Trabalho Final de Curso

Aquisição de Termos Automática

Relatório

Orientador:

Nuno Mamede

Aluna:

Joana Lúcio Paulo, 44048

(Regime de Mestrado Integrado)

Lisboa, 11 de Julho de 2001

Índice

1. Introdução

O desenvolvimento de terminologias tornou-se nos últimos anos uma actividade importante. As terminologias são úteis para todos os tipos de trabalho com um domínio específico. Com o aumento de *corpus* em formato electrónico e a necessidade de extrair informação deles, os termos desempenham um papel fundamental, quando os textos são especializados.

As terminologias são necessárias aos tradutores. Todas as aplicações no campo do Processamento da Língua Natural que querem processar textos especializados num certo domínio ganharão precisão se usarem como base uma boa terminologia. Por exemplo, para indexação automática de textos, o reconhecimento de termos tem um papel central. Para a tradução automática, ou para tradução assistida por computador, são necessárias terminologias bilingues quando o texto a traduzir é especializado.

Além disso, de um modo geral, o conhecimento dos termos presentes num texto pode melhorar os resultados obtidos por uma qualquer análise automática de textos. O desenvolvimento de terminologias é uma tarefa contínua, uma vez que as terminologias mudam constantemente, pelo que, além de criar novas terminologias, os terminologistas têm a responsabilidade de manter as existentes. Fazer esse trabalho manualmente pode ser bastante aborrecido e demorado.

1.1. Trabalho a desenvolver

Neste projecto pretende-se desenvolver uma ferramenta que facilite o trabalho dos linguistas quando têm que extrair manualmente termos de um texto. Quer-se construir um extractor semi-automático de termos que faça parte desse trabalho.

Considerando que os termos são unidades linguísticas especiais, que obedecem a certas restrições linguísticas – restrições sintácticas quando esses termos são compostos por mais que uma unidade léxica (termos multi-léxicos ou termos compostos) e restrições morfológicas, quando esses termos são compostos por apenas uma unidade léxica (termos monoléxicos, ou termos simples) – e considerando que hoje em dia estão mais facilmente disponíveis grandes *corpora* especializados, propõe-se o desenvolvimento de uma ferramenta que seja capaz de extrair desses *corpora* uma lista de termos.

1.1.1. Tipos de Termos

Neste trabalho vão-se considerar dois tipos de termos: simples e compostos.

Os termos simples são constituídos por apenas uma unidade léxica. A complexidade da sua detecção reside na ambiguidade que podem ter por serem aparentemente iguais a qualquer outro item léxico.

Os termos compostos são constituídos por mais que uma unidade léxica. Têm a vantagem de serem menos ambíguos do que os termos simples, mas têm a desvantagem de requererem um estudo sintáctico para se verificar se a combinação das unidades léxicas deste termo constituem uma estrutura sintáctica.

1.2. Objectivo Final

O objectivo final é a construção de um programa para ambiente Windows[1]. Este programa receberá como entrada um *corpus* especializado e, depois de um tempo de processamento razoável, devolve uma lista dos candidatos a termos (simples e compostos) do domínio que aparecem no *corpus*.

Este projecto tem duas vertentes:

1. Aquisição automática de candidatos a termos simples e compostos de um *corpus* especializado, sem qualquer definição prévia da parte dos terminologistas. O programa extrairá do texto todas as sequências que correspondam às estruturas sintácticas pré-definidas, os candidatos a termos compostos, e devolverá uma lista de termos simples. Neste caso espera-se que todos os termos estejam presentes na lista, mas, algumas expressões que não correspondem a termos também serão extraídas. O terminologista terá que as eliminar manualmente no fim do processo.
2. Aquisição de termos simples e compostos guiada pelo utilizador, onde os terminologistas terão a possibilidade de:

   - Definir estruturas sintácticas para os termos compostos, diferentes das pré-definidas e eliminar a utilização de algumas (ou mesmo todas) as pré-definidas. Se se quiser, podem incluir-se alguns lemas nas suas construções misturando informação léxica e sintáctica.
   - Introduzir nestas estruturas sintácticas itens léxicos.

   A ideia deste modo de funcionamento é permitir ao linguista a introdução do seu próprio conhecimento no sistema, aumentando a produtividade e o desempenho do processo e a qualidade dos resultados obtidos.

   Deste modo, podem-se refinar os resultados obtidos no modo automático e alguns parâmetros podem ser controlados pelo utilizador, dando à ferramenta final alguma flexibilidade.

   Este modo é importante por várias razões. Se o utilizador acha que os termos no domínio em questão têm alguma construção sintáctica especial e muito frequente (ou que possa ajudar especialmente a tarefa da ferramenta) e se esta construção não foi definida previamente[2], então, poderá adicioná-la. Pelo contrário, se uma das estruturas pré-definidas não é útil no domínio em causa, pode desactivá-la, ignorando-a.

   Se um terminologista sabe que um determinado termo é muito importante para o seu domínio, poderá procurar todas as estruturas (pré-definidas, ou não) que contenham o lema da palavra. Esta possibilidade vai permitir-lhe organizar a sua terminologia. Isto porque queremos que o programa que vai ser construído tenha a flexibilidade de se adaptar a qualquer domínio.

---

[1] Se as restrições temporais o permitirem poder-se-á considerar a hipótese de se desenharem interfaces para outros sistemas operativos como o Macintosh, o Unix ou o Linux.

[2] Este teste é importante para evitar a duplicação de testes. Independentemente de se repetir, ou não, um teste, o resultado será sempre o mesmo.

1.3. Objectivo Inicial: PAsMo

Para se conseguir atingir o objectivo final, é necessário começar-se por construir uma ferramenta intermédia, o PAsMo, que será responsável pelo processo de modificação do texto para desambiguação morfológica através de regras. Esse será o objectivo do trabalho desenvolvido durante a fase inicial do trabalho, ou seja, durante o Trabalho Final de Curso.

