Exercício para Avaliação n.º 5

Bruno Azevedo*and Miguel Costa†

Módulo Engenharia Gramatical,
UCE30 Engenharia de Linguagens,
Mestrado em Engenharia Informatica,
Universidade do Minho

9 de Julho de 2012

Resumo

Este documento apresenta as resoluções do Exercício Prático n.º 5 do módulo de Engenharia Gramatical. O ecercício está relacionado com a implementação de uma linguagem para definir Mapas de Conceitos em que o output seja um WebSite.

*Email: azevedo.252@gmail.com
†Email: miguelpintodacosta@gmail.com

Conteúdo

1 Ambiente de Trabalho

Foi necessário usar um Gerador de Compiladores para gerar o nosso próprio compilador, por isso usámos o AnTLR que é também usado nas aulas. Para facilitar o processo de debugging durante a resolução do problema dado, usámos a ferramenta AnTLRWorks, que tem uma interface bastante agradável e simpática para ajudar a resolver problemas desta natureza.

A linguagem de programação adoptada foi o JAVA. De forma a tornar a nossa solução mais legível e estruturada. Para guardar a informação usamos uma Base de Dados relacional em MySQL e para o WebSite dinâmico usamos a linguagem PHP.

2 Descrição do problema

O pretendido para este exercício era gerar um pequeno WebSite, para navegar num Mapa de Conceitos, a partir de uma linguagem simples para descrever esses mesmos mapas.

A linguagem criada por nós tem de ser validada para depois ser apresentada a informação visualmente, terá ainda de haver uma forma de mostrar as ocorrências de cada conceito.

3 Gramática

Neste capítulo, iremos mostrar a gramática criada para definir uma linguagem para descrição de Mapa de Conceitos. Para isso, enunciaremos o que é pretendido da linguagem e estabeleceremos um mapeamento entre as necessidades da linguagem e as produções criadas.

De seguida, falaremos da abordagem utilizada (AST e Tree Grammar) para realizar as validações requeridas, assim como o armazenamento da informação contida no ficheiro de input para posterior geração de instruções de povoamento da base de dados.

Conceitos

```
conceitos
        :   conceito (';' conceito)*
        ;
    E possÃvel definir mais que um conceito, utilizando um ';' para separar cada
       definicao.

    conceito
        :   CONCEITO '(' STRING ')'
        ;
```

Um conceito é definido utilizando a palavra reservada 'conceito' seguida do conceito entre parênteses.

Relacionamento entre dois conceitos:

```
mapasConceitos
        :   mapaConceitos (';' mapaConceitos )*
        ;
```

É possível definir mais que um relacionamento, utilizando um ';' para separar cada definição.

```
mapaConceitos
        :   MAPACONCEITOS '('ID ','  STRING ',' STRING ')'
        ;
```

Um relacionamento é definido utilizando a palavra reservada 'mapaConceitos' seguida de um identificador do relacionamento e dois conceitos entre parênteses.

Este tipo de relacionamento só por si limita a informação que pretendemos armazenar no Mapa de Conceitos, por isso, surge a necessidade de se poder definir propriedades sobre os conceitos, melhorando assim a semântica associada a um conceito. Estas propriedades serão dividas em dois tipos: propriedades de dados e propriedades de conceitos.

As propriedades de dados associam uma determinada propriedade a um conceito, por exemplo, é possível definir que o conceito "Pessoa"possui a propriedade "tem nome", para isso criámos um mapeamento entre um conceito, uma propriedade de dados e um tipo de dados que determina o tipo de dados da propriedade.

Surgem assim, dois novos elementos na nossa gramática.

Propriedades de dados:

```
propriedadesDados
        :   propriedadeDados (';' propriedadeDados)*
        ;
```

É possível definir mais que uma propriedade de dados, utilizando um ';' para separar cada definição.

```
propriedadeDados
        :   PROPRIEDADEDADOS '(' STRING ')'
        ;
```

Uma propriedade de dados é definida utilizando a palavra reservada 'propriedadeDados' seguida do propriedade entre parênteses.

Mapeamento entre um conceito, uma propriedade de dados e um tipo de dados:

```
mapasConceitoPropDados
        :   mapaConceitoPropDados (';' mapaConceitoPropDados )*
        ;
```

É possível definir mais que um mapeamento, utilizando um ';' para separar cada definição.

```
mapaConceitoPropDados
        :   MAPACONCEITOPROPDADOS '('ID ','  STRING ',' STRING ',' tipo ')'
        ;
```

Um mapeamento é definido utilizando a palavra reservada 'mapaConceitoPropDados' seguida de um identificador do mapeamento, dois conceitos e o tipo de dados da propriedade entre parênteses.

```
tipo
        :   'STRING'
        |   'INT'
        |   ID
        ;
```

O tipo tomar os valores STRING, INT e pode ser também um identificador.

As propriedades de conceito definem um novo tipo de relacionamento entre dois conceitos, expandindo o tipo de relacionamento que nos era permitido. Desta forma, deixamos de ter apenas relacionamentos do tipo ?é um? e passamos a ter, por exemplo, o relacionamento ?nasceu em?, que relaciona o conceito ?emigrante? com o conceito ?local? e dá-nos a saber que determinado emigrante nasceu em determinado local. Para isto ser possível, criámos um mapeamento entre um conceito, uma propriedade de conceito e um segundo conceito.

Com isto, adicionámos mais dois elementos na nossa gramática:

Propriedades de conceito:

```
propriedadesConceito
        :   propriedadeConceito (';' propriedadeConceito)*
        ;
```

É possível definir mais que uma propriedade de conceito, utilizando um ';' para separar cada definição.

```
propriedadeConceito
        :   PROPRIEDADECONCEITO '(' STRING ')'
        ;
```

Uma propriedade de conceito é definida utilizando a palavra reservada 'propriedadeConceito' seguida da propriedade entre parênteses.

Mapeamento entre Conceito e Propriedade de Conceito:

```
mapasConceitoPropConceito
        :   mapaConceitoPropConceito (';' mapaConceitoPropConceito )*
        ;
```

É possível definir mais que um mapeamento, utilizando um ';' para separar cada definição.

```
mapaConceitoPropConceito
        :   MAPACONCEITOPROPCONCEITO '('ID ','  STRING ',' STRING ',' STRING    ')'
        ;
```

Um mapeamento é definido utilizando a palavra reservada 'mapaConceitoPropConceito' seguida de um identificador do mapeamento, um conceito, a propriedade de conceito e outro conceito entre parênteses.

Os elementos enunciados anteriormente não tem realmente valor semântico associado, por exemplo, sabemos que um conceito ?Emigrante ? está relacionado com um conceito ?Local? através de uma propriedade ?nasceu em?, mas de facto, não sabemos quem é o emigrante, nem o o local onde este terá nascido. Também sabemos que um conceito ?Emigrante? está relacionado com uma propriedade de dados ?tem nome?, no entanto, não se sabe que valor esta propriedade toma.

Para colmatar estas falhas, surgiu a necessidade de se criarem instâncias, que tal como o nome indicam representam instâncias de conceitos, sobre as quais se podem utilizar as propriedades para relacioná-las entre si. Por exemplo, uma instância do conceito ?Emigrante? chamada António e uma propriedade de conceito ?nasceu em? podem ser utilizadas para relacionar a instância António com a instância Fafe que é uma instância do conceito ?Local?. Da mesma forma, é possível relacionar a instância António com a propriedade de dados ?tem nome? e o valor da propriedade ?António José?.

Para finalizar, adicionámos à gramática três novos elementos para suportar o que foi introduzido acima:

Instâncias:

```
instancias
        :   instancia (';' instancia)*
        ;
```

É possível definir mais que uma instância, utilizando um ';' para separar cada definição.

```
instancia
        :   INSTANCIA '(' ID ',' STRING')'
        ;
```

Uma instância é definida utilizando a palavra reservada 'instancia' seguida de um identificador e um conceito entre parênteses.

Mapeamento entre Instância e Propriedade de dados:

```
mapasInstanciaPropDados
        :   mapaInstanciaPropDados (';' mapaInstanciaPropDados )*
        ;
```

É possível definir mais que um mapeamento, utilizando um ';' para separar cada definição.

```
mapaInstanciaPropDados
        :   MAPAINSTANCIAPROPDADOS '('ID ','  ID ',' STRING ')'
        ;
```

Um mapeamento é definido utilizando a palavra reservada 'mapaInstanciaPropDados' seguida de uma instância, um mapeamento entre um conceito e uma propriedade de dados e o valor da propriedade entre parênteses.

Mapeamento entre Instância e Propriedade de conceito:

```
mapasInstanciaPropConceito
        :   mapaInstanciaPropConceito (';' mapaInstanciaPropConceito )*
        ;
```

É possível definir mais que um mapeamento, utilizando um ';' para separar cada definição.

```
mapaInstanciaPropConceito
        :   MAPAINSTANCIAPROPCONCEITO '('ID ','  ID ',' ID ')'
        ;
```

Um mapeamento é definido utilizando a palavra reservada 'mapaInstanciaPropConceito' seguida de uma instância, um mapeamento entre um conceito e uma propriedade de conceito e outra instância entre parênteses.

Finalmente, o símbolo não terminal, onde o processo de derivação inicia, o axioma da nossa gramática e a produção associada:

```
cmc
        :   conceitos ';' (propriedadesDados ';')? (propriedadesConceito ';')?
            mapasConceitos ';' (mapasConceitoPropDados ';')? (
            mapasConceitoPropConceito ';')? (instancias ';')? (
            mapasInstanciaPropDados ';')? (mapasInstanciaPropConceito ';')?
        ;
```

Após criada a gramática, o próximo passo seria validar o texto de input. Mas como fazê-lo? Existem várias abordagens, uma delas e a que iremos utilizar neste trabalho é a geração de uma representação intermédia para que a partir dela se possam efetuar as validações requeridas, essencialmente, validações semânticas. Na secção seguinte, iremos falar desta abordagem e a nossa implementação da mesma.

3.1 AST

Uma Representação Intermédia (RI) é uma versão independente de qualquer linguagem ou máquina do código original. A utilização de uma RI traz algumas vantagens tais como o aumento do nível de abstração e uma separação mais limpa entre o produto inicial e o final.

Existem várias representações intermédias e a que iremos utilizar é a AST (Abstract Syntax Tree) que é uma representação em árvore da estrutura sintática abstrata do código fonte. A sintaxe é abstrata no sentido em que não representa cada detalhe que aparece na sintaxe real, ou seja, elementos como parênteses de agrupamento estão implícitos na estrutura da árvore e uma construção sintática tal como uma condição if e os seus blocos then e else pode ser representada através de um único nodo e dois ramos, e símbolos intermédios e palavras reservadas são tipicamente eliminados. Basicamente, mantém-se uma estrutura suficiente para realizar processos semânticos e geração de código.

O próximo passo será, então, criar a AST, para isso é necessário criar regras de reescrita sobre a gramática criada, um mecanismo que o ANTLR oferece. Enquanto que uma gramática de parsing especifica como reconhecer input, as regras de reescrita são gramáticas geradoras, ou seja, especificam como gerar output.

Assim sendo, por cada produção da gramática vamos criar uma regra de reescrita e cada regra representa um novo nodo na AST. Cada regra possui um token imaginário para agrupar os elementos presentes numa produção, ou seja, referências a tokens que não se encontram na produção original.

Tal como foi dito, elementos tais como ';', ou parênteses são eliminados e elementos com o mesmo nome numa produção são agrupados numa única lista.

A gramática final com as regras de reescrita pode ser consultada em anexo.

3.2 Tree Grammar

O próximo passo consiste na construção de um parser da AST gerada, que permitirá atravessá-la (tree walker) e manipulá-la, transformando-a gradualmente em diversas fases de tradução até que se obtenha uma forma final que satisfaça as nossas necessidades. Este parser será construído, utilizando um mecanismo fornecido pelo ANTLR, uma Tree Grammar (TG). As ações numa TG possuem um contexto muito nítido e conseguem aceder a informação passada das regras invocadas.

