesso em: 22 ago. 2007.

[8] Jena – A Semantic Web Framework for Java. Disponível em <http://jena.sourceforge.net/>. Acesso em 25 ago. 2007.

[9] The World Wide Web Consortium (W3C). Resource Description Framework (RDF). Disponível em: <http://www.w3c.org/RDF>. Acesso em: 04 set. 2007

[10] Venâncio, L. R., Fileto, R. Medeiros, C. B. Aplicando Ontologias de Objetos Geográficos para Facilitar Navegação em GIS. GeoInfo, 2003.

Disponível em: <http://www.geoinfo.info/geoinfo2003/papers/geoinfo2003-45.pdf>. Acesso em: 04 set. 2007

[11] The World Wide Web Consortium (W3C). Semantic Web. Disponível em: <http://www.w3c.org/2001/sw>. Acesso em: 04 set. 2007

[12] Eduardo de Castro, Laura Mastella, Mara Abel, Luiz Fernando De Ros. PetroQuery: Uma ferramenta para consulta e navegação sobre ontologias. Escola Regional de Banco de Dados(ERDB). Rio Grande do Sul, abr 2005.

[13] The World Wide Web Consortium (W3C). IsaViz: A Visual Authoring Tool for RDF. Disponível em: <http://www.w3.org/2001/11/IsaViz/>. Acesso em: 04 set. 2007

[14] The World Wide Web Cons. (W3C). Scalable Vector Graphics (SVG). Disponível em: <http://www.w3c.org/Graphics/SVG>. Acesso em: 04 set. 2007

[15] ZVTM - Zoomable Visual Transformation Machine. Disponível em <http://zvtm.sourceforge.net/> Acesso em : 04 set. 2007.

[16] Noy, N; Musen, M. (2004). Specifying Ontology Views by Traversal. Submission to ISWC 2004, Hiroshima, Japan.

[17] Grigoris Antoniou and Frank van Harmelen A Semantic Web Primer, The MIT Press, Cambridge, MA, USA, 2004.

[18] Uceda-Sosa, R., C. X. Chen et al. (2004) CLOVE: A Framework to

Design Ontology Views. 23th International Conference on Conceptual

Modeling (ER ’04), Shanghai, China, Springer-Verlag.

[19] The World Wide Web Consortium (W3C). DAML+OIL. Disponível em: < http://www.w3.org/TR/daml+oil-reference/>. Acesso em: 14 set. 2007

[20] The World Wide Web Consortium (W3C). Ontology Web Language (OWL). Disponível em: < http://www.w3.org/TR/owl-guide/>. Acesso em: 14 set. 2007

[21] World Wide Web Consortium (W3C). SPARQL Query Language for RDF, Disponível em: <http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217>. Acesso em: 15 set. 2007.

[22] J. Lamping, R. Rao, e P. Pirolli. A Focus+Context Techinique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. Proc. ACM SIGCHI Conf. on Human Factor in Computing System, 1995.

