RICARDO PEREIRA BEATO JÚNIOR

GERAÇÃO DE GRAFOS SEMÂNTICOS

São Paulo
2012

RICARDO PEREIRA BEATO JÚNIOR

GERAÇÃO DE GRAFOS SEMÂNTICOS

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Bacharel em Engenharia Mecatrônica.

Área de Concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Marcos Ribeiro Pereira-Barretto

São Paulo
2012

FICHA CATALOGRÁFICA

Beato Júnior, Ricardo Pereira
Geração de grafos semânticos / R.P. Beato Júnior. -- São Paulo, 2012.
111 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Grafos semânticos 2. Fala humana (Interpretação) 3. Interpretadores I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecâni-cos II. t.

A Don Ramón, inesgotável fonte de inspiração.

AGRADECIMENTOS

Às famílias Beato, Granja e Ricardo, a base.

À Escola Politécnica, a ferramenta.

Ao paciente Professor Barretto.

Às noites em claro com Cleber e Jonas.

Aos relevantes comentários e ensinamentos de Guilherme Amantea, que direta e indiretamente contribuiu para que o trabalho evoluísse.

Às dicas linguísticas de Arnaldo Sobrinho.

Ao apoio do William Collen, do Projeto CoGrOO.

“Quando um rio corta, corta-se de vez
o discurso-rio de água que ele fazia;
cortado, a água se quebra em pedaços,
em poços de água, em água paralítica.
Em situação de poço, a água equivale
a uma palavra em situação dicionária:
isolada, estanque no poço dela mesma,
e porque assim estanque, estancada;
mais: porque assim estancada, muda,
e **muda porque com nenhuma comunica**,
porque cortou-se a sintaxe desse rio,
o fio de água por que ele discorria.

O curso de um rio, seu discurso-rio,
chega raramente a se reatar de vez;
um rio precisa de muito fio de água
para refazer o fio antigo que o fez.
Salvo a grandiloquência de uma cheia lhe
impondo interina outra linguagem,
**um rio precisa de muita água em fios
para que todos os poços se enfrasem**:
se reatando, de um para outro poço,
em frases curtas, então frase a frase,
até a sentença-rio do discurso único
em que se tem voz a seca ele combate”

(João Cabral de Melo Neto)

RESUMO

O gerenciamento de diálogos em robôs sociáveis exige uma série de desenvolvimentos menores que, integrados, fazem parecer que o discurso de um robô é, de fato, humano.

Um desses desenvolvimentos, que está em foco neste trabalho, é transformar textos escritos em estruturas de grafos semânticos. Para tanto,utiliza-se a árvore de análise sintática obtida do projeto CoGrOO. A partir dela, o objetivo é construir uma estrutura de grafos semânticos com formato UNL.

Uma vez construídos os grafos semânticos, estes serão usados para a interpretação do discurso humano (ou serão a interpretação dele em si). Isso possibilitará uma resposta menos “mecânica” do computador ao estímulo humano, permitindo a criação de um robô sociável.

Palavras-chave: Robô Sociável, UNL, Árvores Sintáticas, Grafos Semânticos.

ABSTRACT

The management of dialogs in social robots demands a series of smaller developments that, integrated, make a robot speech sound in fact human.

One of these developments, which is focused in this dissertation, is the transformation of written texts into semantic graphs. To do so, the syntactic tree obtained from the project CoGrOO will be used. From this tree, it will be possible to build the output: a structure of semantic graphs in the UNL format.

Once these semantic graphs have been built, they can be used for the interpretation of the human speech (or they will rather be the interpretation itself). Then it will be possible for the computers to answer less mechanically to human stimulation.

Keywords: Social Robots, UNL, Syntactic Trees, Semantic Graphs.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Sigla/Abreviatura	Significado
CoGrOO	Corretor Gramatical para OpenOffice
POS Tagger	*Part-of-Speech Tagger*
TTS	*Text-to-Speech*
UNL	*Universal Networking Language*
UNLKB	*UNL Knowledge Base*
LS	*Language Server*
IAS	*Institute of Advanced Studies*
UNU	*United Nations University*
UW	*Universal Word*
KRR	*Knowledge and Representation Reasoning*
DRL	*Drools Rule System*

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

1.1.1 Robôs sociáveis

Robôs têm sido usados de diversas maneiras para auxiliar seres humanos, desde em tarefas repetitivas, como apertar parafusos em montadoras, até em mais insalubres, como explorar o fundo de oceanos ou planetas desconhecidos.

Com a evolução das técnicas de construção de robôs, vem a possibilidade de uso deles em situações menos óbvias do cotidiano. Com a inteligência artificial, é possível que robôs se adaptem ao ambiente, desenvolvendo atividades como estacionar carros ou encontrar rotas em mapas.

Na indústria do entretenimento, *videogames* são programados para receber comandos, e responder a eles de forma autônoma, e muitas vezes imprevisível. Nesse mesmo ramo, estão os robôs sociáveis.

Desde seu esboço no filme *Star Wars* com a dupla R2-D2 e C-3PO, robôs com características humanas têm sido almejados. Em relação à comunicação com humanos, ainda se busca um C-3PO, ou seja, um robô que compreenda e responda aos milhões de idiomas que possivelmente existem em todo o universo.

Um produto comercial difundido no mercado por ser acoplado aos *iPhones* a partir do modelo 4S da *Apple* é a Siri. Ela é capaz de desenvolver conversas em inglês com usuários, ou seja, ela compreende o que lhe é dito, e responde com voz.

Um robô com as características tais como a da Siri pode ser utilizado em diversas aplicações. Entre elas, ele poderia fazer o papel de companhia para pessoas solitárias, como idosos em asilos. Outra aplicação poderia ser a facilitação da

interface de pessoas cegas com máquinas, como caixas automáticos de bancos. Além disso, autistas, que interagem pouco com seres humanos, aparentemente respondem bem a estímulos de máquinas [15].

1.1.2 Gerenciadores de diálogos

Os robôs que se comunicam com humanos podem ser vistos de dois modos: aqueles em que a única característica importante é a comunicação, e os que devem ter aparência humana.

Para os humanoides, existem trabalhos a serem realizados na parte mecânica, para que eles apresentem não somente a capacidade de falar como homens, mas também feições humanas (demonstradas em expressões faciais, por exemplo).

Em ambas as vertentes, no entanto, é necessário que o robô compreenda a fala humana, e se expresse com uma linguagem parecida com, ou igual a, a do seu interlocutor.

Os gerenciadores de diálogos permitem que a conversa entre humano e robô flua. Eles têm algumas funções indispensáveis como transformar a fala de áudio para texto, interpretar o texto obtido, e então criar uma resposta plausível. Por fim, a resposta textual deve ser transformada em som para que o interlocutor a entenda.

1.2 Objetivo

Este trabalho está inserido em um projeto que visa a criar um robô que se comunique com seres humanos.

O objetivo desta parte do projeto é desenvolver um programa que receba como entrada um texto em lingua portuguesa, e depois de seu processamento, dê como saída relações semânticas entre pares de conceitos presentes nesse texto.

Mais especificamente, a partir do texto de entrada deverá ser criada uma árvore de interpretação sintática de sentenças que será usada para análise do texto, e para posterior formação de uma estrutura com grafos de análise semântica.

Essa estrutura de grafos semânticos conterá relações entre as palavras das sentenças. Por fim, essas relações serão utilizadas em outros trabalhos para que o robô crie respostas, e então mantenha um diálogo com seu interlocutor.

1.3 Estrutura

Este trabalho está segmentado em quatro partes. A primeira delas é um estudo das ferramentas que poderão ser utilizadas para interpretação automática de textos, como a formação de árvores sintáticas e a padronização de relações predicativas.

A segunda parte contém a descrição de como essas ferramentas foram combinadas para, a partir de um texto em língua portuguesa, e de sua análise sintática, chegar-se à sua interpretação semântica.

Em seguida é feita a apresentação de resultados. Alguns exemplos de saídas são expostos, e depois um sistema que possibilita a geração simplificada de novas regras de interpretação é descrito.

Na última parte, são feitas as considerações finais levando em consideração as dificuldades e soluções encontradas ao longo da execução do projeto, bem como a possibilidade de futuros trabalhos.

2 REVISÃO DA LITERATURA

2.1 Proposta de estruturação de um gerenciador de diálogos

Em [3] é apresentada uma arquitetura para o gerenciador de diálogos. Um diagrama que facilita o entendimento dela está na Figura 2.1.

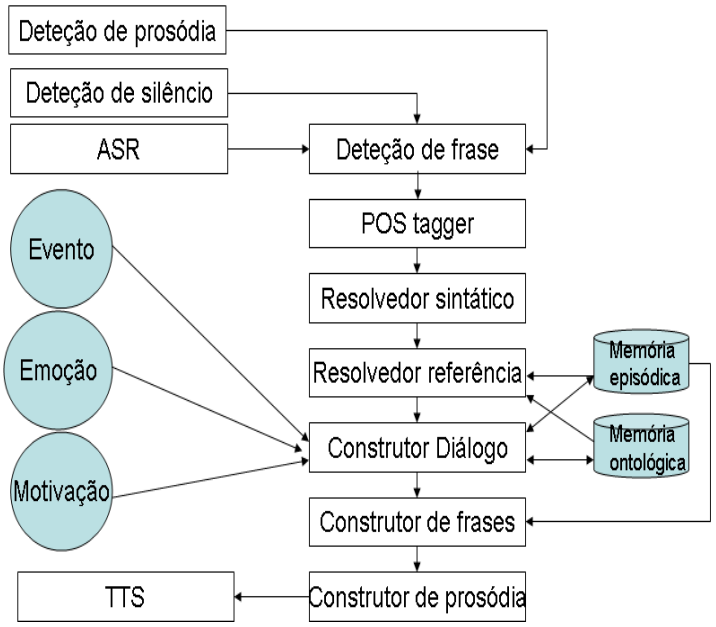

Figura 2.1 – Arquitetura de um gerenciador de diálogos, de [3]

Nessa arquitetura existem basicamente 11 elementos, divididos em: *Detecção de fala humana*, *Interpretação de texto*, *Criação de linha de diálogo* e *Fala do robô*. Em destaque, os itens 2b e 2c estarão em foco neste trabalho.

1. Detecção de fala humana:
   a) *Automatic Speech Recognition* – é responsável por detectar a fala humana e transformá-la num texto;
   b) Detecção de silêncio – dá ao robô a oportunidade de "puxar um assunto" quando o humano interlocutor fica em silêncio;
   c) Detecção de prosódia – identifica, de acordo com a entonação do sujeito, se a frase em questão é uma afirmação, interrogação ou exclamação;

2. Interpretação de texto:
    a) Detecção de frase – pega um texto cru (já escrito), e normaliza suas frases por meio do uso de pontuação;
    b) ***Part-of-Speech (POS) tagger*** – detecta a classe e a função gramatical de cada palavra, determinando sua flexão e sua forma primitiva;
    c) **Resolvedor sintático** – analisa sintaticamente as frases;

3. Criação de linha de diálogo:
    a) Resolvedor de referência – busca na memória referências ao assunto em pauta para mostrar familiaridade com o interlocutor;
    b) Gerenciador de diálogo – determina respostas para perguntas, busca os melhores meios para continuar as conversas (ou para sair do silêncio) e passa informações necessárias ao Construtor de frases;
    c) Construtor de frases – transforma as informações do Gerenciador de diálogo em "*human-like sentences*" (ou frases com "cara humana");

4. Fala do Robô:
    a) Construtor de prosódia – indica ao TTS (explicado a seguir) a entonação que deve ser dada à próxima frase;
    b) *Text-to-Speech* (TTS) – transforma um texto escrito em texto falado.

Com essa arquitetura, o objetivo deste trabalho pode ser refinado: deseja-se criar um programa que, dado um texto de entrada, etiquete-o morfológica e sintaticamente, e em seguida estabeleça relações semânticas entre seus conceitos.

A seção 2.2 mostra um estudo sobre etiquetadores para decidir qual tipo deverá ser utilizado neste trabalho; a seção 2.3 contém uma introdução à ferramenta escolhida para estabelecer as relações semânticas entre os conceitos do texto: a UNL.

2.2 Etiquetadores

2.2.1 Visão geral

Os etiquetadores de textos têm duas grandes funções: classificar gramaticalmente (morfológica e sintaticamente) as palavras de um texto, e posteriormente desambiguar a análise de palavras às quais foram atribuídas duas ou mais etiquetas possíveis.

Dada uma palavra e suas possíveis classificações (por exemplo, o verbo “comer”, que pode ser transitivo direto, como em “Comi arroz ontem”; ou intransitivo, como em “Comi hoje cedo”), o etiquetador tem que ser capaz de decidir qual das etiquetas aplicar.

A princípio, o etiquetador elimina possíveis etiquetas de uma palavra por meio de regras linguísticas, mas algumas vezes duas ou mais etiquetas são atribuídas a esta palavra, então faz-se necessário um desambiguador.

Em relação a *POS Taggers*, pode-se dizer que existem dois grandes tipos de etiquetadores sintáticos: aqueles baseados em restrições, e os probabilísticos – [1].

a) Etiquetadores baseados em restrições

Partem de um conjunto de regras elaboradas por um linguista, e atribuem etiquetas às palavras de acordo com seus contextos.

Apesar de as regras parecem unívocas a priori, no final da etiquetagem ainda existem muitos conflitos de ambiguidade para serem resolvidos.

Para resolver tais ambiguidades, existem regras de segunda ordem, denominadas em [1] como “Heurísticas *ad-hoc*”. Estas regras são menos exatas, mas ainda bastante determinísticas. Por exemplo, quando “o” aparece no começo de uma frase

e ainda há ambiguidade, o etiquetador deve escolher artigo, em detrimento de pronome oblíquo átono.

Há ainda um terceiro nível de regras, estas completamente não contextuais, para tratar todas as ambiguidades que ainda não tiverem sido removidas até este ponto. São regras como: “se houver ambiguidade entre substantivo e verbo, substantivo deverá ser escolhido” independentemente do contexto.

b) Etiquetadores probabilísticos

Os etiquetadores probabilísticos dispensam a imposição de regras humanas; eles apenas exigem um *corpus*[1] manualmente etiquetado.

Esse *corpus* será usado para treinamento[2] do etiquetador, de modo que, dado um *corpus* devidamente etiquetado, é possível etiquetar outros textos que um usuário deseje interpretar.

Assim, os etiquetadores probabilísticos transcendem as línguas, e podem ser aplicados em qualquer lugar, desde que haja *corpora* de treinamento devidamente etiquetados na língua desejada.

Dentre os etiquetadores probabilísticos, existem três tipos mais convencionais apresentados, mas não criados, em [1]. São os seguintes:

1) *TreeTagger*;
2) Sistema Baseado em Transformação (*Transformation-Based Learning* – TBL);
3) *MXPOST* (Modelo da Máxima Entropia).

[1] *Corpus* é um texto de determinado gênero (jornalístico, didático, literário, etc.) manualmente etiquetado que serve para treinamento e teste de etiquetadores automáticos de textos.
[2] O treinamento de um *corpus* é a forma como cada etiquetador aprende automaticamente a marcar os textos a partir de um texto já marcado.

Com base em dados coletados em [1] e [2], foi possível construir a **Error! Reference source not found.**, de comparação entre as precisões encontradas em testes para o português brasileiro dos três etiquetadores citados.

Tabela 2-1 – Precisão de etiquetadores

Etiquetadores	Precisão
TreeTagger	87,12%
TBL	86,42%
MXPOST	88,73%

A precisão obtida com o uso do modelo da máxima entropia, o *MXPOST*, é a maior das três. Assim, haverá preferência pela utilização de uma árvore sintática gerada a partir desse tipo de etiquetador.

2.2.2 CoGrOO: Corretor Gramatical para Open Office – [5]

É um projeto em andamento no IME-USP que contém a geração de árvores sintáticas incorporada ao seu escopo. Embora não haja dados publicados sobre a precisão de seu etiquetador, ele utiliza o *MXPOST* para análise sintática de sentenças.

Sua saída final, o corretor gramatical, não será utilizada, mas existe uma saída intermediária que é interessante aos objetivos do presente projeto: a árvore de análise sintática.

O CoGrOO é uma ferramenta que deve ser usada apenas como corretor gramatical [5]. Os erros encontrados na escrita são classificados em [6] em quatro categorias: erros ortográficos, gramaticais, de estilo e semânticos.

Mais especificamente, o CoGrOO se propõe a detectar seis tipos de problemas gramaticais:

1) Concordância verbal:
   a. “Ela são feia.”;
2) Concordância nominal:
   a. “Os homem é folgado.”;
3) Regência verbal:
   a. “O homem precisa de casar logo.”;
4) Regência nominal:
   a. “Este exemplo é análogo com o outro.”;
5) Crase:
   a. “Fui à Santos ontem”;
6) Erros comuns da língua falada:
   a. “Moro com meus pais a 24 anos”;

Para tanto, ele conta com a arquitetura apresentada na Figura 2.2. Nessa arquitetura ficam bem claras a entrada e a saída do programa, bem como seus passos intermediários. Estes passos estão explicados a seguir.

1) Entrada

O programa deve receber como entrada um texto digitado por um usuário. No caso do projeto no qual o presente trabalho se insere, em [3] é proposta uma série de passos para, a partir do texto falado, chegar-se a um texto escrito para a entrada do CoGrOO. Tal proposta está discutida na seção 2.1.