A utilização de TG, para além da utilização referida acima, também nos fornece algumas vantagens:

- uma especificação formal, concisa e independente de um sistema da estrutura da AST;
- as ações têm um contexto implícito graças à sua localização na gramática;
- os dados podem ser passados entre as ações de forma livre utilizando parâmetros (atributos), valores de retorno e variáveis locais.

A estratégia que utilizámos para efetuar a validação exigida consiste, na passagem de uma instância de uma classe Java Tabela (esta será explicada de seguida) assim como uma string de erros por todas as regras da TG. Todo o código externo à TG está escrito na linguagem de programação Java.

A Tabela consiste numa classe Java que armazena os elementos da nossa linguagem, ou seja, os conceitos, as propriedades de dados, as propriedades de conceito, os mapeamentos entre conceitos, os mapeamentos entre um conceito e uma propriedade de dados, os mapeamentos entre dois conceitos e uma propriedade de conceito, as instâncias, os mapeamentos entre uma instância, uma propriedade de dados e um valor para a propriedade e finalmente os mapeamentos entre duas instâncias e uma propriedade de conceito. Adicionalmente, criámos um método nesta classe que gera instruções de inserção na base de dados criada dos dados presentes na instância Tabela.

As classes Java criadas podem ser consultadas em anexo.

A string de erros consiste numa string à qual, por cada erro resultante das verificações feitas ao longo da TG, é anexado esse erro.

No final da execução da TG a tabela e a string de erros são retornadas à Interface Java criada (esta será explicada posteriormente).

As ações criadas para o processo de validação e povoamento da instância Tabela são bastante simples e o processo de criação foi sistemático. Ou seja, as ações criadas são de dois tipos, no que diz respeito, à forma de construção.

Ações de passagem de parâmetros (atributos herdados) aos símbolos não terminais presentes na produção e retorno dos resultados dos mesmos símbolos (atributos sintetizados). Para ilustrar este tipo de ação, basta apenas um exemplo:

```
conceitos [Tabela tab_in, String erro_in] returns [Tabela tab_out, String erro_out]
:   ^(CONCEITOS (conceito[$conceitos.tab_in, $conceitos.erro_in]
    {
        $conceitos.tab_in = $conceito.tab_out;
        $conceitos.erro_in = $conceito.erro_out;
    }
    )+
    {
        $conceitos.tab_out = $conceito.tab_out;
        $conceitos.erro_out = $conceito.erro_out;
    }
    )
    ;
```

Tal como foi explicado, a produção conceitos recebe como atributos a instância da Tabela e a string de erros.

Esta produção ilustra uma lista de conceitos, ou seja, um ou mais conceitos.

Um conceito recebe como atributos de entrada os atributos recebidos por conceitos. Após ter processado conceito os atributos são atualizados com os valores de retorno de conceito, para que cada conceito da lista tenha sempre uma tabela e uma string de erros atualizada.

No final, os valores de retorno da produção conceitos são atualizadas com os valores de retorno do último conceito da lista.

Ações de povoamento da tabela com a informação presente na produção. Adicionalmente em algumas produções é feita uma verificação de existência de erros, que condiciona o povoamento. Para ilustrar este tipo de ação iremos exibir dois casos representativos:

```
conceito [Tabela tab_in, String erro_in] returns [Tabela tab_out, String erro_out]
:   ^(CONCEITO STRING)
    {
        Tabela t = $conceito.tab_in;
        // adiciona o conceito Ã  tabela
        TreeSet<String> conceitos = t.getConceitos();
        conceitos.add($STRING.text);
        t.setConceitos(conceitos);

        $conceito.tab_out = t;
        $conceito.erro_out = $conceito.erro_in;
    }
    ;
```

Tal como foi explicado, a produção conceito recebe como atributos a instância da Tabela e a string de erros.

Esta produção ilustra a definição de um conceito, sendo que o conceito em si pode ser obtido a partir da propriedade text do token STRING.

O conceito é adicionado à instância tabela através da variável de instância conceitos.

Neste exemplo, não existe verificação de erros porque a definição de conceitos não possui quaisquer tipo de restrições. No final, os valores de retorno da produção são atualizados.

```
instancia [Tabela tab_in, String erro_in] returns [Tabela tab_out, String erro_out]
    @init{
        String erro = $instancia.erro_in;
        Tabela t = $instancia.tab_in;
    }
        :   ^(INSTANCIA ID STRING)
        {
            Boolean cSemErro = true;

            // verifica se existem erros e constroi string de erros
            if (!(cSemErro = t.getConceitos().contains($STRING.text)))
                erro += "\n\t("+$STRING.line+":"+$STRING.pos+")\tO conceito "+
                    $STRING.text+" nao foi definido!";

            // se nao existirem erros insere instancia na tabela
            if (cSemErro) {
                TreeMap<String, Instancia> instancias = t.getInstancias();
                instancias.put($ID.text, new Instancia($ID.text, $STRING.text));
                t.setInstancias(instancias);
            }

            $instancia.erro_out = erro;
            $instancia.tab_out = t;
        }
        ;
```

À semelhança de todas as produções, a produção instancia recebe como atributos a instância da Tabela e a string de erros.

Esta produção ilustra a definição de uma instância, sendo que o identificador e o conceito podem ser obtidos através da propriedade text dos tokens ID e STRING, respetivamente.

É feita uma verificação da existência do conceito na tabela, se este não existir, ou seja, não tiver sido definido previamente, então uma mensagem de erro é adicionada à variável erro. Esta mensagem de erro é constituída pela linha e coluna do conceito inexistente seguidas da mensagem de erro.

Se não tiver sido detetado nenhum erro, então a instância é adicionada à tabela, utilizando a variável de instância instancias.

Analogamente, esta verificação da existência de erros é feita nas produções mapaConceitos, mapaConceitoPropDados, mapaConceitoPropConceito, mapaInstanciaPropDados e mapaInstanciaPropConceito.

A produção inicial também é do primeiro tipo, mas ao invés de receber atributos de entrada com a tabela e a string de erros, estes são inicializados nesta produção.

```
1  cmc returns [Tabela tab_out, String erro_out]
2      @init{
3          Tabela tab = new Tabela();
4          String erro = "Erros:";
5      }
6          :   ^(CMC (conceitos[tab, erro] {tab = $conceitos.tab_out; erro =
               $conceitos.erro_out;})
7                      (propriedadesDados[tab, erro] {tab =    $propriedadesDados.
                           tab_out; erro = $propriedadesDados.erro_out;})?
8                      (propriedadesConceito[tab, erro] {tab =     $propriedadesConceito
                           .tab_out; erro = $propriedadesConceito.erro_out;})?
9                      (mapasConceitos[tab, erro] {tab = $mapasConceitos.tab_out; erro
                              = $mapasConceitos.erro_out;})
10                     (mapasConceitoPropDados[tab, erro] {tab =
                           $mapasConceitoPropDados.tab_out; erro =
                           $mapasConceitoPropDados.erro_out;})?
11                     (mapasConceitoPropConceito[tab, erro] {tab =
                           $mapasConceitoPropConceito.tab_out; erro =
                           $mapasConceitoPropConceito.erro_out;})?
12                     (instancias[tab, erro] {tab = $instancias.tab_out; erro =
                           $instancias.erro_out;})?
13                     (mapasInstanciaPropDados[tab, erro] {tab =
                           $mapasInstanciaPropDados.tab_out; erro =
                           $mapasInstanciaPropDados.erro_out;})?
14                     (mapasInstanciaPropConceito[tab, erro] {tab =
                           $mapasInstanciaPropConceito.tab_out; erro =
                           $mapasInstanciaPropConceito.erro_out;})?
15             )
16             {
17                 $cmc.erro_out = erro;
18                 $cmc.tab_out = tab;
19             }
20             ;
```

Pode-se observar que para cada símbolo não terminal nesta produção, a tabela e a string de erros são passados como atributos de entrada e, após a execução do mesmo, esses atributos são atualizados com os valores de retorno do símbolo.

No final, os valores de retorno da produção são atualizados para que possam ser passados à interface Java.

A Tree Grammar pode ser consultada em anexo.

3.3 Interface Java

É necessário criar uma interface Java para obter o output do ANTLR, obter a AST da gramática e executar o parsing da Tree Grammar (tree walking).

O funcionamento desta interface resume-se no seguinte esquema:

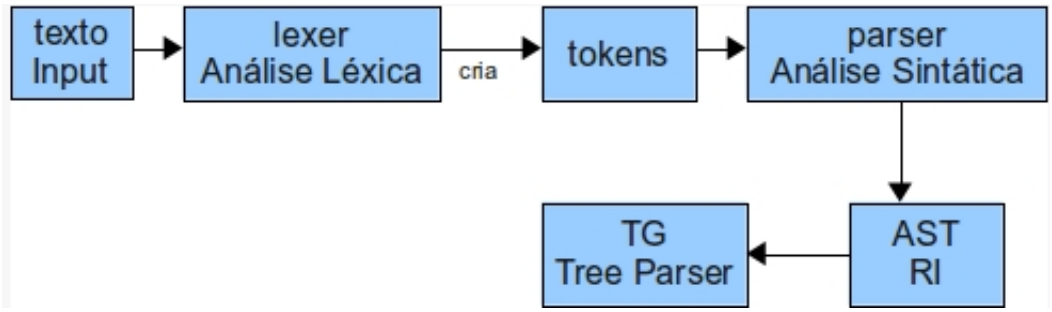

Figura 1:

Primeiro é necessário ligar todas as componentes da gramática geradas pelo ANTLR, nomeadamente o lexer (cmcLexer), o parser (cmcParser).

```
CharStream in = new ANTLRFileStream(args[0], "UTF8");
cmcLexer lexer = new cmcLexer(in);
CommonTokenStream tokens = new CommonTokenStream(lexer);
cmcParser parser = new cmcParser(tokens);
cmcParser.cmc_return ret = parser.cmc();
```

É criado então um ANTLRFileStream para receber o ficheiro de input. Depois alimenta-se o lexer com o input e finalmente alimenta-se o parser com os tokens gerados a partir do lexer. A última instrução inicia o processo de parsing.

Após o parsing ter terminado, é possível obter a árvore (AST) da gramática através do método getTree(), que devolve a raiz da árvore.

```
// obtem a AST utilizando as regras de reescrita da gramática criada
CommonTreeNodeStream tree = new CommonTreeNodeStream(ret.getTree());
```

O último passo consiste na navegação e manipulação da árvore, utilizando para isso a Tree Grammar.

```
// Tree Walking. Utiliza a Tree Grammar criada
mapaconceitosTGValidacao walker = new mapaconceitosTGValidacao(tree);
mapaconceitosTGValidacao.cmc_return walker_ret = walker.cmc();
```

A AST é então passada como argumento à TG e depois o processo de parsing da árvore é iniciado na ultima instrução.

Nesta altura, através de walker_ret podemos aceder aos valores de retorno da TG, ou seja, a instância da classe Tabela já povoada e a string de erros com os possíveis erros encontrados.

Esta interface pode ser consultada em anexo.

4 Base de Dados

Neste capítulo, iremos apresentar uma contextualização sobre a necessidade desta base de dados e a nossa solução final de base de dados. Para isso enunciaremos as tabelas criadas e explicaremos a sua razão de ser.

4.1 Contextualização

Para poder guardar a informação relativa ao mapa de conceitos, foi decido que esta seria armazenada numa base de dados relacional para que esta fosse de fácil acesso aquando da navegação pelo web site criado. Com base na linguagem criada, decidimos que o esquema final da nossa BD seria muito semelhante em termos de estrutura, senão igual, à gramática criada e então chegámos a um modelo final que passaremos a descrever na secção seguinte.

4.2 Modelo Lógico de Dados

As chaves primárias são representadas por atributos sublinhados e as chaves estrangeiras por atributos em itálico.

Conceitos {conceito}
Esta é a tabela que armazena os conceitos definidos na linguagem.
Atributos:
- conceito: é chave primária, já que identifica univocamente cada conceito; string que é o próprio conceito.

PropriedadesDados {propriedadeDados}
Tabela que armazena as propriedades de dados definidas na linguagem.
Atributos:
- propriedadeDados: é chave primária, já que identifica univocamente cada propriedade de dados; string que é a própria propriedade.