Fontes

```
package OntoCover;

import java.awt.Color;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.MalformedURLException;
import java.net.URL;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;
import java.util.TreeSet;

import treebolic.applet.Treebolic;
import util.Conexao;
import util.OCConfig;
import util.model.DefaultTree;
import util.model.OCLinha;
import util.model.OCTabela;

import com.hp.hpl.jena.db.DBConnection;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.ModelMaker;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.ResIterator;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.rdf.model.StmtIterator;
import com.hp.hpl.jena.rdf.model.impl.PropertyImpl;
import com.hp.hpl.jena.vocabulary.DC;

// import org.apache.axis.utils.XMLUtils;

/**
 * @(#)OCModel.java 1.00 07/08/03
 *                  <p>
 *                  Title: OntoCover Model
 *                  </p>
 *                  <p>
 *                  Description: Loads and maintains the ontological tree
 *                  </p>
 *                  <p>
 *                  Copyright: Copyright (c) 2003
 *                  </p>
```

```
 *                  <p>
 *                  Company: IC-Unicamp and Embrapa
 *                  </p>
 * @author Renato Fileto and Lauro Ramos Venancio
 * @version 1.0
 */

public class OCModel {

      private Model modelRDF, modelRDFS;

      protected String nsRDF, // RDF NameSpace
                              nsRDFS, // RDF-Schema NameSpace
                              nsAPL,  // Application NameSpace
                              nsPTG;      // Protege NameSpace

      private PropertyImpl propSubClassOf, propNameBR, propDescr,
propType, propDomain,
                  propInvSlot, propRange;

      private Resource poesiaDimClass, poesiaRootClass, poesiaShowClass,
poesiaEncompassSlot;

      private Map<String, ArrayList<Resource>> classEncompassSlots;

      private Connection dbcon = null;

      private OCNode top = null;

      private int node_number = 1;

      private OCConfig config;

      private FileWriter xmlView;

      public OCModel() {
            try {
                  config = new OCConfig();

            } catch (Exception e) {
                  System.out.println("Arquivo de configuracao não
encontrado/ danificado");
            }

            clear();
      }

      private void clear() {
            top = null;
            modelRDF = modelRDFS = null;
            nsRDF = nsRDFS = nsAPL = nsPTG = "";
            propSubClassOf = propNameBR = propDescr = propType =
propDomain = propInvSlot = propRange = null;
            classEncompassSlots = null;
            poesiaDimClass = poesiaRootClass = poesiaShowClass =
poesiaEncompassSlot = null;
```

```
	}

	/**
	 * Informa se foi carregada a Ontologia
	 * @return
	 * boolean
	 */
	public boolean isEmpty() {
		return (top == null);
	}

	/**
	 *           Connect to database instances holding Jena Models for
the
	 *           previously
	 *           loaded RDF and RDF-Schema files

	 * @throws OCException
	 * connectDB
	 * void
	 */
	public void connectDB() throws OCException {
		try {

			// Connect to database instances holding Jena Models
for the
			// previously
			// loaded RDF and RDF-Schema files
			//

			/*
			 * System.out.println("Connecting to previously loaded
DB ...");
			 * DBConnection dbconRDF = new DBConnection(
			 * "jdbc:mysql://localhost/PoesiaRDF", "root", "");
modelRDF =
			 * ModelRDB.open(dbconRDF, "Mysql"); DBConnection
dbconRDFS = new
			 * DBConnection( "jdbc:mysql://localhost/PoesiaRDFS",
"root", "");
			 * modelRDFS = ModelRDB.open(dbconRDFS, "Mysql");
			 */

			System.out.println("Connecting to previously loaded DB
...");

			/*
			 * DBConnection dbconRDF = new DBConnection(
			 *
"jdbc:postgresql://apoena.lis.dcc.unicamp.br/PoesiaRDF",
			 * "poesia", ""); // This function was call just to set
the jdbc
			 * driver because of a bug in Jena.
			 * dbconRDF.getDriver("Generic","Postgresql"); if
			 * (dbconRDF.isFormatOK()) modelRDF =
ModelRDB.open(dbconRDF);
			 * DBConnection dbconRDFS = new DBConnection(
```

```
                  *
"jdbc:postgresql://apoena.lis.dcc.unicamp.br/PoesiaRDFS",
                  * "poesia", ""); // This function was call just to set
the jdbc
                  * driver because of a bug in Jena.
                  * dbconRDFS.getDriver("Generic","Postgresql"); if
                  * (dbconRDFS.isFormatOK()) modelRDFS =
ModelRDB.open(dbconRDFS); //
                  * Build the tree model // setModelVariables();
buildRDFProjTree();
                  */

            } catch (Exception error1) {
                  error1.printStackTrace();
                  throw new OCException(error1.getMessage());
            } finally {

                  // Clear RDF and RDFS models
                  //
                  modelRDF.close();
                  modelRDFS.close();

            }
      }
      /**
       *
       *
       * encerra a sessão criada ao carregar a Ontologia na memória/BD.
       */
      public void closeOntology(){
            if (modelRDF != null) {
                  modelRDF.close();
                  modelRDFS.close();
                  System.out.println("ontologia fechada");
            }else{
                  System.out.println("nao foi aberta ainda");
            }
      }

      /**
       * Executa uma Query SPARQL na Ontologia
       *
       * @param query: instrução no padrão SPARQL
       *
       * @return <b>List</b>: Lista composta por elementos do tipo
OCLinha com os resultados da busca.
       * O primeiro elemento da lista é os metadados, os demais elementos
são os valores que satisfazem a query.
       */
      public List<OCLinha> runQuery(String query){

//          String NL = System.getProperty("line.separator") ;
//          String prefix = "PREFIX dc: <"+DC.getURI()+">" ;

//          String querySPARQL = NL + prefix + query;
```

```
            QueryExecution ocQe = QueryExecutionFactory.create(query,
modelRDFS);
            com.hp.hpl.jena.query.ResultSet ocResults =
ocQe.execSelect();
//          ocResults.getResultVars();

            List<OCLinha> tabela = OCTabela.setResultados(ocResults);
            ocQe.close();

            return tabela;
      }

      /**
       *
       *
       * Carrega uma Ontologia padrão.
       */
      public void loadDefaultTree() {
            clear();
/*          top = new OCNode(null, "PoesiaDefault", Color.white,
Color.black);
            DefaultTree.create(top);*/

            //TODO : descomentar o metodo abaixo
            //por motivos de teste esta iniciando com uma Ontologia mais
leve
            buildDefaultTree();
      }

      /**
       *
       * @return
       * buildDefaultTree
       * boolean
       */
      private boolean buildDefaultTree() {

            Boolean resultado =false;
            try {
                  //identifica o nome que deve ser atribuido ao xml da AH
                  System.out.println("carregando config.getRDFs()");

                  String home =
System.getProperties().getProperty("user.dir");
                  String path = home.replace("\\", "/") +
"/"+config.getRDFs();
                  System.out.println(path);
                  loadRDF( path, false, false);
                  resultado = true;
            } catch (Exception e) {
                  e.printStackTrace();
            }

            return resultado;
      }

      //trabalha com o esquema(rdfs)
```

```
	/**
	 * Método para carregar o RDF no modelo
	 * @param pathRDFS: caminho do arquivo da Ontologia
	 * @param loadIntoDB: boolean para carregar a Ontologia no BD ao
invés da Memória
	 * @param instanceBD: boolean para indicar se a intancia esta no BD
	 */
	public void loadRDF(String pathRDFS, boolean loadIntoDB, boolean
instanceDB) {
		URL sourceRDFS = createURL(pathRDFS);
		URL sourceRDF = null;
		if (!instanceDB) {
			//se a instancia nao estiver no banco carrega
			String home =
System.getProperties().getProperty("user.dir");
			String path = home.replace("\\", "/") +
"/"+config.getRDF();
			sourceRDF = createURL(path);
		}
		loadInstanceRDF(sourceRDF, sourceRDFS, loadIntoDB);
	}

	//trabalha com a instancia(rdf)
	private void loadInstanceRDF(URL sourceRDF, URL sourceRDFS, boolean
loadIntoDB) {
		if (sourceRDFS != null) {
			try {
				if (loadIntoDB) {
					//carrega o modelo no BD ao inves da Memória
					loadOntologyIntoDB(sourceRDF);
				} else {

					// Create Jena models in memory to hold the
RDF and
					// RDF-Schema
					//
					System.out.println("Loading model in memory
...");
					modelRDF =
ModelFactory.createDefaultModel();
					modelRDFS =
ModelFactory.createDefaultModel();

				}

				// Load the RDF and the RDF-Schema files into the
respective
				// Jenal models
				//
				if (sourceRDF != null) {
					Reader readerRDF = new
InputStreamReader(sourceRDF.openStream(), "UTF-8");
					modelRDF.read(readerRDF,
sourceRDF.toString());
				} else
					modelRDF = null;
```

```
                        Reader readerRDFS = new
InputStreamReader(sourceRDFS.openStream(), "UTF-8");
                        modelRDFS.read(readerRDFS, sourceRDFS.toString());

                        // Build the tree model
                        //
                        setModelVariables();
                        if (modelRDF == null) {
                              try {

      Class.forName("org.postgresql.Driver");
                                    dbcon =
DriverManager.getConnection("jdbc:postgresql://apoena.lis.dcc.unicamp.br/
PoesiaAZ","poesia", "");
                              } catch (ClassNotFoundException ex4) {
                                    System.err.println(ex4);
                              }

                        }
                        //inicia o processo de geracao do xml para a
arvore hiperbolica
                        buildRDFProjTree();

                        if (dbcon != null)
                              dbcon.close();

                  } catch (Exception e) {
                        System.err.println(e/* .getMessage() */);
                  }
            }
      }

      private void loadOntologyIntoDB(URL sourceRDF) {
            // Format database instances to hold Jena Models for the RDF
            // and RDF-Schema
            //
            /*
             * System.out.println("Loading model into DB ...");
             * DBConnection dbconRDF = new DBConnection(
             * "jdbc:mysql://localhost/PoesiaRDF", "root", ""); modelRDF
=
             * ModelRDB.create(dbconRDF, "Generic", "Mysql");
             * DBConnection dbconRDFS = new DBConnection(
             * "jdbc:mysql://localhost/PoesiaRDFS", "root", "");
             * modelRDFS = ModelRDB.create(dbconRDFS, "Generic",
             * "Mysql");
             */

            System.out.println("Loading model into DB ...");
            if (sourceRDF != null) {
                  DBConnection dbconRDF = Conexao.getConexaoRDF();
                  if (dbconRDF.isFormatOK())

                        modelRDF = ((ModelMaker)
ModelFactory.createModelRDBMaker(dbconRDF)).createModel(config.getBdTipo(
), true);
```

```
            } else
                  modelRDF = null;
            DBConnection dbconRDFS = Conexao.getConexaoRDFs();

            if (dbconRDFS.isFormatOK())
                  modelRDFS = ((ModelMaker)
ModelFactory.createModelRDBMaker(dbconRDFS)).createModel(config.getBdTipo
(), true);
      }

      private URL createURL(String sourceName) {
            URL source = null;
            try {
                  source = new URL(sourceName);
            } catch (MalformedURLException e1) {
                  try {
                        source = new URL("file", null, sourceName);
                  } catch (MalformedURLException e2) {
                        System.err.println(e2.getMessage());
                  }
            }
            return source;
      }

      private void setModelVariables() throws OCException {

            try {

                  // Set OCModel's namespaces
                  //
                  TreeSet<String> set = new TreeSet<String>();
                  StmtIterator iter = modelRDFS.listStatements();
                  while (iter.hasNext()) {
                        Statement stmt = (Statement) iter.next();
                        set.add(stmt.getSubject().getNameSpace());
                        set.add(stmt.getPredicate().getNameSpace());
                        RDFNode node = stmt.getObject();
                        if (node instanceof Resource) {
                              set.add(((Resource) node).getNameSpace());
                        }
                  }
                  iter.close();
                  Iterator<String> nsiter = set.iterator();
                  while (nsiter.hasNext()) {
                        String ns = (String) nsiter.next();
                        if (ns.indexOf("http://www.w3.org/") != -1) {
                              if (ns.indexOf("rdf-syntax-ns") != -1) {
                                    nsRDF = ns;
                              } else if (ns.indexOf("rdf") != -1) {