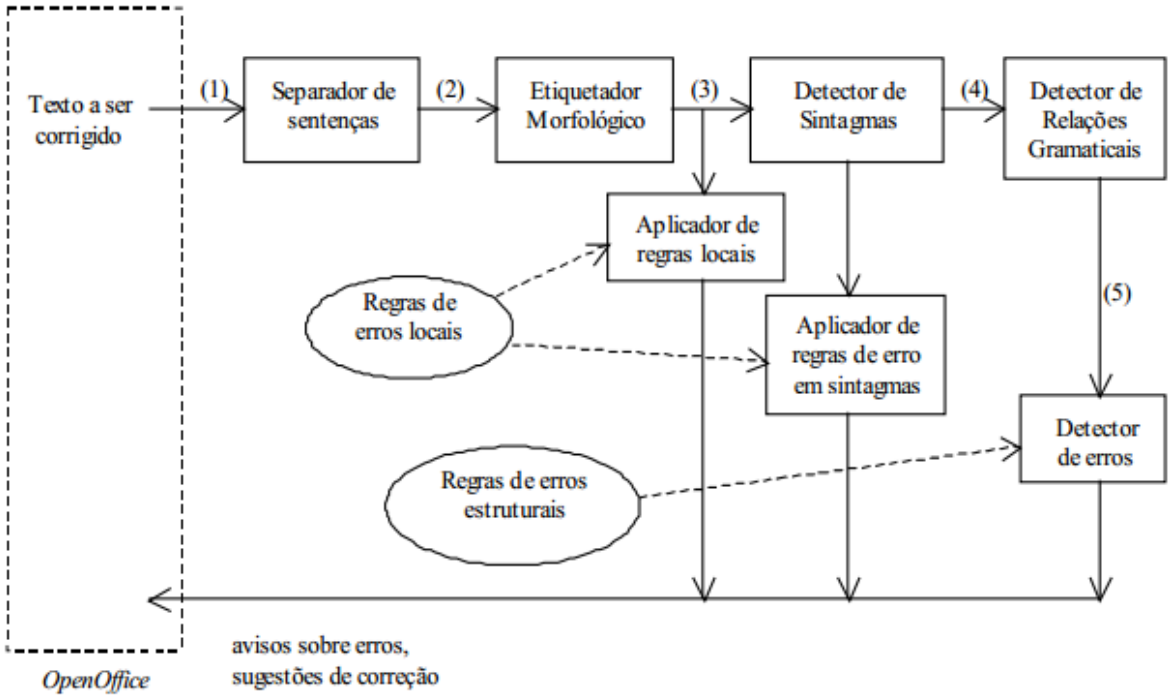

Figura 2.2 – Arquitetura do Projeto CoGrOO [5]

Aqui será apresentado um exemplo que será utilizado em alguns dos módulos do corretor gramatical apresentados a seguir. Seja a entrada digitada pelo usuário a seguinte: “O Sr. Marcos é meu professor. Ele é uma pessoa boa.”

2) Separador de sentenças

Este módulo recebe o texto cru digitado pelo usuário, e identifica sentenças analisando sinais especiais, como ‘.’, ’?’ ou ’!’, por exemplo.

É importante que ele reconheça, por exemplo, quando um ponto separa orações, ou quando este mesmo ponto simplesmente separa letras em uma sigla, ou ainda quando ele corta uma palavra abreviada.

No exemplo, o separador devolveria duas sentenças, pois seria capaz de distinguir o ponto depois de “Sr” do ponto depois de “professor”:

a) “O Sr. Marcos é meu professor.”;
b) “Ele é uma pessoa boa.”;

3) Etiquetador morfológico

Recebe o texto preparado pelo separador de sentenças, e em seguida aplica o procedimento do *MXPOST*, da seguinte forma:

A) Associa etiquetas morfológicas possíveis às palavras da sentença:
   a. Para isso ele consulta um léxico de possíveis etiquetas;
B) Avalia a probabilidade de cada etiqueta:
   a. De acordo com um *corpus* já etiquetado, de modo que tudo seja feito automaticamente;
   b. Cria um banco de dados com os trios de palavras consecutivas e suas classificações;
   c. Para cada palavra do texto a ser analisado, ele verifica as palavras em volta, e busca trios semelhantes no banco de dados;
   d. De acordo com as probabilidades dos trios, ele define uma probabilidade para cada palavra solta;
C) Avalia a etiqueta mais provável para cada palavra:
   a. Se houver empate de probabilidades, o etiquetador usa regras *ad-hoc* para decidir a classe gramatical de cada palavra:
      i. Regras como: “Se a palavra *uma* está no início da frase, ela deve ser considerada artigo”;
      ii. Em último caso, são adotadas regras totalmente determinísticas como: “Em ainda havendo empate de probabilidades, a palavra deve ser classificada como substantivo”.

No exemplo, analisemos a palavra “uma”. Depois do primeiro passo, ela poderia ser classificada como o número 1, ou como um artigo indefinido feminino.

No passo dois ele atribui probabilidades com base em um *corpus* anotado[3]. Diga-se que essas probabilidades sejam 20% para o número, e 80% para o artigo. Como não há empate, o terceiro passo é descartado.

[3] *Corpus* anotado é um texto com classificação morfológica feita manualmente.

4) Aplicador de regras de erros locais

As regras são denominadas locais porque envolvem um contexto muito curto, sem levar em conta a estrutura das sentenças como um todo.

O aplicador aponta erros comparando em tempo real a sequência de etiquetas morfológicas das palavras com padrões armazenados nas regras locais.

Por exemplo, caso a sequência apresente uma ordem incomum, o sistema sugere a ordem que seria mais comum, ou correta, para a frase.

Outros erros apontados são uso de crase antes de palavra masculina ou de verbo, o provável uso flexionado de alguns advérbios (“Estou meia cansada.”, por exemplo), etc.

Uma entrada no arquivo de regras locais possui quatro componentes:

a) Regra de erro;
b) Mensagem explicativa sobre o diagnóstico do erro;
c) Um exemplo de uso adequado do padrão gramatical;
d) Um exemplo de uso inadequado do padrão gramatical.

A **Error! Reference source not found.** mostra um exemplo de regra local para o uso do advérbio “meio”.

Tabela 2-2 – Sistematização da definição das regras de erros locais [5]

"(meia|meias)" _ADJ	
Mensagem	A palavra “meio” usada no sentido de “um pouco” é advérbio, portanto invariável.
Exceção	A palavra “meia” é usada como substantivo e, portanto, concorda com o adjetivo.
Correto	A conclusão está meio confusa/ As conclusões estão meio confusas.
Incorreto	A conclusão está meia confusa/ As conclusões estão meias confusas.

5) Detector de sintagmas

Responsável por localizar sintagmas[4] dos tipos nominal e verbal. A implementação do detector de sintagmas no CoGrOO é feita com base em uma máquina de estados que varre a sentença etiquetada morfologicamente na busca de dois tipos de sequência.

O primeiro tipo é de sequências compostas por artigos, substantivos e adjetivos – para sintagmas nominais; para sintagmas verbais, buscam-se sequências de verbos e advérbios.

No exemplo que vinha sendo usado, teríamos uma etiquetagem sintática tal como é mostrada na **Error! Reference source not found.**, na qual “NP” significa *Noun Phrase* (Sintagma Nominal), “VP” significa *Verbal Phrase* (Sintagma Verbal).

“3S” indica que os termos designam terceira pessoa do singular; o número zero é a numeração dada aos sintagmas dentro da sentença. Caso algum sintagma se repetisse, o número do segundo seria 1, o do terceiro, 2, e assim por diante.

Tabela 2-3 – Etiquetagem sintática de uma sentença

Palavras	Etiquetas sintáticas
Ele	NP_3S_M_0_
É	VP_3S_0_
Uma pessoa boa	NP_3S_F_0_
.	-PNT_NS

6) Aplicador de regras de erro em sintagmas

No caso de sintagmas nominais, são aplicadas regras de erro locais, como problemas de concordância entre artigo e substantivo, ou entre substantivo e adjetivo.

[4] Sintagma é uma unidade mínima de significado de um texto; o sintagma é nominal quando seu núcleo é um nome (substantivo), e é verbal quando seu núcleo é um verbo.

Na **Error! Reference source not found.** é possível verificar um eventual erro de concordância, bem como a saída do aplicador de regras de erro em sintagmas.

Tabela 2-4 – Exemplo de saída do aplicador de regras de erro em sintagmas [5]

N_F_S_ [5] ADJ_F_P_ [6]	Faces rosada da criança.
Mensagem	O adjetivo concorda em gênero (masculino ou feminino) e número (singular ou plural) com o substantivo a que se refere.
Correto	Faces rosadas da criança ou a face rosada da criança
Incorreto	Face rosadas da criança.

7) Detector de relações gramaticais

Neste módulo são levantadas relações gramaticais do tipo sujeito-verbo, verbo-objeto, verbo-preposição, etc. na frase a ser analisada.

Na implementação apresentada em [5], decidiu-se por marcar inícios de sentenças com '!'. Assim, pode-se verificar a concordância verbal entre NP e VP. Nos *corpora* estudados por eles, em 85% dos casos em que a sequência '! NP VP' apareceu, NP era sujeito de VP.

Devido a essa alta frequência, esse padrão é adotado para detecção de sujeito nas sentenças a serem analisadas pelo corretor gramatical.

8) Correção gramatical

Na fase de detecção de relações gramaticais, a correção gramatical é feita em duas frentes:

a) Por meio de verificação das informações contidas nas etiquetas sintáticas;
b) Com base em regras de erros estruturais.

[5] N_F_S_ é a sigla em inglês para Substantivo Feminino Singular.
[6] ADJ_F_P_ é a sigla em inglês para Adjetivo Feminino Plural.

Com a primeira, é possível, por exemplo, identificar problemas de concordância verbal. Com as informações contidas nas etiquetas sintáticas, pode-se verificar a correspondência entre a pessoa do sujeito e a do verbo a ele associado. Se as etiquetas forem diferentes, há aí um erro a ser apontado.

Por exemplo, na frase “Os corretores foi procurar uma casa”, “Os corretores” recebeu a etiqueta sintática [SUBJ>]_M_3P_ (Sujeito masculino, terceira pessoa do plural), que não concorda em número com “foi procurar”, cuja etiqueta é VP_3S_ (Sintagma verbal, terceira pessoa do singular) – exemplo presente em [5].

Na segunda frente, de correção gramatical com base em regras de erros estruturais, são detectados erros devidos ao mau uso da língua em situações específicas.
Regras estruturais de erros envolvem não somente palavras e etiquetas morfológicas, mas também informações sintáticas. No exemplo “Fazem três meses que não chove”, é necessário que o corretor verifique que “Fazem” não é verbo relacionado a sujeito, e então ele deve sugerir o uso de singular). Por outro lado, na frase “Hoje os gêmeos fazem aniversário”, o corretor não deve fazer correções.

9) Desvios gramaticais detectados pelo CoGrOO 2.0

Em [7] encontra-se a classificação de oito tipos de inadequações gramaticais da versão 2.0 do CoGrOO:

a) Colocação pronominal:
    i) Uso de próclise;
b) Concordância nominal:
    i) Gênero entre substantivos e adjetivos;
    ii) Número entre substantivos e adjetivos;
    iii) Gênero entre artigos e substantivos;
    iv) Número entre artigos e substantivos;
c) Concordância entre sujeito e verbo;
d) Concordância verbal:
    i) “Fazer” indicando tempo;

ii) “Haver” indicando existência;

e) Ocorrência de crase:
   i) Antes de substantivo masculino;
   ii) Antes de verbo;
   iii) Em expressões indicativas de horas;
   iv) Em expressões indicativas de modo, tempo, lugar, etc.;
   v) Antes de pronomes de tratamento;
   vi) Em expressões como “em relação à”, “com relação à” e “devido à”;
   vii) Em expressões do tipo “segunda a sexta”;
f) Regência nominal;
g) Regência verbal;
h) Erros comuns na língua portuguesa falada no Brasil:
   i) Confusão entre os usos de “meio” como adjetivo ou advérbio;
   ii) Uso de “mim” como pronome pessoal do caso reto;
   iii) Etc.

10) Detalhes da programação

Esta seção é importante porque a programação do presente projeto deverá acessar a programação do CoGrOO.

O projeto em Java do CoGrOO apresenta divisão de camadas da seguinte forma:

1) **Camada de interface com o usuário**: nela estão contidas as classes que permitem a ligação entre o OpenOffice.org e a camada de negócios;
2) **Camada de negócios**: nela se localizam os módulos de análise e verificação de textos (é o núcleo do programa);
3) **Camada de dados:** são os modelos e os dicionários.

A Figura 2.3 – Arquitetura simplificada do CoGroo [7] apresenta de forma bastante simplificada a arquitetura apresentada na Figura 2.2 – Arquitetura do Projeto CoGrOO [5].

A estrutura de dados interna do programa é definida pelas classes criadas em [7], e está exposta a seguir.

1. Corpus: uma classe que serve de interface entre o programa e o arquivo contendo sentenças para o levantamento das estatísticas.
    a. Atributos:
        i. Contador de sentenças: o número de sentenças contidas no corpus;
        ii. Localização das sentenças: um vetor com o mesmo número de posições quanto o número de sentenças existentes no arquivo; o valor de cada elemento representa a posição de início de cada sentença dentro desse corpus;
        iii. Número da sentença atual: um contador interno das sentenças, cujo valor é atualizado conforme as sentenças são lidas;
        iv. Arquivo do corpus: uma referência ao arquivo "físico" do corpus; permite acesso aleatório a qualquer parte do corpus independentemente do momento do acesso;
2. Sentença:
    a. Atributos:
        i. Sentença: a sentença original inserida pelo usuário;
        ii. Tokens: a sentença dividida em um vetor de tokens;
        iii. Sintagmas: a sentença dividida em um vetor de sintagmas;
3. Token:
    a. Atributos:
        i. Lexema: o próprio token escrito pelo usuário (palavra, pontuação, etc.);
        ii. Primitiva: no caso das palavras, é aquela da qual se originou o lexema, que não apresenta flexões (de gênero, número, etc.);
        iii. Etiqueta de agrupamento: provê informações se o token é início ou continuação de um sintagma, ou nenhum dos dois;
        iv. Etiqueta sintática: indica se o token é núcleo de um sujeito, de uma locução verbal, ou nenhum dos dois;
        v. Sintagma: indica a qual sintagma pertence o token;

vi. Localização: indica início e fim de um token em uma sentença;

4. Sintagma:
    a. Atributos:
        i. Tokens: um vetor apontando para os tokens que formam o sintagma;
        ii. Primeiro token: informação sobre o índice do primeiro token deste sintagma na sentença;
        iii. Etiqueta morfológica: indica o tipo de sintagma, e suas flexões (gênero, número, etc.);
        iv. Etiqueta sintática: indica se representa um sujeito ou um verbo;
5. Etiqueta:
    a. Atributos:
        i. Etiqueta: a etiqueta em formato de texto

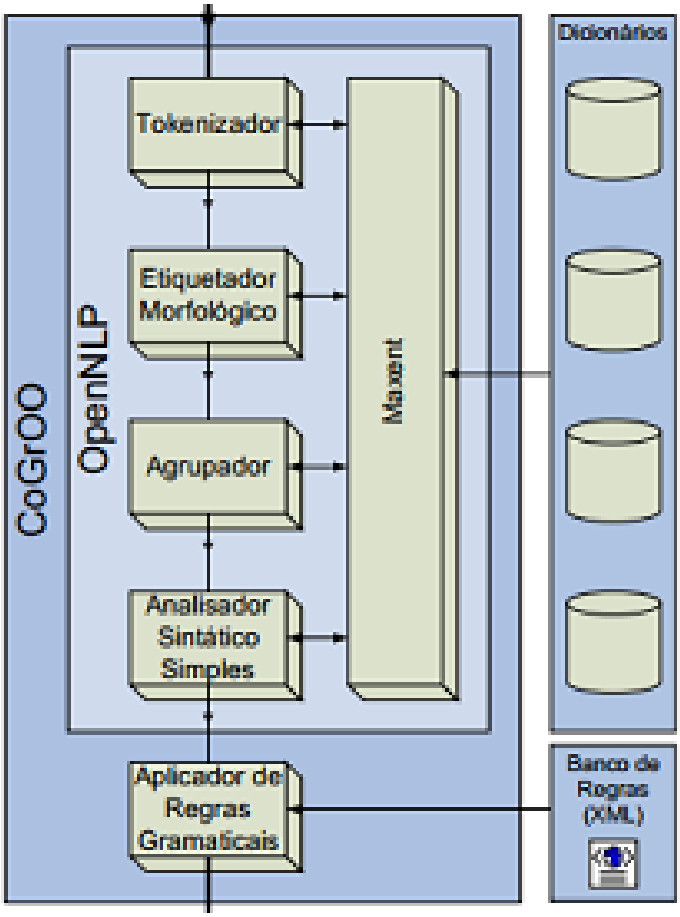

Figura 2.3 – Arquitetura simplificada do CoGroo [7]

11) CoGrOO API

A saída principal do CoGrOO é uma interface indicando ao usuário possíveis inadequações de seu texto em relação à norma culta da língua portuguesa, mas durante seu processamento, ele produz outras saídas intermediárias.

Em especial, o interesse aqui é na árvore sintática gerada
figuras para fins ilustrativos. Na

Figura 2.4 – Interface do CoGrOO com o usuário, é possível ver a interface gerada pelo CoGrOO no *OpenOffice* para uma entrada gramaticalmente incorreta de um usuário.

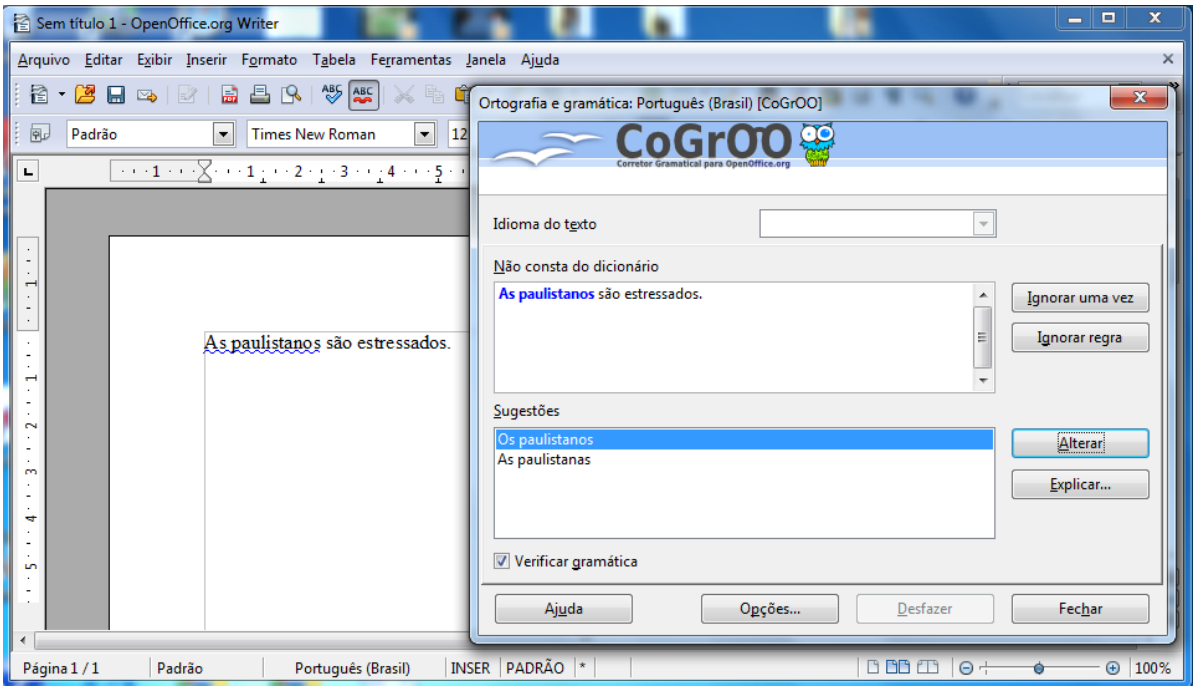

Figura 2.4 – Interface do CoGrOO com o usuário

Já na Figura 2.5, pode-se ver um exemplo de árvore sintática gerada para uma saída intermediária do projeto CoGrOO.