PropriedadesConceito {propriedadeConceito}
Tabela que armazena as propriedades de conceito definidas na linguagem.
Atributos:
- propriedadeConceito: é chave primária, já que identifica univocamente cada propriedade de conceito; string que é a própria propriedade.

MapasConceitos {id, *conceitoFilho*, *conceitoPai*}
Tabela que armazena os relacionamentos entre conceitos definidos na linguagem.
Atributos:
- id: é chave primária, já que identifica univocamente cada mapa de conceitos; string introduzida pelo utilizador;
- conceitoFilho: chave estrangeira da tabela Conceitos. String que identifica o conceito que faz parte de um conceito mais abrangente;
- conceitoPai: chave estrangeira da tabela Conceitos. String que identifica o conceito mais abrangente.

MapasConceitoPropDados {id, *conceito*, *propriedadeDados*, tipoDados}
Tabela que armazena os mapeamentos entre conceitos e propriedades de dados definidos na linguagem.
Atributos:
- id: é chave primária, já que identifica univocamente cada mapeamento; string introduzida pelo utilizador;
- conceito: chave estrangeira da tabela Conceitos. String que identifica o conceito ao qual se irá mapear uma propriedade de dados;
- propriedadeDados: chave estrangeira da tabela PropriedadesDados. String que identifica a propriedade de dados que será mapeada a um conceito;
- tipoDados: string que identifica o tipo de dados que propriedadeDados pode receber.

MapasConceitoPropConceito {id, *conceitoPai*, *propriedadeConceito*, *conceitoFilho*}
Tabela que armazena os mapeamentos entre conceitos e propriedades de cocneito definidos na linguagem.
Atributos:
- id: é chave primária, já que identifica univocamente cada mapeamento; string introduzida pelo utilizador;
- conceitoPai: chave estrangeira da tabela Conceitos. String que identifica o conceito que servirá de destino à propriedade de conceito;
- propriedadeConceito: chave estrangeira da tabela PropriedadesConceitos. String que identifica a propriedade de conceito que relacionará os dois conceitos;
- conceitoFilho: chave estrangeira da tabela Conceitos. String que identifica o conceito que servirá de partida à propriedade de conceito;

Instancias {instancia, *conceito*}
Esta é a tabela que armazena as instâncias definidos na linguagem.
Atributos:
- instancia: é chave primária, já que identifica univocamente cada instância; string que é a própria instância;
- conceito: chave estrangeira da tabela Conceitos. String que identifica o conceito sobre o qual se está a criar uma instância.

MapasInstanciaPropDados {*instancia*, *mapaConceitopropDados*, valor}
Tabela que armazena os mapeamentos entre uma instância, um mapeamentos entre um conceito e uma propriedade de dados e o valor da propriedade de dados para a instância definidos na linguagem.
Atributos:
- instancia: chave estrangeira da tabela Instancias;
- mapaConceitopropDados: chave estrangeira da tabela MapasConceitopropDados;
- valor: string que é o valor da propriedade de dados para a instância.
A chave primária é composta pelos atributos instancia e mapaConceitopropDados.

MapasInstanciaPropConceito {*instanciaPai*, *mapaConceitopropConceito*, *instanciaFilho*}
Tabela que armazena os mapeamentos entre duas instâncias e um mapeamento entre um conceito e uma propriedade de conceito definidos na linguagem.
Atributos:
- instanciaPai: chave estrangeira da tabela Instancias que servirá de destino ao mapeamento;
- mapaConceitopropConceito: chave estrangeira da tabela MapasConceitopropConceito;
- instanciaFilho: chave estrangeira da tabela Instancias que servirá de partida ao mapeamento.

5 WebSite

No site criado é possível instroduzir um texto de acordo com a nossa gramática e depois navegar por todas as navegações que possam existir entre os conceitos e as suas instãncias.

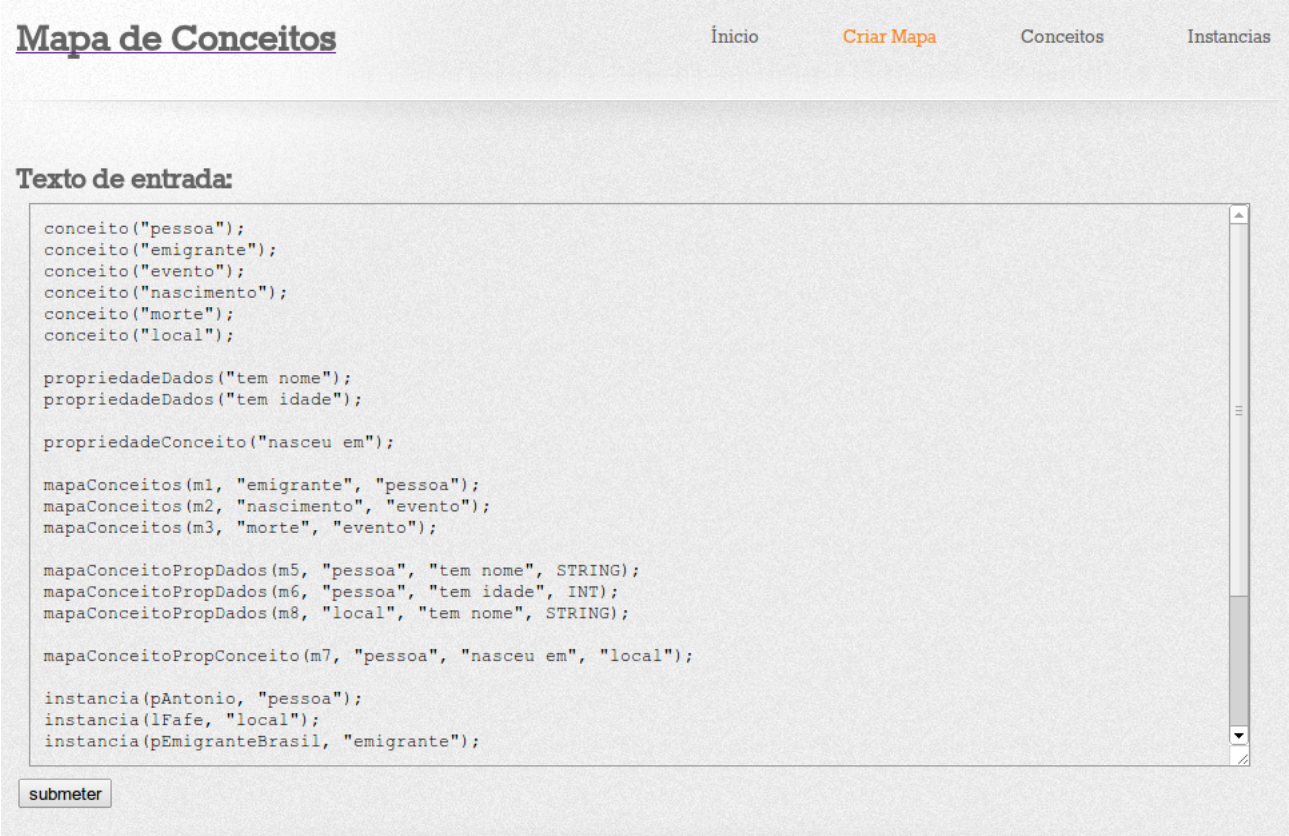

Figura 2: Página para criar uma mapa de conceitos

Depois se submetida a informação, é possível ver o resultado em ”Conceitos”:

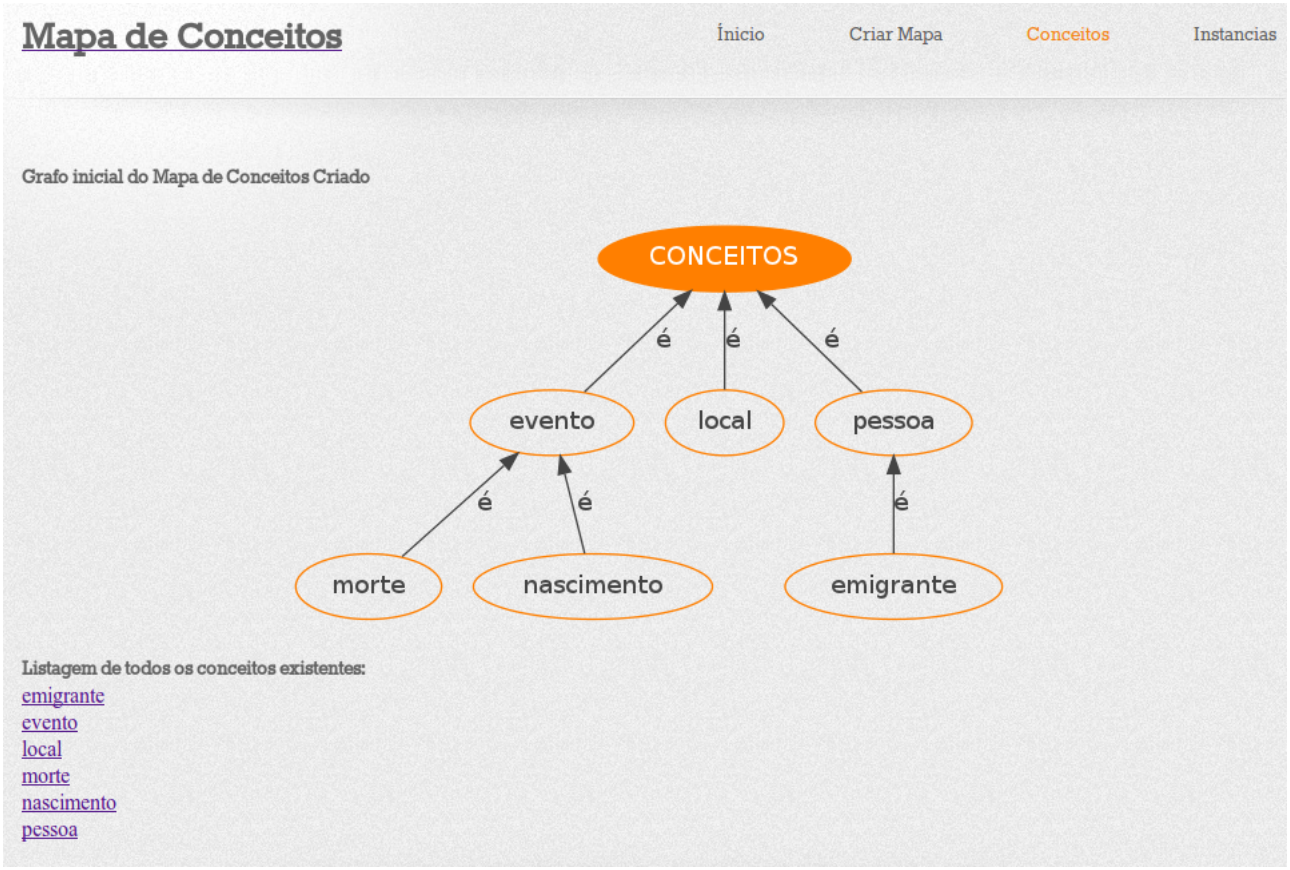

Figura 3: Página dos conceitos

É possível clicar nos nodos para se conseguir ver mais informação, se clicarmos em ”pessoa”vai ser apresentada a seguinte página:

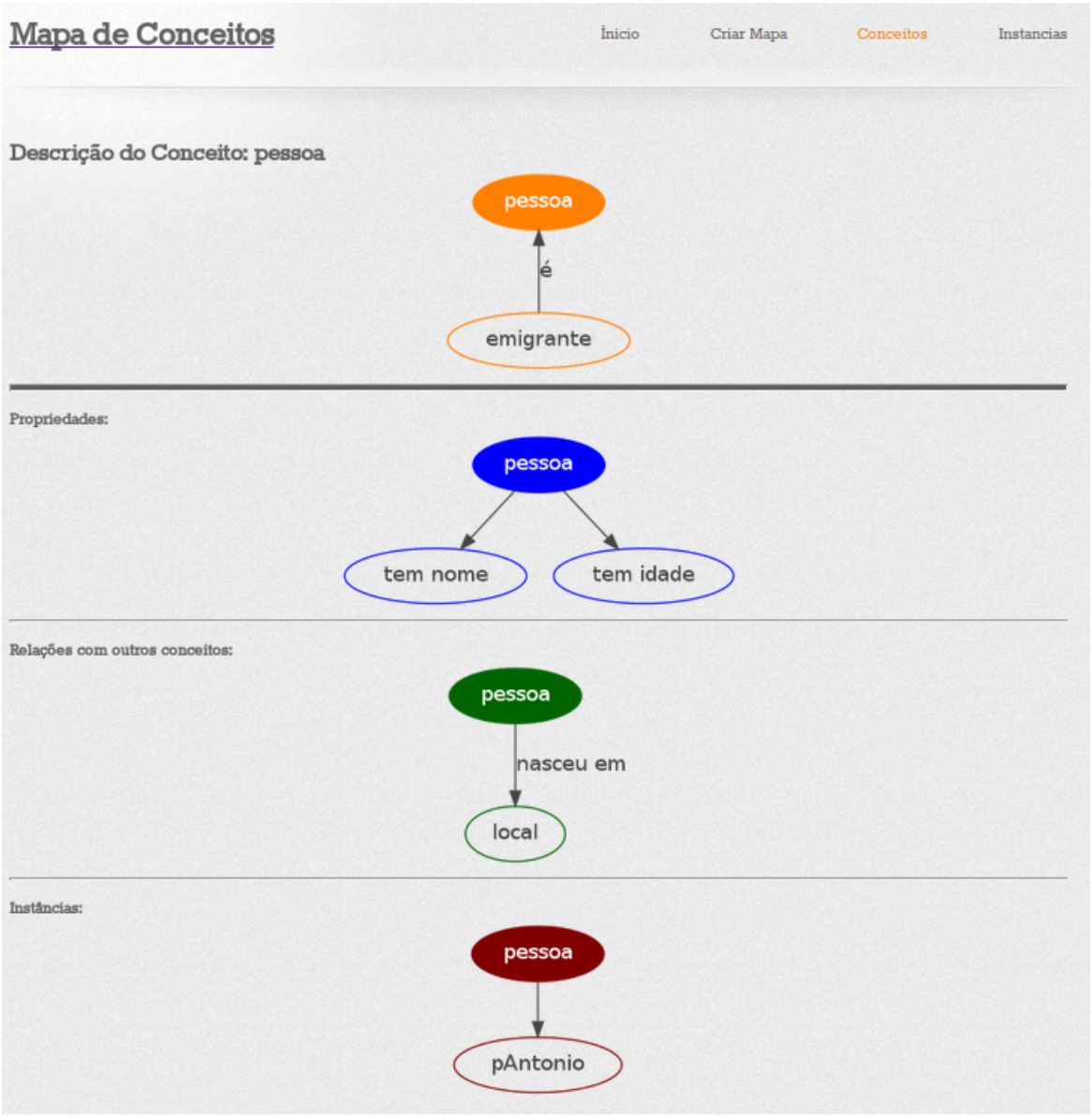

Figura 4: Página de detalhe de um conceito

Esta página está dividida em 4 partes:

- Parte em que mostra os ”filhos”do conceito em questão, neste caso só existe ”emigrante”;
- Propriedades que o conceito possa ter associadas a si;
- Relações que existem com outros conceitos;
- Instãncias que possam existir daquele conceito.

Nesta página, é também possível clicar num dos nodos para obter mais informação, se formos para as intâncias e clicarmos em ”pAntonio”é apresentada a seguinte página:

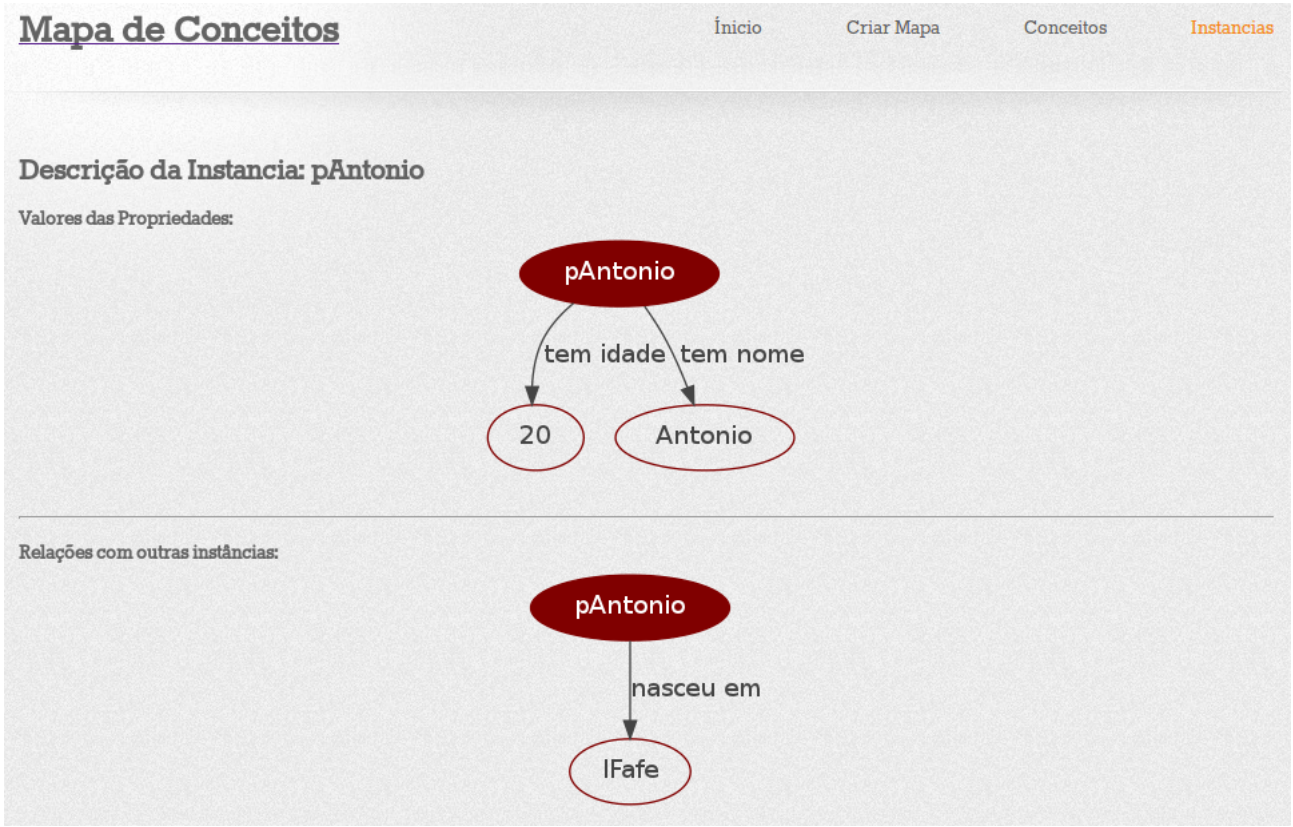

Figura 5: Página de detalhe de um conceito

Nesta página é possível ver as relações que uma instância tem para com outras relações e ainda ver os valores das propriedades que lhe estão associadas.

6 Conclusões

A utilização de Tree Grammars levou a que a construção de qualquer módulo se tornasse uma tarefa mais simples, já que o seu processo de construção se revelou sistemático. O facto de podermos fazer a travessia de uma AST através de uma gramática de atributos é uma mais valia pois de outra forma seria mais trabalhoso. Acima de tudo a utilização desta metodologia agilizou e simplificou todo o processo.

A linguagem criada revelou-se ser eficaz e muito simples para especificar ontologias, apesar de não ter tadas as propriedades de outras linguagens para o mesmo efeito, é de fácil leitura e contêm as característica essenciais para desenvolver uma árvore de conhecimento.

7 Anexos

7.1 Gramática final

```
grammar cmc;

options{
    backtrack = true;
    output = AST;
}

tokens {
    CMC;
    CONCEITOS;
    CONCEITO = 'conceito';
    PROPRIEDADESDADOS;
    PROPRIEDADEDADOS = 'propriedadeDados';
    PROPRIEDADESCONCEITO;
    PROPRIEDADECONCEITO = 'propriedadeConceito';
    MAPASCONCEITOS;
    MAPACONCEITOS = 'mapaConceitos';
    MAPASCONCEITOPROPDADOS;
    MAPACONCEITOPROPDADOS = 'mapaConceitoPropDados';
    MAPASCONCEITOPROPCONCEITO;
    MAPACONCEITOPROPCONCEITO = 'mapaConceitoPropConceito';
    INSTANCIAS;
    INSTANCIA = 'instancia';
    MAPASINSTANCIAPROPDADOS;
    MAPAINSTANCIAPROPDADOS = 'mapaInstanciaPropDados';
    MAPASINSTANCIAPROPCONCEITO;
    MAPAINSTANCIAPROPCONCEITO = 'mapaInstanciaPropConceito';
}

cmc
    :   conceitos ';' (propriedadesDados ';')? (propriedadesConceito ';')?
        mapasConceitos ';' (mapasConceitoPropDados ';')? (mapasConceitoPropConceito '
        ;')? (instancias ';')? (mapasInstanciaPropDados ';')? (
        mapasInstanciaPropConceito ';')?
    -> ^(CMC conceitos propriedadesDados? propriedadesConceito? mapasConceitos
        mapasConceitoPropDados? mapasConceitoPropConceito? instancias?
        mapasInstanciaPropDados? mapasInstanciaPropConceito?)
    ;

conceitos
    :   conceito (';' conceito)*
    -> ^(CONCEITOS conceito+)
    ;

conceito
    :   CONCEITO '(' STRING ')'
    -> ^(CONCEITO STRING)
    ;

propriedadesDados
    :   propriedadeDados (';' propriedadeDados)*
    -> ^(PROPRIEDADESDADOS propriedadeDados+)
    ;

propriedadeDados
    :   PROPRIEDADEDADOS '(' STRING ')'
```

```
52      -> ^(PROPRIEDADEDADOS STRING)
53      ;
54
55  propriedadesConceito
56      :   propriedadeConceito (';' propriedadeConceito)*
57      -> ^(PROPRIEDADESCONCEITO propriedadeConceito+)
58      ;
59
60  propriedadeConceito
61      :   PROPRIEDADECONCEITO '(' STRING ')'
62      -> ^(PROPRIEDADECONCEITO STRING)
63      ;
64
65  mapasConceitos
66      :   mapaConceitos (';' mapaConceitos )*
67      -> ^(MAPASCONCEITOS mapaConceitos+)
68      ;
69
70  mapaConceitos
71      :   MAPACONCEITOS '('ID ','  STRING ',' STRING ')'
72      -> ^(MAPACONCEITOS ID STRING STRING)
73      ;
74
75
76  mapasConceitoPropDados
77      :   mapaConceitoPropDados (';' mapaConceitoPropDados )*
78      -> ^(MAPASCONCEITOPROPDADOS mapaConceitoPropDados+)
79      ;
80
81  mapaConceitoPropDados
82      :   MAPACONCEITOPROPDADOS '('ID ','  STRING ',' STRING ',' tipo ')'
83      -> ^(MAPACONCEITOPROPDADOS ID STRING STRING tipo)
84      ;
85
86  mapasConceitoPropConceito
87      :   mapaConceitoPropConceito (';' mapaConceitoPropConceito )*
88      -> ^(MAPASCONCEITOPROPCONCEITO mapaConceitoPropConceito+)
89      ;
90
91  mapaConceitoPropConceito
92      :   MAPACONCEITOPROPCONCEITO '('ID ','  STRING ',' STRING ',' STRING ')'
93      -> ^(MAPACONCEITOPROPCONCEITO ID STRING STRING STRING)
94      ;
95
96  instancias
97      :   instancia (';' instancia)*
98      -> ^(INSTANCIAS instancia+)
99      ;
100
101 instancia
102     :   INSTANCIA '(' ID ',' STRING')'
103     -> ^(INSTANCIA ID STRING)
104     ;
105
106 mapasInstanciaPropDados
107     :   mapaInstanciaPropDados (';' mapaInstanciaPropDados )*
108     -> ^(MAPASINSTANCIAPROPDADOS mapaInstanciaPropDados+)
109     ;
110
111 mapaInstanciaPropDados
112     :   MAPAINSTANCIAPROPDADOS '('ID ','  ID ',' STRING ')'
```

```
113     -> ^(MAPAINSTANCIAPROPDADOS ID ID STRING)
114     ;
115
116 mapasInstanciaPropConceito
117     :   mapaInstanciaPropConceito (';' mapaInstanciaPropConceito )*
118     -> ^(MAPASINSTANCIAPROPCONCEITO mapaInstanciaPropConceito+)
119     ;
120
121 mapaInstanciaPropConceito
122     :   MAPAINSTANCIAPROPCONCEITO '(' ID ',' ID ',' ID ')'
123     -> ^(MAPAINSTANCIAPROPCONCEITO ID ID ID)
124     ;
125
126 tipo
127     :   'STRING' -> 'STRING'
128     |   'INT' -> 'INT'
129     |   ID  ->  ID
130     ;
131
132
133
134 ID  :   ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
135     ;
136
137 COMMENT
138     :   '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
139     |   '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
140     ;
141
142 WS  :   ( ' '
143         | '\t'
144         | '\r'
145         | '\n'
146         ) {$channel=HIDDEN;}
147     ;
148
149 STRING
150     :  '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
151     ;
152
153 fragment
154 HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;
155
156 fragment
157 ESC_SEQ
158     :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
159     |   UNICODE_ESC
160     |   OCTAL_ESC
161     ;
162
163 fragment
164 OCTAL_ESC
165     :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
166     |   '\\' ('0'..'7') ('0'..'7')
167     |   '\\' ('0'..'7')
168     ;
169
170 fragment
171 UNICODE_ESC
172     :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
173     ;
```