                                    nsRDFS = ns;
                              }
                        } else {
                              if
(ns.indexOf("http://protege.stanford.edu/") != -1) {
                                    nsPTG = ns;
                              } else {
                                    nsAPL = ns;
```

```
                        }
                    }
                }

                // Set properties used to load the RDF into the Tree
                //
                propSubClassOf = new PropertyImpl(nsRDFS,
"subClassOf");
                propNameBR = new PropertyImpl(nsAPL, "nameBR");
                propDescr = new PropertyImpl(nsAPL, "Descricao");
                propType = new PropertyImpl(nsRDF, "type");
                propDomain = new PropertyImpl(nsRDFS, "domain");
                propInvSlot = new PropertyImpl(nsPTG,
"inverseProperty");
                propRange = new PropertyImpl(nsRDFS, "range");

                // You could also print the model as seen by Jena in
order to check
                // it
                //
//                printRDFNamespaces(modelRDFS);
//                printRDFModel(modelRDF);
//                printRDFTriples(modelRDF);
//                printRDFNamespaces(modelRDFS);
//                printRDFModel(modelRDFS);
//                printRDFTriples(modelRDFS);
                // printNameSpaces();

            } catch (Exception error1) {
                error1.printStackTrace();
                throw new OCException(error1.getMessage());
            }
      }

      private void buildRDFProjTree(){
            poesiaDimClass = modelRDFS.getResource(nsAPL + "POESIA-DIM-
CLASS");
            poesiaRootClass = modelRDFS.getResource(nsAPL + "POESIA-ROOT-
CLASS");
            poesiaShowClass = modelRDFS.getResource(nsAPL + "POESIA-SHOW-
CLASS");
            poesiaEncompassSlot = modelRDFS.getResource(nsAPL + "POESIA-
ENCOMPASS-SLOT");
            classEncompassSlots = new TreeMap<String,
ArrayList<Resource>>();

            //gera a arvore hiperbolica e a visao em xml
            gerarXMLparaArvoreHiperbolica();

      }

      private void gerarXMLparaArvoreHiperbolica() {
            try {
                Color cor[][] =
                            {     { Color.cyan,     Color.black },
                                  { Color.pink,     Color.black },
                                  { Color.yellow, Color.black },
```

```
                                    { Color.green,    Color.black } };

                  xmlView = new FileWriter(config.getArquivoDeSaida(),
false);

                  top = new OCNode(null, getAplicationURI(), Color.white,
Color.black);

                  System.out.println("Building tree ...");
                  ResIterator iterDim =
modelRDFS.listSubjectsWithProperty(propType,poesiaDimClass);
                  int i = 0;
                  while (iterDim.hasNext()) {
                        Resource resDim = (Resource) iterDim.next();

                        xmlView.write("<node id=\"Xabc123456789"); //
generate XML
                        xmlView.write(String.valueOf(node_number++)); //
generate XML
                        xmlView.write("\">\n"); // generate XML
                        xmlView.write("<label>"); // generate XML

      xmlView.write(cleanStr(resDim.getLocalName()).replaceAll("_",""));
// generate XML
                        xmlView.write("</label>\n"); // generate XML
                        xmlView.write("<content>"); // generate XML
                        xmlView.write("[Nível:"); // generate XML

      xmlView.write(cleanStr(resDim.getLocalName().replaceAll("_", "")));
// generate XML
                        xmlView.write("]  "); // generate XML
                        xmlView.write("</content>\n"); // generate XML

                        OCNode noDim =
top.addChild(resDim.getLocalName().replaceAll("_", ""), cor[i][0],
cor[i][1]);
                        addSubClassesTree(noDim, resDim, true, cor[i]);
                        i = (i + 1) % cor.length;

                        xmlView.write("</node>\n"); // generate XML
                  }
                  iterDim.close();
                  System.out.println("Tree is built !!!");

                  xmlView.close();

            } catch (IOException e) {
                  System.err.println("Cannot write on file
"+config.getArquivoDeSaida()+" !!!");
            }
      }

      private void addSubClassesTree(OCNode node, Resource res,
                  boolean searchRoot, Color[] cor){
            ResIterator iterSub = modelRDFS.listSubjectsWithProperty(
                        propSubClassOf, res);
            while (iterSub.hasNext()) {
```

```
                  Resource resSub = (Resource) iterSub.next();
                  if (searchRoot) {
                        if (resSub.hasProperty(propType, poesiaRootClass))
{

                              try {

                                    xmlView.write("<node
id=\"Xabc123456789"); // generate

                              // XML

      xmlView.write(String.valueOf(node_number++)); // generate

                                    // XML
                                    xmlView.write("\">\n"); // generate
XML
                                    xmlView.write("<label>"); // generate
XML

      xmlView.write(cleanStr(resSub.getLocalName()
                                          .replaceAll("_", ""))); //
generate XML
                                    xmlView.write("</label>\n"); //
generate XML
                                    xmlView.write("<content>"); //
generate XML
                                    xmlView.write("[Nível:"); // generate
XML

      xmlView.write(cleanStr(resSub.getLocalName()
                                          .replaceAll("_", ""))); //
generate XML
                                    xmlView.write("]  "); // generate XML
                                    xmlView.write("</content>\n"); //
generate XML

                                    OCNode nodeSub =
node.addChild(resSub.getLocalName()
                                          .replaceAll("_", ""),
cor[0], cor[1]);
                                    addSubClassesTree(nodeSub, resSub,
false, cor);
                                    addInstancesTree(nodeSub, resSub,
cor);

                                    xmlView.write("</node>\n"); //
generate XML

                              }

                              catch (IOException e) {
                                    System.err
                                          .println("Cannot write on
file \"treeView.xml\" !!!");
                              }
```

```
                        } else {
                              addSubClassesTree(node, resSub, true, cor);
                        }
                  } else {
                        if (resSub.hasProperty(propType, poesiaShowClass))
{

                              try {

                                    xmlView.write("<node
id=\"Xabc123456789"); // generate

                              // XML

      xmlView.write(String.valueOf(node_number++)); // generate

                                    // XML
                                    xmlView.write("\">\n"); // generate
XML
                                    xmlView.write("<label>"); // generate
XML

      xmlView.write(cleanStr(resSub.getLocalName()
                                          .replaceAll("_", ""))); //
generate XML
                                    xmlView.write("</label>\n"); //
generate XML
                                    xmlView.write("<content>"); //
generate XML
                                    xmlView.write("[Nível:"); // generate
XML

      xmlView.write(cleanStr(resSub.getLocalName()
                                          .replaceAll("_", ""))); //
generate XML
                                    xmlView.write("]  "); // generate XML
                                    xmlView.write("</content>\n"); //
generate XML

                                    OCNode nodeSub =
node.addChild(resSub.getLocalName()
                                          .replaceAll("_", ""),
cor[0], cor[1]);
                                    addSubClassesTree(nodeSub, resSub,
false, cor);
                                    addInstancesTree(nodeSub, resSub,
cor);

                                    xmlView.write("</node>\n"); //
generate XML

                              }

                              catch (IOException e) {
                                    System.err
                                          .println("Cannot write on
file \"treeView.xml\" !!!");
```

```
                              }

                        } else {
                              addSubClassesTree(node, resSub, false, cor);
                        }
                  }
            }
            iterSub.close();
            // if (res.hasProperty(RDF.type,
            // modelRDFS.getResource(nsAPL+"POESIA-ROOT-CLASS")) ||
            // res.hasProperty(RDF.type,
            // modelRDFS.getResource(nsAPL+"POESIA-SHOW-CLASS"))) {
            // }
      }

      private void addInstancesTree(OCNode node, Resource res, Color[]
cor){
            if (modelRDF == null) {
                  java.sql.Statement stmt = null;
                  ResultSet inst = null;
                  try {
                        stmt = dbcon.createStatement();
                        inst = stmt.executeQuery("SELECT " +
res.getLocalName()
                                    + ",NAMEBR FROM " + res.getLocalName()
                                    + " ORDER BY NAMEBR");
                        while (inst.next()) {
                              addDbNodeIntancesTree(node, res, inst, cor,
dbcon);
                        }

                  } catch (SQLException ex) {
                        System.err.println("Erro SQL: " + ex);
                  } finally {
                        if (inst != null) {
                              try {
                                    inst.close();
                              } catch (SQLException ex1) {
                              }
                        }
                        if (stmt != null) {
                              try {
                                    stmt.close();
                              } catch (SQLException ex2) {
                              }
                        }

                  }

            } else {
                  ResIterator iterInst =
modelRDF.listSubjectsWithProperty(propType,
                              res);
                  while (iterInst.hasNext()) {
                        Resource resInst = (Resource) iterInst.next();
                        addNodeInstanceTree(node, res, resInst, cor);
                  }
```

```
                iterInst.close();
          }
     }

     private void addNodeInstanceTree(OCNode node, Resource resClass,
                Resource resInst, Color[] cor){
          if (resInst.hasProperty(propNameBR)) {

                try {
                     xmlView.write("<node id=\"Xabc123456789"); //
generate XML
                     xmlView.write(String.valueOf(node_number++)); //
generate XML
                     xmlView.write("\">\n"); // generate XML
                     xmlView.write("<label>"); // generate XML

     xmlView.write(cleanStr(resInst.getProperty(propNameBR)
                                    .getObject().toString())); // generate
XML
                     xmlView.write("</label>\n"); // generate XML
                     xmlView.write("<content>"); // generate XML
                     xmlView.write("[Nível:"); // generate XML

     xmlView.write(cleanStr(resClass.getLocalName().replaceAll("_",
                                    ""))); // generate XML
                     xmlView.write("]  "); // generate XML
                     if (resInst.hasProperty(propDescr)) { // generate
XML
                          // XMLview.write("\n"); // generate XML

     xmlView.write(cleanStr(resInst.getProperty(propDescr)
                                         .getObject().toString())); //
generate XML
                     }
                     xmlView.write("</content>\n"); // generate XML

                     OCNode subNode =
node.addChild(resClass.getLocalName()
                                    .replaceAll("_", ""),
resInst.getProperty(propNameBR)
                                    .getObject().toString(), cor[0],
cor[1]);

                     Iterator<Resource> iter =
findEncompassSlots(resClass).iterator();
                     while (iter.hasNext()) {
                          StmtIterator stmts = resInst
                                    .listProperties(new
PropertyImpl(((Resource) iter
                                                  .next()).getURI()));
                          while (stmts.hasNext()) {
                               Statement stmt = (Statement)
stmts.next();
                               // Resource slotRDFS =
                               //
modelRDFS.getResource(stmt.getPredicate().toString());