O sistema PAsMo, "Pós AnáliSe MOrfológica", é uma evolução do sistema MPS, "Module Post-SMorph" [Faiza 99] e tem como primeiro objectivo a adaptação dos resultados produzidos pelo analisador morfológico às necessidades específicas de cada analisador sintáctico, respondendo assim a três necessidades:

- modificação da segmentação feita pelo analisador morfológico. Por exemplo, se o analisador morfológico delimitar de modo separado um prefixo e uma forma base como por exemplo "ex-" e "marido", pode-se desejar agrupar estes dois segmentos num só ("ex-marido"). Por outro lado, quando o analisador morfológico delimita como um só segmento a sequência "nas" pode-se querer obter na entrada da análise sintáctica os dois segmentos "em" e "as". A modificação de segmentação também pode ser usada para realizar tarefas que são do domínio da pré-sintaxe como, por exemplo, o reconhecimento de números, de datas e de expressões fixas.
- modificação da informação associada às palavras reconhecidas pelo analisador morfológico. Com efeito, conforme o tratamento sintáctico que se pretenda fazer, a granularidade da informação linguística presente à saída de um analisador pode ser demasiada fina para o analisador sintáctico. Por exemplo, o facto de um segmento ter sido analisado morfologicamente como um diminutivo de um nome pode não ser pertinente na entrada da análise sintáctica onde só o facto de este segmento ser um nome é interessante.
- modificação do formato de saída do analisador morfológica para esta saída se adequar aos formatos esperados pelo analisador sintáctico. A sintaxe da saída do analisador morfológico pode não ser idêntica à sintaxe exigida pelo analisador sintáctico, sendo assim necessário proceder a uma adaptação.

Esta capacidade de modificação expressa-se através de regras declarativas de transformação que se baseiam no conceito de emparelhamento de padrões. Nesta primeira versão, assume-se como formato de entrada para os dados a processar pelo PAsMo o formato de saída produzido pelo SMorph (opção saída *Prolog*).

Uma descrição mais detalhada deste sistema pode ser encontrada na secção 4.

2. Estado da Arte

O trabalho que aqui se apresenta resulta do cruzamento de vários campos. Poderá ser útil olhar para eles, ver as técnicas actualmente utilizadas em cada um, estudar o funcionamento das ferramentas existentes, qual o trabalho feito, quais os resultados obtidos, quais os problemas detectados e quais as soluções encontradas.

Neste capítulo começa-se por introduzir alguns conceitos sobre este domínio, para, depois, se analisar o *estado da arte* das várias áreas relevantes para a extracção de termos. Começa-se pela análise de *corpus*, depois vê-se a lematização, a obtenção das frequências de lemas, o processamento e extracção de unidades com múltiplas palavras e, para terminar, o funcionamento dos analisadores sintácticos de superfície. Finalmente, descreve-se o estado da arte da área em que se vai trabalhar, onde já foi produzido algum trabalho relevante, útil e interessante. Isso poderá dar um bom ponto de partida, muito embora, a maioria do trabalho feito se destine especialmente a línguas estrangeiras.

2.1. Abstracção e Conceitos

O objectivo desta secção é, antes de mais, introduzir de forma breve os conceitos relevantes para a exposição que se vai seguir. Explicam-se alguns conceitos, discutem-se as várias visões e teorias envolvidas e indicam-se as características mais importantes das unidades terminológicas que queremos extrair e que vão guiar o processo.

Não se tem a intenção de indicar uma definição precisa do que é um termo, nem entrar em grandes detalhes nas discussões das várias visões possíveis. Quer-se apenas esboçar aquilo em que se pode basear uma ferramenta cuja função e tarefa seja extrair os termos contidos num qualquer texto especializado.

Nesta descrição serão dadas algumas das ideias principais subjacentes e indicam-se algumas referências para um leitor mais interessado em aprofundar melhor certas noções.

2.1.1. Conceito de Termo

A definição daquilo que constitui um termo, proposta pela aproximação computacional, é diferente daquela que é proposta tradicionalmente pela escola de Viena. A caracterização de um termo no contexto da computação deve ter em conta a nova dimensão do estudo dos termos em relação às propostas de aplicação da engenharia terminológica.

Visão clássica

No sentido tradicional, um termo é considerado como a etiqueta linguística de um conceito. Nascido no período do movimento positivista, a doutrina clássica da terminologia assenta na visão unificadora do conhecimento. Assume que o conhecimento está organizado em domínios, sendo cada domínio equivalente a uma rede de conceitos. Em cada domínio, cada conceito é (idealmente) associado a um termo, que é a sua etiqueta linguística. Tal aproximação de terminologia, tão centrada

no conceito, é adequada à normalização. No entanto, já não é tão adequada a uma aproximação computacional da análise de termos.

Problemas com a aplicabilidade da visão clássica

Em primeiro lugar, a visão clássica assume que os peritos na área têm um mapa conceptual na cabeça sobre um certo domínio. Esta aproximação é enganadora e não é produtiva porque os peritos não conseguem construir mapas conceptuais a partir de introspecção. Os terminologistas consultam constantemente dados e analisam os elementos léxicos da terminologia de dados de forma a adquirir e validar uma descrição conceptual.

Em segundo lugar, a construção semi-automática de recursos terminológicos e a sua exploração fazem crescer uma grande variedade de dados terminológicos. Há por exemplo, dois tipos de aplicações possíveis para este tipo de sistema: procura de sinónimos para indexação automática e aquisição de informação, índices estruturados para grandes documentos, etc.

Engenharia de Terminologia

Numa definição de termos mais adequada a uma terminologia baseada em *corpus*, um termo deve ser a saída de um procedimento de análise morfológica. Uma única palavra ou várias palavras podem ser um termo apenas porque foi decidido que eram. O processo de decisão pode envolver uma comunidade de investigadores ou especialistas, uma instituição de normalização, ou mesmo um simples engenheiro ou terminologista encarregado de construir uma terminologia com objectivos específicos.