7.2 Tree Grammar

```
tree grammar mapaconceitosTGValidacao;

options{
    tokenVocab=cmc;
    ASTLabelType = CommonTree;
    backtrack = true;
}

@header{
    import java.util.TreeSet;
    import java.util.HashSet;
    import java.util.TreeMap;
}



cmc returns [Tabela tab_out, String erro_out]
@init{
    Tabela tab = new Tabela();
    String erro = "Erros:";
}
@after{
}
    :   ^(CMC (conceitos[tab, erro] {tab = $conceitos.tab_out; erro = $conceitos.
        erro_out;})
                (propriedadesDados[tab, erro] {tab = $propriedadesDados.tab_out; erro
                    = $propriedadesDados.erro_out;})?
                (propriedadesConceito[tab, erro] {tab = $propriedadesConceito.
                    tab_out; erro = $propriedadesConceito.erro_out;})?
                (mapasConceitos[tab, erro] {tab = $mapasConceitos.tab_out; erro =
                    $mapasConceitos.erro_out;})
                (mapasConceitoPropDados[tab, erro] {tab = $mapasConceitoPropDados.
                    tab_out; erro = $mapasConceitoPropDados.erro_out;})?
                (mapasConceitoPropConceito[tab, erro] {tab =
                    $mapasConceitoPropConceito.tab_out; erro =
                    $mapasConceitoPropConceito.erro_out;})?
                (instancias[tab, erro] {tab = $instancias.tab_out; erro = $instancias
                    .erro_out;})?
                (mapasInstanciaPropDados[tab, erro] {tab = $mapasInstanciaPropDados.
                    tab_out; erro = $mapasInstanciaPropDados.erro_out;})?
                (mapasInstanciaPropConceito[tab, erro] {tab =
                    $mapasInstanciaPropConceito.tab_out; erro =
                    $mapasInstanciaPropConceito.erro_out;})?
        )
        {
            $cmc.erro_out = erro;
            $cmc.tab_out = tab;
        }
    ;

conceitos [Tabela tab_in, String erro_in] returns [Tabela tab_out, String erro_out]
    :   ^(CONCEITOS (conceito[$conceitos.tab_in, $conceitos.erro_in]
    {
        $conceitos.tab_in = $conceito.tab_out;
        $conceitos.erro_in = $conceito.erro_out;
    }
    )+
    {
        $conceitos.tab_out = $conceito.tab_out;
```

```
49          $conceitos.erro_out = $conceito.erro_out;
50      }
51      )
52      ;
53
54  conceito [Tabela tab_in, String erro_in] returns [Tabela tab_out, String erro_out]
55      :   ^(CONCEITO STRING)
56      {
57          Tabela t = $conceito.tab_in;
58          // adiciona o conceito Ã  tabela
59          TreeSet<String> conceitos = t.getConceitos();
60          conceitos.add($STRING.text);
61          t.setConceitos(conceitos);
62
63          $conceito.tab_out = t;
64          $conceito.erro_out = $conceito.erro_in;
65      }
66      ;
67
68  propriedadesDados [Tabela tab_in, String erro_in] returns [Tabela tab_out, String
        erro_out]
69      :   ^(PROPRIEDADESDADOS (propriedadeDados[$propriedadesDados.tab_in,
            $propriedadesDados.erro_in]
70      {
71          $propriedadesDados.tab_in = $propriedadeDados.tab_out;
72          $propriedadesDados.erro_in = $propriedadeDados.erro_out;
73      }
74      )+
75      {
76          $propriedadesDados.tab_out = $propriedadeDados.tab_out;
77          $propriedadesDados.erro_out = $propriedadeDados.erro_out;
78      }
79      )
80      ;
81
82  propriedadeDados [Tabela tab_in, String erro_in] returns [Tabela tab_out, String
        erro_out]
83      :   ^(PROPRIEDADEDADOS STRING)
84      {
85          Tabela t = $propriedadeDados.tab_in;
86          // adiciona a propriedade de dados Ã  tabela
87          TreeSet<String> propriedadesDados = t.getPropriedadesDados();
88          propriedadesDados.add($STRING.text);
89          t.setPropriedadesDados(propriedadesDados);
90
91          $propriedadeDados.tab_out = t;
92          $propriedadeDados.erro_out = $propriedadeDados.erro_in ;
93      }
94      ;
95
96  propriedadesConceito [Tabela tab_in, String erro_in] returns [Tabela tab_out, String
        erro_out]
97      :   ^(PROPRIEDADESCONCEITO (propriedadeConceito[$propriedadesConceito.tab_in,
            $propriedadesConceito.erro_in]
98      {
99          $propriedadesConceito.tab_in = $propriedadeConceito.tab_out;
100         $propriedadesConceito.erro_in = $propriedadeConceito.erro_out;
101     }
102     )+
103     {
104         $propriedadesConceito.tab_out = $propriedadeConceito.tab_out;
```

```
105         $propriedadesConceito.erro_out = $propriedadeConceito.erro_out;
106     }
107     )
108     ;
109 
110 propriedadeConceito [Tabela tab_in, String erro_in] returns [Tabela tab_out, String
    erro_out]
111     :   ^(PROPRIEDADECONCEITO STRING)
112     {
113         Tabela t = $propriedadeConceito.tab_in;
114         // adiciona a propriedade de conceito Ã  tabela
115         TreeSet<String> propriedadesConceito = t.getPropriedadesConceito();
116         propriedadesConceito.add($STRING.text);
117         t.setPropriedadesConceito(propriedadesConceito);
118 
119         $propriedadeConceito.tab_out = t;
120         $propriedadeConceito.erro_out = $propriedadeConceito.erro_in ;
121     }
122     ;
123 
124 
125 mapasConceitos [Tabela tab_in, String erro_in] returns [Tabela tab_out, String
    erro_out]
126 @init{
127     Tabela t = $mapasConceitos.tab_in;
128 }
129     :   ^(MAPASCONCEITOS (mapaConceitos[t, $mapasConceitos.erro_in]
130     {
131         $mapasConceitos.erro_in = $mapaConceitos.erro_out;
132         t = $mapaConceitos.tab_out;
133     }
134     )+
135     {
136         $mapasConceitos.tab_out = $mapaConceitos.tab_out;
137         $mapasConceitos.erro_out = $mapaConceitos.erro_out;
138     }
139     )
140     ;
141 
142 mapaConceitos [Tabela tab_in, String erro_in] returns [Tabela tab_out, String
    erro_out]
143 @init{
144     String erro = $mapaConceitos.erro_in;
145     Tabela t = $mapaConceitos.tab_in;
146 }
147     :   ^(MAPACONCEITOS ID ci=STRING cf=STRING)
148     {
149         Boolean ciSemErro = true;
150         Boolean cfSemErro = true;
151 
152         // verifica se existem erros e constroi string de erros
153         if (!(ciSemErro = t.getConceitos().contains($ci.text)))
154             erro += "\n\t("+$ci.line+":"+$ci.pos+")\tO conceito "+$ci.text+" nao foi
                definido!";
155         if (!(cfSemErro = t.getConceitos().contains($cf.text)))
156             erro += "\n\t("+$cf.line+":"+$cf.pos+")\tO conceito "+$cf.text+" nao foi
                definido!";
157 
158         // se nao existirem erros insere Mapa na tabela
159         if (ciSemErro && cfSemErro) {
160             TreeMap<String, MapaConceitos> mapas = t.getMapasConceitos();
```

```
161             mapas.put($ID.text, new MapaConceitos($ID.text, $ci.text, $cf.text));
162             t.setMapasConceitos(mapas);
163         }
164 
165         $mapaConceitos.erro_out = erro;
166         $mapaConceitos.tab_out = t;
167     }
168     ;
169 
170 mapasConceitoPropDados [Tabela tab_in, String erro_in] returns [Tabela tab_out,
    String erro_out]
171 @init{
172     Tabela t = $mapasConceitoPropDados.tab_in;
173 }
174     :   ^(MAPASCONCEITOPROPDADOS (mapaConceitoPropDados[t, $mapasConceitoPropDados.
        erro_in]
175     {
176         $mapasConceitoPropDados.erro_in = $mapaConceitoPropDados.erro_out;
177         t = $mapaConceitoPropDados.tab_out;
178     }
179     )+
180     {
181         $mapasConceitoPropDados.tab_out = $mapaConceitoPropDados.tab_out;
182         $mapasConceitoPropDados.erro_out = $mapaConceitoPropDados.erro_out;
183     }
184     )
185     ;
186 
187 mapaConceitoPropDados [Tabela tab_in, String erro_in] returns [Tabela tab_out, String
     erro_out]
188 @init{
189     String erro = $mapaConceitoPropDados.erro_in;
190     Tabela t = $mapaConceitoPropDados.tab_in;
191 }
192     :   ^(MAPACONCEITOPROPDADOS ID c=STRING prop=STRING tipo)
193     {
194         Boolean cSemErro = true;
195         Boolean pSemErro = true;
196         Boolean tSemErro = true;
197 
198         // verifica se existem erros e constroi string de erros
199         if (!(cSemErro = t.getConceitos().contains($c.text)))
200             erro += "\n\t("+$c.line+":"+$c.pos+")\tO conceito "+$c.text+" nao foi
                definido!";
201         if (!(pSemErro = t.getPropriedadesDados().contains($prop.text)))
202             erro += "\n\t("+$prop.line+":"+$prop.pos+")\tA propriedade de dados "+
                $prop.text+" nao foi definida!";
203         if (!(tSemErro = ($tipo.val.equals("STRING") || $tipo.val.equals("INT"))))
204             erro += "\n\t("+$c.line+")\tO tipo da propriedade de dados "+$tipo.val+"
                nao foi definido!";
205 
206         // se nao existirem erros insere Mapa na tabela
207         if (cSemErro && pSemErro && tSemErro) {
208             TreeMap<String, MapaConceitoPropDados> mapas = t.
                getMapasConceitoPropDados();
209             mapas.put($ID.text, new MapaConceitoPropDados($ID.text, $c.text, $prop.
                text, $tipo.val));
210             t.setMapasConceitoPropDados(mapas);
211         }
212 
213         $mapaConceitoPropDados.erro_out = erro;
```

```
214         $mapaConceitoPropDados.tab_out = t;
215     }
216     ;
217
218 mapasConceitoPropConceito [Tabela tab_in, String erro_in] returns [Tabela tab_out,
        String erro_out]
219 @init{
220     Tabela t = $mapasConceitoPropConceito.tab_in;
221 }
222     :   ^(MAPASCONCEITOPROPCONCEITO (mapaConceitoPropConceito[t,
            $mapasConceitoPropConceito.erro_in]
223     {
224         $mapasConceitoPropConceito.erro_in = $mapaConceitoPropConceito.erro_out;
225         t = $mapaConceitoPropConceito.tab_out;
226     }
227     )+
228     {
229         $mapasConceitoPropConceito.tab_out = $mapaConceitoPropConceito.tab_out;
230         $mapasConceitoPropConceito.erro_out = $mapaConceitoPropConceito.erro_out;
231     }
232     )
233     ;
234
235 mapaConceitoPropConceito [Tabela tab_in, String erro_in] returns [Tabela tab_out,
        String erro_out]
236 @init{
237     String erro = $mapaConceitoPropConceito.erro_in;
238     Tabela t = $mapaConceitoPropConceito.tab_in;
239 }
240     :   ^(MAPACONCEITOPROPCONCEITO ID c=STRING prop=STRING cp=STRING)
241     {
242         Boolean cSemErro = true;
243         Boolean pSemErro = true;
244         Boolean cpSemErro = true;
245
246         // verifica se existem erros e constroi string de erros
247         if (!(cSemErro = t.getConceitos().contains($c.text)))
248             erro += "\n\t("+$c.line+":"+$c.pos+")\tO conceito "+$c.text+" nao foi
                    definido!";
249         if (!(pSemErro = t.getPropriedadesConceito().contains($prop.text)))
250             erro += "\n\t("+$prop.line+":"+$prop.pos+")\tA propriedade de conceito "+
                    $prop.text+" nao foi definida!";
251         if (!(cpSemErro = t.getConceitos().contains($c.text)))
252             erro += "\n\t("+$cp.line+":"+$cp.pos+")\tO conceito "+$cp.text+" nao foi
                    definido!";
253
254         // se nao existirem erros insere Mapa na tabela
255         if (cSemErro && pSemErro && cpSemErro) {
256             TreeMap<String, MapaConceitoPropConceito> mapas = t.
                    getMapasConceitoPropConceito();
257             mapas.put($ID.text, new MapaConceitoPropConceito($ID.text, $cp.text,
                    $prop.text, $c.text));
258             t.setMapasConceitoPropConceito(mapas);
259         }
260
261         $mapaConceitoPropConceito.erro_out = erro;
262         $mapaConceitoPropConceito.tab_out = t;
263     }
264     ;
265
266 instancias [Tabela tab_in, String erro_in] returns [Tabela tab_out, String erro_out]
```

```
@init{
    Tabela t = $instancias.tab_in;
}
    :   ^(INSTANCIAS (instancia[t, $instancias.erro_in]
    {
        $instancias.erro_in = $instancia.erro_out;
        t = $instancia.tab_out;
    }
    )+
    {
        $instancias.tab_out = $instancia.tab_out;
        $instancias.erro_out = $instancia.erro_out;
    })
    ;

instancia [Tabela tab_in, String erro_in] returns [Tabela tab_out, String erro_out]
@init{
    String erro = $instancia.erro_in;
    Tabela t = $instancia.tab_in;
}
    :   ^(INSTANCIA ID STRING)
    {
        Boolean cSemErro = true;

        // verifica se existem erros e constroi string de erros
        if (!(cSemErro = t.getConceitos().contains($STRING.text)))
            erro += "\n\t("+$STRING.line+":"+$STRING.pos+")\tO conceito "+$STRING.
                text+" nao foi definido!";

        // se nao existirem erros insere instancia na tabela
        if (cSemErro) {
            TreeMap<String, Instancia> instancias = t.getInstancias();
            instancias.put($ID.text, new Instancia($ID.text, $STRING.text));
            t.setInstancias(instancias);
        }

        $instancia.erro_out = erro;
        $instancia.tab_out = t;
    }
    ;

mapasInstanciaPropDados [Tabela tab_in, String erro_in] returns [Tabela tab_out,
    String erro_out]
@init{
    Tabela t = $mapasInstanciaPropDados.tab_in;
}
    :   ^(MAPASINSTANCIAPROPDADOS (mapaInstanciaPropDados[t, $mapasInstanciaPropDados
        .erro_in]
        {
            $mapasInstanciaPropDados.erro_in = $mapaInstanciaPropDados.erro_out;
            t = $mapaInstanciaPropDados.tab_out;
        }
        )+
        {
            $mapasInstanciaPropDados.tab_out = $mapaInstanciaPropDados.tab_out;
            $mapasInstanciaPropDados.erro_out = $mapaInstanciaPropDados.erro_out;
        })
    ;

mapaInstanciaPropDados [Tabela tab_in, String erro_in] returns [Tabela tab_out,
    String erro_out]
```

```
@init{
    String erro = $mapaInstanciaPropDados.erro_in;
    Tabela t = $mapaInstanciaPropDados.tab_in;
}
    :   ^(MAPAINSTANCIAPROPDADOS inst=ID mapaConcProp=ID val=STRING)
    {
        Boolean instSemErro = true;
        Boolean mapaConcPropSemErro = true;

        // verifica se existem erros e constroi string de erros
        if (!(instSemErro = t.getInstancias().containsKey($inst.text)))
            erro += "\n\t("+$inst.line+":"+$inst.pos+")\tA instancia "+$inst.text+"
                nao foi definida!";
        if (!(mapaConcPropSemErro = t.getMapasConceitoPropDados().containsKey(
            $mapaConcProp.text)))
            erro += "\n\t("+$mapaConcProp.line+":"+$mapaConcProp.pos+")\tO mapa entre
                 conceito e propriedade de dados "+$mapaConcProp.text+" nao foi
                definido!";

        // se nao existirem erros insere Mapa na tabela
        if (instSemErro && mapaConcPropSemErro) {
            HashSet<MapaInstanciaPropDados> mapasInstanciaProp = t.
                getMapasInstanciaPropDados();
            mapasInstanciaProp.add(new MapaInstanciaPropDados($inst.text,
                $mapaConcProp.text, $val.text));
            t.setMapasInstanciaPropDados(mapasInstanciaProp);
        }

        $mapaInstanciaPropDados.erro_out = erro;
        $mapaInstanciaPropDados.tab_out = t;
    }
    ;

mapasInstanciaPropConceito [Tabela tab_in, String erro_in] returns [Tabela tab_out,
    String erro_out]
@init{
    Tabela t = $mapasInstanciaPropConceito.tab_in;
}
    :   ^(MAPASINSTANCIAPROPCONCEITO (mapaInstanciaPropConceito[t,
        $mapasInstanciaPropConceito.erro_in]
    {
        $mapasInstanciaPropConceito.erro_in = $mapaInstanciaPropConceito.erro_out;
        t = $mapaInstanciaPropConceito.tab_out;
    }
    )+
    {
        $mapasInstanciaPropConceito.tab_out = $mapaInstanciaPropConceito.tab_out;
        $mapasInstanciaPropConceito.erro_out = $mapaInstanciaPropConceito.erro_out;
    })
    ;

mapaInstanciaPropConceito [Tabela tab_in, String erro_in] returns [Tabela tab_out,
    String erro_out]
@init{
    String erro = $mapaInstanciaPropConceito.erro_in;
    Tabela t = $mapaInstanciaPropConceito.tab_in;
}
    :   ^(MAPAINSTANCIAPROPCONCEITO instFilho=ID mapaConc=ID instPai=ID)
    {
        Boolean instFilhoSemErro = true;
        Boolean mapaConceitoSemErro = true;
```

```
376         Boolean instPaiSemErro = true;
377 
378         // verifica se existem erros e constroi string de erros
379         if (!(instFilhoSemErro = t.getInstancias().containsKey($instFilho.text)))
380             erro += "\n\t("+$instFilho.line+":"+$instFilho.pos+")\tA instancia "+
                    $instFilho.text+" nao foi definida!";
381         if (!(mapaConceitoSemErro = t.getMapasConceitoPropConceito().containsKey(
                $mapaConc.text)))
382             erro += "\n\t("+$mapaConc.line+":"+$mapaConc.pos+")\tO mapa entre
                    conceitos e propriedade de conceito "+$mapaConc.text+" nao foi
                    definido!";
383         if (!(instPaiSemErro = t.getInstancias().containsKey($instPai.text)))
384             erro += "\n\t("+$instPai.line+":"+$instPai.pos+")\tA instancia "+$instPai
                    .text+" nao foi definida!";
385 
386         // se nao existirem erros insere Mapa na tabela
387         if (instFilhoSemErro && mapaConceitoSemErro && instPaiSemErro) {
388             HashSet<MapaInstanciaPropConceito> mapasInstancias = t.
                    getMapasInstanciaPropConceito();
389             mapasInstancias.add(new MapaInstanciaPropConceito($instFilho.text,
                    $mapaConc.text, $instPai.text));
390             t.setMapasInstanciaPropConceito(mapasInstancias);
391         }
392 
393         $mapaInstanciaPropConceito.erro_out = erro;
394         $mapaInstanciaPropConceito.tab_out = t;
395     }
396     ;
397 
398 tipo returns [String val]
399     :   'STRING' { $tipo.val = "STRING"; }
400     |   'INT' { $tipo.val = "INT"; }
401     |   ID { $tipo.val = $ID.text; }
402     ;
```