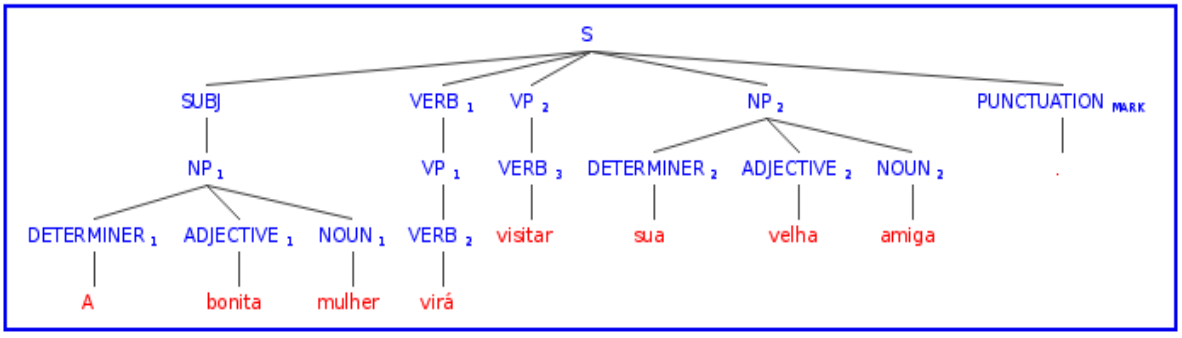

Figura 2.5 – Árvore sintática gerada pelo CoGrOO [4]

2.3 Introdução à UNL – *Universal Networking Language*

A UNL foi lançada em 1996 como um programa do Instituto de Estudos Avançados (IAS) da Universidade das Nações Unidas (UNU), em Tóquio. Sua difusão ganhou escala com o trabalho de [8], que lançou as primeiras diretrizes e padrões para UNL. De acordo com [11], existem patentes da ONU em nome de Hiroshi Ushida, em japonês, e Meiying Zhu, em chinês, estabelecendo os padrões da UNL. Como as patentes são da ONU, elas são de livre acesso e uso para todo ser humano.

Em janeiro de 2001, a UNU criou uma organização autônoma, a Fundação UNDL, para se responsabilizar pelo desenvolvimento e gerenciamento do Programa UNL.

2.3.1 Objetivos da UNL

A UNL tem dois grandes objetivos. Em primeiro lugar, ela é uma linguagem que possibilita que os computadores tratem a linguagem natural de modo eficiente, facilitando o processamento de informação e conhecimento.

Além disso, a UNL apresenta capacidade de transportar toda a significação da linguagem falada em algo palpável computacionalmente, então ela pode ser usada para distribuição, receptação e entendimento de informações vindas de diversas linguagens.

Essa capacidade representa uma grande ruptura com as linhas de pesquisa anteriores. Antes da UNL, o processamento semântico era tido como algo custoso e extremamente complexo, que ia além das capacidades humanas.

Como a UNL possibilita a interpretação semântica de textos, ela é interessante a este trabalho, pois quer-se aqui possibilitar o desenvolvimento de um robô que responda mais “humanamente” a estímulos de interlocutores.

2.3.2 Funcionamento da UNL

Neste item serão estudadas as ferramentas da UNL que têm relevância para este trabalho.

a) Expressões UNL

Uma expressão em UNL guarda apenas o conteúdo semântico de textos, então é comum que se percam informações sobre artigos, por exemplo.

Para guardar esse conteúdo, são criados grafos direcionados cujos nós são conceitos (denominados *Universal Words*, que são armazenadas sempre em inglês).

As arestas desse grafo representam as relações entre os conceitos. Além disso, cada conceito pode ser anotado por atributos.

Com esses elementos, o formato de uma expressão UNL deve ser parecido com o seguinte:

<relation>[:<scope-ID](<from-node>, <to-node>);

a. <relation> → etiqueta;
b. <scope-ID> → todas as relações binárias de um escopo devem receber um mesmo ID, e um <scope-ID> pode ser omitido;
c. <from,to-node> → uma UW pode ser seguida por um <node-ID>, e um scope node é apontado por um <scope-ID> precedido de dois pontos.

Adiante serão apresentados exemplos que deixarão o formato mais inteligível.

b) *Unviersal Words* (UW)

Elas constituem o vocabulário UNL, e são assim representadas:

<uw> =:: <headword>[constraint list]

A *headword* é necessariamente uma palavra, frase ou oração em inglês que define a UW.

Em [11] há um exemplo sobre a necessidade da definição da UW, uma vez que existem muitas palavras com diversos possíveis significados. O exemplo trata do termo *state*, em inglês, cuja lista de possíveis classificações é a seguinte:

1) state(icl>abstract thing);
2) state(icl>country);
3) state(icl>government);
4) state(icl>region);
5) state(icl>governmental(mod<thing));
6) state(icl>official(mod<thing));
7) state(icl>fix(agt>thing,obj>thing));
8) state(icl>say(agt>thing,obj>thing)).

Os significados das siglas serão abordados posteriormente.

c) Relações

São usadas para conectar as UW. Existem 46 tipos, tais como agente, objetivo, objeto, etc.

Em [14] é apresentada uma lista com explicação e exemplos de cada tipo de relação existente.

d) Atributos

São usados para descrever informações de subjetividade, tais como sentimento, julgamento, ênfase, etc. A **Error! Reference source not found.** sistematiza os tipos de atributos.

Tabela 2-5 – Tipos de atributos [11]

Tipo de descrição	Nome do atributo
Describing logicality	@transitive,@symmetric,@identifiable,@disjointed
Describing times	@future,@past,@present
Describing aspects	@begin,@complete,@continue,@end,@progress,@state
Describing generallity and specificity	@generic,@def,@indef,@not,@ordinal
Describing emphasis, focus and topic	@emphasis,@entry,@focus,@topic
Describing attitudes	@affirmative,@imperative,@interrogative,@request,…
Describing feelings and judgments	@ability,@grant,@wish,@will,@obligation,@possible,@regret
For convention	@passive,@pl,@parenthesis,…

e) Código UNL

Até aqui já foram apresentadas as ferramentas mínimas necessárias para o entendimento de uma sentença representada em UNL.

Utilizemos um dos exemplos apresentados em [11]. Seja a sentença "*I can hear a dog barking outside*". A expressão UNL gerada a partir dela está apresentada na Figura 2.6.

```
{unl}
  (1)  agt(hear(icl>perceive(agt>person,obj>thing)):06.@ability.@entry, I(icl>person):00.@topic)
  (2)  obj(hear(icl>perceive(agt>person,obj>thing)):06.@ability.@entry, :01)
  (3)  agt:01(bark(agt>dog):0H.@progress.@entry, dog(icl>mammal):0D.@indef)
  (4)  plc:01(bark(agt>dog):0H.@progress.@entry, outside(icl>area):0P)
{/unl}
```

Figura 2.6 – Exemplo de expressão UNL [11]

Na linha (1), pode-se notar a relação (*agt*) entre as UWs (*hear* e *I*). Também é possível ver a definição da UW *hear*, que neste caso está especificada como ato de percepção auditiva.

Além disso, notamos a sigla *icl*, que está inserida em um conceito mais amplo que será tratado no item f).

Na linha (2) é criada uma etiqueta (":01"), pois o objeto da sentença é composto. Assim, o objeto é explicado nas linhas (3) e (4), representado respectivamente por relações de *agt* de *plc*.

f) Sistema de UW

Existe uma hierarquia de UWs, e todas estão ligadas a um de três possíveis conectores, quais sejam:

a) icl → *subclass of*;
b) iof → *instance of*;
c) equ → *equivalent to*.

Assim, *hear* (ouvir) é subclasse de *perceive* (perceber), pois audição é um tipo de percepção.

Como as UW mais baixas herdam as características de suas superclasses, pode-se dizer que "ouvir" possui todas as características de "perceber", mas a recíproca não é verdadeira. Uma UW imediatamente superior a outra deve ser a palavra mais próxima em significado da primeira, com apenas um grau de generalidade a mais.

g) Ontologia UNL

i. Conhecimento Linguísitico em Conceitos

Uma UW é uma etiqueta para um conceito. Este conceito é definido pela descrição do conjunto de possíveis relações que uma UW pode ter com outras.

A ontologia é usada em algumas situações. Por exemplo, quando há ambiguidade na análise de sentenças é possível verificar no contexto qual seria a melhor escolha de palavra para aquela situação.

Outro exemplo de uso da ontologia é na geração de sentenças. Quando um conceito desconhecido é encontrado, a UNL pode criar a sentença utilizando o conceito imediatamente superior que, apesar de mais geral, não deixa a frase sem sentido.

ii. Conhecimento Semântico em Conceitos

As definições de conceitos são dadas em expressões com formato UNL. Por exemplo, o título de um livro é guardado com um link para ‘book(icl>document)’, de modo que o computador processe a informação do título sabendo a categoria na qual esse nome se enquadra.

2.3.3 Sistema UNL [10]

Consiste, basicamente, de três partes:

1) Recursos linguísticos;
2) Software para processamento desses recursos;
3) Ferramentas de suporte para manter software e recursos.

O recursos linguísticos podem ser divididos em duas partes: a independente de linguagem, que armazena conhecimento sobre conceitos e relações entre conceitos universais a todas as línguas (o armazenamento é feito na UNLKB – UNL *Knowledge Base*); e a dependente da linguagem, que armazena dicionários de palavras e conjuntos de regras em cada LS (*Language Server*).

A Figura 2.7 representa o mecanismo de conversão entre textos UNL e linguagens naturais.

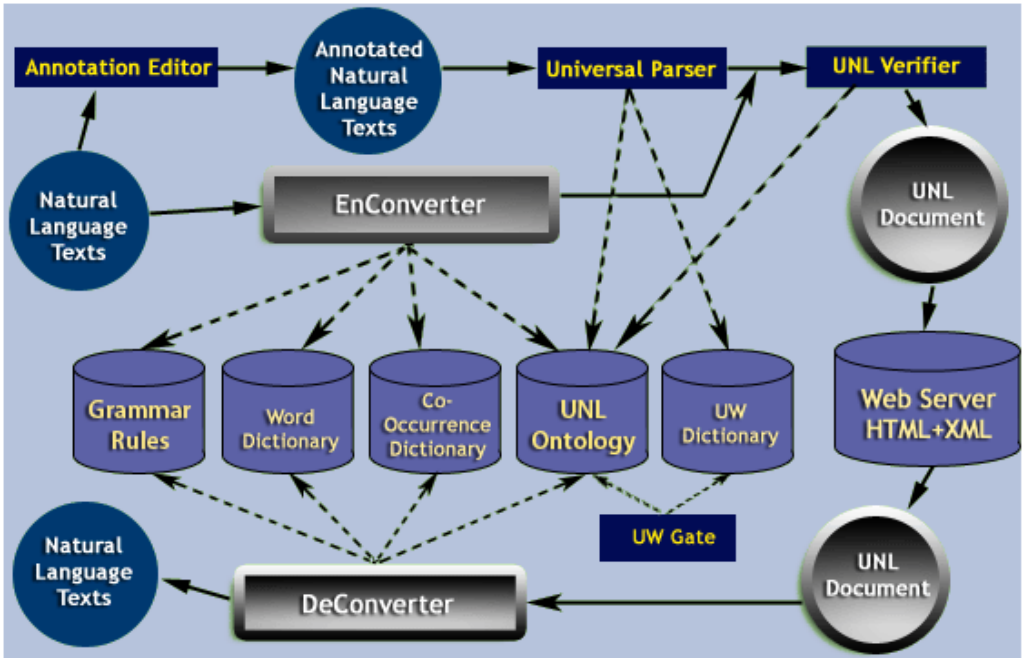

Figura 2.7 – Mecanismo de conversão entre UNL e linguagens naturais [10]

1. *EnConverter*

O *EnConverter* é responsável por transformar frases de línguas naturais em UNL. Para tanto, ele trabalha baseado em um dicionário de palavras e um conjunto de regras de "enconversão". Assim, ele pode funcionar para diversas línguas, apenas alterando seu dicionário de palavras e o conjunto de regras. Ele funciona da seguinte forma:

1) Escaneia a *string* de entrada da esquerda para a direita;
2) Cada morfema encontrado enquanto o escaneamento é feito é comparado com entradas do dicionário, em busca de morfemas candidatos;
3) Os morfemas escolhidos como candidatos são ordenados de acordo com a prioridade que lhes é atribuída;
4) A seleção de palavras é feita pela aplicação das regras gramaticais sobre esses morfemas candidatos;

5) As análises sintática e semântica são conduzidas pela aplicação das regras às palavras já selecionadas para construir uma árvore sintática e uma rede semântica para a sentença de entrada;
6) O *EnConverter* ainda é responsável por verificar a Ontologia UNL.

2. *DeConverter*

Faz o serviço oposto ao do *EnConverter*. Dada uma sentença na estrutura UNL, ele é responsável por retransformá-la em linguagem natural. Seu funcionamento é análogo:

1) Transforma a entrada de uma expressão UNL em uma estrutura de grafos direcionados:
a. O nó-raiz é chamado nó de entrada, e representa a cabeça (verbo principal) de uma sentença;
2) As regras de "deconversão" são aplicadas a cada nó do grafo:
a. Começa a partir do nó de entrada, e vai tentando encontrar palavras apropriadas para cada nó da rede, gerando uma sequência de palavras;
3) A deconversão termina quando todos os nós têm uma palavra atribuída a si, e a sequência-alvo termina de ser montada.

3. Word Dictionary

Deve existir um dicionário de palavras para cada língua. Toda informação sobre palavras de uma língua natural deve ser guardada no dicionário de palavras.

Uma entrada do dicionário de palavras em geral possui três partes: uma *headword*, uma UW, que expressa o significado da *headword*, e um conjunto de atributos gramaticais, que definem como uma palavra se comporta em uma sentença.

O modelo de uma entrada desse dicionário é:

[HW]{ID} "UW" (ATTR,ATTR,…) <FLG,FRE,PRI>;

A **Error! Reference source not found.**explica cada sigla utilizada.

Tabela 2-6 – Significado das entradas do dicionário de palavras [10]

Sigla	Significado
HW	***headword* para determinada língua**
ID	***identifier* (pode estar vazio)**
UW	***Universal Word* (pode estar vazio)**
ATTR	***grammar code***
FLG	***language flag*, um caractere em ASCII**
FRE	***frequency* para ser usada pelo *EnCoverter***
PRI	***priority* para ser usada pelo *DeConverter***

Com base nesse modelo, são apresentados aqui alguns exemplos de entradas reais do dicionário de palavras de língua inglesa retirados de [10].

1) [a]{} "" (ART,IART) <E,1,1>;
2) [book]{} "book(icl>document)" (BA,C,N,PLS) <E,1,1>;
3) [buy]{}"buy(icl>purchase(agt>thing,obj>thing))"(3SGS,AGT.S,BA,INGING,IRG, OBJ.DO,V,VDO,VDON) <E,1,1>;
4) [bought]{}"buy(icl>purchase(agt>thing,obj>thing))" (AGT.S,ED,EN,IRG,OBJ.DO, V,VDO,VDON) <E,1,1>;
5) [I]{} "I(icl>person)" (1SG,HPRON,PRON,SUBJ) <E,1,1>;
6) [me]{} "I(icl>person)" (1SG,HPRON,OBJ,PRON) <E,1,1>.

4. *UNL Verifier*

Existe um controle para verificação de expressões UNL (para determinar se dada expressão é ou não é válida). Esse controle é realizado pelo Verificador UNL. Ele verifica se uma expressão é válida em três aspectos:

1) Sintaticamente, seguindo especificações da UNL;
2) Lexicamente, averiguando se todas as palavras da expressão se encontram posicionadas na Ontologia UNL;
3) Semanticamente, verificando se todas as relações binárias entre as UW da expressão são possíveis de acordo com a Ontologia UNL.

5. Enciclopédia UNL

Possui uma estrutura em níveis. Em todos esses níveis, Enciclopédia UNL apontam para documentos UNL com significados. Além disso, existe um grafo de hierarquia de UWs. A

Figura 2.8 ilustra o conceito apresentado neste parágrafo.

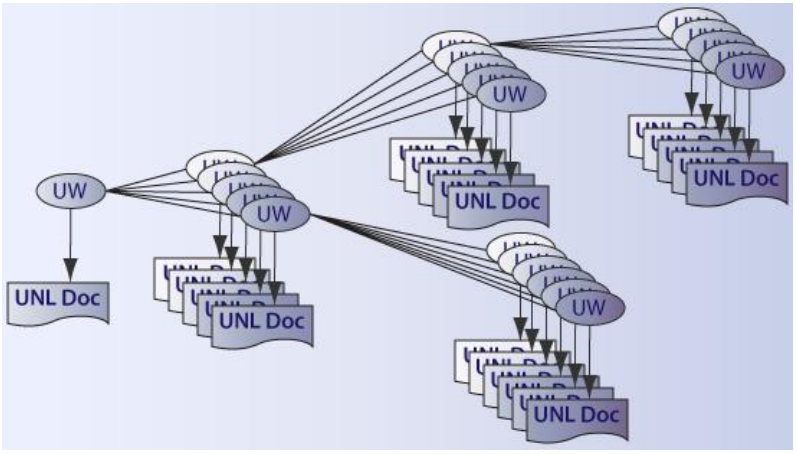

Figura 2.8 – Ilustração do conceito de enciclopédia em grafo [9]

Os níveis diferentes se apresentam no conteúdo armazenado. Nos níveis mais altos é guardado conhecimento geral sobre conceitos (descrições sobre conceitos se referem ao que eles são, como fazê-los ou usá-los, quando eles são verdadeiros, etc.).

Nos níveis mais baixos ficam descrições mais pesadas. Em vez de significados de conceitos, podem ser guardados conteúdos de arquivos e livros, que são considerados conceitos compostos.

3 MÉTODOS E MEIOS

3.1 Desenvolvimento

a. Ferramentas

As ferramentas necessárias para o desenvolvimento deste projeto já foram apresentadas, e esta seção trata do modo como elas serão articuladas para que os objetivos propostos sejam atingidos.

Para se chegar a uma estrutura de relações semânticas entre palavras de frases com formato UNL, será utilizada a árvore sintática obtida do projeto CoGrOO, e em seguida uma série de regras desenvolvidas em Java.