São dois os tipos de termos que se consideram: os simples e os compostos.

Termos Simples

Os termos simples são caracterizados por serem constituídos por apenas uma unidade léxica. Normalmente, são nomes e tal como todos os outros nomes podem ser ambíguos. Para tratar estes termos há que se basear nas suas características morfológicas. Podem-se encontrar num texto na sua forma singular ou plural, neste caso a palavra será diferente mas o conceito será o mesmo. Por isso, na aproximação que se vai adoptar, usar-se-á a raiz, ou lema, das palavras que não depende nem do número, nem do género, nem do tempo gramatical. Muitas vezes são a simplificação de um termo complexo, isto é, a separação de unidades léxicas de um termo composto (por exemplo, no termo complexo "Rede Mundial de Computadores", pode-se considerar "Rede" como um termo simples).

Para detectar um termo simples pode-se usar como base dois critérios: as frequências e os itens léxicos desconhecidos, que podem ser nomes (possivelmente associado a um filtro de frequências).

Frequências

De acordo com um trabalho de [Neto 96], todas as unidades léxicas de uma palavra para as quais a frequência relativa num texto especializado for catorze[3] vezes superior à

[3] Note-se que o trabalho em causa usa *corpus* que não estão lematizados pelo que se podem obter resultados diferentes numa abordagem em que se comparam as frequências relativas de lemas e não as das palavras. Por exemplo, as palavras "galáxia" e "galáxias" seriam detectadas como o mesmo candidato

sua frequência de referência em *corpora* gerais, será considerada como um possível termo.

Da lista resultante é necessário distinguir nomes de outras unidade léxicas que possam ser consideradas como termos. Na lista final, por se estar a trabalhar com os lemas das palavras, podem aparecer itens de diferentes categorias, como por exemplo adjectivos. No entanto, apesar destas unidades léxicas não serem consideradas como termos simples, poderão ser úteis para a segunda etapa do processo, o reconhecimento de termos compostos.

Palavras desconhecidas

Os *corpora* serão inicialmente lematizados. O dicionário usado nesse processo será representativo do português geral. O facto de uma palavra não ser reconhecida num *corpus* especializado pode ser um bom indício de que esta palavra pode ser um termo. Se, além disso, a palavra desconhecida é um nome[4], o indício é ainda mais forte. Finalmente, se a palavra desconhecida aparece repetidamente no *corpus*, a palavra será considerada um candidato a termo simples.

O dicionário que se vai usar contém termos suficientes para cobrir o vocabulário geral do português. O *corpus* será analisado por um analisador sintáctico de superfície, que trata das palavras desconhecidas detectando-lhes a categoria a que pertencem, considerando o contexto e a posição em que aparecem. Esta tarefa pode ser ajudada por um sistema de previsão que use índices morfológicos, ou seja, que detecte os sufixos e prefixos das palavras desconhecidas. Finalmente, podemos filtrar os termos candidatos aplicando-lhes restrições baseadas nas frequências[5].

Termos Compostos

Estes termos são compostos por mais de uma unidade léxica. São sintagmas nominais[6] que obedecem a certas restrições.

Para o reconhecimento de termos ingleses a estrutura dos termos é muito restrita. De acordo com [Justeson 95] esta estrutura pode ser filtrada com expressões regulares definidas por eles. De acordo com [Anscombre 91], uma das pistas para detectar um termo é o facto de ter uma estrutura nominal mas não ter determinante[7]. Em [Bourigault 92] a ausência de determinantes também é utilizada no seu sistema de aquisição de termos compostos.

---

a termo num, mas não no outro. O número de catorze será provavelmente diminuído quando se usa frequência de lemas. Pode-se, também, dar ao utilizador a possibilidade de variar este valor.

[4] Sabe-se que uma palavra desconhecida é um nome (ou outra categoria qualquer) pela sua posição na frase. Pelas restrições gramaticais existentes na língua quanto às categorias de palavras que podem ocorrer na posição em causa é possível deduzir uma lista de categorias possíveis para a palavra desconhecida. Outra ajuda a essa determinação são as terminações das palavras. Por exemplo, palavras terminadas em "mente" geralmente são advérbios.

[5] Neste momento ainda não se tem a certeza da utilidade deste refinamento final, mas pretende-se testar se é necessário, para melhorar os resultados, ou não.

[6] Será interessante e útil definir rigorosamente quais as estruturas sintácticas que poderão ser utilizadas pelo sistema na detecção deste tipo de termos.

[7] Esta restrição, apesar de não ser muito comum, também se verifica no português. Seria útil identificar outras restrições a que obedeçam os termos compostos no português.

Um terminologista com conhecimentos linguísticos será capaz de definir uma gramática regular das construções sintácticas mais habituais dos termos compostos. A extracção destas estruturas de *corpora* especializados será feita por um analisador sintáctico de superfície. Introduzindo nesta estrutura os lemas nominais e não nominais detectados pela comparação de frequências relativas poder-se-ia aumentar a probabilidade de detectar termos.

2.2. Análise de Corpus

Um *corpus*[8] é uma colecção de dados linguísticos, compilados tal como foram escritos pelo autor ou transcritos de gravações de fala. Um *corpus computacional* é um grande corpo de textos em formato electrónico. Actualmente, *corpora linguísticos computacionais* podem ser armazenados contendo muitos milhões de palavras cujas características podem ser analisadas. Por exemplo, o CEMTEM Público.

A análise de *corpus* é uma das actividades mais importantes do campo da Linguística de c*orpus* (sobre isto, consultar [Kennedy 98]). Ocupa-se de todas as actividades linguísticas que podem ser feitas sobre grandes *corpora*. Entre eles distinguem-se: a criação de *corpus* e a sua análise.

O desenho e construção de grandes *corpora* está muito além dos objectivos deste projecto. Nenhum *corpus* será desenvolvido aqui, mas o uso de *corpora* será, obviamente, necessário para se conseguir trabalhar.