7.3 Classes Java

7.3.1 Run.java

```
1  import org.antlr.runtime.*;
2  import org.antlr.runtime.tree.*;
3  //import org.antlr.stringtemplate.*;
4  import java.io.*;
5
6  public class Run {
7
8      public static void main(String[] args) throws Exception {
9          try{
10             CharStream in = new ANTLRFileStream(args[0], "UTF8");
11             cmcLexer lexer = new cmcLexer(in);
12             CommonTokenStream tokens = new CommonTokenStream(lexer);
13             cmcParser parser = new cmcParser(tokens);
14             cmcParser.cmc_return ret = parser.cmc();
15
16             // obtem a AST utilizando as regras de reescrita da gramÃ¡tica
                   criada
17             CommonTreeNodeStream tree = new CommonTreeNodeStream(ret.getTree
                   ());
18
19             // Tree Walking. Utiliza a Tree Grammar criada
20             mapaconceitosTGValidacao walker = new mapaconceitosTGValidacao(
                   tree);
21             mapaconceitosTGValidacao.cmc_return walker_ret = walker.cmc();
22
23             //System.out.println(walker_ret.tab_out);
24             //System.out.println(walker_ret.erro_out);
25
26         }
27         catch(Exception e){
28             e.printStackTrace();
29         }
30     }
31
32 }
```