A utilização dessas duas ferramentas possui um custo inicial de configuração. Para o uso do projeto CoGrOO, é necessário o *download* e a instalação de todos os arquivos usados internamente pela API para montar a árvore de análise sintática.

A Figura 3.1 apresenta um diagrama UML de classes adaptado. Dele foram omitidos os objetos para que os métodos presentes nas classes ficassem em foco.

b. Estrutura de Classes

A partir desse diagrama, é possível compreender o funcionamento do sistema. Ele possui apenas seis classes e um arquivo *xml*.

O código completo de cada classe utilizada, e também do dicionário construído estão expostos nos apêndices de A até G.

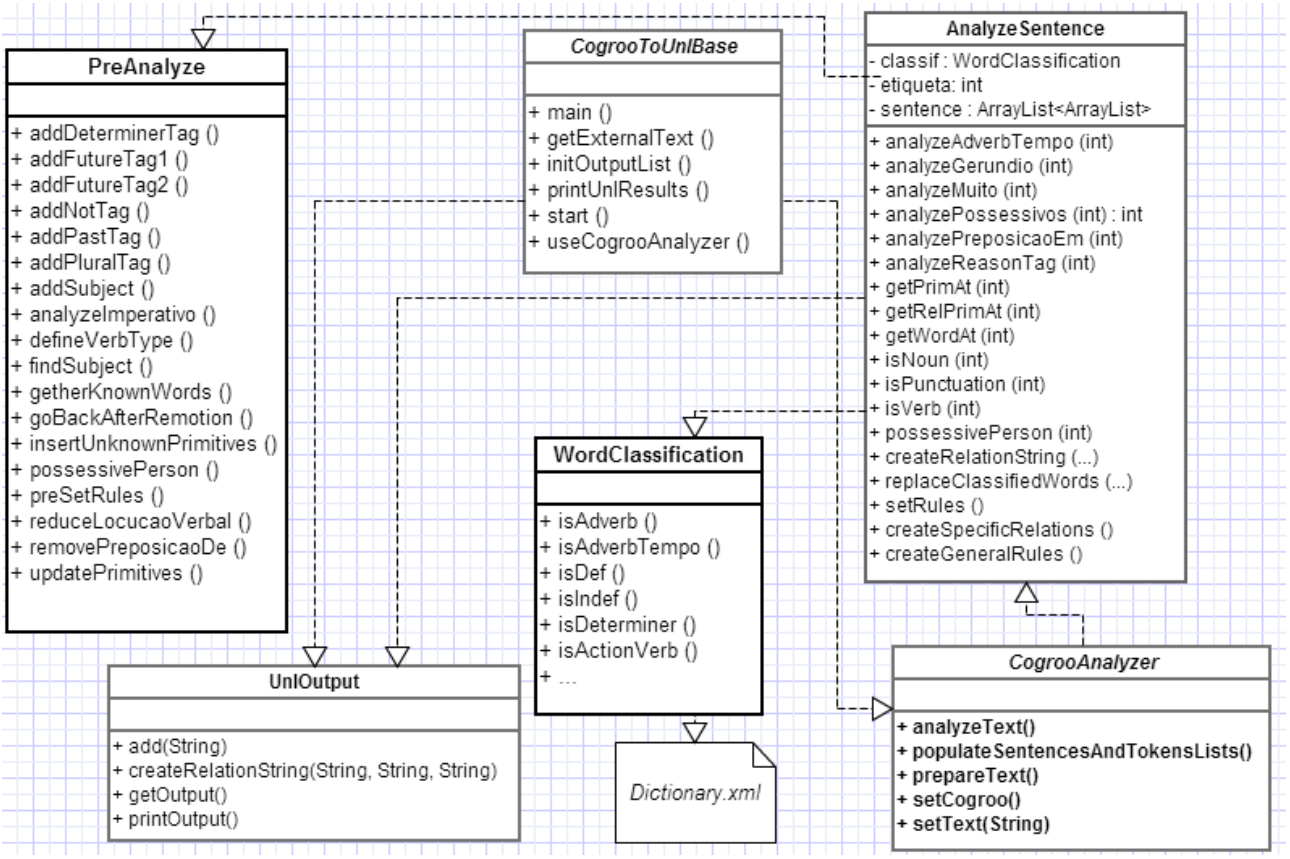

Figura 3.1 – Diagrama de Classes

1. *CogrooToUnlBase*

Inicia a execução do programa. O método principal desta classe, *main()*, chama o método *start()* para que possam ser criados métodos não estáticos no desenvolvimento do programa.

Essa classe é responsável ainda por acessar o arquivo que contém o texto a ser analisado, e enviá-lo para análise na classe *CogrooAnalyzer*.

2. *CogrooAnalyzer*

Faz a análise do texto utilizando a ferramenta CoGrOO, e envia o texto já analisado, cada sentença de uma vez, para que a classe *AnalyzeSentence* elabore os métodos de acesso a cada palavra.

Com cada palavra analisada, obtêm-se os vetores de análise gramatical, como os exibidos na Figura 3.2, que mostra a análise de cada palavra da frase “Ana come doces.”

```
[Ana, ana, proper noun,female,singular, SUBJECT, 0..3]
[come, comer, verb,singular,third,present,indicative,finite, VERB, 4..8]
[doces, doce, adjective,neutral,plural, NONE, 9..14]
[., ., punctuation mark,abs, NONE, 14..15]
```

Figura 3.2 – Análise do CoGrOO para a sentença “Ana come doces.”

Existem cinco campos compostos por *strings* em cada vetor, e eles estão descritos a seguir. O vetor relativo a “comer” será usado como exemplo.

1) A palavra em si – ‘come’;
2) Primitiva da palavra – sem flexão de gênero ou número – ‘comer’;
3) Análise morfológica – ‘*verb,singular,third,present,indicative,finite*’;
4) Análise sintática – no caso, ‘VERB’ indica que a palavra pertence ao predicado verbal;
5) Posição dos caracteres da palavra na sentença – ‘4..8’.

3. *AnalyzeSentence*

É a classe central deste projeto. Ela tem duas responsabilidades mais importantes:

1) Ativa a classe *PreAnalyze* para que ela faça alguns ajustes na sentença para facilitar a análise;
2) Passa a sentença por uma série de regras de interpretação, responsáveis pela criação da estrutura de grafos semânticos.

Uma descrição mais detalhada do funcionamento deta classe pode ser encontrada na seção 4, de Resultados.

4. *PreAnalyze*

Esta classe faz uma adaptação da sentença a ser analisada para que ela esteja pronta para passar pelo sistema de regras montado na classe *AnalyzeSentence*. A adaptação percorre cinco passos:

1) *insertUnknownPrimitives()*
    a. O CoGrOO não reconhece todas as palavras de entrada, e para as que ele não conhece, ele deixa a primitiva em branco. Como a primitiva é a parte da palavra acessada para leitura, este método copia a palavra original no campo primitiva;
2) *gatherKnownWords()*
    a. Este método reúne expressões reconhecidas em uma única palavra. Por exemplo, a expressão “de vez em quando” deve ser tratada como uma única palavra, que seria um advérbio de frequência;
3) *addSubject()*
    a. Adiciona sujeito às frases com sujeito oculto, pois a análise na classe AnalyzeSentence parte do princípio que todos os verbos conjugados, inclusive no imperativo, têm sujeito. As exceções são os casos de subjuntivo e de sujeito inexistente;
4) *defineVerbType()*
    a. Define se um verbo é de ação, estado ou sentimento; esta informação será utilizada no momento em que as regras de análise forem testadas. Essa separação se justifica no fato de que a interpretação das sentenças “Eu amo carne” e “Eu como carne” são de fato diferentes;
5) *updatePrimtives()*
    a. Este método será mais discutido na seção 4; ele é responsável por atribuir etiquetas tais como as mostradas na Tabela 2-5 a cada palavra que convier.

5. *UnlOutput*

Responsável por criar uma lista de expressões em formato UNL durante a execução do sistema de regras.

Uma vez terminada a execução desse sistema, esta classe é chamada pela classe *UnlToCogrooBase* para que imprima a lista de expressões no formato final de UNL.

Um exemplo de lista obtida para a entrada “Ela está elegante.” é apresentada na Figura 3.3. Essa lista é a saída do projeto.

```
{unl}
aoj(elegante, ela)
{/unl}
```

Figura 3.3 – Exemplo de saída do projeto

6. *WordClassification*

Esta classe tem o objetivo de acessar o Dicionário.xml, e devolver valores *true* ou *false* dependendo da existência ou não de uma palavra na posição desejada do léxico.

Como exemplo, pode-se citar que no campo a *tag* ‘*verb*’ tem uma *subtag* ‘*feeling*’; dentre os elementos desta *subtag* estão ‘amar’, ‘detestar’ e ‘idolatrar’. Assim, a classe *WordClassification* devolve *true* para a chamada *isFeelingVerb*(‘amar’), e devolve *false* para as chamadas *isActionVerb*(‘detestar’) e *isAdverb*(‘idolatrar’).

7. *Dictionary.xml*

É um dicionário em formato xml, cujas *tags* são tipos de palavras. Entre elas, há: verbos e suas subdivisões, como ação, estado e sentimento, advérbios (de lugar, frequência, tempo, etc), artigos, pronomes possessivos, meios de transporte, etc.

Adicionar novas entradas ao dicionário é possível e simples a qualquer momento, o que permite a extensão deste projeto para futuras melhorias.

4 RESULTADOS

Os resultados obtidos com o desenvolvimento deste projeto serão apresentados nesta seção. O programa desenvolvido é capaz de analisar diversos tipos de sentença da língua portuguesa os quais serão expostos, e há ainda a descrição de um método que simplifica a criação de futuras regras.

A Tabela 4-1 mostra o fluxo de trabalho desenvolvido neste projeto. O programa começa na classe CogrooToUnlBase, e as classes acionadas subsequentemente são listadas na ordem de execução.

Tabela 4-1 – Fluxo de atividades

Fluxo (classes e métodos)	Atividade	Exemplo (tipo de sentença analisada)
1) CogrooToUnlBase		
a. getExternalText	Adquirir texto externo	-
b. prepareText	Separar nomes próprios por '_'	"São Paulo" vira "São_Paulo"
c. printUnlResults	Imprimir os resultados armazenados na classe UnlOutput	-
2) CogrooAnalyzer		
a. prepareText	Realizar a análise do Cogroo, e depois enviar cada sentença separada por '.' para as classes de análise	-
3) PreAnalyze		
a. insUnknownPrimitives	Inserir no campo Primitiva de cada palavra uma cópia da própria palavra, nos casos em que o Cogroo não reconhece o termo em questão	-
b. gatherKnownWords	Transformar expressões compostas comuns em apenas uma palavra	"De vez em quando" passa a ser tratada como apenas uma palavra
c. addSubject	Adicionar sujeito às sentenças com sujeito	"Vou trabalhar" vira

	oculto	"Eu vou trabalhar"
d. defineVerbType	Decidir, de acordo com *Dictionary*, se um dado verbo é de estado, ação ou sentimento	-
e. updatePrimitives	Atribuir etiquetas de especificidade	-
i. addNotTag	".@not" quando há negação	
ii. addPastTag	".@past" quando algum verbo está no passado	-
iii. addPluralTag	".@pl" quando algum adjetivo ou substantivo está no plural	-
iv. addFutureTag1	".@future" quando o verbo "ir" indica o futuro do próximo verbo	"Ele vai viajar semana que vem"
v. addFutureTag2	".@future" quando o verbo da sentença está no futuro	"Ele viajará semana que vem"
vi. addDeterminerTag	".@def" ou ".@indef" quando há artigos ou pronomes definidos ou indefinidos, respectivamente	
vii. reduceLocVerbal	Reduzir locuções verbais compostas pelos verbos "poder" e "saber", e acrescentar as etiquetas ".@permission" e ".@ability" ao segundo verbo da locução	"Nós sabemos jogar futebol" vira "Nós jogamos.@ability futebol"
f. removePrepDe	Remover a preposição "de" quando ela estiver no texto somente por regência verbal, como depois dos verbos "precisar" e "gostar"	"Eu gosto de você" vira "Eu gosto você"
***4)* AnalyzeSentence**		
a. createSpecificRelations	-	-
i. analyzeImperativo	Verificar se uma estrutura contém verbos imperativos, acrescentar a etiqueta ".@imperative" e criar relação de agente ou de objeto	"Retire a torta" ou "Bruno, saia daqui!"
ii. analyzePrepEm	Estudar casos em que a preposição "em" aparece	"Estou em casa", "Ela estava linda na festa", "O gato subiu no telhado"

iii.	*analyzeGerundio*	Remover o verbo “estar”, e acrescentar a etiqueta “@.progress” ao verbo seguinte para ações contínuas	“Estamos comprando botas”
iv.	*analyzePossessivos*	Criar relações de posse, ***pos***, para situações em que o pronome está junto ao substantivo ou não	“Lavei meu carro” ou “Aquele carro é meu”
v.	*analyzeMuito*	Analisar situalções em que “muito” indica quantidade, ***qua***, ou intensidade, ***man***	“Trabalhei muito hoje” ou “Comi muitas peras anteontem” ou “Ela está muito bonita”
vi.	*analyzeAdvTempo*	Criar relação de tempo, ***tim***	“Li um livro ontem”
vii.	*analyzeAdvFreq*	Criar relação de modo, ***man*** (em relação à frequência)	“Leio de vez em quando”
viii.	*analyzeAdvLugar*	Criar relação de modo, ***man*** (em relação ao lugar ou à posição)	“Estou perto de casa” ou “O boné está sobre a mesa”
ix.	*analyzeLocVerbal*	Criar relação de objeto, ***obj***, em locuções verbais	“Eu quero comer açaí”
x.	*analyzeSintNomSuj*	Reduzir sintagmas nominais em posição de sujeito, criando relação de atributo, ***aoj***, entre possíveis adjetivos e núcleos de sujeitos	“O rei Arthur é sábio” vira “Arthur é sábio”
xi.	*analyzeSintNomGeral*	Reduzir sintagmas nominais em qualquer posição de uma sentença, criando relação de atributo, ***aoj***, entre seus componentes	“Ela comprou uma calça nova”
xii.	*analyzeCoordConj*	Criar relação do tipo ***and*** em sujeitos compostos	“A Anna e o Ricardo são amigos”
xiii.	*analyzeVirVoltar*	Criar relações de método, ***met***, ou de local de origem, ***plf***, quando os verbos “vir” ou ”voltar” aparecerem seguidos de preposição “de”	“Vim de ônibus” ou “Subi de escada” ou “Voltei de São Paulo”
xiv.	*analyzePrepPara*	Criar relações de destino, ***plt***, e de finalidade, ***gol***, quando os verbos “ir”, ”vir” e ”voltar” forem seguidos da preposição “para”, ou de sua forma coloquial, “pra”	“Fui para casa” ou “Voltei para comer”

b. createGeneralRelations	-	-
i. analyzeVerbIntrans	Criar relação de agente, ***agt***, em orações cujo verbo é seguido de ponto final ou de advérbios	"Estudei muito"
ii. analyzeSujeitoVerbo	Criar relações de agente, ***agt***, quando um verbo de ação seguir o sujeito, de atributo, ***aoj***, quando um verbo de sentimento seguir o sujeito ou quando um adjetivo segue um verbo de estado que segue o sujeito, e de local, ***plc***, quando um advérbio de lugar segue um verbo de estado que segue o sujeito	"Eu jogo..." ou "Eu amo..." ou "Eu estou feliz" ou "Eu esto aqui"
iii. analyzeObjeto	Criar relação de objeto, ***obj***, em uma locução verbal qualquer, ou quando um substantivo segue um verbo	"Quero comer frutas" ou "Comprei goiabas"
5) UnlOutput		
a. createRelationString	Montar o formato da relação enviada pela classe *AnalyzeSentence*, substituindo eventuais *underscores,* '_', por espaços em branco	"agt(comprar, ele)"

Como a UNL não autorizou a utilização de sua Ontologia até a publicação deste trabalho, ele se limitou a construir as relações no formato desejado. Quando a UNL liberar a utilização da Ontologia, será necessário substituir as primitivas de cada palavra analisada pelas primitivas encontradas na Ontologia, e então o trabalho será completado.

4.1 Classe *PreAnalyze*

A classe *PreAnalyze* atribui etiquetas às primitivas para adicionar a elas informações contextuais relevantes. Essas etiquetas são atribuídas nos seguintes casos:

1) Negativa:

   a. Busca-se a posição do verbo que está sendo negado pela palavra ‘não’, e à sua primitiva é acrescentada a etiqueta ‘.@not’;
   b. Caso o verbo que está sendo negado faça parte de uma locução verbal, a etiqueta é acrescentada ao verbo principal; por exemplo, na sentença “Eu não vou viajar amanhã”, a etiqueta vai para o verbo ‘viajar’;
2) Passado:
   a. O CoGrOO indica se um dado verbo está no passado; quando isso acontece, a etiqueta ‘.@past’ é acrescentada à primitiva;
3) Plural:
   a. Idem ao passado;
4) Futuro informal:
   a. Quando o verbo ‘ir’ aparece com função de indicação de futuro, como por exemplo em “Eu vou comer mais tarde”, ele é removido da sentença, e o verbo seguinte ganha a etiqueta ‘.@future’;
5) Futuro formal:
   a. O CoGrOO indica se um dado verbo está no futuro; quando isso acontece, a etiqueta ‘.@future’ é acrescentada à primitiva;
6) Artigos e pronomes demonstrativos (*determiners*):
   a. Quando eles definem o substantivo, a etiqueta ‘.@def’ é atribuída. Por exemplo, na sentença “A mulher está dirigindo”, a primitiva de ‘mulher’ fica ‘mulher.@def’, e o artigo ‘a’ é removido da sentença;
   b. Quando são indefinidos, o processo é o mesmo, mas a etiqueta apresentada é ‘.@indef’;
7) Imperativo:
   a. Quando um verbo aparece na primeira posição da sentença com a identificação de subjuntivo do CoGrOO, ele recebe a etiqueta ‘.@imperative’; o mesmo para quando o verbo vem depois de uma vírgula;
8) Locuções verbais com os verbos ‘saber’ e ‘poder’:
   a. O verbo ‘saber’ é removido, e a etiqueta .@ability é aplicada ao verbo subsequente;
   b. O verbo ‘poder’ também é removido, mas a etiqueta aplicada é .@permission.

4.2 Classe *AnalyzeSentence*

Para esta classe, serão mostrados resultados obtidos, e seus exemplos, para cada tipo de análise.