Pelo contrário, as pesquisas no domínio da extracção de termos são bastante importantes no campo da análise de *corpus* que quer tratar robusta e eficientemente textos muito grandes (os *corpora*), de forma a conseguir extrair deles a maior quantidade de informação linguística possível.

Neste trabalho usa-se em especial a lematização, a correspondência de etiquetas linguísticas para desambiguação da informação morfológica, a análise sintáctica e a obtenção de frequências de lemas.

2.3. Lematização

A primeira ferramenta a ser necessária neste projecto é um lematizador para segmentar o texto inicial (o *corpus* do qual queremos extrair os termos) e para associar a cada unidade alguma informação morfológica e sintáctica (lema, categoria, etc.). Esta ferramenta é o primeiro passo de qualquer aplicação que queira processar texto escrito. Existem diferentes tipos de lematizadores (usam morfologia de dois níveis, ou regras). O lematizador a ser utilizado, o SMorph, é baseado em regras morfológicas.

2.4. Frequência dos Lemas

A obtenção da frequência é algo bastante linear, se o texto tiver sido previamente lematizado. A ideia é contar o número de vezes que cada termo aparece e extrair do resultado informação linguística. Contar os lemas em vez de contar as palavras será uma vantagem. Por exemplo, a forma singular e plural de um nome seriam contadas como duas palavras diferentes, mas têm o mesmo lema. O mesmo acontece para os tempos

[8] *Corpus*: palavra latina que significa “corpo”, cujo plural é *corpora*.

verbais ou para o género: as várias conjugações de um verbo são palavras diferentes mas têm o mesmo lema.

2.5. Extracção de Unidades com Múltiplas Palavras

Este processo é baseado em informação estatística extraída de grandes *corpora*. A ideia principal é medir o grau de coesão[9] entre duas ou mais palavras (sobre isto, consultar [Silva 99]). Apesar do aparecimento de alguns resultados interessantes, pensamos que os termos são uma espécie especial de unidades com múltiplas palavras e que o uso de uma descrição estrutural pode melhorar significativamente os resultados obtidos. No entanto, este tipo de processamento só é válido para termos compostos[10].

2.6. Analisador Sintáctico de Superfície

Analisador sintáctico de superfície é a designação comum de vários trabalhos de análise sintáctica que estão mais interessados numa percepção superficial do que numa descrição estrutural de sequências encontradas no texto. Nos últimos anos tem havido um novo interesse nesta técnica a robustez e a rapidez.

A robustez porque este analisador se comporta bem mesmo na presença de palavras desconhecidas, resistindo bem a erros e incorrecções[11]. Muitos textos reais de grande dimensão podem ser analisados, mesmo que contenham algumas incorrecções sem que o processo seja interrompido pela detecção de erros ou de palavras desconhecidas.

A rapidez porque o tempo de análise recorrendo a este processo é melhor do que o dos analisadores que trabalham com gramáticas estruturais[12].

A investigação neste tipo de gramáticas e analisadores tem sido muito rica nos últimos anos (Hudson, Tapanainen, Sparkle, Xerox, etc.). E hoje, para o tratamento linguístico de grandes *corpora* parecem ser a única solução. Note-se que também se podem usar métodos estatísticos e híbridos. A comparação de métodos e a discussão comparada dos resultados assim obtidos foge ao âmbito deste trabalho pelo que não se alongará esta discussão.

2.7. Aquisição de Termos e Indexação Automática

Os métodos de análise de *corpus* para a aquisição automática de termos pode ser dividida em duas grandes famílias: métodos de indexação e métodos de aquisição. O

---

[9] O grau de coesão mede a probabilidade de duas palavras constituírem um bigrama, isto é, qual a probabilidade das duas palavras aparecerem juntas.

[10] Como já se teve a oportunidade de referir, os termos simples são nomes que, estruturalmente, são em tudo iguais aos outros nomes que não são termos.

[11] Isto deve-se antes de mais à função para que a ferramenta é desenhada. O facto desta ferramenta se destinar à análise de grandes quantidades de texto exige que seja desenhada de tal forma que um erro ortográfico não seja suficiente para parar o processo.

[12] Quando se vão analisar grandes quantidades de texto, qualquer segundo ganho num processo mais simples é relevante para a redução do tempo gasto globalmente que é um factor determinante e muito relevante não só para a funcionalidade da ferramenta, como também, para o uso e aplicação do sistema em problemas reais.

primeiro método quer dar descritores ao texto de forma a permitir aceder-lhe mais facilmente (identificação). O segundo método quer enriquecer uma lista de termos já existente ou criar de raiz uma nova lista de termos para uso posterior.

Sistemas de aquisição de termos já foram desenvolvidos para várias línguas. A detecção de termos compostos e a detecção de termos simples parecem necessitar de um tratamento diferente devido às suas características. Estes sistemas lidam com termos compostos e com termos simples. Podem ser orientados pela estrutura, ou seja, baseados em descrições linguísticas; orientados por métodos estatísticos, ou seja, onde as deduções se fazem com base em medidas de frequência; ou por métodos híbridos, onde há uma mistura das duas técnicas anteriores.

Independentemente da escolha técnica subjacente, há diferentes tipos de sistemas a considerar:

1. Aqueles que criam uma lista de termos candidatos de raiz baseando-se nas características;
2. Aqueles que trabalham com terminologias e *corpus* já existentes. O processo de aquisição de termos é baseado em variações linguísticas (morfológicas e sintácticas);
3. Aqueles que apenas lidam com termos simples recorrendo a métodos estatísticos que aplicam a *corpus*.

A saída destes sistemas é uma lista de termos compostos e de termos simples.