7.3.2 Tabela.java

```
1  import java.util.*;
2
3  public class Tabela {
4      private TreeSet<String> conceitos;
5      private TreeSet<String> propriedadesDados;
6      private TreeSet<String> propriedadesConceito;
7      private TreeMap<String, MapaConceitos> mapasConceitos;
8      private TreeMap<String, MapaConceitoPropDados> mapasConceitoPropDados;
9      private TreeMap<String, MapaConceitoPropConceito>
           mapasConceitoPropConceito;
10     private TreeMap<String, Instancia> instancias;
11     private HashSet<MapaInstanciaPropDados> mapasInstanciaPropDados;
12     private HashSet<MapaInstanciaPropConceito> mapasInstanciaPropConceito;
13
```

```
14      public Tabela (){
15          this.setConceitos(new TreeSet<String>());
16          this.setPropriedadesDados(new TreeSet<String>());
17          this.setPropriedadesConceito(new TreeSet<String>());
18          this.setMapasConceitos(new TreeMap<String, MapaConceitos>());
19          this.setMapasConceitoPropDados(new TreeMap<String,
                MapaConceitoPropDados>());
20          this.setMapasConceitoPropConceito(new TreeMap<String,
                MapaConceitoPropConceito>());
21          this.setInstancias(new TreeMap<String, Instancia>());
22          this.setMapasInstanciaPropDados(new HashSet<MapaInstanciaPropDados
                >());
23          this.setMapasInstanciaPropConceito(new HashSet<
                MapaInstanciaPropConceito>());
24      }
25
26
27      /**
28       * @return the conceitos
29       */
30      public TreeSet<String> getConceitos() {
31          return conceitos;
32      }
33
34
35      /**
36       * @param conceitos the conceitos to set
37       */
38      public void setConceitos(TreeSet<String> conceitos) {
39          this.conceitos = conceitos;
40      }
41
42
43      /**
44       * @return the propriedadesDados
45       */
46      public TreeSet<String> getPropriedadesDados() {
47          return propriedadesDados;
48      }
49
50
51      /**
52       * @param propriedadesDados the propriedadesDados to set
53       */
54      public void setPropriedadesDados(TreeSet<String> propriedadesDados) {
55          this.propriedadesDados = propriedadesDados;
56      }
57
58
59      /**
60       * @return the propriedadesConceito
61       */
62      public TreeSet<String> getPropriedadesConceito() {
63          return propriedadesConceito;
64      }
65
```

```
66
67      /**
68       * @param propriedadesConceito the propriedadesConceito to set
69       */
70      public void setPropriedadesConceito(TreeSet<String> propriedadesConceito
            ) {
71          this.propriedadesConceito = propriedadesConceito;
72      }
73
74
75      /**
76       * @return the mapasConceitos
77       */
78      public TreeMap<String, MapaConceitos> getMapasConceitos() {
79          return mapasConceitos;
80      }
81
82
83      /**
84       * @param mapasConceitos the mapasConceitos to set
85       */
86      public void setMapasConceitos(TreeMap<String, MapaConceitos>
            mapasConceitos) {
87          this.mapasConceitos = mapasConceitos;
88      }
89
90
91      /**
92       * @return the mapasConceitoPropDados
93       */
94      public TreeMap<String, MapaConceitoPropDados> getMapasConceitoPropDados
            () {
95          return mapasConceitoPropDados;
96      }
97
98
99      /**
100      * @param mapasConceitoPropDados the mapasConceitoPropDados to set
101      */
102     public void setMapasConceitoPropDados(
103             TreeMap<String, MapaConceitoPropDados> mapasConceitoPropDados) {
104         this.mapasConceitoPropDados = mapasConceitoPropDados;
105     }
106
107
108     /**
109      * @return the mapasConceitoPropConceito
110      */
111     public TreeMap<String, MapaConceitoPropConceito>
            getMapasConceitoPropConceito() {
112         return mapasConceitoPropConceito;
113     }
114
115
116     /**
117      * @param mapasConceitoPropConceito the mapasConceitoPropConceito to set
```

```
118         */
119        public void setMapasConceitoPropConceito(
120                TreeMap<String, MapaConceitoPropConceito>
                       mapasConceitoPropConceito) {
121            this.mapasConceitoPropConceito = mapasConceitoPropConceito;
122        }
123
124
125        /**
126         * @return the instancias
127         */
128        public TreeMap<String, Instancia> getInstancias() {
129            return instancias;
130        }
131
132
133        /**
134         * @param instancias the instancias to set
135         */
136        public void setInstancias(TreeMap<String, Instancia> instancias) {
137            this.instancias = instancias;
138        }
139
140
141        /**
142         * @return the mapasInstanciaPropDados
143         */
144        public HashSet<MapaInstanciaPropDados> getMapasInstanciaPropDados() {
145            return mapasInstanciaPropDados;
146        }
147
148
149        /**
150         * @param mapasInstanciaPropDados the mapasInstanciaPropDados to set
151         */
152        public void setMapasInstanciaPropDados(
153                HashSet<MapaInstanciaPropDados> mapasInstanciaPropDados) {
154            this.mapasInstanciaPropDados = mapasInstanciaPropDados;
155        }
156
157
158        /**
159         * @return the mapasInstanciaPropConceito
160         */
161        public HashSet<MapaInstanciaPropConceito> getMapasInstanciaPropConceito
               () {
162            return mapasInstanciaPropConceito;
163        }
164
165
166        /**
167         * @param mapasInstanciaPropConceito the mapasInstanciaPropConceito to
                   set
168         */
169        public void setMapasInstanciaPropConceito(
170                HashSet<MapaInstanciaPropConceito> mapasInstanciaPropConceito) {
```

```
171             this.mapasInstanciaPropConceito = mapasInstanciaPropConceito;
172         }
173 
174 
175         /**
176          * @return the SQL instructions to load tabela's data into the CMC
                Database
177          */
178         public String geraInstrucoesSQL() {
179             StringBuilder sb = new StringBuilder();
180                 sb.append(conceitosToString(this.conceitos));
181                 sb.append(propriedadesDadosToString(this.propriedadesDados));
182                 sb.append(propriedadesConceitoToString(this.propriedadesConceito
                        ));
183                 sb.append(mapasConceitosToString(this.mapasConceitos));
184                 sb.append(mapasConceitoPropDadosToString(this.
                        mapasConceitoPropDados));
185                 sb.append(mapasConceitoPropConceitoToString(this.
                        mapasConceitoPropConceito));
186                 sb.append(instanciasToString(this.instancias));
187                 sb.append(mapasInstanciaPropDadosToString(this.
                        mapasInstanciaPropDados));
188                 sb.append(mapasInstanciaPropConceitoToString(this.
                        mapasInstanciaPropConceito));
189             return sb.toString();
190         }
191 
192         /**
193          * @return the SQL instructions to load conceitos into the CMC Database
194          */
195         public String conceitosToString(TreeSet<String> conceitos){
196             StringBuilder sb = new StringBuilder();
197             for(String conceito : conceitos){
198                 sb.append("INSERT INTO `mapaconceitos`.`Conceitos` ");
199                 sb.append("(`conceito`) ");
200                 sb.append("VALUES ");
201                 sb.append("("+conceito+");\n");
202             }
203             sb.append("\n");
204             return sb.toString();
205         }
206 
207         /**
208          * @return the SQL instructions to load propriedadesDados into the CMC
                Database
209          */
210         public String propriedadesDadosToString(TreeSet<String>
                propriedadesDados){
211             StringBuilder sb = new StringBuilder();
212             for(String propriedadeDados : propriedadesDados){
213                 sb.append("INSERT INTO `mapaconceitos`.`PropriedadesDados` ");
214                 sb.append("(`propriedadeDados`) ");
215                 sb.append("VALUES ");
216                 sb.append("("+propriedadeDados+");\n");
217             }
218             sb.append("\n");
```

```
219             return sb.toString();
220         }
221 
222         /**
223          * @return the SQL instructions to load propriedades Conceito into the
                CMC Database
224          */
225         public String propriedadesConceitoToString(TreeSet<String>
            propriedadesConceito){
226             StringBuilder sb = new StringBuilder();
227             for(String propriedadeConceito : propriedadesConceito){
228                 sb.append("INSERT INTO `mapaconceitos`.`PropriedadesConceito` ")
                    ;
229                 sb.append("(`propriedadeConceito`) ");
230                 sb.append("VALUES ");
231                 sb.append("("+propriedadeConceito+");\n");
232             }
233             sb.append("\n");
234             return sb.toString();
235         }
236 
237         /**
238          * @return the SQL instructions to load mapasConceitos into the CMC
                Database
239          */
240         public String mapasConceitosToString(TreeMap<String, MapaConceitos>
            mapas){
241             StringBuilder sb = new StringBuilder();
242             for(MapaConceitos mapa : mapas.values()){
243                 sb.append(mapa.sqlToString());
244             }
245             sb.append("\n");
246             return sb.toString();
247         }
248 
249         /**
250          * @return the SQL instructions to load mapasConceitoPropDados into the
                CMC Database
251          */
252         public String mapasConceitoPropDadosToString(TreeMap<String,
            MapaConceitoPropDados> mapas){
253             StringBuilder sb = new StringBuilder();
254             for(MapaConceitoPropDados mapa : mapas.values()){
255                 sb.append(mapa.sqlToString());
256             }
257             sb.append("\n");
258             return sb.toString();
259         }
260 
261         /**
262          * @return the SQL instructions to load mapasConceitoPropConceito into
                the CMC Database
263          */
264         public String mapasConceitoPropConceitoToString(TreeMap<String,
            MapaConceitoPropConceito> mapas){
265             StringBuilder sb = new StringBuilder();
```

```
266             for(MapaConceitoPropConceito mapa : mapas.values()){
267                 sb.append(mapa.sqlToString());
268             }
269             sb.append("\n");
270             return sb.toString();
271         }
272 
273         /**
274          * @return the SQL instructions to load instancias into the CMC Database
275          */
276         public String instanciasToString(TreeMap<String, Instancia> instancias){
277             StringBuilder sb = new StringBuilder();
278             for(Instancia instancia : instancias.values()){
279                 sb.append(instancia.sqlToString());
280             }
281             sb.append("\n");
282             return sb.toString();
283         }
284 
285         /**
286          * @return the SQL instructions to load mapasInstanciaPropDados into the
                 CMC Database
287          */
288         public String mapasInstanciaPropDadosToString(HashSet<
                MapaInstanciaPropDados> mapasInstanciaPropDados){
289             StringBuilder sb = new StringBuilder();
290             for(MapaInstanciaPropDados mapaInstanciaPropDados :
                    mapasInstanciaPropDados){
291                 sb.append(mapaInstanciaPropDados.sqlToString());
292             }
293             sb.append("\n");
294             return sb.toString();
295         }
296 
297         /**
298          * @return the SQL instructions to load mapasInstanciaPropConceito into
                 the CMC Database
299          */
300         public String mapasInstanciaPropConceitoToString(HashSet<
                MapaInstanciaPropConceito> mapasInstanciaPropConceito){
301             StringBuilder sb = new StringBuilder();
302             for(MapaInstanciaPropConceito mapaInstanciaPropConceito :
                    mapasInstanciaPropConceito){
303                 sb.append(mapaInstanciaPropConceito.sqlToString());
304             }
305             sb.append("\n");
306             return sb.toString();
307         }
308 
309         @Override
310         public String toString() {
311             return "Tabela [\n" +
312                     "conceitos=" + conceitos + ",\n" +
313                     "propriedadesDados=" + propriedadesDados + ",\n" +
314                     "propriedadesConceito=" + propriedadesConceito + ",\n" +
315                     "mapasConceitos=" + mapasConceitos + ",\n" +
```

```
                "mapasConceitoPropDados=" + mapasConceitoPropDados + ",\n" +
                "mapasConceitoPropConceito=" + mapasConceitoPropConceito + ",\n" +
                "instancias=" + instancias + ",\n" +
                "mapasInstanciaPropDados=" + mapasInstanciaPropDados + ",\n" +
                "mapasInstanciaPropConceito=" + mapasInstanciaPropConceito + "\n" +
                "]";
    }
}
```

7.3.3 MapaConceitos.java

```

public class MapaConceitos {
    private String id;
    private String conceitoFilho;
    private String conceitoPai;

    /**
     * @param id
     * @param conceitoFilho
     * @param conceitoPai
     */
    public MapaConceitos(String id, String conceitoFilho, String conceitoPai) {
        super();
        this.id = id;
        this.conceitoFilho = conceitoFilho;
        this.conceitoPai = conceitoPai;
    }

    /**
     * @return the id
     */
    public String getId() {
        return id;
    }

    /**
     * @param id the id to set
     */
    public void setId(String id) {
        this.id = id;
    }

    /**
     * @return the conceitoFilho
     */
    public String getConceitoFilho() {
        return conceitoFilho;
    }

    /**
     * @param conceitoFilho the conceitoFilho to set
     */
```

```
43      public void setConceitoFilho(String conceitoFilho) {
44          this.conceitoFilho = conceitoFilho;
45      }
46
47      /**
48       * @return the conceitoPai
49       */
50      public String getConceitoPai() {
51          return conceitoPai;
52      }
53
54      /**
55       * @param conceitoPai the conceitoPai to set
56       */
57      public void setConceitoPai(String conceitoPai) {
58          this.conceitoPai = conceitoPai;
59      }
60
61      public String sqlToString(){
62          StringBuilder sb = new StringBuilder();
63          sb.append("INSERT INTO `mapaconceitos`.`MapasConceitos` ");
64          sb.append("(`id`, `conceitoFilho`, `conceitoPai`) ");
65          sb.append("VALUES ");
66          sb.append("(\""+id+"\", "+ conceitoFilho +", "+ conceitoPai +");\n")
                ;
67      return sb.toString();
68      }
69
70      @Override
71      public String toString() {
72          return "MapaConceitos [\n\t" +
73                  "id=" + id + ",\n\t" +
74                  "conceitoFilho=" + conceitoFilho + ",\n\t" +
75                  "conceitoPai=" + conceitoPai + "\n\t" +
76                  "]";
77      }
78  }
```