1) Preposição ‘em’ (Figura 4.1):
    a. Precedida de verbos de estado, como em “Estou na escola”;
    b. Precedida de adjetivo, como em “Ela estava linda na festa”;
    c. Precedida de verbos de ação, como em “Um gato subiu no telhado”.

```
Estou na escola.
{unl} - 1
   plc(eu:01, escola.@def:02)
{/unl}

Ela estava linda na festa.
{unl} - 2
   plc(lindo.@past:03, festa.@def:04)
   aoj(lindo.@past:03, ela:05)
{/unl}

Um gato subiu no telhado.
{unl} - 3
   plc(subir.@past:06, telhado.@def:07)
   agt(subir.@past:06, gato.@indef:08)
{/unl}
```

Figura 4.1 – Análise da preposição ‘em’

2) Gerúndio (Figura 4.2):
    a. Verbo ‘estar’ seguido de verbo no gerúndio, como em “Eles estão correndo muito!”;

```
Eles estão correndo muito!
{unl} - 1
   agt(correr.@progress:01, eles:02)
   man(correr.@progress:01, muito:03)
{/unl}
```

Figura 4.2 – Análise de gerúndio

3) Pronomes possessivos (Figura 4.3):
    a. Precedidos por substantivos, como em “Inimigos meus não gostam de pizza”;
    b. Sucedidos por substantivos, como em “Meus avós são simpáticos”;
    c. Precedidos pelo verbo ‘ser’, como em “A taça é nossa”;

```
Inimigos meus não gostam de pizza.
{unl} - 1
   pos(inimigo.@pl:01, eu:02)
   obj(gostar.@not:03, pizza:04)
{/unl}

Meus avós são simpáticos.
{unl} - 2
   pos(avó.@pl:05, eu:06)
   aoj(simpático.@pl:07, avó.@pl:05)
{/unl}

A taça é nossa.
{unl} - 3
   pos(taça.@def:08, nós:09)
{/unl}
```

Figura 4.3 – Pronomes possessivos

4) ‘Muito’ indicando quantidade ou intensidade (Figura 4.6):
    a. Seguido de adjetivo ou de advérbio, como em “Nós estamos muito felizes”, ou “Eles leem muito rápido!”;
    b. Seguido de substantivo, como em “Muitas mulheres comem maçã”;
    c. Precedido de verbo, como em “Eu leio muito”;

5) Advérbio de tempo (Figura 4.4):
    a. Relacionado a um verbo da sentença, como em “Viemos de carro ontem”;

```
Viemos de carro ontem.
{unl} - 1
   met(vir.@past:01, carro:02)
   tim(vir.@past:01, ontem:03)
   agt(vir.@past:01, nós:04)
{/unl}
```

Figura 4.4 – Advérbio de tempo

6) Advérbio de frequência (Figura 4.5):
    a. Relacionado a um verbo da sentença, como em “Estudamos de vez em quando”;

```
Estudamos de vez em quando .
{unl} - 1
   man(estudar:02, de vez em quando:01)
   agt(estudar:02, nós:03)
{/unl}
```

Figura 4.5 – Advérbio de frequência

```
Nós estamos muito felizes.
{unl} - 1
   man(feliz.@pl:01, muito:02)
   aoj(feliz.@pl:01, nós:03)
{/unl}

Eles leem muito rápido.
{unl} - 2
   qua(rápido:04, muito:05)
   agt(leem.@past:06, eles:07)
   obj(leem.@past:06, rápido:04)
{/unl}

Muitas mulheres comem maçã.
{unl} - 3
   qua(mulher.@pl:08, muito.@pl:09)
   agt(comer:010, mulher.@pl:08)
   obj(comer:010, maçã:011)
{/unl}

Eu leio muito.
{unl} - 4
   agt(ler:012, eu:013)
   man(ler:012, muito:014)
{/unl}
```

Figura 4.6 – Análise de ‘muito’

7) Advérbio de lugar (Figura 4.7):
    a. Relacionado a um verbo da sentença, como em “O cachorro está em cima do sofá”;

```
O cachorro está em cima do sofá .
{unl} - 1
   mod(sofá.@def:02, em cima de:01)
   plc(cachorro.@def:03, sofá.@def:02)
{/unl}
```

Figura 4.7 – Advérbio de lugar

8) Verbos intransitivos (Figura 4.8):
    a. Seguidos de pontuação ou por advérbios, como em “Cheguei!” e “Dormi cedo.”;

```
Cheguei.
{unl} - 1
   agt(chegar.@past:01, eu:02)
{/unl}

Dormi cedo.
{unl} - 2
   agt(dormir.@past:03, eu:04)
   tim(dormir.@past:03, cedo:05)
{/unl}
```

Figura 4.8 – Verbos intransitivos

9) Locuções verbais (Figura 4.9):
    a. Com o verbo ‘querer’, como em “Quero comprar uma bola”;

```
Quero comprar uma bola.
{unl} - 1
   obj(querer:01, comprar:02)
   agt(comprar:02, eu:03)
   obj(comprar:02, bola.@indef:04)
{/unl}
```

Figura 4.9 – Locução verbal

10) Verbos ‘vir’ e ‘voltar’ seguidos de preposição ‘de’ (Figura 4.10):
    a. Com meio de transporte, como em “Vim de ônibus”;
    b. Com localizações, como em “Cheguei de São Paulo”;

```
Vim de ônibus.
{unl} - 1
   met(vir.@past:01, ônibus:02)
   agt(vir.@past:01, eu:03)
{/unl}

Cheguei de São Paulo.
{unl} - 2
   plf(chegar.@past:05, São Paulo:04)
   agt(chegar.@past:05, eu:06)
{/unl}
```

Figura 4.10 – Verbos 'vir' e 'voltar'

11) Verbos de locomoção seguidos da preposição 'para' (Figura 4.11):
    a. Seguidos de substantivo, como em "Vou para casa";
    b. Seguidos de infinitivo, como em "Parti para ficar";

```
Venho para casa.
{unl} - 1
   plt(vir:01, casa:02)
   agt(vir:01, eu:03)
{/unl}

Voltei para ficar.
{unl} - 2
   gol(voltar.@past:04, ficar:05)
   agt(voltar.@past:04, eu:06)
{/unl}
```

Figura 4.11 – Indicação de destino ou de finalidade

12) Conjunções (Figura 4.12):
    a. Aditiva 'e' entre substantivos, como em "o Ricardo Beato e o Marcos Barretto trabalharam muito";

```
O Ricardo Beato e o Marcos Barretto trabalharam muito .
{unl} - 1
   and(Marcos Barretto.@def:01, Ricardo Beato.@def:02)
   man(trabalhar.@past:03, muito:04)
   agt(trabalhar.@past:03, Marcos Barretto.@def:01)
{/unl}
```

Figura 4.12 – Conjunção aditiva 'e'

13) Formação de orações (Figura 4.13):
    a. Com verbo de ação, como em "Eu corro";

b. Com verbo de estado e adjetivo, como em “Ela está linda”;
c. Com verbo de estado e localização, como em “O sapato está embaixo do sofá”;
d. Com verbo de sentimento, como em “Eles amam paçoca”;

14) Simplificação de sintagmas nominais (Figura 4.14):
a. Em posição de sujeito, como em “O deputado Roberto Jefferson denunciou o mensalão”;
b. Em posições genéricas na frase, como em “O Madruguinha tomou um bom banho”;

```
Eu corro .
{unl} - 1
   agt(correr:01, eu:02)
{/unl}

Ela está linda .
{unl} - 2
   aoj(lindo:03, ela:04)
{/unl}

O sapato está embaixo do sofá .
{unl} - 3
   mod(sofá.@def:06, embaixo de:05)
   plc(sapato.@def:07, sofá.@def:06)
{/unl}

Eles amam paçoca .
{unl} - 4
   agt(amar:08, eles:09)
   obj(amar:08, paçoca:010)
{/unl}
```

Figura 4.13 – Formação de orações

```
O senador Eduardo Suplicy criou algumas leis .
{unl} - 1
   aoj(senador.@def:01, Eduardo Suplicy:02)
   agt(criar.@past:03, Eduardo Suplicy:02)
   obj(criar.@past:03, lei.@indef.@pl:04)
{/unl}

O Madruguinha tomou um bom banho .
{unl} - 2
   aoj(bom:05, banho.@indef:06)
   agt(tomar.@past:07, O Madruguinha:08)
   obj(tomar.@past:07, banho.@indef:06)
{/unl}
```

Figura 4.14 – Simplificação de sintagmas nominais

15) Identificação de objeto (Figura 4.15):

a. Em locuções verbais, como em “Eu preciso correr”;
b. Em orações simples, como em “Acendo fósforos”;

```
Eu preciso correr .
{unl} - 1
   agt(precisar:01, eu:02)
   obj(precisar:01, correr:03)
{/unl}

Acendo fósforos .
{unl} - 2
   agt(acender:04, eu:05)
   obj(acender:04, fósforo.@pl:06)
{/unl}
```

Figura 4.15 – Identificação de objeto

4.3 Método para elaboração de novas regras

A classe *AnalyzeSentence* tem dois métodos que podem conter novas regras. O método *createSpecificRules()* deve conter novas regras que sejam específicas devido a determinadas palavras. Por exemplo, se se quer definir uma regra para contextos que contenham a preposição ‘em’, isso deve ser feito nesse método.

Há ainda um outro método, chamado *createGeneralRules()*, que deverá conter regras gerais da língua portuguesa a partir apenas da análise morfológica de cada palavra em uma sentença. Como exemplo, pode-se citar a criação de uma regra que estabeleça que sempre que um advérbio de tempo for encontrado, ele deve ser associado ao verbo mais próximo a ele dentro da sentença.

A Figura 4.16 apresenta uma regra presente no programa para ilustrar a explicação do método de desenvolvimento de novas regras.

O método consiste de três passos:

1. Estabelecer a regra; no exemplo da Figura 4.16, dentro do laço de execução do método *createSpecificRules()*, o programa busca pelo verbo 'ir' seguido da preposição 'para';
2. Uma vez satisfeito o requisito da regra, dois métodos devem ser acionados;
   i. *createRelationString()*: este método envia uma nova relação semântica e um par de palavras relacionadas por esse atributo; no caso, se a palavra seguinte – posição *i+2* – for um verbo, há a criação de uma relação de finalidade, ou 'goal', representada por 'gol';
   ii. *replaceClassifiedWords()*: este método reduz a sentença; no exemplo da Figura 4.16, caso um verbo siga a preposição 'para', como em "Vou para comer", ele substitui todas as palavras da oração por apenas uma palavra indicando que a oração já foi analisada.

```
//Example: "Eu vou para Santos."
if (sentence.size() > 2) {
    if (getPrimAt(i).equals("ir")) {
        if (getPrimAt(i+1).equals("para") || getPrimAt(i+1).equals("pra")) {
            if (getWordAt(i+2).contains("noun")) {
                createRelationString("plt", getRelPrimAt(i-1), i-1, getRelPrimAt(i+2), i+2);
                replaceClassifiedWords(i-1, i+2, getRelPrimAt(i), "action verb");
            }
            if (isVerb(i+2)) {
                createRelationString("gol", getRelPrimAt(i), i, getRelPrimAt(i+2), i+2);
                replaceClassifiedWords(i-1, i+2, getRelPrimAt(i+2), "verb");
            }
        }
    }
}
```

Figura 4.16 – Criação de regras

O resultado da operação descrita está na Figura 4.17.

```
Vou para comer .
{unl} - 1
   gol(ir:01, comer:02)
   agt(ir:01, eu:03)
{/unl}
```

Figura 4.17 – Método

Com o método apresentado nesta seção, a divisão de trabalho entre programadores e linguistas fica simplificada, e o trabalho dos primeiros fica direcionado, de forma que o processo possa ser realizado mais rapidamente.

5 CONSIDERAÇÕES FINAIS

a) Realizações do projeto

O trabalho de construção de relações semânticas entre conceitos presentes em sentenças da língua portuguesa é extenso, e uma parte desse trabalho foi realizada com sucesso.

A abrangência deste projeto pode ser observada na Tabela 4-1, na qual os tipos de relações analisados aqui são apresentados. É importante notar que a ordem na qual as análises são realizadas é extremamente importante para que o programa funcione corretamente.

Primeiro, devem ser criadas etiquetas, eliminando das sentenças elementos não centrais, como artigos e preposições; em seguida, devem-se realizar análises específicas, pois as muitas exceções às regras da língua devem ser observadas; por fim, caso nenhuma regra específica tenha sido atendida, as regras gerais devem ser verificadas.

O trabalho de análise semântica pode ser indefinidamente estendido, pois a língua é dinâmica e se apresenta de novas formas de tempos em tempos. Assim, foi desenvolvido um método que possibilita a criação mais eficaz de regras de interpretação semântica – seção 4.3.

b) Trabalhos futuros

Existem alguns trabalhos que podem tornar este projeto mais poderoso, como a extensão do dicionário criado. Além disso, a criação de novas regras aumentaria o escopo das análises, mas seria útil a contratação de linguistas para definição de regras mais consistentes.

REFERÊNCIAS

[1] Aires, R. (2000). *Implementação, adaptação, combinação e avaliação de etiquetadores para português do Brasil.* Dissertação de Mestrado apresentada no ICMC – USP;

[2] Aires, R., Aluísio, S., Kuhn, D., Andreeta, M., & Oliveira Jr, M. (2000). *Combining Multiple Classifiers to Improve Part-of-Speech Taggers: a Case Study for Brazilian Portuguese.* São Carlos: In the proceedings of the Brazilian AI Symposium (SBIA 2000);

[3] ALFENAS, D. A. ; Pereira-Barretto, Marcos R. . Adaptatividade em Robôs Sociáveis. In: Workshop de Tecnologias Adaptativas (WTA2012), 2012, São Paulo. Anais do WTA2012, 2012.

[4] *CCSL-IME-USP*. (2006). Acesso em 10 de Junho de 2012, disponível em http://ccsl.ime.usp.br/redmine/projects/cogroo/wiki/CoGrOO_Funciona;

[5] Kinoshita, J., Salvador, L. d., & Menezes, C. E. (2005). CoGrOO - Um Corretor Gramatical para a língua portuguesa, acoplável ao OpenOffice. *In Proceedings of XXXI Latin American Informatics Conference.* Cali, Colômbia;

[6] Naber, D. (2003). Tese de mestrado: "*A Rule-Based Style and Grammar Checker".* Technische Fakultät, Universität Bielefeld;

[7] Pires, D. A., Gusukuma, F. W., Suzumura, M., & Colen, W. D. (2006). *Corretor Gramatical Acoplável ao OpenOffice.org CoGrOO 2.0.* São Paulo: Projeto de Formatura - Escola Politécnica da USP;

[8] Uchida, H., Zhu, M., & Senta, T. D. (1999). *A gift for a Millenium* – Publicado por UNU/IAS;

[9] *UNDL - Aplicações*. (2010). Acesso em 18 de Junho de 2012, disponível em http://www.undl.org/index.php?option=com_content&view=article&id=50&Itemid=81&lang=en;

[10] *UNDL - UNL System*. (2006). Acesso em 17 de Junho de 2012, disponível em http://www.undl.org/index.php?option=com_content&view=article&id=49&Itemid=78&lang=en#Universal Parser;

[11] *UNDL*. (2010). Acesso em 09 de Junho de 2012, disponível em http://www.undl.org/index.php?option=com_content&view=article&id=46&Itemid=58&lang=en;

[12] *The JBoss Drools team - Drools Expert User Guide version 5.4.0.Final* (encontrado também em http://www.jboss.org/drools/team.html);

[13] Forgy, C (1978-79) – Tese de doutorado: "*Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem*";

[14] *UNDL.* (2006). Acesso em 23 de setembro de 2012, disponível em http://www.undl.org/unlsys/unlman/

[15] Projeto Florescer. Acesso em 02 de dezembro de 2012, disponível em http://www.projetofloreser.blogspot.com.br/2010/08/cientistas-usam-robo-para-estimular.html

APÊNDICE A – *CogrooToUnlBase*

```
/*
 * @author Ricardo Pereira Beato Junior
 * @date 02/Oct/2012
 */

package hellodrools;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

// Classe principal, que inicializa tudo o que eh necessario ao programa
public class CogrooToUnlBase {

   // Variavel para captar o texto de entrada do usuario
   private static String textToCogroo;

   // Variavel que contem o texto a ser analisado
   private String text;

   // Texto corrigido com uniao de nomes compostos
   private StringBuilder newText = new StringBuilder();

   // Acesso ao dicionario
   public WordClassification classif = new WordClassification("../HelloDrools/build/"
                                    + "classes/hellodrools/Dictionary");

   // Metodo que inicializa o programa
   public static void main(final String[] args) throws IOException {
      new CogrooToUnlBase().start();
   }

   // Chamada dos métodos de configuracao do sistema de regras, da
   // inicializacao da lista de saida, da captacao do texto a ser analisado,
```

```
// da configuracao do analisador CoGrOO e da impressao da saida
public void start() throws IOException {
   initOutputList();
   getExternalText();
   prepareText();
   useCogrooAnalyzer();
   printUnlResults();
}

// Inicializa a lista que conterah a saida do programa em formato UNL
private void initOutputList() {
   new UnlOutput();
}

// Chama o construtor da classe CogrooAnalyzer para que o CoGrOO analise o
// texto
private static void useCogrooAnalyzer() {
   new CogrooAnalyzer(CogrooToUnlBase.textToCogroo);
}

// Imprime a saida do projeto
private static void printUnlResults() {
    UnlOutput.printOutput();
}

// Le e armazena o texto do usuario, presente no arquivo de "textEntry.txt"
private void getExternalText() throws IOException {
   File file = new File("D:/Ricardinho/Desktop/TF/textEntry2.txt");
   StringBuilder contents = new StringBuilder();
   BufferedReader reader;
   reader = new BufferedReader(new FileReader(file));
   String textBuilder;
   while ((textBuilder = reader.readLine()) != null) {
      contents.append(textBuilder).append(System.getProperty("line.separator"));
   }
   text = contents.toString();
   if (reader != null) reader.close();
}
```

```
// Reune palavras seguidas comecando com letras maiusculas em apenas uma string
private void prepareText() {
   String[] words;
   addSpaces();
   words = text.split(" ");
   int point = 0;
   for (int i = 0; i < (words.length); i++) {
      if (i == point && classif.isDeterminer(words[point].toString().toLowerCase())) {
         printPalavSolta(words, i);
         i++;
      }
      int j = i;
      int numPalavras = 0;
      if (Character.isUpperCase(words[i].charAt(0)))
         numPalavras = findUpperCases(j, words) -  i + 1;
      if (numPalavras > 0) {
         unirPalavras(numPalavras, i, words);
         i = i + numPalavras - 1;
      } else printPalavSolta(words, i);
   }
   CogrooToUnlBase.textToCogroo = newText.toString();

}

private void addSpaces() {
   text = text.replace(",", " ,");
   text = text.replace(";", " ;");
   text = text.replace(".", " .");
}

private int findUpperCases(int j, String[] words) {
   int a = j+1;
   if (words[a].equals("de") || words[a].equals("da") || words[a].equals("das") ||
      words[a].equals("do") || words[a].equals("dos")) {
      if (Character.isUpperCase(words[a+1].charAt(0))) {
         j = j + 2;
         a = a + 2;
```

```
            }
        }
        while (a < words.length - 1) {
            if (Character.isUpperCase(words[a].charAt(0))) {
                j++;
                a++;
            } else a = words.length;
        }
        return j;
    }

    private void unirPalavras(int numPalavras, int i, String[] words) {
        for (int p = 0; p < words.length; p++) {
            if (i <= p && p < i + numPalavras - 1)
                newText.append(words[p] + "_");
            if (p == i + numPalavras - 1) newText.append(words[p] + " ");
        }
    }

    private void printPalavSolta(String[] words, int i) {
        newText.append(words[i] + " ");
    }
}
```