```

```
                                    // if (slotRDFS.hasProperty(propType))
{
                                    // if
                                    //
(slotRDFS.getProperty(propType).getObject().toString().
                                    // equals(nsAPL + "POESIA-ENCOMPASS-
SLOT")) {
                                    Resource resSubInst = (Resource)
stmt.getObject();
                                    if (resSubInst.hasProperty(propType))
{
                                          Resource resSubClass =
(Resource) resSubInst

      .getProperty(propType).getObject();
                                          addNodeInstanceTree(subNode,
resSubClass,
                                                      resSubInst, cor);
                                    }
                                    // }
                                    // }
                              }
                              stmts.close();
                        }

                        xmlView.write("</node>\n"); // generate XML

                  }

                  catch (IOException e) {
                        System.err.println("Cannot write on file
\"treeView.xml\" !!!");
                  }

            }

      }

      private String getAplicationURI() {
            return (nsAPL);
      }

      private ArrayList<Resource> findEncompassSlots(Resource resClass) {
            ArrayList<Resource> encompassSlots =
classEncompassSlots.get(resClass.getURI());
            if (encompassSlots == null) {
                  encompassSlots = new ArrayList<Resource>();
                  ResIterator iterSlot =
modelRDFS.listSubjectsWithProperty(
                              propDomain, resClass);
                  while (iterSlot.hasNext()) {
                        Resource slot = (Resource) iterSlot.next();
                        if (modelRDFS.contains(slot, propType,
poesiaEncompassSlot)) {
                              encompassSlots.add(slot);
                        }
                  }
```

```
                  iterSlot.close();
                  if (resClass.hasProperty(propSubClassOf)) {

      encompassSlots.addAll(findEncompassSlots((Resource) resClass

      .getProperty(propSubClassOf).getObject()));
                  }
                  classEncompassSlots.put(resClass.getURI(),
encompassSlots);
            }
            return encompassSlots;
      }

      private void addDbNodeIntancesTree(OCNode node, Resource resClass,
                  ResultSet inst, Color[] cor, Connection dbcon){

            OCNode subNode = null;
            try {

                  try {

                        xmlView.write("<node id=\"Xabc123456789"); //
generate XML
                        xmlView.write(String.valueOf(node_number++)); //
generate XML
                        xmlView.write("\">\n"); // generate XML
                        xmlView.write("<label>"); // generate XML
                        xmlView.write(cleanStr(inst.getString("NAMEBR")));
// generate

                              // XML
                        xmlView.write("\n</label>\n"); // generate XML
                        xmlView.write("<content>"); // generate XML
                        xmlView.write("[Nível:"); // generate XML

      xmlView.write(cleanStr(resClass.getLocalName().replaceAll("_",
                                    "")));  // generate XML
                        xmlView.write("]  "); // generate XML
                        // XMLview.write("\n"); // generate XML

      xmlView.write(cleanStr(inst.getString("Descricao"))); // generate

                                    // XML
                        xmlView.write("</content>\n"); // generate XML

                        subNode =
node.addChild(resClass.getLocalName().replaceAll("_",
                                    ""), inst.getString("NAMEBR"), cor[0],
cor[1]);

                        Iterator<Resource> iter =
findEncompassSlots(resClass).iterator();
                        while (iter.hasNext()) {
                              Resource slot = (Resource) iter.next();
                              Resource inverseSlot = (Resource)
modelRDFS.getProperty(
                                          slot, propInvSlot).getObject();
```

```
                              Resource targetClass = (Resource)
modelRDFS.getProperty(
                                          slot, propRange).getObject();
                              java.sql.Statement stmt =
dbcon.createStatement();
                              String tabela = targetClass.getLocalName();
                              ResultSet subInst =
stmt.executeQuery("SELECT " + tabela
                                          + ",NAMEBR FROM " + tabela + "
WHERE "
                                          + inverseSlot.getLocalName() +
"="
                                          +
inst.getObject(resClass.getLocalName())
                                          + " ORDER BY NAMEBR");
                              while (subInst.next()) {
                                    addDbNodeIntancesTree(subNode,
targetClass, subInst,
                                                cor, dbcon);
                              }
                              subInst.close();
                              stmt.close();
                        }
                        xmlView.write("</node>\n"); // generate XML

                  }

                  catch (IOException e) {
                        System.err.println("Cannot write on file
\"treeView.xml\" !!!");
                  }

            } catch (SQLException ex) {
                  System.err.println(ex);
            }

      }

      /**
       * Retorna uma implementação da interface OCTree a partir da
Ontologia carregada.
       *
       * @param boolean isHyperbolic: <ui><li>true: instância da OCTree
do tipo OCTreebolicFactory (Árvore Hiperbólica)</li>
       *  <li>false: instância da OCTree do tipo OCJTree (Pastas)</li>
       * @return OCTree: retorna uma implementação da interface OCTree do
tipo Árvore Hiperbólica ou Pastas.
       */
      public OCTree getOCTree(boolean isHyperbolic) {
            OCTree ocTree = null;
            if (top != null) {
                  if (isHyperbolic) {
                        Treebolic treebolicTree = new Treebolic();
                        OCTreebolicFactory factory = new
OCTreebolicFactory(top,treebolicTree);
                        treebolicTree.setStub(factory);
                        treebolicTree.setStandalone(true);
```

```
                        treebolicTree.init(factory);

                        ocTree = factory;
                  } else {
                        OCJTree ocJTree = new OCJTree(top);
                        ocJTree.revalidate();
                        ocTree = ocJTree;
                  }
            }
            return ocTree;
      }

      private String cleanStr(String strIn) {
            String strOut;
            strOut = strIn.replaceAll("<=", " menor ou igual a ");
            strOut = strOut.replaceAll(">=", " maior ou igual a ");
            strOut = strOut.replaceAll("=<", " menor ou igual a ");
            strOut = strOut.replaceAll("=>", " maior ou igual a ");
            strOut = strOut.replaceAll("<", " menor que ");
            strOut = strOut.replaceAll(">", " maior que ");
            strOut = strOut.replaceAll("\n", " ");
            return (strOut);
      }
}

package OntoCover;

import java.awt.*;


/**
 * @(#)OCTest.java      1.00 07/08/03
 * <p>Title: OntoCover Test </p>
 * <p>Description: Ontological tree</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: IC-Unicamp and Embrapa </p>
 * @author Lauro Ramos Venancio and Renato Fileto
 * @version 1.0
 */


public interface OCTree {

//  public OCNode addChild(OCNode node, String ClassName);
//  public OCNode addChild(OCNode node, String ClassName, String
InstanceName);
//  public OCTerm[] getSelectedTerms ();
  public OCTerm getSelectedTerm ();
  public OCTuple getSelectedTuple ();
  public void setMultiSelection (boolean multiSelection);
  public Component getVisualTree ();
}


package OntoCover;

import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;

/**
 * @(#)OCChoose.java    1.00 07/08/03
 * <p>Title: OntoCover Choose Dialog</p>
 * <p>Description: Presents an ontological tree maintained in an OCModel
 *                 allowing the user to choose an ontological coverage
 *                 from that ontological tree</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: IC-Unicamp and Embrapa </p>
 * @author Renato Fileto
 * @version 1.0
 */


public class OCChoose extends JDialog {

  /**
       *
       */
      private static final long serialVersionUID = 1L;
// Possible results of the dialog
  //
  public static final int SET = 1;
  public static final int ADD = 2;
  public static final int CANCEL = 3;

  // Effective result of the dialog
  //
  private int answer = CANCEL;

  private boolean showHyper;

  private OCModel ocModel;
  private OCTree ocTree;

  private JPanel topPanel = new JPanel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JPanel treePanel = new JPanel();
  private BorderLayout borderLayout2 = new BorderLayout();
  private JPanel controlPanel = new JPanel();
  private JPanel setTreePanel = new JPanel();
  private BorderLayout borderLayout3 = new BorderLayout();
  private JPanel buttonsPanel = new JPanel();
  private JRadioButton jRadioButtonHyper = new JRadioButton();
  private JRadioButton jRadioButtonConv = new JRadioButton();
  private JButton jButtonSet = new JButton();
  private JButton jButtonAdd = new JButton();
  private JButton jButtonCancel = new JButton();
  private BorderLayout borderLayout4 = new BorderLayout();
  private ButtonGroup buttonGroup1 = new ButtonGroup();

  public OCChoose() {
    try {
      jbInit();
```

```
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  public OCChoose(OCModel ocModel) throws HeadlessException {
    this.ocModel = ocModel;
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    showTree();
  }

  public OCChoose(OCModel ocModel, JDialog owner) throws
HeadlessException {
    super(owner);
    this.ocModel = ocModel;
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    showTree();
  }

  public OCChoose(OCModel ocModel, JDialog owner, boolean modal) throws
HeadlessException {
    super(owner, modal);
    this.ocModel = ocModel;
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    showTree();
  }

  private void jbInit() throws Exception {
    topPanel.setLayout(borderLayout1);
    topPanel.setBorder(BorderFactory.createRaisedBevelBorder());
    topPanel.setDoubleBuffered(false);
    topPanel.setMinimumSize(new Dimension(400, 310));
    topPanel.setPreferredSize(new Dimension(800, 600));
    treePanel.setLayout(borderLayout2);
    controlPanel.setDoubleBuffered(false);
    controlPanel.setMinimumSize(new Dimension(350, 40));
    controlPanel.setPreferredSize(new Dimension(350, 40));
    controlPanel.setLayout(borderLayout3);
    setTreePanel.setBorder(BorderFactory.createEtchedBorder());
    setTreePanel.setDoubleBuffered(false);
    setTreePanel.setMinimumSize(new Dimension(130, 40));
```

```
    setTreePanel.setPreferredSize(new Dimension(130, 40));
    setTreePanel.setLayout(borderLayout4);
    jRadioButtonHyper.setMinimumSize(new Dimension(107, 25));
    jRadioButtonHyper.setPreferredSize(new Dimension(107, 18));
    jRadioButtonHyper.setToolTipText("View ontology as hyperbolic tree");
    jRadioButtonHyper.setMnemonic('0');
    jRadioButtonHyper.setSelected(false);
    jRadioButtonHyper.setText("Hyperbolic tree");
    jRadioButtonHyper.addChangeListener(new
OCChoose_jRadioButtonHyper_changeAdapter(this));
    jRadioButtonHyper.addMouseListener(new
OCChoose_jRadioButtonHyper_mouseAdapter(this));
    jRadioButtonHyper.addActionListener(new
OCChoose_jRadioButtonHyper_actionAdapter(this));
    jRadioButtonConv.setAlignmentX((float) 0.0);
    jRadioButtonConv.setMinimumSize(new Dimension(121, 25));
    jRadioButtonConv.setPreferredSize(new Dimension(121, 18));
    jRadioButtonConv.setToolTipText("View ontology as convetional tree");
    jRadioButtonConv.setSelected(true);
    jRadioButtonConv.setText("Conventional tree");
    jRadioButtonConv.addChangeListener(new
OCChoose_jRadioButtonConv_changeAdapter(this));
    jRadioButtonConv.addMouseListener(new
OCChoose_jRadioButtonConv_mouseAdapter(this));
    jRadioButtonConv.addActionListener(new
OCChoose_jRadioButtonConv_actionAdapter(this));
    treePanel.setBorder(BorderFactory.createEtchedBorder());
    treePanel.setMinimumSize(new Dimension(350, 250));
    treePanel.setPreferredSize(new Dimension(350, 250));
    buttonsPanel.setBorder(BorderFactory.createEtchedBorder());
    buttonsPanel.setDoubleBuffered(false);
    buttonsPanel.setMinimumSize(new Dimension(205, 40));
    buttonsPanel.setPreferredSize(new Dimension(205, 40));
    jButtonSet.setToolTipText("Set ontological coverage to the selected
terms");
    jButtonSet.setText("Set");
    jButtonSet.addActionListener(new
OCChoose_jButtonSet_actionAdapter(this));