Têm duas características em comum:

- Massividade: a saída é uma grande lista que, no final, tem que ser corrigida manualmente;
- Não estruturadas: não são reconhecidas as relações conceptuais entre os termos,

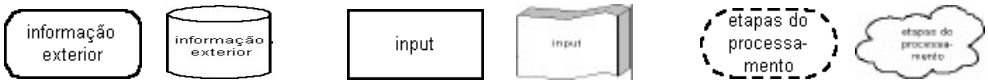
Os resultados podem ser melhorados pela possibilidade de interagir com o utilizador no processo de aquisição de termos especialmente se este for um terminologista. Isto pretende ser feito neste projecto.

Os sistemas do primeiro tipo são muito úteis para construir novas terminologias, terminologias que ainda não existem. Os do segundo tipo têm uma grande vantagem: oferecem uma maneira de enriquecer uma terminologia previamente existente e organizar os termos numa hierarquia conceptual (por exemplo, numa rede semântica). Mas isto só poderá ser feito quando estiverem disponíveis terminologias em formato electrónico.

3. Arquitectura do ATA

Neste capítulo faz-se a descrição da arquitectura do sistema a construir. Começa-se a análise do sistema global e dos processos que o compõem: lematizador, PAsMo, analisador sintáctico de superfície e extractor de termos. De seguida e porque se quer estudar com mais pormenor o último processo que é aquele que se vai implementar, faz-se um estudo mais pormenorizado do extractor de termos.

Na descrição das arquitecturas e dos processos envolvidos usa-se a seguinte notação:



3.1. Sistema Global: Arquitectura Geral

O sistema que nos propomos construir é constituído por quatro grandes processos como se pode observar na Figura 1.

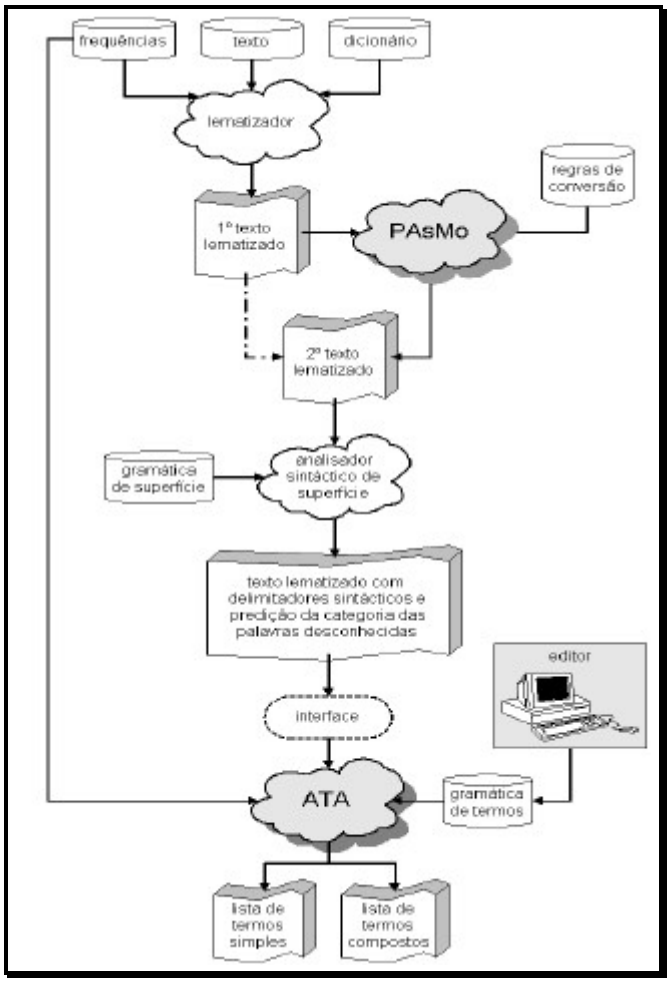


Figura 1: Arquitectura Geral do ATA

O primeiro processo é o de lematização que enriquece um texto, o *corpus* recebido, com os lemas de cada palavra. Para isso precisa de um dicionário da língua em questão.

O segundo adapta os resultados produzidos pelo analisador morfológico às necessidades específicas de cada analisador sintáctico. Este processo pode não existir se a saída produzida estiver logo de acordo com aquilo que o analisador sintáctico espera (daí a linha a tracejado entre os dois textos lematizados, na Figura 1).

O terceiro é o da análise sintáctica de superfície que vai etiquetar as frases com os possíveis papéis sintácticos de cada palavra ou grupo de palavras. Este processo vai necessitar para isso de uma gramática de superfície.

O quarto processo, o extractor de termos baseia-se numa gramática de termos (que poderá, eventualmente, ser editada pelo linguista) que indica quais as estruturas sintácticas a que, tipicamente, um termo composto obedece. Outro critério útil a considerar é a relação entre a frequência com que uma palavra costuma aparecer (a chamada, frequência de referência de cada palavra) e a frequência com que aparece no texto em questão.

Deste processo obtém-se uma lista de nomes que são possíveis termos simples; uma lista que contém as palavras não sendo nomes, podem ser parte de um termo; e uma lista de termos simples e uma lista de termos compostos detectados pelo sistema.

Este processo está ainda encarregado dum último processamento de verificação dos dados obtidos no passo anterior, de forma a melhorar os resultados.

Por exemplo, identificação e tradução de datas e números, agrupamento de palavras compostas, tratamento de contracções, revisão dos dados previamente obtidos com base em informação estatística e no grau de coesão entre palavras.

No final deste processo o sistema devolve como *saída*, duas listas: uma de termos simples e outra de termos compostos. Essa lista deverá, no final, ser revista manualmente por um terminologista ou por um linguista.

No âmbito deste trabalho vai ser desenvolvido um sistema baseado em regras que trata os resultados obtidos do lematizador e um sistema que faça a extracção de termos. Como lematizador vai ser utilizado o SMorph. Basta, portanto, definir as interfaces e as funcionalidades, ou seja, basta indicar aquilo que devem receber e produzir ambos os processos e qual o formato final da saída do segundo processo, que será a entrada do processo que se vai implementar.