APÊNDICE B – *CogrooAnalyzer*

```
/*
 * @author Ricardo Pereira Beato Junior
 * @date 02/Oct/2012
 */

package hellodrools;

import br.usp.pcs.lta.cogroo.configuration.LegacyRuntimeConfiguration;
import br.usp.pcs.lta.cogroo.configuration.RuntimeConfigurationI;
import br.usp.pcs.lta.cogroo.entity.Sentence;
import br.usp.pcs.lta.cogroo.entity.Token;
import br.usp.pcs.lta.cogroo.grammarchecker.CheckerResult;
import br.usp.pcs.lta.cogroo.grammarchecker.Cogroo;
import br.usp.pcs.lta.cogroo.tag.LegacyTagInterpreter;
import br.usp.pcs.lta.cogroo.tag.TagInterpreterI;
import java.util.ArrayList;

public class CogrooAnalyzer {

   // Variaveis que possibilitam a configuracao do CoGrOO, e posterior
   // analise do texto desejado
   private Cogroo cogroo;
   private String text;
   private static final TagInterpreterI tagInterpreter = new LegacyTagInterpreter();
   private CheckerResult results;
   private ArrayList<String> eachSentence  = new ArrayList<>();
   private ArrayList<ArrayList> pairs = new ArrayList<>();
   public ArrayList<ArrayList> wordsTags = new ArrayList<>();

   // Configura os parametros da arvore sintatica para que sejam usados no
   // sistema de regras
   public CogrooAnalyzer(String text) {
      setCogroo();
      setText(text);
      prepareText();
```

```
}

// Inicializa a base de trabalho do CoGrOO
private void setCogroo() {
   RuntimeConfigurationI config = new LegacyRuntimeConfiguration();
   this.cogroo = new Cogroo(config);
}

// Texto externo
private void setText(String text) {
   this.text = text;
}

// Armazena cada sentenca do texto em posicoes consecutivas de eachSentence
// Armazena cada palavra de uma sentenca dentro de eachToken
// Separa sentencas por uma string '.' entre palavras armazenadas
private void prepareText() {
   analyzeText();
   populateSentencesAndTokensLists();
}

// Analisa o texto, atribuindo etiquetas a palavras
private void analyzeText() {
   this.results = this.cogroo.analyseAndCheckText(this.text);
}

// Cria um ArrayList de ArrayLists.
// Cada elemento (wordTag) de wordsTags contem 5 tags.
// [Palavra][Primitiva][Tags Gramaticais][Sintatica][Tamanho]
private void populateSentencesAndTokensLists() {
   for (Sentence sentence : this.results.sentences) {

      // Inicializa a saida em UNL
      UnlOutput.output.add("{unl}");

      StringBuilder formSentence = new StringBuilder();

      for (Token token : sentence.getTokens()) {
```

```
            ArrayList<String> wordTags = new ArrayList<>();

            wordTags.add(token.getLexeme().toString()); //Ela
            wordTags.add(token.getPrimitive().toString()); //ela
            wordTags.add(token.getMorphologicalTag().toString());          //personal pronoun,
                                                                           //female, singular,
                                                                        //nominative, third
            wordTags.add(token.getSyntacticTag().toString()); //SUBJECT
            wordTags.add(token.getSpan().toString()); //0..3

            for (int i = 0; i<wordTags.size(); i++) {
               if (!this.wordsTags.contains(wordTags)) {
                  this.wordsTags.add(wordTags);
               }
            }

            formSentence.append(token.getLexeme()).append(" ");
         }
         new Word(this.wordsTags);
         this.wordsTags.clear();
         UnlOutput.output.add("{/unl} \n");
         this.eachSentence.add(formSentence.toString());
      }
   }
}
```

APÊNDICE C – *AnalyzeSentence*

```
/*
 * @author Ricardo Pereira Beato Junior
 * @date 02/Oct/2012
 */

package hellodrools;

import java.util.ArrayList;

// Classe que contem metodos para acesso das partes da frase que serao
// analisadas pelo arquivo de regras.
public class AnalyzeSentence {

   // Lista que deverah conter, em cada posicao, a palavra com todos
   // os seus atributos, tais como a palavra em si, sua primitiva, sua
   // classe gramatical, seu genero, seu numero, etc (o conteudo exato de)
   // cada posicao dessa lista depende de cada tipo de palavra.
   //
   // Palavras podem ser de varios tipos: pontos como '.' ou ':', adjetivos,
   // substantivos, etc.
   public static ArrayList<ArrayList> sentence = new ArrayList<>();

   // Criacao de uma variavel global para que o numero de etiquetas cresca
   // conforme seu uso
   public static int etiqueta = 1;

   // Acesso ao dicionario
   public WordClassification classif = new WordClassification("D:/Ricardinho/Desktop/TF/"
            + "Projeto HelloDrools/HelloDrools/build/classes/hellodrools/Dictionary");

   // Contrutor da classe AnalyzeSentence, que dispara o arquivo de regras.
   public AnalyzeSentence (ArrayList<ArrayList> sentence) {
      AnalyzeSentence.sentence = sentence;
      setRules();
   }
```

```
    private void setRules() {
        PreAnalyzer p = new PreAnalyzer();
        p.preSetRules(sentence);
// Loops para testes
//      for (int a = 0; a < AnalyzeSentence.sentence.size(); a++)
//          System.out.println(AnalyzeSentence.getWordAt(a) + "66666");
        createSpecificRelations();
//      for (int a = 0; a < AnalyzeSentence.sentence.size(); a++)
//          System.out.println(AnalyzeSentence.getWordAt(a) + "77777");
        createGeneralRelations();
//      for (int a = 0; a < AnalyzeSentence.sentence.size(); a++)
//          System.out.println(AnalyzeSentence.getWordAt(a) + "88888");
    }

    // Coloca na primitiva da primeira palavra da frase a etiqueta
    // correspondente ao que jah foi analisado.
    // Exemplo: "A Anna e eu compramos um carro." passa a ser ":01 compramos
    // um carro."
    public static void replaceClassifiedWords(int first, int last, String tag, String grammar) {
        if (sentence.size() > first && first >= 0) {
            sentence.get(first).set(0, "Ok");
            if (!tag.contains(":0")) {
                sentence.get(first).set(1, tag);
                sentence.get(first).set(4, tag + ":0" + etiqueta);
                etiqueta++;
            }
            else {
                sentence.get(first).set(1, tag);
                sentence.get(first).set(4, tag);
            }
            sentence.get(first).set(2, grammar);
            for(int j = first + 1; j <= last; j++) {
                sentence.remove(first + 1);
            }
        }
    }
```

```
// Envia relacoes prontas para a classe UnlOutput criar a saida
public static void createRelationString(String rel, String word01, int i, String word02, int j) {
   if (!word01.contains(":0")) {
      word01 = word01 + ":0" + etiqueta;
      sentence.get(i).set(4, word01);
      etiqueta++;
   }
   if (!word02.contains(":0")) {
      word02 = word02 + ":0" + etiqueta;
      sentence.get(j).set(4, word02);
      etiqueta++;
   }
   UnlOutput.createRelationString(rel, word01, word02);
}

// Retorna apenas a primitiva da palavra da posicao i.
// Essa primitiva serah usada na saida UNL.
public static String getPrimAt(int i) {
   if (sentence.size() > i && i >= 0) {
      return sentence.get(i).get(1).toString();
   }
   return "   ";
}

// Primitiva alterada com tags para construir relacoes
public static String getRelPrimAt(int i) {
   if (sentence.size() > i && i >= 0) {
      return sentence.get(i).get(4).toString();
   }
   return "   ";
}

// Palavra completa - com classificacoes
public static String getWordAt(int i) {
   if (sentence.size() > i && i >= 0) {
      return sentence.get(i).toString();
   }
   return "   ";
```

```
}

public static boolean isPunctuation(int i) {
   String p = getPrimAt(i);
   if (p.equals(".") || p.equals(",") || p.equals(";") || p.equals("!"))
      return true;
   return false;
}

public static boolean isVerb(int i) {
   if (getWordAt(i).contains("verb") && !getWordAt(i).contains("adverb"))
      return true;
   return false;
}

public static boolean isNoun(int i) {
   if (sentence.size() > i) {
      if (getWordAt(i).contains("noun") && !getWordAt(i).contains("pronoun"))
         return true;
   }
   return false;
}

private String possessivePerson(int i) {
   String person = null;
   String prim = getPrimAt(i);
   if (classif.isPosEu(prim)) person = "eu";
   if (classif.isPosVoce(prim)) person = "você";
   if (classif.isPosTu(prim)) person = "tu";
   if (classif.isPosNos(prim)) person = "nós";
   if (classif.isPosEle(prim)) person = "ele";
   if (classif.isPosEla(prim)) person = "ela";
   if (classif.isPosEles(prim)) person = "eles";
   if (classif.isPosElas(prim)) person = "elas";
   return person;
}

private void createSpecificRelations() {
```

```
    for (int i = 0; i < sentence.size(); i++) {
        analyzeImperativo();
        analyzePreposicaoEm(i);
        analyzeGerundio(i);
        i = analyzePossessivos(i);
        analyzeMuito(i);
        analyzeAdverbTempo(i);
        analyzeAdverbFreq(i);
        analyzeAdverbLugar(i);
        analyzeLocucaoVerbal(i);
        analyzeSintagmaNomSujeito(i);
        analyzeSintagmaNomGeral(i);
        analyzeCoordConj(i);
        analyzeVirVoltar(i);
        analyzePrepPara(i);
    }
}

private void createGeneralRelations() {
    boolean goBack = false;
    for (int i = 0; i < sentence.size(); i++) {
        i = isGoBackTrue(goBack, i);
        i = analyzeVerbosIntransitivos(i);
        goBack = analyzeSujeitoVerbo(i, goBack);
        analyzeObjeto(i);
        analyzeReasonTag(i);
    }
}

private void analyzePreposicaoEm(int i) {
    // "Em" como preposicao de lugar
    if (getPrimAt(i).equals("em") && getWordAt(i+1).contains("noun")) {
        // "Estou em casa."
        if (classif.isStateVerb(getPrimAt(i-1))) {
            createRelationString("plc", getRelPrimAt(i-2), i-2, getRelPrimAt(i+1), i+1);
            replaceClassifiedWords(i-2, i+1, getRelPrimAt(i-2), "oracao ou lugar");
        }
        // "Ela estava linda na festa."
```

```
      if (getWordAt(i-1).contains("adjective")) {
         createRelationString("plc", getRelPrimAt(i-1), i-1, getRelPrimAt(i+1), i+1);
         replaceClassifiedWords(i-1, i+1, getRelPrimAt(i-1), "adjective");
      }
      // "O gato subiu no telhado."
      if (classif.isActionVerb(getPrimAt(i-1))) {
         createRelationString("plc", getRelPrimAt(i-1), i-1, getRelPrimAt(i+1), i+1);
         replaceClassifiedWords(i-1, i+1, getRelPrimAt(i-1), "action");
      }
   }
}

private void analyzeGerundio(int i) {
   // Gerundio
   if (getPrimAt(i).equals("estar") || getPrimAt(i).equals("ficar")) {
      if (getWordAt(i+1).contains("gerund")) {
         replaceClassifiedWords(i, i+1, getRelPrimAt(i+1) + ".@progress", "action verb");
         i--;
      }
   }
}

private int analyzePossessivos(int i) {
   // Possessivos
   if (possessivePerson(i) != null) {
      String owner = possessivePerson(i);
      int position = -1;
      if (getWordAt(i-1).contains("noun")) position = i-1;
      if (getWordAt(i+1).contains("noun")) position = i+1;
      if (position != -1) {
         createRelationString("pos", getRelPrimAt(position), position, owner, i);
         if (position == i+1) {
            replaceClassifiedWords(i, i+1, getRelPrimAt(position), "noun");
            i--;
         }
         else {
            replaceClassifiedWords(i-1, i, getRelPrimAt(position), "noun");
            i--;
```

```
            }
        }
        else if (getPrimAt(i-1).equals("ser")) {
                for (int k = 0; k < sentence.size(); k++)
                    if (isNoun(k)) {
                        createRelationString("pos", getRelPrimAt(k), k, owner, i);
                        replaceClassifiedWords(i-1, i, getRelPrimAt(k), "SUBJECT");
                        i--;
                    }
        }
    }
    return i;
}

private void analyzeMuito(int i) {
    // "Muito"
    if (getPrimAt(i).equals("muito")) {
        if (getWordAt(i+1).contains("adjective") || getWordAt(i+1).contains("adverb")) {
            createRelationString("man", getRelPrimAt(i+1), i+1, getRelPrimAt(i), i);
            replaceClassifiedWords(i, i+1, getRelPrimAt(i+1), "adjective or adverb");
        } else if (isNoun(i+1)) {
            createRelationString("qua", getRelPrimAt(i+1), i+1, getRelPrimAt(i), i);
            replaceClassifiedWords(i, i+1, getRelPrimAt(i+1), "noun");
        } else if (isVerb(i-1)) {
            createRelationString("man", getRelPrimAt(i-1), i-1, getRelPrimAt(i), i);
            replaceClassifiedWords(i-1, i, getRelPrimAt(i-1)/* + ":0" + etiqueta*/, "verb");
        }
    }
}

private void analyzeAdverbTempo(int i) {
    // Adverbios de tempo
    if (classif.isAdverbTempo(getPrimAt(i))) {
        boolean bool = true;
        for (int b = 0 ; b < sentence.size() && bool; b++) {
            if (isVerb(b)) {
                createRelationString("tim", getRelPrimAt(b), b, getRelPrimAt(i), i);
                replaceClassifiedWords(b, b, getRelPrimAt(b), "action verb");
```

```
            sentence.remove(i);
            bool = false;
          }
        }
      }
    }

    private void analyzeAdverbFreq(int i) {
      // Adverbios de tempo
      if (classif.isAdverbFreq(getPrimAt(i))) {
        boolean bool = true;
        for (int b = 0 ; b < sentence.size() && bool; b++) {
          if (isVerb(b)) {
            createRelationString("man", getRelPrimAt(b), b, getRelPrimAt(i), i);
            replaceClassifiedWords(b, b, getRelPrimAt(b), "action verb");
            sentence.remove(i);
            bool = false;
          }
        }
      }
    }

    private void analyzeAdverbLugar(int i) {
      // Adverbios de tempo
      if (classif.isAdverbLugar(getPrimAt(i))) {
        if (isNoun(i+1)) {
          createRelationString("mod", getRelPrimAt(i+1), i+1, getRelPrimAt(i), i);
          replaceClassifiedWords(i, i+1, getRelPrimAt(i+1), "noun place");
        }
      }
    }


    private void analyzeImperativo() {
      // Example: "Retire a torta."
      // No imperativo os verbos tendem a aparecer na primeira posicao da sentenca
      if (getWordAt(0).contains("subjunctive")) {
        if (sentence.size() == 2 && isPunctuation(1)) {
```

```
            UnlOutput.createRelationString("agt", getRelPrimAt(0) + ".@imperative", "você");
            sentence.remove(0);
        } else {
            createRelationString("obj", getRelPrimAt(0) + ".@imperative", 0, getRelPrimAt(1), 1);
            replaceClassifiedWords(0, 1, getRelPrimAt(0), "verb");
        }
    } else if (sentence.size() > 2 && (getWordAt(0).contains("proper noun") ||
            getWordAt(0).contains("noun")) && getPrimAt(1).equals(",") &&
            getWordAt(2).contains("subjunctive")) {
            sentence.remove(1);
            createRelationString("agt", getRelPrimAt(1) + ".@imperative", 1, getRelPrimAt(0), 0);
            replaceClassifiedWords(0, 1, getRelPrimAt(1), "verb");
    }
}

private void analyzeReasonTag(int i) {
    boolean bool01 = true;
    boolean bool02 = true;
    if (getWordAt(i).contains("porque")) {
        for (int j = 0; bool01 && j < i; j++) {
            if (isVerb(j)) {
                bool01 = false;
                for (int k = j+1; bool02 && k < sentence.size(); k++) {
                    if (isVerb(k)) {
                        bool02 = false;
                        createRelationString("rsn", getRelPrimAt(j), j, getRelPrimAt(k), k);
                        replaceClassifiedWords(j, k, getRelPrimAt(k), "verb");
                    }
                }
            }
        }
    }
}

private void analyzeLocucaoVerbal(int i) {
    // Locucoes verbais
    if (isVerb(i) && getWordAt(i+1).contains("infinitive")) {
        // "Querer"
```

```
      if (getPrimAt(i).equals("querer") && !isPunctuation(i+1)) {
        createRelationString("obj", getRelPrimAt(i), i,getRelPrimAt(i+1), i+1);
        replaceClassifiedWords(i, i+1, getRelPrimAt(i+1), "verb");
      }
      // Feeling verbs
      if (classif.isFeelingVerb(getPrimAt(i)) && !isPunctuation(i+1)) {
        createRelationString("obj", getRelPrimAt(i), i, getRelPrimAt(i+1), i+1);
        replaceClassifiedWords(i, i+1, getRelPrimAt(i+1), "feeling verb");
      }
    }
  }

  private void analyzeSintagmaNomSujeito(int i) {
    // Simplificacao de sintagmas nominais em posicao de sujeito
    // Exemplo: "O Deputado Cordovil foi eleito."
    if (getWordAt(i).contains("SUBJECT") &&
      (getWordAt(i+1).contains("proper noun") ||
       Character.isUpperCase(getPrimAt(i+1).charAt(0)))) {
        createRelationString("aoj", getRelPrimAt(i), i, getRelPrimAt(i+1), i+1);
        replaceClassifiedWords(i, i+1, getRelPrimAt(i+1), "SUBJECT");
    }
  }

  private void analyzeSintagmaNomGeral(int i) {
    // Exemplo: "Calca nova"
    if (isNoun(i)) {
      if (getWordAt(i-1).contains("adjective")) {
        createRelationString("aoj", getRelPrimAt(i-1), i-1, getRelPrimAt(i), i);
        replaceClassifiedWords(i-1, i, getRelPrimAt(i), "noun");
      } else if (getWordAt(i+1).contains("adjective")) {
        createRelationString("aoj", getRelPrimAt(i+1), i+1, getRelPrimAt(i), i);
        replaceClassifiedWords(i, i+1, getRelPrimAt(i), "noun");
      }
    }
  }

  private void analyzeCoordConj(int i) {
    // Exemplo: "A Anna e o Ricardo riscaram o caderno."
```

```
    if (getWordAt(i).contains("coordinating conjunction")) {
      if (getWordAt(i-1).contains("noun") && getWordAt(i+1).contains("noun")) {
        createRelationString("and", getRelPrimAt(2), 2, getRelPrimAt(0), 0);
        replaceClassifiedWords(0, 2, getRelPrimAt(2), "noun");
      }
    }
}

private void analyzeVirVoltar(int i) {
    // Verbos "vir" e "voltar" seguidos de preposicao "de"
    if (classif.isActionVerb(getPrimAt(i))) {
      // "Vir de"
      if (getPrimAt(i+1).equals("de")) {
        // "Eu voltei de onibus."
        if (classif.isTransportation(getPrimAt(i+2))) {
          createRelationString("met", getRelPrimAt(i), i, getRelPrimAt(i+2), i+2);
          replaceClassifiedWords(i, i+2, getRelPrimAt(i), "action verb");
          // "Eu vim de carro de Sao Paulo."
          if (getPrimAt(i+1).equals("de") && isNoun(i+2)) {
            createRelationString("plf", getRelPrimAt(i), i, getRelPrimAt(i+2), i+2);
            replaceClassifiedWords(i, i+2, getRelPrimAt(i), "action verb");
          }
        }
        else {
          createRelationString("plf", getRelPrimAt(i), i, getRelPrimAt(i+2), i+2);
          replaceClassifiedWords(i, i+2, getRelPrimAt(i), "action verb");
        }
      }
    }
}

private void analyzePrepPara(int i) {
    // Verbos "ir", "vir" e "voltar" seguidos de preposicao "para"
    if (getPrimAt(i).equals("ir") || getPrimAt(i).equals("voltar") ||
      getPrimAt(i).equals("vir") || getPrimAt(i).equals("partir")) {
      if (getPrimAt(i+1).equals("para") || getPrimAt(i+1).equals("pra")) {
        if (isNoun(i+2)) {
          createRelationString("plt", getRelPrimAt(i), i, getRelPrimAt(i+2), i+2);
```

```
            replaceClassifiedWords(i, i+2, getRelPrimAt(i), "action verb");
          } else if (isVerb(i+2)) {
            createRelationString("gol", getRelPrimAt(i), i, getRelPrimAt(i+2), i+2);
            replaceClassifiedWords(i, i+2, getRelPrimAt(i), "action verb");
          }
        }
      }
    }

    private int isGoBackTrue(boolean goBack, int i) {
      if (goBack) {
        i--;
        goBack = false;
      }
      return i;
    }

    private int analyzeVerbosIntransitivos(int i) {
      // Verbos intransitivos
      if (classif.isActionVerb(getPrimAt(i)) && !getWordAt(i).contains("subjunctive")) {
        if (sentence.size() > 2 && (isPunctuation(i+1) || getWordAt(i+1).contains("adverb"))) {
          createRelationString("agt", getRelPrimAt(i), i, getRelPrimAt(i-1), i-1);
          replaceClassifiedWords(i-1, i, getRelPrimAt(i), "action verb");
          i--;
        }
      }
      return i;
    }

    private boolean analyzeSujeitoVerbo(int i, boolean goBack) {
      if (getWordAt(i).contains("SUBJECT") ||
        getWordAt(i).contains("noun") ||
        getWordAt(i).contains("pronoun") ||
        getWordAt(i).contains("proper noun")) {
        // Exemplo: "Eu jogo."
        if ((classif.isActionVerb(getPrimAt(i+1)) || classif.isFeelingVerb(getPrimAt(i+1)))
          && !getWordAt(i+1).contains("subjunctive")) {
          createRelationString("agt", getRelPrimAt(i+1), i+1, getRelPrimAt(i), i);
```

```
            replaceClassifiedWords(i, i+1, getRelPrimAt(i+1), "action verb");
            goBack = true;
        }
        // Exemplo: "Eu estou."
        if (classif.isStateVerb(getPrimAt(i+1))) {
            if (getWordAt(i+2).contains("adjective")) {
                createRelationString("aoj", getRelPrimAt(i+2), i+2, getRelPrimAt(i), i);
                replaceClassifiedWords(i, i+2, getRelPrimAt(i+2), "oracao com state verb");
            } else if (getWordAt(i+2).contains("noun place")) {
                createRelationString("plc", getRelPrimAt(i), i, getRelPrimAt(i+2), i+2);
                replaceClassifiedWords(i, i+2, getRelPrimAt(i+2), "oracao com state verb");
            }
        }
        if (classif.isFeelingVerb(getPrimAt(i))) {
                createRelationString("aoj", getRelPrimAt(i), i, getRelPrimAt(i-1), i-1);
                replaceClassifiedWords(i-1, i, getRelPrimAt(i),"feeling verb");
        }
    }
    return goBack;
  }

  private void analyzeObjeto(int i) {
    //Example: "Compro doces."
    if (classif.isActionVerb(getPrimAt(i)) && !getWordAt(i).contains("subjunctive")) {
        if (isVerb(i+1) && !classif.isStateVerb(getPrimAt(i+1))) {
            createRelationString("obj", getRelPrimAt(i), i, getRelPrimAt(i+1), i+1);
            replaceClassifiedWords(i, i+1, getRelPrimAt(i+1), "verb");
        }
        if (isNoun(i+1)) {
            createRelationString("obj", getRelPrimAt(i), i, getRelPrimAt(i+1), i+1);
            replaceClassifiedWords(i, i+1, getRelPrimAt(i), "verb");
        }
    }
  }
}
```