    jButtonAdd.setToolTipText("Add selected terms to ontological
coverage");
    jButtonAdd.setText("Add");
    jButtonAdd.addActionListener(new
OCChoose_jButtonAdd_actionAdapter(this));
    jButtonCancel.setToolTipText("Do not change the ontological
coverage");
    jButtonCancel.setText("Cancel");
    jButtonCancel.addActionListener(new
OCChoose_jButtonCancel_actionAdapter(this));
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    this.setSize(new Dimension(screenSize.width, screenSize.height));
    this.setTitle("Choose an ontological coverage");
    this.getContentPane().add(topPanel, BorderLayout.CENTER);
    topPanel.add(treePanel, BorderLayout.CENTER);
    topPanel.add(controlPanel,  BorderLayout.SOUTH);
    controlPanel.add(setTreePanel,  BorderLayout.WEST);
    controlPanel.add(buttonsPanel, BorderLayout.CENTER);
    setTreePanel.add(jRadioButtonHyper,  BorderLayout.SOUTH);
```

```
    setTreePanel.add(jRadioButtonConv, BorderLayout.CENTER);
    buttonsPanel.add(jButtonSet, null);
    buttonsPanel.add(jButtonAdd, null);
    buttonsPanel.add(jButtonCancel, null);
    buttonGroup1.add(jRadioButtonHyper);
    buttonGroup1.add(jRadioButtonConv);

    showHyper = jRadioButtonHyper.isSelected();
  }

  void jRadioButtonHyper_actionPerformed(ActionEvent e) {
  }

  void jRadioButtonConv_actionPerformed(ActionEvent e) {
  }

  void jRadioButtonHyper_mouseClicked(MouseEvent e) {
//    showTree();
  }

  void jRadioButtonConv_mouseClicked(MouseEvent e) {
//    showTree();
  }

  void jRadioButtonHyper_stateChanged(ChangeEvent e) {
    if (jRadioButtonHyper.isSelected() && (! showHyper)) {
      showHyper = true;
      showTree();
    }
  }

  void jRadioButtonConv_stateChanged(ChangeEvent e) {
    if (jRadioButtonConv.isSelected() && showHyper) {
      showHyper = false;
      showTree();
    }
  }

  void showTree () {
    ocTree = ocModel.getOCTree(showHyper);
    treePanel.removeAll();
    if (ocTree != null) {
      if (showHyper) {
        treePanel.add(ocTree.getVisualTree(), BorderLayout.CENTER);
      } else {
        JScrollPane jScrollPaneTree = new JScrollPane();
        jScrollPaneTree.getViewport().add(ocTree.getVisualTree(), null);

jScrollPaneTree.setBorder(BorderFactory.createLineBorder(Color.black));
        jScrollPaneTree.setMaximumSize(new Dimension(32767, 32767));
        jScrollPaneTree.setMinimumSize(new Dimension(21, 21));
        jScrollPaneTree.setOpaque(true);
        jScrollPaneTree.setPreferredSize(new Dimension(350, 250));
        jScrollPaneTree.setRequestFocusEnabled(true);
        treePanel.add(jScrollPaneTree, BorderLayout.CENTER);
      }
      ocTree.setMultiSelection(true);
```

```
      treePanel.setVisible(false);
      treePanel.setVisible(true);
    }
  }

  void jButtonSet_actionPerformed(ActionEvent e) {
    answer = SET;
    dispose();
  }

  void jButtonAdd_actionPerformed(ActionEvent e) {
    answer = ADD;
    dispose();
  }

  void jButtonCancel_actionPerformed(ActionEvent e) {
    answer = CANCEL;
    dispose();
  }

  public int getAnswer () {
    return answer;
  }

  public OCTuple getSelectedTuple () {
    return ocTree.getSelectedTuple();
  }



private class OCChoose_jRadioButtonHyper_actionAdapter implements
java.awt.event.ActionListener {
  OCChoose adaptee;

  OCChoose_jRadioButtonHyper_actionAdapter(OCChoose adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jRadioButtonHyper_actionPerformed(e);
  }
}

private class OCChoose_jRadioButtonConv_actionAdapter implements
java.awt.event.ActionListener {
  OCChoose adaptee;

  OCChoose_jRadioButtonConv_actionAdapter(OCChoose adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jRadioButtonConv_actionPerformed(e);
  }
}

private class OCChoose_jButtonSet_actionAdapter implements
java.awt.event.ActionListener {
  OCChoose adaptee;
```

```
  OCChoose_jButtonSet_actionAdapter(OCChoose adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButtonSet_actionPerformed(e);
  }
}

private class OCChoose_jButtonAdd_actionAdapter implements
java.awt.event.ActionListener {
  OCChoose adaptee;

  OCChoose_jButtonAdd_actionAdapter(OCChoose adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButtonAdd_actionPerformed(e);
  }
}

private class OCChoose_jButtonCancel_actionAdapter implements
java.awt.event.ActionListener {
  OCChoose adaptee;

  OCChoose_jButtonCancel_actionAdapter(OCChoose adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jButtonCancel_actionPerformed(e);
  }
}

private class OCChoose_jRadioButtonHyper_mouseAdapter extends
java.awt.event.MouseAdapter {
  OCChoose adaptee;

  OCChoose_jRadioButtonHyper_mouseAdapter(OCChoose adaptee) {
    this.adaptee = adaptee;
  }
  public void mouseClicked(MouseEvent e) {
    adaptee.jRadioButtonHyper_mouseClicked(e);
  }
}

private class OCChoose_jRadioButtonConv_mouseAdapter extends
java.awt.event.MouseAdapter {
  OCChoose adaptee;

  OCChoose_jRadioButtonConv_mouseAdapter(OCChoose adaptee) {
    this.adaptee = adaptee;
  }
  public void mouseClicked(MouseEvent e) {
    adaptee.jRadioButtonConv_mouseClicked(e);
  }
}
```

```
private class OCChoose_jRadioButtonHyper_changeAdapter implements
javax.swing.event.ChangeListener {
  OCChoose adaptee;

  OCChoose_jRadioButtonHyper_changeAdapter(OCChoose adaptee) {
    this.adaptee = adaptee;
  }
  public void stateChanged(ChangeEvent e) {
    adaptee.jRadioButtonHyper_stateChanged(e);
  }
}

private class OCChoose_jRadioButtonConv_changeAdapter implements
javax.swing.event.ChangeListener {
  OCChoose adaptee;

  OCChoose_jRadioButtonConv_changeAdapter(OCChoose adaptee) {
    this.adaptee = adaptee;
  }
  public void stateChanged(ChangeEvent e) {
    adaptee.jRadioButtonConv_stateChanged(e);
  }
}
}
package OntoCover;

/**
 * @(#)OCException.java 1.00 07/08/03
 * <p>Title: OntoCover Exception </p>
 * <p>Description: Specific exceptions related with ontological
coverages</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: IC-Unicamp and Embrapa</p>
 * @author Renato Fileto
 * @version 1.0
 */

public class OCException extends Exception {

  public OCException() {
    super();
  }

  public OCException(String message) {
    super(message);
  }

  public OCException(String message, Throwable cause) {
    super(message, cause);
  }

  public OCException(Throwable cause) {
    super(cause);
  }
}
package OntoCover;
```

```
import java.io.*;
import java.util.*;

import javax.swing.filechooser.FileFilter;

/**
 * @(#)OCFileFilter.java     1.00 07/08/03
 * <p>Title: OntoCover File Filter </p>
 * <p>Description: File filter for choosing the ontological reference</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: IC-Unicamp and Embrapa</p>
 * @author Renato Fileto
 * @version 1.0
 *
 * Adapted from Sun's SDK 2 "ExampleFileFilter"
 *
 * Documentation of the original author:
 */

/**
 <p>A convenience implementation of FileFilter that filters out
 all files except for those type extensions that it knows about.</p>

<p> Extensions are of the type ".foo", which is typically found on
 Windows and Unix boxes, but not on Macinthosh. Case is ignored.</P>

<p> Example - create a new filter that filerts out all files
 but gif and jpg image files:</p>
<pre>
     JFileChooser chooser = new JFileChooser();
     OCFileFilter filter = new OCFileFilter(new String{"gif", "jpg"},
"JPEG & GIF Images")
     chooser.addChoosableFileFilter(filter);
     chooser.showOpenDialog(this);
</pre>
 * @version 1.12 12/03/01
 * @author Jeff Dinkins
 */
public class OCFileFilter extends FileFilter {

    private static String TYPE_UNKNOWN = "Type Unknown";
    private static String HIDDEN_FILE = "Hidden File";

    private Hashtable<String, OCFileFilter> filters = null;
    private String description = null;
    private String fullDescription = null;
    private boolean useExtensionsInDescription = true;

    /**
     *<p>Creates a file filter. If no filters are added, then all
     * files are accepted.</p>
     *
     * @see #addExtension
     */
    public OCFileFilter() {
        this.filters = new Hashtable<String, OCFileFilter>();
    }
```

```
    /**
     * Creates a file filter that accepts files with the given
extension.<br>
     * Example: new OCFileFilter("jpg");
     *
     * @see #addExtension
     */
    public OCFileFilter(String extension) {
        this(extension,null);
    }

    /**
     *<p> Creates a file filter that accepts the given file type.<br>
     * Example: new OCFileFilter("jpg", "JPEG Image Images");</p>
     *
     *<p> Note that the "." before the extension is not needed. If
     * provided, it will be ignored.</p>
     *
     * @see #addExtension
     */
    public OCFileFilter(String extension, String description) {
        this();
        if(extension!=null) addExtension(extension);
         if(description!=null) setDescription(description);
    }

    /**
     * <p>Creates a file filter from the given string array.<br>
     * Example: new OCFileFilter(String {"gif", "jpg"});</p>
     *
     * <p>Note that the "." before the extension is not needed adn
     * will be ignored.</p>
     *
     * @see #addExtension
     */
    public OCFileFilter(String[] filters) {
        this(filters, null);
    }

    /**
     * <p>Creates a file filter from the given string array and
description.<br>
     * Example: new OCFileFilter(String {"gif", "jpg"}, "Gif and JPG
Images");</p>
     *
     *<p>Note that the "." before the extension is not needed and will be
ignored.</p>
     *
     * @see #addExtension
     */
    public OCFileFilter(String[] filters, String description) {
        this();
        for (int i = 0; i < filters.length; i++) {
            // add filters one by one
            addExtension(filters[i]);
        }
```

```
         if(description!=null) setDescription(description);
    }

    /**
     * <p>Return true if this file should be shown in the directory pane,
     * false if it shouldn't.</p>
     *
     * <p>Files that begin with "." are ignored.</p>
     *
     * @see #getExtension
     *
     */
    public boolean accept(File f) {
        if(f != null) {
            if(f.isDirectory()) {
                return true;
            }
            String extension = getExtension(f);
            if(extension != null && filters.get(getExtension(f)) != null)
{
                return true;
            };
        }
        return false;
    }

    /**
     * <p>Return the extension portion of the file's name .</p>
     *
     *
     * @see #accept
     */
     public String getExtension(File f) {
        if(f != null) {
            String filename = f.getName();
            int i = filename.lastIndexOf('.');
            if(i>0 && i<filename.length()-1) {
                return filename.substring(i+1).toLowerCase();
            };
        }
        return null;
    }

    /**
     * <p>Adds a filetype "dot" extension to filter against.</p>
     *
     * <p>For example: the following code will create a filter that
filters
     * out all files except those that end in ".jpg" and ".tif":</p>
     *<pre>
     *   OCFileFilter filter = new OCFileFilter();
     *   filter.addExtension("jpg");
     *   filter.addExtension("tif");
     *</pre>
     * <p>Note that the "." before the extension is not needed and will
be ignored.</p>
     */
```

```
    public void addExtension(String extension) {
        if(filters == null) {
            filters = new Hashtable<String, OCFileFilter>(5);
        }
        filters.put(extension.toLowerCase(), this);
        fullDescription = null;
    }


    /**
     * <p>Returns the human readable description of this filter. For
     * example: "JPEG and GIF Image Files (*.jpg, *.gif)"</p>
     *
     * @see #setDescription
     * @see #setExtensionListInDescription
     * @see #isExtensionListInDescription
     *
     */
    public String getDescription() {
        if(fullDescription == null) {
            if(description == null || isExtensionListInDescription()) {
                 fullDescription = description==null ? "(" : description
+ " (";
                // build the description from the extension list
                Enumeration extensions = filters.keys();
                if(extensions != null) {
                    fullDescription += "." + (String)
extensions.nextElement();
                    while (extensions.hasMoreElements()) {
                        fullDescription += ", ." + (String)
extensions.nextElement();
                    }
                }
                fullDescription += ")";
            } else {
                fullDescription = description;
            }
        }
        return fullDescription;
    }

    /**
     *<p> Sets the human readable description of this filter. For
     * example: filter.setDescription("Gif and JPG Images");<p>
     *
     *
     * @see #setExtensionListInDescription
     * @see #isExtensionListInDescription
     */
    public void setDescription(String description) {
        this.description = description;
        fullDescription = null;
    }

    /**
     * <p>Determines whether the extension list (.jpg, .gif, etc) should