Posteriormente, será necessário verificar a adequação das categorias escolhidas ao trabalho que se pretende desenvolver. É possível que se possa simplificar as categorias escolhidas ou diminuir o seu número.

Assim sendo, na secção que se segue, faz-se uma descrição pormenorizada do processo de extracção de termos[13].

[13] Assume-se a existência de pequenos processos de interface que possibilitam a utilização conjunta dos dois sistemas que assim podem ser desenhados e implementados de forma independente.

3.2. O Processo de Extracção Semi-Automática de Termos

Para testar a ferramenta a desenvolver, a título de exemplo, usar-se-ão duas ferramentas em particular que estão descritas com pormenor em artigos a isso dedicados, pelo que não se explicitarão rigorosamente os métodos utilizados.

Veja-se, então, com mais pormenor, o processo relevante para o sistema que ora se descreve, o processo de extracção semi-automática de termos.

A Figura 2 descreve a arquitectura do processo de extracção de termos.

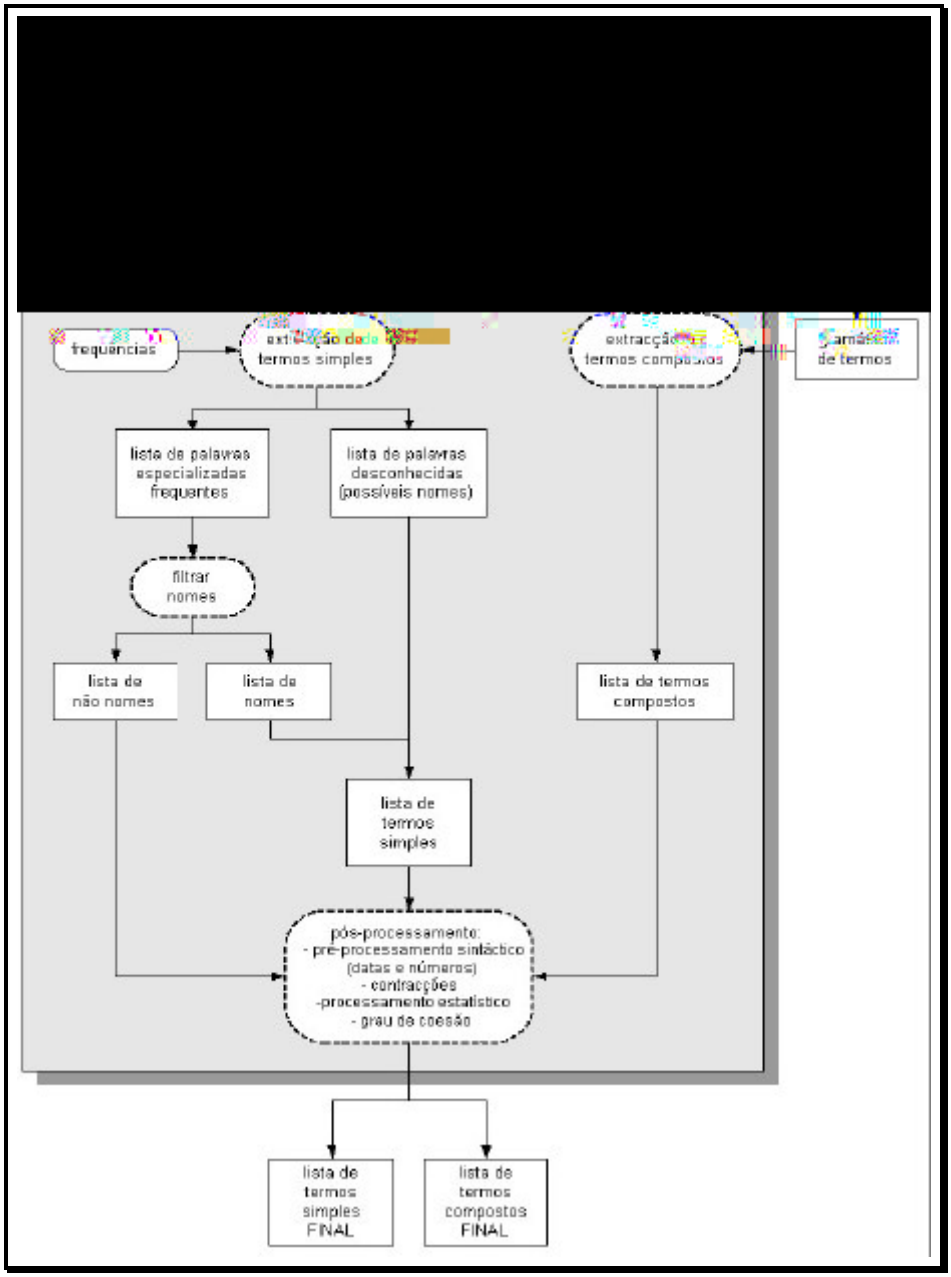


Figura 2: Arquitectura dos processos relevantes do sistema.

O sistema recebe um texto já lematizado e com delimitadores sintácticos.

Como a lematização e a análise sintáctica são feitas por módulos totalmente independentes deste módulo, é necessário, antes de mais, converter as etiquetas e delimitadores usados no texto recebido, de forma a serem reconhecidas pelo sistema. É essa a função do processo.

Uma vez feita a conversão, estando o texto devidamente lematizado e agrupado segundo as suas estruturas sintácticas, já se pode dar início ao processo de extracção semi-automática[14] de termos.

O processo de extracção de termos, que é o mais importante deste módulo, pode, de certa forma, dividir-se em dois. Isto porque, devido às características próprias dos dois tipos de termos considerados, há dois processamentos iniciais que se podem fazer previamente em separado: o da extracção de candidatos a termos simples e o da extracção de candidatos a termos compostos. No fim dos dois processos de extracção dos dois tipos de candidatos, é necessário fazer um pós-processamento das listas obtidas, por razões que se perceberão melhor mais adiante. Aí são feitos os agrupamentos necessários, onde se consideram as frequências de ocorrência de cada palavra e o grau de coesão de cada grupo de palavras e onde se tenta extrair informação adicional a partir da formatação do texto.