APÊNDICE D – *PreAnalyzer*

```
/*
 * @author Ricardo Pereira Beato Junior
 * @date 09/Dec/2012
 */

package hellodrools;

import java.util.ArrayList;

public class PreAnalyze {

   // Acesso ao dicionario
   public WordClassification classif = new WordClassification("../HelloDrools/build/"
                                    + "classes/hellodrools/Dictionary");

   // Instancia para uso dos metodos de analise de AnalyzeSentence
   AnalyzeSentence as;

   // Permite que a contagem de palavras regrida no metodo de atualizacao
   // das primitivas, caso haja remocao de uma palavra da sentenca
   private boolean goBack = false;


   public void preSetRules(ArrayList<ArrayList> sentence) {
// Loops para testes
//      for (int a = 0; a < AnalyzeSentence.sentence.size(); a++)
//         System.out.println(AnalyzeSentence.getWordAt(a) + "00000");
      insertUnknownPrimitives(sentence);
//      for (int a = 0; a < AnalyzeSentence.sentence.size(); a++)
//         System.out.println(AnalyzeSentence.getWordAt(a) + "11111");
      gatherKnownWords(sentence);
//      for (int a = 0; a < AnalyzeSentence.sentence.size(); a++)
//         System.out.println(AnalyzeSentence.getWordAt(a) + "22222");
      addSubject(sentence); // Sujeito Oculto
//      for (int a = 0; a < AnalyzeSentence.sentence.size(); a++)
```

```
//          System.out.println(AnalyzeSentence.getWordAt(a) + "33333");
        defineVerbType(sentence);
//      for (int a = 0; a < AnalyzeSentence.sentence.size(); a++)
//          System.out.println(AnalyzeSentence.getWordAt(a) + "44444");
        updatePrimitives(sentence);
//      for (int a = 0; a < AnalyzeSentence.sentence.size(); a++)
//          System.out.println(AnalyzeSentence.getWordAt(a) + "55555");
    }

    // Quando ha palavras desconhecidas, o cogroo deixa o campo de
    // primitiva em branco. Este metodo copia a palavra original
    // no campo da primitiva
    public void insertUnknownPrimitives(ArrayList<ArrayList> sentence) {
        for (int i = 0; i < sentence.size(); i++)
            if (as.getPrimAt(i).equals("")) {
                sentence.get(i).set(1, sentence.get(i).get(0));
                sentence.get(i).set(4, sentence.get(i).get(0));
            }
    }

    // Todas as palavras sao tratadas em separado, mas "de vez em quando", "Sao paulo"
    // e outras expressoes deveriam ser encaradas como uma palavra soh.
    public void gatherKnownWords(ArrayList<ArrayList> sentence) {
        for (int i = 0; i < sentence.size(); i++) {
            for (int j = sentence.size() - 1; j > i; j--) {
                StringBuilder subString = new StringBuilder();
                for (int k = i; k <= j; k++) {
                    if (k < j) subString.append(as.getPrimAt(k) + " ");
                    if (k == j) subString.append(as.getPrimAt(k));
                }
                if (classif.isWordInDictionary(subString.toString())) {
                    as.replaceClassifiedWords(i, j, subString.toString(), "indef");
                }
            }
        }
    }

    // Sujeito Oculto
```

```
private void addSubject(ArrayList<ArrayList> sentence) {
   for (int j = 0; j < AnalyzeSentence.sentence.size(); j++) {
      int pos = j;
      if (as.isVerb(j) && !as.getWordAt(j).contains("gerund") &&
        !as.getWordAt(j).contains("infinitive") && !as.getWordAt(j).contains("subjunctive")) {

        // Normalmente o sujeito fica imediatamente antes do verbo
        // Excecao: quando ha adverbios entre sujeito e verbo
        int a;
        if (as.getPrimAt(j-1).equals("não") || as.getPrimAt(j-1).equals("sempre") ||
           as.getPrimAt(j-1).equals("nunca")) {
           if (as.getWordAt(j-2).contains("determiner")) {
              a = j-3;
              pos = j-2;
           }
           else {
              a = j-2;
              pos = j-1;
           }
        } else a = j-1;

        if (!as.getWordAt(a).contains("proper noun") && !as.getWordAt(a).contains("noun") &&
           !as.getWordAt(a).contains("pronoun") && !as.getWordAt(a).contains("SUBJECT") &&
           !as.getWordAt(a).contains("quem")) {
              String subject = findSubject(j, sentence);
              ArrayList<String> newWord = new ArrayList<>();
              newWord.add(subject);
              newWord.add(subject.toLowerCase());
              newWord.add("SUBJECT");
              newWord.add("");
              newWord.add(subject.toLowerCase());
              sentence.add(pos, newWord);
              j++;
        }
      }
   }
}
```

```
private String findSubject(int j, ArrayList<ArrayList> sentence) {
   String subject = null;
   String classe = sentence.get(j).get(2).toString();
   if (classe.contains("singular") && classe.contains("first")) subject = "Eu";
   else {
      // Ignorando a ambiguidade entre "ele", "ela" e "voce"
      if (classe.contains("singular") && classe.contains("third")) subject = "Você";
      else {
         if (classe.contains("plural") && classe.contains("first")) subject = "Nós";
         // Ignorando a ambiguidade entre "eles", "elas" e "voces"
         else if (classe.contains("third") && classe.contains("plural")) subject = "Eles";
         }
   }
   return subject;
}

// Define se dado verbo eh de estado, acao ou sentimento
public void defineVerbType(ArrayList<ArrayList> sentence) {
   for (int i = 0; i < sentence.size(); i++) {
      String word = as.getPrimAt(i);
      if (as.isVerb(i)) {
         if (classif.isStateVerb(word))
            sentence.get(i).set(3, "state verb");
         else if(classif.isFeelingVerb(as.getPrimAt(i))) sentence.get(i).set(3, "feeling verb");
         else sentence.get(i).set(3, "action verb");
      }
   }
}

// Atualiza as primitivas acrescentando etiquetas como @pl, @past, @future,
// @def, @indef
// Alem disso, simplifica sintagmas verbais e nominais com o objetivo de
// tornar a interpretacao no sistema de regras mais simples
private void updatePrimitives(ArrayList<ArrayList> sentence) {
   for (int i = 0; i < sentence.size(); i++) {
      i = goBackAfterRemotion(i);
      addNotTag(i, sentence);
      addPastTag(i, sentence);
```

```
        addPluralTag(i, sentence);
        addFutureTag1(i,sentence);
        addFutureTag2(i, sentence);
        i = addDeterminerTag(i, sentence);
        removePreposicaoDe(i, sentence);
        reduceLocucaoVerbal(i);
    }
}

private int goBackAfterRemotion(int i) {
    if (goBack) {
        i--;
        goBack = false;
    }
    return i;
}

private void addNotTag(int i, ArrayList<ArrayList> sentence) {
    // Negacao
    if (as.getPrimAt(i).equals("não")) {
        if (sentence.size() > i+1) {
        if ((as.getWordAt(i+1).contains("estar") && as.getWordAt(i+2).contains("gerund")) ||
            (as.getPrimAt(i+1).equals("ir") && as.getWordAt(i+2).contains("infinitive")) ||
           ((as.getWordAt(i+1).contains("estar") || as.getWordAt(i+2).contains("ser")) &&
            as.getWordAt(i+2).contains("adjective"))) {
           sentence.get(i+2).set(4, as.getRelPrimAt(i+2) + ".@not");
        } else sentence.get(i+1).set(4, as.getRelPrimAt(i+1) + ".@not");
        sentence.remove(i);
        goBack = true;
    }
    goBack = false;
    }
}

private void addPastTag(int i, ArrayList<ArrayList> sentence) {
    // .@past
    if (as.getWordAt(i).contains("preterito")) {
        if (classif.isStateVerb(as.getPrimAt(i))) {
```

```
            sentence.get(i+1).set(4, as.getRelPrimAt(i+1) + ".@past");
        } else sentence.get(i).set(4, as.getRelPrimAt(i) + ".@past");
    }
}

private void addPluralTag(int i, ArrayList<ArrayList> sentence) {
    // .@pl
    if (as.getWordAt(i).contains("plural") && !as.getWordAt(i).contains("proper noun") &&
        !as.isVerb(i) && possessivePerson(i) == null &&
        !as.getWordAt(i).contains("personal pronoun")) {
            sentence.get(i).set(4, as.getRelPrimAt(i) + ".@pl");
    }
}

private void addFutureTag1(int i, ArrayList<ArrayList> sentence) {
    // .@future
    if (as.getWordAt(i).contains("future") && !as.getWordAt(i).contains(".@future")) {
        sentence.get(i).set(4, as.getRelPrimAt(i) + ".@future");
    }
}

private void addFutureTag2(int i, ArrayList<ArrayList> sentence) {
    // Futuro com "ir" -> "Vou comprar uma camiseta."
    if (as.getPrimAt(i).equals("ir")) {
        if (as.getWordAt(i+1).contains("infinitive")) {
            sentence.get(i+1).set(4, as.getRelPrimAt(i+1) + ".@future");
            sentence.remove(i);
            goBack = true;
        }
    }
}

private int addDeterminerTag(int i, ArrayList<ArrayList> sentence) {
    // Define se o substantivo ao qual um pronome ou artigo se referem
    // eh determinado ou indeterminado
    if (classif.isDeterminer(as.getPrimAt(i))){
        String def = null;
        if (classif.isIndef(as.getPrimAt(i))) def = ".@indef";
```

```
      if (classif.isDef(as.getPrimAt(i))) def = ".@def";
      //noun, pronoun ou proper noun
      if (as.getWordAt(i+1).contains("noun")) {
         sentence.get(i+1).set(4, as.getRelPrimAt(i+1) + def);
         sentence.remove(i);
      } else
      // "Uma bela arquitetura"
      if (as.getWordAt(i+1).contains("adjective")) {
         System.out.println(as.getPrimAt(i+1));
         System.out.println(as.getPrimAt(i+2));
         if (as.isNoun(i+2)) {
            sentence.get(i+2).set(4, as.getRelPrimAt(i+2) + def);
            sentence.remove(i);
         }
      } else
      if (classif.isPossessive(as.getPrimAt(i+1))) {
         sentence.get(i+2).set(4, as.getRelPrimAt(i+2) + def);
         sentence.remove(i);
      }
      i--;
   }
   return i;
}

private void removePreposicaoDe(int i, ArrayList<ArrayList> sentence) {
   // Eliminacao da preposicao "de" posposta a verbos somente por questao
   // de regencia - "gostar", "precisar"
   if (as.getPrimAt(i).equals("de")) {
      String pre01 = as.getPrimAt(i-1);
      String pre02 = as.getPrimAt(i-2);
      if (pre01.equals("gostar") || pre02.equals("gostar") ||
         pre01.equals("precisar") || pre02.equals("precisar") ||
         pre01.equals("sair") || pre02.equals("sair")) {
         sentence.remove(i);
         goBack = true;
      }
   }
   goBack = false;
```

```
    }

    private void reduceLocucaoVerbal(int i) {
        // Locucoes verbais com saber - .@ability - e poder - .@permission
        if (as.isVerb(i) && as.getWordAt(i+1).contains("infinitive")) {
        // "Poder"
            if (as.getPrimAt(i).contains("poder")) {
                if (as.getWordAt(i).contains("@not")) {
                    as.replaceClassifiedWords(i, i+1, as.getRelPrimAt(i+1) + ".@permission"
                                                + ".@not", "verb");
                } else as.replaceClassifiedWords(i, i+1, as.getRelPrimAt(i+1) + ".@permission", "verb");
                goBack = true;
            }
            // "Saber"
            if (as.getPrimAt(i).equals("saber")) {
                if (as.getRelPrimAt(i).contains("@not")) {
                    as.replaceClassifiedWords(i, i+1, as.getRelPrimAt(i+1) + ".@ability" + ".@not", "verb");
                } else as.replaceClassifiedWords(i, i+1, as.getRelPrimAt(i+1) + ".@ability", "verb");
                goBack = true;
            }
        }
    }

    private String possessivePerson(int i) {
        String person = null;
        String prim = as.getPrimAt(i);
        if (classif.isPosEu(prim)) person = "eu";
        if (classif.isPosVoce(prim)) person = "você";
        if (classif.isPosTu(prim)) person = "tu";
        if (classif.isPosNos(prim)) person = "nós";
        if (classif.isPosEle(prim)) person = "ele";
        if (classif.isPosEla(prim)) person = "ela";
        if (classif.isPosEles(prim)) person = "eles";
        if (classif.isPosElas(prim)) person = "elas";
        return person;
    }
}
```