     * show up in the human readable description.</p>
```

```
     *
     * <p>Only relevent if a description was provided in the constructor
     * or using setDescription();</p>
     *
     * @see #getDescription
     * @see #setDescription
     * @see #isExtensionListInDescription
     */
    public void setExtensionListInDescription(boolean b) {
        useExtensionsInDescription = b;
        fullDescription = null;
    }

    /**
     * <p>Returns whether the extension list (.jpg, .gif, etc) should
     * show up in the human readable description.</p>
     *
     * <p>Only relevent if a description was provided in the constructor
     * or using setDescription();</p>
     *
     * @see #getDescription
     * @see #setDescription
     * @see #setExtensionListInDescription
     */
    public boolean isExtensionListInDescription() {
        return useExtensionsInDescription;
    }
}

package OntoCover;

import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;

//import javax.swing.tree.DefaultTreeSelectionModel;

/**
 * @(#)OCJTree.java    1.00 07/08/03
 * <p>Title: OntoCover JTree</p>
 * <p>Description: Presents an ontological tree maintained in an OCModel
 *                  as a JTree</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: IC-Unicamp and Embrapa </p>
 * @author Renato Fileto
 * @version 1.0
 */

public class OCJTree extends JTree implements OCTree {

      /**
       *
       */
      private static final long serialVersionUID = 1L;

      public OCJTree() {
            super();
```

```
	}

	public OCJTree(OCNode top) {
		super(top);
	}

	public void setMultiSelection(boolean multiSelection) {
		if (multiSelection) {
			getSelectionModel().setSelectionMode
			(TreeSelectionModel.DISCONTIGUOUS_TREE_SELECTION);
		}
		else {
			getSelectionModel().setSelectionMode
			(TreeSelectionModel.SINGLE_TREE_SELECTION);
		}
	}

	public OCTerm[] getSelectedTerms() {
		OCTerm[] terms;
		TreePath[] paths = this.getSelectionPaths();
		if (paths == null) {
			terms = new OCTerm[1];
			terms[0] = new OCTerm(null);
		}
		else {
			terms = new OCTerm[paths.length];
			for (int i=0;i<paths.length;i++)
				terms[i] =
((OCNode)paths[i].getLastPathComponent()).getTerm();
		}
		return terms;
	}

	public OCTerm getSelectedTerm() {
		return getSelectedTerms()[0];
	}

	public OCTuple getSelectedTuple() {
		return new OCTuple(getSelectedTerms());
	}

	public Component getVisualTree () {
		return this;
	}

}
package OntoCover;

import java.util.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.tree.*;

import treebolic.kernel.*;
```

```
/**
 * @(#)OCNode.java      1.00 07/08/03
 * <p>Title: OntoCover Node </p>
 * <p>Description: Node of an ontological tree</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: IC-Unicamp and Embrapa </p>
 * @author Lauro Ramos Venancio
 * @version 1.0
 */

/**
 *<p>Node of an ontological tree.</p>
 *
 * @see OCTree
 *
 */
public class OCNode implements TreeNode,INode {
      /** The Children of node.   */
      private Vector<INode> theChildren = null;
      /** The Parent of node.   */
      private OCNode theParent = null;
      /** The HyperCircle of node. This variable is used by Hyperbolic
Tree. */
      private HyperCircle theHyperCircle;
      /** The Weight of node. This variable is used by Hyperbolic Tree.
*/
      private double theWeight;
      /** The Children's weight of node. This variable is used by
Hyperbolic Tree.   */
      private double theChildrenWeight;
      /** The Children of node. This variable is used by Hyperbolic Tree.
*/
      private double theMinWeight;
      /** The label of Node. This string is show on trees. */
      private String theLabel;
      /** This variable is necessary for Hyperbolic Tree, but in this
project don't have use. */
      private String theContent;
      /** This variable is necessary for Hyperbolic Tree, but in this
project don't have use. */
      private String theLink;
      /** This variable is necessary for Hyperbolic Tree, but in this
project don't have use. */
      private String theTarget;
      /** The back-color of node in Hyperbolic Tree. */
      private Color theBackColor;
      /** The line-color of node in Hyperbolic Tree. */
      private Color theForeColor;
      /** This variable is necessary for Hyperbolic Tree, but in this
project don't have use. */
      private String theImageFile;
      /** This variable is necessary for Hyperbolic Tree, but in this
project don't have use. */
      private Image theImage;
      /** This variable is necessary for Hyperbolic Tree, but in this
project don't have use. */
      private Mountpoint theMountpoint;
```

```
	/** This variable is necessary for Hyperbolic Tree, but in this
project don't have use. */
	private Vector<IEdge> theFromEdges;
	/** This variable is necessary for Hyperbolic Tree, but in this
project don't have use. */
	private Vector<IEdge> theToEdges;
	/** tree[0] is the instance of OCTreebolicFactory what contains the
node. tree[0] is global for all tree.*/
	private OCTreebolicFactory[] tree;


	/**
	 * Creates and adds a class node child.
	 *
	 * @param className The class' name of child node.
	 * @param backColor The back-color of child node.
	 * @param lineColor The line-color of child node.
	 * @return The child node.
	 */
	public OCNode addChild(String className,Color backColor,Color
lineColor) {
		OCNode child = new
OCNode(this,className,backColor,lineColor);
		return child;

	}
	/**
	 * Creates and adds a instance node child.
	 *
	 * @param className The class' name of child node.
	 * @param instanceName The intance's name of chil node.
	 * @param backColor The back-color of child node.
	 * @param lineColor The line-color of child node.
	 * @return The child node.
	 */
	public OCNode addChild(String className, String instanceName,Color
backColor,Color lineColor) {
		OCNode child = new
OCNode(this,className+"("+instanceName+")",backColor,lineColor);
		return child;
	}

//	TREENODE INTERFACE
	/**
	 * Get a child at a determined position.
	 *
	 * @param pos Position of child in the Vector theChildren.
	 * @return The Child at pos position.
	 *
	 */
	public TreeNode getChildAt(int pos) {
		if (isLeaf()) return null;
		else return (TreeNode)this.theChildren.elementAt(pos);
	}
	/**
	 * Count the quantidy of node's children.<br>
	 * Same function of getTChildCount().
```

```
 *
 * @return The quantidy of children.
 * @see #getTChildCount
 */
public int getChildCount() {
      return getTChildCount();
}
/**
 * Return the parent of node.
 *
 * @return The parent of node.
 */
public TreeNode getParent() {
      return this.theParent;
}
/**
 * Return the position of child node in the Vector theChildren.
 *
 * @param child - The child node.
 * @return The position of child node.
 */
public int getIndex(TreeNode child) {
      return this.theChildren.indexOf(child);
}
/**
 * Method necessary for JTree, but don't used in this package.
 *
 * @return true
 */
public boolean getAllowsChildren() {
      return true;
}
/**
 * Determines if the node is a leaf.
 *
 * @return true if is a leaf or false if is not a leaf.
 */
public boolean isLeaf() {
      return (this.theChildren==null);
}
/**
 * Returns the childen.
 *
 * @return The children enumeration.
 */
public Enumeration children() {
      return getTChildNodes();
}

// C O N S T R U C T O R
/**
 * Contructs a new node.
 *
 * @param thisParent The parent node. null if the new node is root.
 * @param thisLabel The label of node.
 * @param thisBackColor The back-color of node in hyperbolic tree.
 * @param thisForeColor The line-color of node in hyperbolic tree.
```

```
       */
      public OCNode(OCNode thisParent, String thisLabel, Color
thisBackColor, Color thisForeColor)
      {
            theHyperCircle = new HyperCircle();
            theParent = thisParent;
            theLabel = thisLabel;
            theBackColor = thisBackColor;
            theForeColor = thisForeColor;
            theLink = null;
            theTarget = null;
            theImageFile = null;
            theImage = null;
            theMountpoint = null;
            theWeight = 0.;
            theFromEdges = null;
            theToEdges = null;

            if(thisParent != null) {
                  tree = theParent.getTree();
                  thisParent.addTChild(this);
            } else {
                  tree=new OCTreebolicFactory[1];
            }
      }


      // S E T   A C C E S S
      /**
       * Sets a link for node. This link will appear in the statusbar of
Hyperbolic Tree.
       *
       * @param thisLink The link.
       */
      public void setLink(String thisLink)
      {
            theLink = thisLink;
      }
      /**
       * This method is necessary for Hyperbolic Tree, but don't have
function in this package.
       *
       * @param thisTarget
       */
      public void setTarget(String thisTarget)
      {
            theTarget = thisTarget;
      }
      /**
       * Sets the image what will appear in Hyperbolic Tree together with
node.
       *
       * @param thisImage The url of image.
       */
      public void setImageFile(String thisImage)
      {
            theImageFile = thisImage;
```

```
	}

	// E D G E S
	/**
	 * This method is necessary for Hyperbolic Tree, but don't have
function in this package.
	 *
	 * @param thisEdge
	 */
	public void addFromEdge(IEdge thisEdge)
	{
		if(theFromEdges == null)
			theFromEdges = new Vector<IEdge>();
		theFromEdges.addElement(thisEdge);
	}
	/**
	 * This method is necessary for Hyperbolic Tree, but don't have
function in this package.
	 *
	 * @param thisEdge
	 */
	public void addToEdge(IEdge thisEdge)
	{
		if(theToEdges == null)
			theToEdges = new Vector<IEdge>();
		theToEdges.addElement(thisEdge);
	}


	// I N O D E I N T E R F A C E

	// link
	/**
	 * Return the parent of node.
	 *
	 * @return The parent of node.
	 */
	public INode getTParent()
	{
		return theParent;
	}
	/**
	 * Sets the parent of node.
	 *
	 * @param thisParent The parent.
	 */
	public void setTParent(INode thisParent)
	{
		theParent = (OCNode)thisParent;
	}
	/**
	 * Count the quantidy of node's children.<br>
	 * Same function of getChildCount().
	 *
	 * @return The quantidy of children.
	 * @see #getChildCount
	 */
```

```
public int getTChildCount()
{
      return theChildren == null ? 0 : theChildren.size();
}
/**
 * Returns the childen.
 *
 * @return The children enumeration.
 */
public Enumeration getTChildNodes()
{
      return theChildren == null
      ? new Enumeration()
      {
            public boolean hasMoreElements() { return false; }
            public Object nextElement() { return null;}
      }
      : theChildren.elements();
}
/**
 * Adds a child.
 *
 * @param thisChild The child. Must be an instance of OCNode.
 */
public void addTChild(INode thisChild)
{
      if(theChildren == null)
            theChildren = new Vector<INode>(1);
      theChildren.addElement(thisChild);
}

/**
 * Removes a child.
 *
 * @param thisChild The child.
 */
public void removeTChild(INode thisChild)
{
      theChildren.removeElement(thisChild);
}

// internals
public HyperCircle getHyperCircle()
{
      return theHyperCircle;
}

// display
public String toString()
{
      return theLabel;
}

public String getLabel()
{
      return theLabel;
}
```

```
	public void setLabel(String thisLabel)
	{
		theLabel = thisLabel;
	}

	public String getContent()
	{
		return theContent;
	}

	public void setContent(String thisContent)
	{
		theContent = thisContent;
	}

	public Color getBackColor()
	{
		if (isSelected()) return Color.gray;
		else return theBackColor;
	}

	public void setBackColor(Color thisColor)
	{
		theBackColor = thisColor;
	}

	public Color getForeColor()
	{
//		if (selected) return Color.red;
if (isSelected()) return theBackColor;
else return theForeColor;
	}

	public void setForeColor(Color thisColor)
	{
		theForeColor = thisColor;
	}

	public String getLink()
	{
		return theLink;
	}

	public String getTarget()
	{
		return theTarget;
	}

	// image
	public String getImageFile()
	{
		return theImageFile;
	}

	public Image getImage()
	{
```

```
      return theImage;
}

public void setImage(Image thisImage)
{
      theImage = thisImage;
}

// weight
public double getWeight()
{
      return theWeight;
}

public void setWeight(double thisWeight)
{
      theWeight = thisWeight;
}

public double getChildrenWeight()
{
      return theChildrenWeight;
}

public void setChildrenWeight(double thisWeight)
{
      theChildrenWeight = thisWeight;
}

public double getMinWeight()
{
      return theMinWeight;
}

public void setMinWeight(double thisWeight)
{
      theMinWeight = thisWeight;
}

// mountpoint
public Mountpoint getMountpoint()
{
      return theMountpoint;
}

public void setMountpoint(Mountpoint thisMountpoint)
{
      theMountpoint = thisMountpoint;
}

// edge links
public Enumeration getFromEdges()
{
      if(theFromEdges == null)
            return null;
      return theFromEdges.elements();
}
```

```
	public Enumeration getToEdges()
	{
		if(theToEdges == null)
			return null;
		return theToEdges.elements();
	}