7.3.4 Instancia.java

```
1   public class Instancia {
2
3       private String id;
4       private String conceito;
5
6       public Instancia(String id, String conceito) {
7           super();
8           this.id = id;
9           this.conceito = conceito;
10      }
11
12      public String getId() {
13          return id;
14      }
15
16      public void setId(String id) {
17          this.id = id;
```

```
18      }
19
20      public String getConceito() {
21          return conceito;
22      }
23
24      public void setConceito(String conceito) {
25          this.conceito = conceito;
26      }
27
28      public String sqlToString() {
29          StringBuilder sb = new StringBuilder();
30              sb.append("INSERT INTO `mapaconceitos`.`Instancias` ");
31              sb.append("(`instancia`, `conceito`) ");
32              sb.append("VALUES ");
33              sb.append("(\""+id+"\", "+ conceito +");\n");
34          return sb.toString();
35      }
36
37      @Override
38      public String toString() {
39          return "Instancia [\n\t" +
40                  "id=" + id + ",\n\t" +
41                  "conceito=" + conceito + "\n\t" +
42                  "]";
43      }
44
45
46  }
```

7.3.5 MapaConceitoPropConceito.java

```
1   public class MapaConceitoPropConceito {
2
3       private String id;
4       private String conceitoFilho;
5       private String propriedadeConceito;
6       private String conceitoPai;
7
8       /**
9        * @param id
10       * @param conceito
11       * @param propriedadeConceito
12       * @param valor
13       */
14      public MapaConceitoPropConceito(String id, String conceitoFilho,
15              String propriedadeConceito, String conceitoPai) {
16          super();
17          this.id = id;
18          this.setConceitoFilho(conceitoFilho);
19          this.propriedadeConceito = propriedadeConceito;
20          this.setConceitoPai(conceitoPai);
21      }
22
23      /**
24       * @return the id
25       */
```

```
26      public String getId() {
27          return id;
28      }
29
30      /**
31       * @param id the id to set
32       */
33      public void setId(String id) {
34          this.id = id;
35      }
36
37      /**
38       * @return the conceitoFilho
39       */
40      public String getConceitoFilho() {
41          return conceitoFilho;
42      }
43
44      /**
45       * @param conceitoFilho the conceitoFilho to set
46       */
47      public void setConceitoFilho(String conceitoFilho) {
48          this.conceitoFilho = conceitoFilho;
49      }
50
51      /**
52       * @return the propriedadeConceito
53       */
54      public String getPropriedadeConceito() {
55          return propriedadeConceito;
56      }
57
58      /**
59       * @param propriedadeConceito the propriedadeConceito to set
60       */
61      public void setPropriedadeConceito(String propriedadeConceito) {
62          this.propriedadeConceito = propriedadeConceito;
63      }
64
65      /**
66       * @return the conceitoPai
67       */
68      public String getConceitoPai() {
69          return conceitoPai;
70      }
71
72      /**
73       * @param conceitoPai the conceitoPai to set
74       */
75      public void setConceitoPai(String conceitoPai) {
76          this.conceitoPai = conceitoPai;
77      }
78
79      public String sqlToString(){
80          StringBuilder sb = new StringBuilder();
81          sb.append("INSERT INTO `mapaconceitos`.`MapasConceitoPropConceito` "
```

```
            );
82          sb.append("(`id`, `conceitoPai`, `propriedadeConceito`, `
                conceitoFilho`) ");
83          sb.append("VALUES ");
84          sb.append("(\""+id+"\", "+ conceitoPai +", "+ propriedadeConceito +"
                , " + conceitoFilho + ");\n");
85      return sb.toString();
86      }
87
88      @Override
89      public String toString() {
90          return "MapaConceitoPropConceito [\n\t" +
91                  "id=" + id + ",\n\t" +
92                  "conceitoFilho=" + conceitoFilho + ",\n\t" +
93                  "propriedadeConceito=" + propriedadeConceito + ",\n\t" +
94                  "conceitoPai=" + conceitoPai + "\n\t" +
95                  "]";
96      }
97  }
```

7.3.6 MapaConceitoPropDados.java

```
1   public class MapaConceitoPropDados {
2
3       private String id;
4       private String conceito;
5       private String propriedadeDados;
6       private String tipo;
7
8       /**
9        * @param id
10       * @param conceito
11       * @param propriedadeDados
12       * @param tipo
13       */
14      public MapaConceitoPropDados(String id, String conceito,
15              String propriedadeDados, String tipo) {
16          super();
17          this.id = id;
18          this.conceito = conceito;
19          this.propriedadeDados = propriedadeDados;
20          this.tipo = tipo;
21      }
22
23      /**
24       * @return the id
25       */
26      public String getId() {
27          return id;
28      }
29
30      /**
31       * @param id the id to set
32       */
33      public void setId(String id) {
34          this.id = id;
35      }
```

```
36
37      /**
38       * @return the conceito
39       */
40      public String getConceito() {
41          return conceito;
42      }
43
44      /**
45       * @param conceito the conceito to set
46       */
47      public void setConceito(String conceito) {
48          this.conceito = conceito;
49      }
50
51      /**
52       * @return the propriedadeDados
53       */
54      public String getPropriedadeDados() {
55          return propriedadeDados;
56      }
57
58      /**
59       * @param propriedadeDados the propriedadeDados to set
60       */
61      public void setPropriedadeDados(String propriedadeDados) {
62          this.propriedadeDados = propriedadeDados;
63      }
64
65      /**
66       * @return the tipo
67       */
68      public String getTipo() {
69          return tipo;
70      }
71
72      /**
73       * @param tipo the tipo to set
74       */
75      public void setTipo(String tipo) {
76          this.tipo = tipo;
77      }
78
79      public String sqlToString(){
80          StringBuilder sb = new StringBuilder();
81          sb.append("INSERT INTO `mapaconceitos`.`MapasConceitoPropDados` ");
82          sb.append("(`id`, `conceito`, `propriedadeDados`, `tipoDados`) ");
83          sb.append("VALUES ");
84          sb.append("(\""+id+"\", "+ conceito +", "+ propriedadeDados +", \""
                + tipo + "\");\n");
85      return sb.toString();
86      }
87
88      @Override
89      public String toString() {
90          return "MapaConceitoPropDados [\n\t" +
```

```
                "id=" + id + ",\n\t" +
                "conceito=" + conceito + ",\n\t" +
                "propriedadeDados=" + propriedadeDados + ",\n\t" +
                "tipo=" + tipo + "\n\t" +
                "]";
    }


}
```

7.3.7 MapaInstanciaPropConceito.java

```
public class MapaInstanciaPropConceito {

    private String instanciaFilho;
    private String mapaConceitoPropConceito;
    private String instanciaPai;

    /**
     * @param instanciaFilho
     * @param mapaConceitoPropConceito
     * @param instanciaPai
     */
    public MapaInstanciaPropConceito(String instanciaFilho,
            String mapaConceitoPropConceito, String instanciaPai) {
        super();
        this.instanciaFilho = instanciaFilho;
        this.mapaConceitoPropConceito = mapaConceitoPropConceito;
        this.instanciaPai = instanciaPai;
    }

    /**
     * @return the instanciaFilho
     */
    public String getInstanciaFilho() {
        return instanciaFilho;
    }

    /**
     * @param instanciaFilho the instanciaFilho to set
     */
    public void setInstanciaFilho(String instanciaFilho) {
        this.instanciaFilho = instanciaFilho;
    }

    /**
     * @return the mapaConceitoPropConceito
     */
    public String getMapaConceitoPropConceito() {
        return mapaConceitoPropConceito;
    }

    /**
     * @param mapaConceitoPropConceito the mapaConceitoPropConceito to set
     */
    public void setMapaConceitoPropConceito(String mapaConceitoPropConceito)
        {
```

```
45          this.mapaConceitoPropConceito = mapaConceitoPropConceito;
46      }
47
48      /**
49       * @return the instanciaPai
50       */
51      public String getInstanciaPai() {
52          return instanciaPai;
53      }
54
55      /**
56       * @param instanciaPai the instanciaPai to set
57       */
58      public void setInstanciaPai(String instanciaPai) {
59          this.instanciaPai = instanciaPai;
60      }
61
62      public String sqlToString(){
63          StringBuilder sb = new StringBuilder();
64          sb.append("INSERT INTO `mapaconceitos`.`MapasInstanciaPropConceito`
                ");
65          sb.append("(`instanciaPai`, `mapaConceitoPropConceito`, `
                instanciaFilho`) ");
66          sb.append("VALUES ");
67          sb.append("(\""+ instanciaPai +"\", \""+ mapaConceitoPropConceito +"
                \", \"" + instanciaFilho + "\");\n");
68      return sb.toString();
69      }
70
71      @Override
72      public String toString() {
73          return "MapaInstanciaPropConceito [\n\t" +
74                  "instanciaFilho=" + instanciaFilho + ",\n\t" +
75                  "mapaConceitoPropConceito=" + mapaConceitoPropConceito + ",\
                       n\t" +
76                  "instanciaPai=" + instanciaPai + "\n\t" +
77                  "]";
78      }
79
80
81  }
```

7.3.8 MapaInstanciaPropDados.java

```
1   public class MapaInstanciaPropDados {
2
3       private String instancia;
4       private String mapaConceitoPropDados;
5       private String valor;
6
7       /**
8        * @param instancia
9        * @param mapaConceitoPropDados
10       * @param valor
11       */
12      public MapaInstanciaPropDados(String instancia,
13              String mapaConceitoPropDados, String valor) {
```

```
14          super();
15          this.instancia = instancia;
16          this.mapaConceitoPropDados = mapaConceitoPropDados;
17          this.valor = valor;
18      }
19
20      /**
21       * @return the instancia
22       */
23      public String getInstancia() {
24          return instancia;
25      }
26
27      /**
28       * @param instancia the instancia to set
29       */
30      public void setInstancia(String instancia) {
31          this.instancia = instancia;
32      }
33
34      /**
35       * @return the mapaConceitoPropDados
36       */
37      public String getMapaConceitoPropDados() {
38          return mapaConceitoPropDados;
39      }
40
41      /**
42       * @param mapaConceitoPropDados the mapaConceitoPropDados to set
43       */
44      public void setMapaConceitoPropDados(String mapaConceitoPropDados) {
45          this.mapaConceitoPropDados = mapaConceitoPropDados;
46      }
47
48      /**
49       * @return the valor
50       */
51      public String getValor() {
52          return valor;
53      }
54
55      /**
56       * @param valor the valor to set
57       */
58      public void setValor(String valor) {
59          this.valor = valor;
60      }
61
62      public String sqlToString(){
63          StringBuilder sb = new StringBuilder();
64          sb.append("INSERT INTO `mapaconceitos`.`MapasInstanciaPropDados` ");
65          sb.append("(`instancia`, `mapaConceitoPropDados`, `valor`) ");
66          sb.append("VALUES ");
67          sb.append("(\""+ instancia +"\", \""+ mapaConceitoPropDados +"\", "
                + valor + ");\n");
68      return sb.toString();
```

```
69      }
70
71      @Override
72      public String toString() {
73          return "MapaInstanciaPropDados [\n\t" +
74                  "instancia=" + instancia+ ",\n\t" +
75                  "mapaConceitoPropDados=" + mapaConceitoPropDados + ",\n\t" +
76                  "valor=" + valor + "\n\t" +
77                  "]";
78      }
79
80
81  }
```

7.4 Esquema da Base de Dados