De seguida, descrevem-se os processos de aquisição das listas de candidatos a termos simples e a termos compostos, que pelo menos a nível conceptual se podem considerar como independentes, e as várias etapas de pós-processamento a que se deve submeter o texto e os resultados obtidos para melhorar a lista final obtida.

3.2.1. Extracção de Termos Simples

Na lista de possíveis termos simples devem estão nomes ou palavras desconhecidas, que pela sua posição na frase, se suspeite serem nomes; ou que apareçam com uma frequência superior[15] à esperada[16].

Para se extraírem os possíveis termos simples existentes no texto, começa-se por extrair as palavras que se sabem serem nomes e que aparecem com uma frequência *n* vezes superior à esperada. Onde *n* é um parâmetro, que o utilizador poderá fazer variar conforme a precisão que desejar.

Depois, à lista assim obtida juntam-se todas as palavras desconhecidas que, pela sua posição na frase, podem ser nomes e que aparecem mais que um determinado número de vezes[17].

---

[14] Chama-se extracção semi-automática (e não, simplesmente, automática) porque apesar de não ser um processo totalmente independente do utilizador – no final é sempre necessário que um especialista confira os resultados obtidos automaticamente – deve, pelo menos, minimizar o trabalho humano.

[15] A razão entre o número de vezes que a palavra aparece normalmente, é que vai servir para decidir se a palavra é ou não candidata a termo. Pode vir a ser um parâmetro passível de alteração por parte do utilizador. O valor ideal pode ser alvo de estudo, uma vez que se quererá sugerir um valor inicial ao utilizador. Este valor depende, por exemplo, daquilo que se usa para classificar as palavras: quando se usam as raízes das palavras espera-se um número menor do que quando se usam as palavras inteiras.

[16] A frequência de referência, que é a frequência esperada, vem tabelada para o português corrente.

[17] Também este parâmetro deve ser escolhido pelo utilizador. Em princípio, deverá ser sempre maior que um para se tentar garantir que um erro ortográfico não é considerado como um termo.

Deste modo, obtêm-se todos os nomes que aparecem no texto inicial e que podem (com grande probabilidade) ser termos simples.

Esta lista provisória terá que ser analisada por um pós-processamento que se descreverá posteriormente.

3.2.2. Extracção de Termos Compostos

Tal como os termos simples, os termos compostos obedecem a determinadas restrições sintácticas e gramaticais, que podem facilitar a sua identificação.

Se os termos compostos obedecem a várias estruturas sintácticas, então, um sistema que os que queira detectar deve começar por procurar no texto sequências de palavras, que obedeçam a uma determinada gramática descritiva dessas possíveis estruturas.

Assim, o ATA terá uma gramática inicial que deverá descrever a forma típica da estrutura de termos compostos. Essa gramática deverá ser passível de ser modificada pelo utilizador. Para isso o sistema integrará um editor de gramáticas de termos[18].

A estrutura e o vocabulário da gramática serão definidos posteriormente. Poderá conter símbolos representativos de classes de palavras ou de palavras propriamente ditas.

Uma vez identificados os possíveis termos compostos, é preciso, à semelhança dos termos simples, verificar se aparecem mais do que seria esperado. Esse pós-processamento, que reúne a informação dos dois processos que se acabam de descrever é analisado na próxima secção.

3.2.3. Agrupamentos

O sistema deverá ter a capacidade de identificar e traduzir datas e números. Por exemplo, num texto especializado de história, muito provavelmente aparecerão muitas datas. Neste caso, é preferível que o sistema diga que há mais datas do que seria de esperar do que indique “Dezembro” como termo. O mesmo se passa com os números, com as palavras compostas e com as contracções.

Com as ferramentas auxiliares que vão ser utilizadas, parte deste trabalho já chega feito ao sistema. O analisador sintáctico encarrega-se, por exemplo, de agrupar os componentes de uma data ou de um número. Aquilo que é importante no desenho do sistema é ter o cuidado de dotá-lo da capacidade de fazer este tipo de agrupamentos: agrupar o que os sistemas auxiliares ainda não fazem. Outra opção é assumir quais os agrupamentos que vêm feitos e obrigar a que, quando se substituírem as ferramentas, as novas que se utilizarem tragam isso feito. Neste ponto, deve ter-se o cuidado de não tornar o sistema dependente das ferramentas, assim, poder-se-á, por exemplo, dar ao utilizador a opção de indicar que agrupamentos vêm feitos. Se não o fizerem por si será necessário acrescentar um módulo que faça isso.

[18] Ainda que não se chegue a criar uma interface gráfica para este editor, pelo menos, pode-se localizar a informação num ficheiro exterior que o utilizador possa alterar.

3.2.4. Processamento de Frequências

Para uma palavra ser considerada como um termo simples, é preciso que seja um nome e que apareça mais do que um determinado número de vezes. Para um conjunto de palavras ser considerado como um termo composto, é preciso que surjam com um grau de coesão superior a um determinado valor e segundo uma determinada estrutura sintáctica.

Mas suponha-se, a título de exemplo, que, num determinado texto especializado, aparece sete vezes a expressão "rede mundial de computadores". Neste caso, tem-se a palavra "rede" na lista de candidatos a termos simples e tem-se a expressão "rede mundial de computadores" na lista de candidatos a termos compostos. Então, é preciso verificar se a palavra "rede" é de facto um termo simples, ou seja, se aparece vezes suficientes sozinha, de modo a poder-se considerar um termo simples, ou se só aparece com relevância quando contida na expressão "rede mundial de computadores".