//	METODOS UTILIZADOS APENAS QUANDO A ÁRVORE FOR HIPERBÓLICA
	public void clicked(MouseEvent mouseEvent) {
		if
((tree[0].countSelected()>1)&&(!mouseEvent.isControlDown())){
			tree[0].select(this,false);
		} else if (isSelected()) tree[0].deselect(this);
		else if (mouseEvent.isControlDown())
tree[0].select(this,true);
		else tree[0].select(this,false);
	}

	public boolean isSelected(){
		return tree[0].isSelected(this);
	}
	public OCTreebolicFactory[] getTree() {
		return tree;
	}

	public void setTree(OCTreebolicFactory tree) {
		this.tree[0]=tree;
	}

	public String getPath() {
		if (theParent==null) return (theLabel);
		else return(theParent.getPath()+"."+theLabel);
	}

	public OCTerm getTerm() {
		return new OCTerm(getPath());
	}

	public boolean deselectAll() {
		boolean ret = tree[0].deselect(this);
		Enumeration e=children();
		while (e.hasMoreElements()) ret |=
((OCNode)e.nextElement()).deselectAll();
		return ret;
	}

	public boolean isSelectedAnyParent() {
		if (this.theParent==null) return false;
		if (this.theParent.isSelected()) return true;
		else return this.theParent.isSelectedAnyParent();
	}

	public boolean isRoot() {
		return (this.theParent==null);
	}
}
```

```
package OntoCover;

/**
 * @(#)OCTerm.java      1.00 07/08/03
 * <p>Title: OntoCover Term </p>
 * <p>Description: Term of an ontology</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: IC-Unicamp and Embrapa </p>
 * @author Renato Fileto and Lauro Ramos Venancio
 * @version 1.0
 */

public class OCTerm {
  String path="";

  public OCTerm() {
    path="";
  }

  public OCTerm(String path) {
    if (path == null) this.path = "";
    else this.path = path;
  }

  private String getPath() {
    return path;
  }

  private String getLastPathComponent(){
    return path.substring(path.lastIndexOf(".")+1);
  }

  public String toString() {
    return getLastPathComponent();
  }

  public String toPathString() {
    return path;
  }


  public boolean treeEncompass(OCTerm term) {
    return term.getPath().startsWith(path);
  }

  public boolean treeEquivalent(OCTerm term) {
    return path.equals(term.getPath()) ;
//    return (this.treeEncompass(term) &&
//            term.treeEncompass(this));
  }

  public int getPathCount() {
    return path.split(".").length;
  }

  public String getPathComponent(int i) {
```

```
    String[] a = path.split(".");
    if (i<a.length) return a[i];
    else return null;
  }

}
package OntoCover;

import java.applet.*;
import java.io.*;
import java.net.*;
import java.util.*;

import java.awt.*;

import treebolic.applet.*;
import treebolic.in.*;
import treebolic.in.provider.*;

/**
 * @(#)OCTreebolicFactory.java     1.00 07/08/03
 * <p>Title: OntoCover Treebolic Factory </p>
 * <p>Description: Presents an ontological tree maintained in an OCModel
 *                  as a treebolic tree (a kind of hyperbolic tree)</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: IC-Unicamp and Embrapa </p>
 * @author Lauro Ramos Venancio
 * @version 1.0
 */

public class OCTreebolicFactory implements
OCTree,IProviderFactory,AppletStub,AppletContext{
  public OCNode top;
  private TreeSet selected=new TreeSet(new Comparador());
  private boolean multiSelection=false;
  private Treebolic tree;

  class Comparador implements Comparator {
    public int compare(Object a,Object b){
      int ha = a.hashCode();
      int hb = b.hashCode();
      if (ha==hb) return 0;
      else if (ha>hb) return 1;
      else return -1;
    }
  }



  public OCTreebolicFactory(OCNode top,Treebolic tree) {
    this.top=top;
    top.setTree(this);
    this.tree=tree;
    top.deselectAll();
  }
```

```
public void setMultiSelection(boolean multiSelection) {
  this.multiSelection=multiSelection;
  if ((!multiSelection)&&(selected.size()>1)) {
    Object node=selected.first();
    selected.clear();
    selected.add(node);
    tree.theView.repaint();
  }
}

public Provider makeProvider(String string) {
  Settings settings = new Settings();
  settings.theSweep=1.5;
  settings.theFontSize=15;
  settings.theHasToolbarFlag=true;
  settings.theFocusOnHoverFlag=false;
  settings.theOrientation="south";
  settings.thePreserveOrientationFlag=true;
  return new Provider(top,new EdgeList(),settings);
}

public boolean isActive() {
  return true;
}

public URL getDocumentBase() {
  return null;
}

public URL getCodeBase() {
  return null;
}

public String getParameter(String string) {
  return null;
}

public AppletContext getAppletContext() {
  return this;
}

public void appletResize(int int0, int int1) {
}

public AudioClip getAudioClip(URL uRL) {
  return null;
}

public Image getImage(URL uRL) {
  return null;
}

public Applet getApplet(String string) {
  return null;
}

public Enumeration getApplets() {
```

```
  return null;
}

public void showDocument(URL uRL) {
}

public void showDocument(URL uRL, String string) {
}

public void showStatus(String string) {
}

public void setStream(String string, InputStream inputStream) {
}

public InputStream getStream(String string) {
  return null;
}

public Iterator getStreamKeys() {
  return null;
}

public int countSelected() {
  return selected.size();
}

public boolean deselect(OCNode node) {
  return selected.remove(node);
}

public OCTerm[] getSelectedTerms() {
 OCTerm[] terms;
  if (selected.isEmpty()) {
    terms = new OCTerm[1];
    terms[0] = new OCTerm(null);
  }
  else {
    Iterator nodes = (selected.iterator());
    terms = new OCTerm[selected.size()];
    int i = 0;
    while (nodes.hasNext()) {
      terms[i] = ( (OCNode) nodes.next()).getTerm();
      i++;
    }
  }
  return terms;
}

public OCTerm getSelectedTerm () {
  return getSelectedTerms()[0];
}

public OCTuple getSelectedTuple () {
  return new OCTuple(getSelectedTerms());
}
```

```
  public void select(OCNode node, boolean mustAdd) {
    if ((!multiSelection)||(!mustAdd)) {
      this.selected.clear();
    } else if ((multiSelection)&&(mustAdd)){
      if (node.isSelectedAnyParent()) {
        tree.putStatus("Ancestor node is already selected.","");
        return;
      }
      else {
        if (node.deselectAll())
          tree.putStatus("Descendent node has been unselected.","");

        this.selected.add(node);
        if (!node.isRoot()){
          Enumeration enumeration = node.getParent().children();
          boolean allSelected = true;
          while (enumeration.hasMoreElements()) {
            allSelected &= selected.contains(enumeration.nextElement());
          }
          if (allSelected){
            select( (OCNode) node.getParent(), true);
            tree.putStatus("Parent node was selected because all its
siblings have been selected.","");
          }
        }
        return;
      }
    }
    this.selected.add(node);
  }

  public Component getVisualTree () {
    return this.tree;
  }

  public boolean isSelected(OCNode node) {
    return selected.contains(node);
  }
}

package OntoCover;

/**
 * @(#)OCTuple.java    1.00 07/08/03
 * <p>Title: OntoCover Tuple </p>
 * <p>Description: Tuple of terms of an ontology</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: IC-Unicamp and Embrapa </p>
 * @author Renato Fileto
 * @version 1.0
 */

public class OCTuple {

  OCTerm[] terms = null;

  public OCTuple () {
```

```
    terms = new OCTerm[1];
    terms[0] = new OCTerm(null);
  }

  public OCTuple (OCTerm[] terms) {
    if (terms == null)
      new OCTuple();
    else
      this.terms = terms;
  }

  public OCTuple (String ocPathStr) {
    String[] strTerms = ocPathStr.split("\n");
    terms = new OCTerm[strTerms.length];
    for (int i = 0; i < strTerms.length; i++ ) {
      terms[i] = new OCTerm(strTerms[i]);
    }
  }

  public String toPathsString () {
    String pathsStr = terms[0].toPathString();
    for (int i = 1; i < terms.length; i++) {
      pathsStr = pathsStr + '\n' + terms[i].toPathString();
    }
    return pathsStr;
  }

  public boolean treeEncompass (OCTuple tuple) {
    for (int i = 0; i < terms.length; i++) {
      if (! tuple.hasTreeEncompassedTerm(terms[i]))  return false;
    }
    return true;
  }

  public boolean hasTreeEncompassedTerm (OCTerm term) {
    for (int i = 0; i < terms.length; i++) {
      if (term.treeEncompass(terms[i]))  return true;
    }
    return false;
  }

  public boolean treeEquivalent (OCTuple tuple) {
    return (this.treeEncompass(tuple) &&
            tuple.treeEncompass(this));
  }

}
package util;

import java.util.Vector;

/**
 * @author Marcelo Oliveira de Moraes
 * @version 1.0
 *
 * <p>Classe utilizada para resgatar os atributos inseridos no arquivo
xml de configuração.
```

```
 */
public class OCConfig {

	private String rdf, rdfs, arquivoDeSaida,  bdTipo, bdDriver, URL,
URLs, login, senha;

	// construtor da classe
	public OCConfig(  String rdf,
					String rdfs,
					String output,
					String bdTipo,
					String bdDriver,

					String URL,
					String URLs,
					String login,
					String senha) {
		this.rdf = rdf;
		this.rdfs = rdfs;
		this.arquivoDeSaida = output;
		this.bdTipo = bdTipo;
		this.bdDriver = bdDriver;
		this.URL = URL;
		this.URLs = URLs;
		this.login = login;
		this.senha = senha;
	}

	public OCConfig() throws Exception {
		atualizaAtributos();
	}

	private void atualizaAtributos() throws Exception {
		LeitorXML leitor = new LeitorXML("oc-config.xml");

			Vector<Object> v = leitor.ler();
			this.rdf = (String) v.get(0);
			this.rdfs = (String) v.get(1);
			this.arquivoDeSaida = (String) v.get(2);
			this.bdTipo = (String) v.get(3);
			this.bdDriver = (String) v.get(4);
			this.URL = (String) v.get(5);
			this.URLs = (String) v.get(6);
			this.login =(String)  v.get(7);
			this.senha = (String) v.get(8);
	}

	public String getBdDriver() {
		return bdDriver;
	}

	public void setBdDriver(String bdDriver) {
		this.bdDriver = bdDriver;
	}

	public String getBdTipo() {
		return bdTipo;
```

```
}

public void setBdTipo(String bdTipo) {
	this.bdTipo = bdTipo;
}

public String getURL() {
	return URL;
}

public void setURL(String URL) {
	this.URL = URL;
}

public String getURLs() {
	return URLs;
}

public void setURLs(String URLs) {
	this.URLs = URLs;
}


public String getLogin() {
	return login;
}

public void setLogin(String login) {
	this.login = login;
}

public String getRDF() {
	return this.rdf;
}

public void setRDF(String rdf) {
	this.rdf = rdf;
}

public String getRDFs() {
	return rdfs;
}

public void setRDFs(String rdfs) {
	this.rdfs = rdfs;
}

public String getSenha() {
	return senha;
}

public void setSenha(String senha) {
	this.senha = senha;
}

public String getArquivoDeSaida() {
	return arquivoDeSaida;
```

```
	}

	public void setArquivoDeSaida(String output) {
		this.arquivoDeSaida = output;
	}

	public static void main(String[] args) {
		try {
			OCConfig oc = new OCConfig();
			System.out.println( oc.getRDF());
			System.out.println( oc.getRDFs());
			System.out.println( oc.getURL());
			System.out.println( oc.getURLs());

		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}


}
package util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.hp.hpl.jena.db.DBConnection;

/**
 *
 * @author Marcelo Oliveira de Moraes
 * @version 1.0
 *
 * <p>Classe responsável pela conexão com o BD.
 *
 */
public class Conexao {

	/**
	 * Retorna um objeto que faz a conexão com a tabela da instancia do
RDF
	 *
	 * @return DBConnection conexao
	 */
	public static DBConnection getConexaoRDF(){
		DBConnection conexao =null;
		OCConfig oc =null;

		try {
			oc = new OCConfig();
		} catch (Exception e1) {
			e1.printStackTrace();
		}
```

```
            try {
                  Class.forName(oc.getBdDriver());
                  conexao = new DBConnection(oc.getURL(), oc.getLogin(),
oc.getSenha(), oc.getBdTipo());
            } catch (ClassNotFoundException e) {
                  e.printStackTrace();
            }

            return conexao;
      }

      /**
       * Retorna um objeto que faz a conexão com a tabela da instancia do
RDF Schema
       *
       * @return DBConnection conexao
       */
      public static DBConnection getConexaoRDFs(){
            DBConnection conexao =null;
            OCConfig oc =null;

            try {
                  oc = new OCConfig();
            } catch (Exception e1) {
                  e1.printStackTrace();
            }
            try {
                  Class.forName(oc.getBdDriver());
                  conexao = new DBConnection(oc.getURLs(), oc.getLogin(),
oc.getSenha(), oc.getBdTipo());
            } catch (ClassNotFoundException e) {
                  e.printStackTrace();
            }

            return conexao;
      }

      /**
       * Metodo para conexao via jdbc como BD
       *
       * @param driver
       * @param url
       * @param login
       * @param senha
       * @return Connection
       */
      public static Connection getConexao(String driver, String url,
      String login, String senha) {
            Connection conexao =null;

            try {
                  Class.forName(driver);
                  conexao = DriverManager.getConnection(url, login,
senha); // load JDBC-ODBC driver
            } catch (ClassNotFoundException e) {