APÊNDICE E – *UnlOutput*

```
/*
 * @author Ricardo Pereira Beato Junior
 * @date 02/Oct/2012
 */

package hellodrools;

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

public class UnlOutput {

   // Lista com Strings prontas de UNL
   // O primeiro elemento serah a string "{unl}", e o ultimo serah "{/unl}"
   public static List<String> output;

   // Construtor: inicializa a lista com as strings de saida do projeto
   public UnlOutput() {
      UnlOutput.output  = new ArrayList<>();
   }

   // Passa a lista para a impressao no formato final
   public static List<String> getOutput() {
      return UnlOutput.output;
   }

   // Dadas as entradas enviadas pelo arquivo de regras (tipo de relacao,
   // primeira palavra e segunda palavra da relacao), cria uma string com
   // o padrao UNL e a adiciona na lista de saida
   public static void createRelationString(String relationType, String word01, String word02) {
      word01 = word01.replace("_", " ");
      word02 = word02.replace("_", " ");
      UnlOutput.output.add("   " + relationType + "(" + word01 + ", " + word02 + ")");
   }
```

```
    // Metodo chamado ao fim da execucao pela classe CogrooToUnlBase para
    // imprimir a lista de saida no formato UNL
    public static void printOutput() {
        for (int i = 0; i < getOutput().size(); i++) {
            System.out.println(getOutput().get(i).toString());
        }
    }
}
```

APÊNDICE F – *WordClassification*

```
/*
 * @author Ricardo Pereira Beato Junior
 * @date 09/Dec/2012
 */

package hellodrools;

import java.io.IOException;
import java.util.ArrayList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

// Tipo adverbio
class Adverbs {
   private Expressions frequencia;
   private Expressions lugar;
   private Expressions modo;
   private Expressions negacao;
   private Expressions tempo;

   public Adverbs() {
      this.frequencia = new Expressions("frequencia");
      this.lugar = new Expressions("lugar");
      this.modo = new Expressions("modo");
      this.negacao = new Expressions("negacao");
      this.tempo = new Expressions("tempo");
   }
```

```
public void initialize(NodeList adverbsTags) {
   if (adverbsTags == null || adverbsTags.getLength() == 0) return;
   Element e = (Element) adverbsTags.item(0);
   this.frequencia.initialize(e.getElementsByTagName("frequencia"));
   this.lugar.initialize(e.getElementsByTagName("lugar"));
   this.modo.initialize(e.getElementsByTagName("modo"));
   this.negacao.initialize(e.getElementsByTagName("negacao"));
   this.tempo.initialize(e.getElementsByTagName("tempo"));
}

public void buildExpressions(StringBuilder b) {
   if (this.frequencia.isEmpty() && this.lugar.isEmpty() && this.modo.isEmpty() &&
            this.negacao.isEmpty() && this.tempo.isEmpty()) return;
   b.append("adverbs:");
   this.frequencia.buildExpressions(b);
   this.lugar.buildExpressions(b);
   this.modo.buildExpressions(b);
   this.negacao.buildExpressions(b);
   this.tempo.buildExpressions(b);
}

public boolean isAdverb(String string) {
   return this.frequencia.containsExpression(string) ||
   this.lugar.containsExpression(string) ||
   this.modo.containsExpression(string) ||
   this.negacao.containsExpression(string) ||
   this.tempo.containsExpression(string);
}

public boolean isAdverbTempo(String string) {
   return this.tempo.containsExpression(string);
}

public boolean isAdverbFreq(String string) {
   return this.frequencia.containsExpression(string);
}

public boolean isAdverbLugar(String string) {
```

```
        return this.lugar.containsExpression(string);
    }
}

class Determiners {
    private Expressions def;
    private Expressions indef;

    public Determiners() {
        this.def = new Expressions("def");
        this.indef = new Expressions("indef");
    }

    public void initialize(NodeList adverbsTags) {
        if (adverbsTags == null || adverbsTags.getLength() == 0) return;
        Element e = (Element) adverbsTags.item(0);
        this.def.initialize(e.getElementsByTagName("def"));
        this.indef.initialize(e.getElementsByTagName("indef"));
    }

    public void buildExpressions(StringBuilder b) {
        if (this.def.isEmpty() && this.indef.isEmpty()) return;
        b.append("determiners:");
        this.def.buildExpressions(b);
        this.indef.buildExpressions(b);
    }

    public boolean isDeterminer(String string) {
        return this.def.containsExpression(string) ||
               this.indef.containsExpression(string);
    }

    public boolean isDef(String string) {
        return this.def.containsExpression(string);
    }

    public boolean isIndef(String string) {
        return this.indef.containsExpression(string);
```

```
    }

}


class Possessives {
    private Expressions eu;
    private Expressions tu;
    private Expressions voce;
    private Expressions ele;
    private Expressions ela;
    private Expressions nos;
    private Expressions eles;
    private Expressions elas;

    public Possessives() {
        this.eu = new Expressions("eu");
        this.tu = new Expressions("tu");
        this.voce = new Expressions("voce");
        this.ele = new Expressions("ele");
        this.ela = new Expressions("ela");
        this.nos = new Expressions("nos");
        this.eles = new Expressions("eles");
        this.elas = new Expressions("elas");
    }

    public void initialize(NodeList adverbsTags) {
        if (adverbsTags == null || adverbsTags.getLength() == 0) return;
        Element e = (Element) adverbsTags.item(0);
        this.eu.initialize(e.getElementsByTagName("eu"));
        this.tu.initialize(e.getElementsByTagName("tu"));
        this.voce.initialize(e.getElementsByTagName("voce"));
        this.ele.initialize(e.getElementsByTagName("ele"));
        this.ela.initialize(e.getElementsByTagName("ela"));
        this.nos.initialize(e.getElementsByTagName("nos"));
        this.eles.initialize(e.getElementsByTagName("eles"));
        this.elas.initialize(e.getElementsByTagName("elas"));
    }
```

```
public void buildExpressions(StringBuilder b) {
   if (this.eu.isEmpty() && this.tu.isEmpty() && this.voce.isEmpty() &&
      this.ele.isEmpty() && this.ela.isEmpty() && this.nos.isEmpty() &&
      this.eles.isEmpty() && this.elas.isEmpty()) return;
   b.append("possessives:");
   this.eu.buildExpressions(b);
   this.tu.buildExpressions(b);
   this.voce.buildExpressions(b);
   this.ele.buildExpressions(b);
   this.ela.buildExpressions(b);
   this.nos.buildExpressions(b);
   this.eles.buildExpressions(b);
   this.elas.buildExpressions(b);
}

public boolean isPossessivo(String string) {
   return this.eu.containsExpression(string) ||
   this.tu.containsExpression(string) ||
   this.voce.containsExpression(string) ||
   this.ele.containsExpression(string) ||
   this.ela.containsExpression(string) ||
   this.nos.containsExpression(string) ||
   this.eles.containsExpression(string) ||
   this.elas.containsExpression(string);
}

public boolean isPosEu(String string) {
   return this.eu.containsExpression(string);
}

public boolean isPosTu(String string) {
   return this.tu.containsExpression(string);
}

public boolean isPosVoce(String string) {
   return this.voce.containsExpression(string);
}
```

```
    public boolean isPosEle(String string) {
        return this.ele.containsExpression(string);
    }

    public boolean isPosEla(String string) {
        return this.ela.containsExpression(string);
    }

    public boolean isPosNos(String string) {
        return this.nos.containsExpression(string);
    }

    public boolean isPosEles(String string) {
        return this.eles.containsExpression(string);
    }

    public boolean isPosElas(String string) {
        return this.elas.containsExpression(string);
    }

    public boolean isPossessive(String string) {
        return this.eu.containsExpression(string) ||
                this.tu.containsExpression(string) ||
                this.voce.containsExpression(string) ||
                this.ele.containsExpression(string) ||
                this.ela.containsExpression(string) ||
                this.nos.containsExpression(string) ||
                this.eles.containsExpression(string) ||
                this.elas.containsExpression(string);
    }
}

class Verbs {
    private Expressions state;
    private Expressions feeling;
    private Expressions action;
```

```
public Verbs() {
   this.state = new Expressions("state");
   this.feeling = new Expressions("feeling");
   this.action = new Expressions("action");
}

public void initialize(NodeList expressionsTags) {
   if (expressionsTags == null || expressionsTags.getLength() == 0) return;
   Element e = (Element) expressionsTags.item(0);
   this.state.initialize(e.getElementsByTagName("state"));
   this.feeling.initialize(e.getElementsByTagName("feeling"));
   this.action.initialize(e.getElementsByTagName("action"));
}

public void buildExpressions(StringBuilder b) {
   if (state.isEmpty() && feeling.isEmpty() && action.isEmpty()) return;
   b.append("verbs:");
   this.state.buildExpressions(b);
   this.feeling.buildExpressions(b);
   this.action.buildExpressions(b);
}

public boolean isVerb(String string) {
   return this.state.containsExpression(string) ||
   this.feeling.containsExpression(string) ||
   this.action.containsExpression(string);
}

public boolean isFeelingVerb(String string) {
   return this.feeling.containsExpression(string);
}

public boolean isStateVerb(String string) {
   return this.state.containsExpression(string);
}

public boolean isActionVerb(String string) {
   return this.action.containsExpression(string);
```

```
    }

}

class Expressions {
    private ArrayList<String> expressions;
    private String name;

    public Expressions(String name) {
        this.expressions = new ArrayList<>();
        this.name = name;
    }

    public boolean containsExpression(String exp) {
        for (String expression : expressions) {
            if (expression.equals(exp)) return true;
        }
        return false;
    }

    public void initialize(NodeList expressionsTags) {
        if (expressionsTags == null || expressionsTags.getLength() == 0) return;
        Element e = (Element) expressionsTags.item(0);
        for (Node exp = e.getFirstChild(); exp != null; exp = exp.getNextSibling()) {
            if (exp.getNodeType() != Node.ELEMENT_NODE) continue;
            this.expressions.add(exp.getTextContent());
        }
    }

    public void buildExpressions(StringBuilder ret) {
        if (this.expressions != null && this.expressions.size() > 0) {
            ret.append(this.name + ":");
            for (String exp : this.expressions) {
                ret.append("'" + exp + "'");
            }
        }
    }
```

```
    public boolean isEmpty() {
        return this.expressions == null || this.expressions.size() == 0;
    }
}

public class WordClassification {
    private Expressions expressions;
    private Verbs verbs;
    private Adverbs adverbs;
    private Expressions transportation;
    private Determiners determiners;
    private Possessives possessives;

    public WordClassification(String xml) {
        this.expressions = new Expressions("expressions");
        this.verbs = new Verbs();
        this.adverbs = new Adverbs();
        this.determiners = new Determiners();
        this.possessives = new Possessives();
        this.transportation = new Expressions("transportation");
        Element element = readXML(xml);
        if (element == null) return;
        this.expressions.initialize(element.getElementsByTagName("expressions"));
        this.verbs.initialize(element.getElementsByTagName("verbs"));
        this.adverbs.initialize(element.getElementsByTagName("adverbs"));
        this.determiners.initialize(element.getElementsByTagName("determiners"));
        this.possessives.initialize(element.getElementsByTagName("possessives"));
        this.transportation.initialize(element.getElementsByTagName("transportation"));
    }

    private Element readXML(String xml) {
        DocumentBuilderFactory f = DocumentBuilderFactory.newInstance();
        try {
            DocumentBuilder b = f.newDocumentBuilder();
            Document d = b.parse(xml);
            return d.getDocumentElement();
        } catch (SAXException e) {
            System.out.println("SAX Exception: " + e);
```

```
    } catch (IOException e) {
        System.out.println("IO Exception: " + e);
    } catch (ParserConfigurationException e) {
        System.out.println("Parser Configuration Exception: " + e);
    }
    return null;
}

@Override
public String toString() {
    StringBuilder ret = new StringBuilder();
    this.expressions.buildExpressions(ret);
    this.verbs.buildExpressions(ret);
    this.adverbs.buildExpressions(ret);
    this.transportation.buildExpressions(ret);
    this.determiners.buildExpressions(ret);
    this.possessives.buildExpressions(ret);
    return ret.toString();
}

public boolean isVerb(String string) {
    return this.verbs.isVerb(string);
}

public boolean isFeelingVerb(String string) {
    return this.verbs.isFeelingVerb(string);
}

public boolean isStateVerb(String string) {
    return this.verbs.isStateVerb(string);
}

public boolean isActionVerb(String string) {
    return !this.verbs.isFeelingVerb(string) &&
        !this.verbs.isStateVerb(string);
}

public boolean isAdverb(String string) {
```

```
    return this.adverbs.isAdverb(string);
}

public boolean isAdverbTempo(String string) {
    return this.adverbs.isAdverbTempo(string);
}

public boolean isAdverbFreq(String string) {
    return this.adverbs.isAdverbFreq(string);
}

public boolean isAdverbLugar(String string) {
    return this.adverbs.isAdverbLugar(string);
}

public boolean isDeterminer(String string) {
    return this.determiners.isDeterminer(string);
}

public boolean isDef(String string) {
    return this.determiners.isDef(string);
}

public boolean isIndef(String string) {
    return this.determiners.isIndef(string);
}

public boolean isPossessive(String string) {
    return this.possessives.isPossessive(string);
}

public boolean isPosEu(String string) {
    return this.possessives.isPosEu(string);
}

public boolean isPosTu(String string) {
    return this.possessives.isPosTu(string);
}
```

```
    public boolean isPosVoce(String string) {
        return this.possessives.isPosVoce(string);
    }

    public boolean isPosEle(String string) {
        return this.possessives.isPosEle(string);
    }

    public boolean isPosEla(String string) {
        return this.possessives.isPosEla(string);
    }

    public boolean isPosNos(String string) {
        return this.possessives.isPosNos(string);
    }

    public boolean isPosEles(String string) {
        return this.possessives.isPosEles(string);
    }

    public boolean isPosElas(String string) {
        return this.possessives.isPosElas(string);
    }

    public boolean isTransportation(String string) {
        return this.transportation.containsExpression(string);
    }

    public boolean isWordInDictionary(String string) {
        return isVerb(string) || isAdverb(string) || isDeterminer(string) ||
                isPossessive(string) || isTransportation(string);
    }
}
```

APÊNDICE G – *Dictionary*

```
<Classes>

  <adjectives>
    <expression>mais ou menos</expression>
  </adjectives>

  <adverbs>
    <frequencia>
      <expression>a as vezes</expression>
      <expression>de vez em quando</expression>
      <expression>dia sim dia não</expression>
      <expression>jamais</expression>
      <expression>nunca</expression>
      <expression>sempre</expression>
      <expression>todo dia</expression>
    </frequencia>
    <lugar>
      <expression>aqui</expression>
      <expression>em cima de</expression>
      <expression>embaixo</expression>
      <expression>lá</expression>
      <expression>longe</expression>
      <expression>perto</expression>
      <expression>sobre</expression>
    </lugar>
    <modo>
      <expression>devagar</expression>
      <expression>como um</expression>
      <expression>lentamente</expression>
      <expression>rapidamente</expression>
      <expression>rápido</expression>
    </modo>
    <negacao>
      <expression>não</expression>
    </negacao>
```

```
    <posicao>
      <expression>em cima de</expression>
      <expression>de o lado de</expression>
      <expression>em cima de</expression>
    </posicao>
    <tempo>
      <expression>agora</expression>
      <expression>amanhã</expression>
      <expression>anteontem</expression>
      <expression>antes</expression>
      <expression>cedo</expression>
      <expression>depois</expression>
      <expression>hoje</expression>
      <expression>imediatamente</expression>
      <expression>já</expression>
      <expression>ontem</expression>
      <expression>tarde</expression>
    </tempo>
  </adverbs>

  <determiners>
    <def>
      <expression>a</expression>
      <expression>as</expression>
      <expression>o</expression>
      <expression>os</expression>
      <expression>este</expression>
      <expression>estes</expression>
      <expression>esse</expression>
      <expression>esses</expression>
      <expression>aquele</expression>
      <expression>aqueles</expression>
      <expression>aquela</expression>
      <expression>aquelas</expression>
    </def>
    <indef>
      <expression>um</expression>
      <expression>uma</expression>
```

```
      <expression>algum</expression>
      <expression>alguma</expression>
      <expression>uns</expression>
      <expression>umas</expression>
      <expression>alguns</expression>
      <expression>algumas</expression>
    </indef>
  </determiners>

  <possessives>
    <eu>
      <expression>meu</expression>
      <expression>meus</expression>
      <expression>minha</expression>
      <expression>minhas</expression>
    </eu>
    <tu>
      <expression>teu</expression>
      <expression>tua</expression>
      <expression>teus</expression>
      <expression>tuas</expression>
    </tu>
    <voce>
      <expression>seu</expression>
      <expression>sua</expression>
      <expression>seus</expression>
      <expression>suas</expression>
    </voce>
    <nos>
      <expression>nosso</expression>
      <expression>nossos</expression>
      <expression>nossa</expression>
      <expression>nossas</expression>
    </nos>
    <ele>
      <expression>dele</expression>
      <expression>de ele</expression>
    </ele>
```

```
    <ela>
      <expression>dela</expression>
      <expression>de ela</expression>
    </ela>
    <eles>
      <expression>deles</expression>
      <expression>de eles</expression>
    </eles>
    <elas>
      <expression>delas</expression>
      <expression>de elas</expression>
    </elas>
</possessives>

<verbs>
    <feeling>
      <expression>adorar</expression>
      <expression>amar</expression>
      <expression>curtir</expression>
      <expression>detestar</expression>
      <expression>deprezar</expression>
      <expression>gostar</expression>
      <expression>idolatrar</expression>
      <expression>odiar</expression>
      <expression>temer</expression>
    </feeling>
    <state>
      <expression>ser</expression>
      <expression>estar</expression>
    </state>
    <action>
    </action>
</verbs>

<transportation>
      <expression>avião</expression>
      <expression>barco</expression>
      <expression>bicicleta</expression>
```

```
        <expression>bonde</expression>
        <expression>bote</expression>
        <expression>caminhão</expression>
        <expression>canoa</expression>
        <expression>carona</expression>
        <expression>carro</expression>
        <expression>carroça</expression>
        <expression>elevador</expression>
        <expression>escada</expression>
        <expression>escuna</expression>
        <expression>helicóptero</expression>
        <expression>jangada</expression>
        <expression>lotação</expression>
        <expression>ônibus</expression>
        <expression>metrô</expression>
        <expression>navio</expression>
        <expression>táxi</expression>
        <expression>trem</expression>
        <expression>trólebus</expression>
        <expression>veleiro</expression>
    </transportation>

</Classes>
```