                  e.printStackTrace();
            } catch (SQLException e) {
```

```
                e.printStackTrace();
            }

            return conexao;
      }

      /**
       * @param args
       */
      public static void main(String[] args) {

            Connection conn = null;
          Statement stmt = null;
          ResultSet rs = null;

          try {

            DBConnection conJena = getConexaoRDF();

            if (conJena != null){

              System.out.println("fechando a conexao...");
              conJena.close();
            }
          }catch (Exception e) {
                  e.printStackTrace();
            }
      }


}

package util;

import java.util.Vector;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

public class LeitorXML {

      // caminho (path) do arquivo XML
      private String xmlPathname;

      // construtor que seta o caminho do XML
      public LeitorXML(String path) {
            xmlPathname = path;
      }

      // le o XML carregando os dados dos usuários em um Vector.
      // retorna o vector contendo os usuários cadastrados no XML.
      public Vector<Object> ler() throws Exception {
```

```
            DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document doc = db.parse(xmlPathname);
            Element elem = doc.getDocumentElement();

            Vector<Object> vetor = new Vector<Object>();

            // parser em todos os elementos do XML

            // elementos pertencentes a Arvore Hiperbolica
            NodeList nodoArvoreHiperbolica =
elem.getElementsByTagName("arvore-hiperbolica");

            Element tagArvoreHiperbolica = (Element)
nodoArvoreHiperbolica.item(0);
            String rdf = getChildTagValue(tagArvoreHiperbolica, "RDF");

            vetor.add(rdf);

            String rdfS = getChildTagValue(tagArvoreHiperbolica,
"RDFSchema");
            vetor.add(rdfS);

            String output = getChildTagValue(tagArvoreHiperbolica,
"arquivo-output");
            vetor.add(output);

            // elementos pertencentes ao Banco de Dados
            NodeList nodoBancoDeDados = elem.getElementsByTagName("banco-
de-dados");

            Element tagBD = (Element) nodoBancoDeDados.item(0);
            String tipoBD = getChildTagValue(tagBD, "tipo");
            vetor.add(tipoBD);

            String driverBD = getChildTagValue(tagBD, "driver");
            vetor.add(driverBD);

            String urlRDF = getChildTagValue(tagBD, "urlRDF");
            vetor.add(urlRDF);

            String urlRDFs = getChildTagValue(tagBD, "urlRDFs");
            vetor.add(urlRDFs);

            String loginBD = getChildTagValue(tagBD, "login");
            vetor.add(loginBD);

            String senhaBD = getChildTagValue(tagBD, "senha");
            vetor.add(senhaBD);

            return vetor;
      }

      // este método lê e retorna o conteúdo (texto) de uma tag
(elemento)
```

```
	// filho da tag informada como parâmetro. A tag filho a ser
pesquisada
	// é a tag informada pelo nome (string)
	private String getChildTagValue(Element elem, String tagName)
			throws Exception {
		NodeList children = elem.getElementsByTagName(tagName);
		if (children == null)
			return null;
		Element child = (Element) children.item(0);
		if (child == null)
			return null;
		return child.getFirstChild().getNodeValue();
	}
}


package util.model;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;


import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;

/**
 *
 * @author Marcelo Oliveira de Moraes
 *
 * classe utilizada para converter em List o Resultset gerado pelo SPARQL
 */
public class OCTabela {


	public static List<OCLinha> setResultados(ResultSet ocResults) {
		List<OCLinha> tabelaCompleta = new ArrayList<OCLinha>();

		//obtendo o titulo, no caso o nome dos atributos da Ontologia
		Map<Integer, String> titulo =
getTitulos(ocResults.getResultVars());
		List metadados =  Arrays.asList( titulo.values().toArray());
		OCLinha linhaMetadados = new OCLinha();
		linhaMetadados.setAtributos( metadados );


		tabelaCompleta.add(linhaMetadados);

		//preenchendo a tabela
		while (ocResults.hasNext()) {
			QuerySolution elem = (QuerySolution) ocResults.next();
			OCLinha linha = new OCLinha();
```

```
                  for (int indiceColuna =0; indiceColuna <
ocResults.getResultVars().size(); indiceColuna ++){
                        String variavel = titulo.get(indiceColuna);
                        linha.addItem(elem.get(variavel).toString());

                  }

                  tabelaCompleta.add(linha);

            }

            return tabelaCompleta;
      }

      private static Map<Integer, String> getTitulos(List resultVars) {
            Map<Integer, String> titulos = new HashMap<Integer,
String>();

            int i=0;

            for (Iterator iterator = resultVars.iterator();
iterator.hasNext();) {
                  String titulo = (String) iterator.next();
                  titulos.put(i, titulo);
                  i++;
            }
            return titulos;
      }

}

package util.model;

import java.awt.Color;

import OntoCover.OCNode;

/**
 *
 * @author Marcelo Oliveira de Moraes
 * @version 1.0
 *
 *<p>Classe utilizada para gerar uma Árvore para testes.
 *
 */
public class DefaultTree {

      public static void create(OCNode top){
            OCNode n0, n1, n2, n3, n4;

            n0 = top.addChild("Institution", Color.cyan,
Color.black);//nodo 1 da raiz

            n1 = n0.addChild("Consortium", Color.cyan, Color.black);

            n2 = n1.addChild("Consortium(AGRITEMPO)", Color.cyan,
Color.black);
```

```
            n2 = n1.addChild("Consortium(RNA)", Color.cyan, Color.black);

            n3 = n2.addChild("PublicInstitution(EMBRAPA)", Color.cyan,
Color.black);
            n3 = n2.addChild("PublicInstitution(IAC)", Color.cyan,
Color.black);
            n3 = n2.addChild("PublicInstitution(INMET)", Color.cyan,
Color.black);
            n3 = n2.addChild("PublicInstitution(UNICAMP)", Color.cyan,
Color.black);

            n4 = n3.addChild("Unit(CEPAGRI)", Color.cyan, Color.black);
            n4 = n3.addChild("Unit(FEECC)", Color.cyan, Color.black);
            n4 = n3.addChild("Unit(IC)", Color.cyan, Color.black);
            n4 = n3.addChild("Unit(IMECC)", Color.cyan, Color.black);

            n0 = top.addChild("TerritorialDivision", Color.pink,
Color.black);//nodo 2 da raiz

            n1 = n0.addChild("Country", Color.pink, Color.black);

            n2 = n1.addChild("Country(ARGENTINA)", Color.pink,
Color.black);
            n2 = n1.addChild("Country(BRASIL)", Color.pink, Color.black);

            n3 = n2.addChild("Region(CENTRO-OESTE)", Color.pink,
Color.black);
            n3 = n2.addChild("Region(NORDESTE)", Color.pink,
Color.black);
            n3 = n2.addChild("Region(NORTE)", Color.pink, Color.black);
            n3 = n2.addChild("Region(SUDESTE)", Color.pink, Color.black);

            n4 = n3.addChild("State(ESPIRITO SANTO)", Color.pink,
Color.black);
            n4 = n3.addChild("State(MINAS GERAIS)", Color.pink,
Color.black);
            n4 = n3.addChild("State(RIO DE JANEIRO)", Color.pink,
Color.black);
            n4 = n3.addChild("State(SAO PAULO)", Color.pink,
Color.black);

            n3 = n2.addChild("Region(SUL)", Color.pink, Color.black);

            n2 = n1.addChild("Country(ESTADOS UNIDOS)", Color.pink,
Color.black);
            n2 = n1.addChild("Country(PARAGUAI)", Color.pink,
Color.black);
            n2 = n1.addChild("Country(URUGUAI)", Color.pink,
Color.black);
            n2 = n1.addChild("Country(VENEZUELA)", Color.pink,
Color.black);

            n1 = n0.addChild("EcoRegion", Color.pink, Color.black);
            n1 = n0.addChild("Basin", Color.pink, Color.black);

            n0 = top.addChild("AgriculturalProduct", Color.yellow,
Color.black);//nodo 3 da raiz
```

```
            n1 = n0.addChild("KindOfProduct", Color.yellow, Color.black);

            n2 = n1.addChild("KindOfProduct", "FLOWER", Color.yellow,
Color.black);
            n2 = n1.addChild("KindOfProduct", "FRUIT", Color.yellow,
Color.black);
            n2 = n1.addChild("KindOfProduct", "GRAIN", Color.yellow,
Color.black);

            n3 = n2.addChild("Crop", "BEANS", Color.yellow, Color.black);

            n4 = n3.addChild("CommercialGroup(BLACK)", Color.yellow,
Color.black);
            n4 = n3.addChild("CommercialGroup(CARIOCA)", Color.yellow,
Color.black);
            n4 = n3.addChild("CommercialGroup(FRADINHO)", Color.yellow,
      Color.black);
            n4 = n3.addChild("CommercialGroup(KIDNEY)", Color.yellow,
Color.black);

            n3 = n2.addChild("Crop(COFFEE)", Color.yellow, Color.black);
            n3 = n2.addChild("Crop(CORN)", Color.yellow, Color.black);
            n3 = n2.addChild("Crop(RICE)", Color.yellow, Color.black);
            n3 = n2.addChild("Crop(WHEAT)", Color.yellow, Color.black);

            n2 = n1.addChild("KindOfProduct(NUT)", Color.yellow,
Color.black);
            n2 = n1.addChild("KindOfProduct(VEGETABLE)", Color.yellow,
Color.black);
      }

}

package util.model;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/***
 *
 * @author Marcelo Oliveira de Moraes
 * @version 1.0
 *
 * <p>Classe utilizada para representar uma linha contendo atributos.
 */
public class OCLinha {

      private List<String> atributos;

      public OCLinha(){
            this.atributos = new ArrayList<String>();
      }

      /**
       * Adiciona um item na Linha
```

```
	 *
	 * @param item: Valor String inserido na linha
	 */
	public void addItem(String item){
		atributos.add(item);
	}

	/**
	 * Retorna o item inserido na posição
	 *
	 * @param i: posição do elemento na linha.
	 * @return: String
	 */
	public String getItem(int i){

		return atributos.get(i);
	}

	/**
	 * Retorna a linha, que é um Objeto do tipo List que contém os item
inseridos.
	 *
	 * @return
	 * List<String>
	 */
	public List<String> getAtributos() {
		return atributos;
	}

	/**
	 * Atribui um Objeto do tipo List a linha.
	 *
	 * @param linha
	 * setAtributos
	 * void
	 */
	public void setAtributos(List<String> linha) {
		this.atributos = linha;
	}

	public static void main(String[] args) {

		OCLinha rotulos = new OCLinha();
		rotulos.addItem("Esquerda");
		rotulos.addItem("Centro");
		rotulos.addItem("Direita");

		List<OCLinha> dados = new ArrayList<OCLinha>();
		OCLinha linha1 = new OCLinha();
		OCLinha linha2 = new OCLinha();

		linha1.addItem("Coluna1"); linha1.addItem("Coluna2");
linha1.addItem("Coluna3");
		linha2.addItem("Coluna4"); linha2.addItem("Coluna5");
linha2.addItem("Coluna6");

		dados.add(linha1);
```

```
            dados.add(linha2);

            for (Iterator iterator = dados.iterator();
iterator.hasNext();) {
                  OCLinha linha = (OCLinha) iterator.next();
                  for (Iterator iterator2 =
linha.getAtributos().iterator(); iterator2.hasNext();) {
                        String atributo = (String) iterator2.next();
                        System.out.print(atributo+" ");
                  }
                  System.out.println("-----");
            }
      }

}

package util.model;

import java.io.PrintWriter;

import OntoCover.OCModel;

import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.rdf.model.StmtIterator;
import com.hp.hpl.jena.rdf.model.impl.NsIteratorImpl;

public class OCModelAtributes extends OCModel{
      public static void printRDFNamespaces(Model model) {
            System.out.println("--");
            System.out.println("Namespaces:");
            NsIteratorImpl nSpaces = (NsIteratorImpl)
model.listNameSpaces();
            while (nSpaces.hasNext()) {
                  System.out.println(nSpaces.next());
            }
            nSpaces.close();
      }

      public static void printRDFModel(Model model){
            System.out.println("--");
            System.out.println("N-TRIPLEs:");
            model.write(new PrintWriter(System.out), "N-TRIPLE");
      }

      public static void printRDFTriples(Model model){
            StmtIterator iterDim = model.listStatements();
            System.out.println("");
            System.out.println("BEGIN");
            while (iterDim.hasNext()) {
                  Statement triple = (Statement) iterDim.next();
                  System.out.println(triple.getSubject().toString());
                  System.out.println(triple.getPredicate().toString());
                  System.out.println(triple.getObject().toString());
                  System.out.println("--");
            }
            iterDim.close();
```

```
            System.out.println("END");
            System.out.println("");
      }

      public void printNameSpaces() {
            System.out.print("RDF NameSpace: ");
            System.out.println(nsRDF);
            System.out.print("RDF-Schema NameSpace: ");
            System.out.println(nsRDFS);
            System.out.print("Application NameSpace: ");
            System.out.println(nsAPL);
            System.out.print("Protege NameSpace: ");
            System.out.println(nsPTG);
      }
}
```