Assim sendo, no final dos dois processos de extracção de listas de candidatos a termos é preciso cruzar as listas obtidas para se detectar este tipo de dependências. No exemplo dado, o número real de ocorrências do termo simples "rede" seria a diferença entre o número de ocorrências do candidato a termo simples "rede" e o número de ocorrências do candidato a termo composto "rede mundial de computadores". Isto, claro, assumindo que mais nenhuma expressão candidata a termo composto continha a palavra candidata a termos simples.

Se a diferença fosse consideravelmente elevada, isto é, se a palavra "rede" aparecesse vezes suficientes sozinha, podia-se, então, considerar que se tinha detectado um termo simples, "rede", e um termo composto, "rede mundial de computadores".

3.2.5. Processamento de Informação Adicional

Há uma última informação contida no texto inicial que se revela importante: a da formatação. Se o texto recebido vier formatado terá marcas (*tags*) que acrescentam informação ao próprio texto.

É importante notar que a formatação depende do editor de texto onde se produziu e formatou o texto pelo que, se se quiser considerar esta fonte, ter-se-á que desenvolver módulos especializados para cada editor de texto, módulos esses que sejam independentes entre si e do sistema mas que possam ser associados à ferramenta final. Uma possibilidade seria indicar-se como parâmetro qual o editor de texto utilizado, outra será a referência da localização da ferramenta específica de cada processador.

No caso de se usar um lematizador ou um analisador sintáctico de superfície que ignore as marcas de formatação[19] será necessário processar o texto inicial e não apenas o resultante da pré-análise feita por esses dois processos iniciais.

Uma ideia seria atribuir uma bonificação a cada palavra que se suspeitasse ser um termo de acordo com a formatação que tem nas várias ocorrências[20]. Por exemplo, se

[19] O problema não é tanto que estes sistemas ignorem as marcas de formatação, o problema é quando alguns destes processos, reconhecendo as marcas, as eliminam perdendo essa informação durante o processo, e impedindo o ATA de as receber.

uma palavra aparece num título, em vez de se contabilizar apenas uma ocorrência, poder-se-iam contabilizar mais[21].

Para se usar a informação contida na formatação de forma coerente, consistente e robusta seria, talvez, útil criar uma hierarquia de bonificações uma vez que a relevância de uma palavra que surge no título é certamente mais importante do que a de uma que apareça num subtítulo ou a meio do texto em negrito. Essa hierarquia poderia, inclusivamente, ser modificável pelo utilizador.

Assume-se que a formatação foi aplicada pelo criador do texto de forma coerente e consistente, excluindo-se hipóteses de formatação absurdas.

Há vários tipos de informação a considerar quando se considera a formatação.

- ***Títulos de documentos, de secções e de subsecções***

O título, seja qual for o seu nível dentro do texto, obedece a regras diferentes das esperadas seja a níveis gramaticais, seja em termos de importância daquilo que dizem.

Um título pode não ser uma frase completa, pode não ter sujeito ou pode não ter predicado e pode não ter uma construção gramatical completa.

Uma palavra que apareça num título é certamente mais importante que uma que apareça no corpo do texto. Assim, a ocorrência de uma palavra numa posição destas é um indício forte de que possa ser um termo.

Nos editores de texto em que é indicado claramente que uma "frase" é um título, independentemente do "nível" do título, essa informação é muito relevante no que se refere ao discernimento da importância das suas palavras no texto.

- ***Negrito***

Quando uma palavra, no meio do corpo do texto, aparece em negrito (*bold*), tipicamente, isso significa que essa palavra deverá ter uma importância maior do que as outras vizinhas.

Além disso, muitas vezes se escolhe formatar os títulos e subtítulos com este tipo de marca. Também aí é um bom indício.

- ***Itálico***

Quando uma palavra aparece em itálico, geralmente, isso quer dizer que ou a palavra é especial do contexto em questão, ou é um estrangeirismo[22], daí que também essas palavras sejam mais relevantes que as outras.

---

[20] Estas bonificações têm que ser feitas depois das contas que possibilitam a diferenciação entre os candidatos a termos simples e os verdadeiros termos simples, que anteriormente se descreveram.

[21] Onde o factor multiplicativo terá que ser definido com base em experimentação e posterior avaliação dos resultados obtidos. Pode, inclusivamente, vir a ser um parâmetro (ou vários se se quiser ter uma hierarquia de bonificações), que o utilizador fará variar conforme desejar.

[22] Normalmente, os estrangeirismos que forem nomes aparecerão incluídos nas palavras desconhecidas que, pela sua posição na frase, se supõe serem nomes. Estar em itálico é uma indicação que confirma (ou reforça) essa suspeita.

À semelhança do negrito, esta formatação também costuma servir para indicar títulos e subtítulos.

- ***Tipo de letra***

Normalmente, num texto só se muda de tipo de letra nalgumas situações muito particulares: títulos, palavras mais importantes, estrangeirismos ou palavras específicas da área.

Exemplo disto é este relatório onde os títulos de capítulos e secções estão num tipo de letra diferente (*Arial*) do resto do texto (*Times New Roman*)

- ***Tamanho de letra***

Tipicamente, os títulos, as palavras que compõe as capas, os nomes dos capítulos e demais frases de especial importância, aparecem com um tamanho de letra superior ao do resto do texto. Mais uma vez, pode-se utilizar o exemplo deste relatório.

Seria interessante definir uma hierarquia das bonificações dadas a cada palavra pela sua formatação.

- ***Uso de maiúsculas***

Uma palavra que surja com todas as suas letras em maiúsculas será, provavelmente, uma palavra de especial interesse. Antes de mais porque geralmente se evita escrever em maiúsculas por razões de ordem *social*[23], depois, porque o recurso a esse truque é muito usado para, em editores de texto que não têm possibilidades de formatação, sublinhar algo relevante do texto.

- ***Cabeçalhos e rodapés***

Um cabeçalho (*header*), tal como um rodapé (*footer*), é algo que, em princípio, se via repetir coerente e sucessivamente ao longo de todo o texto.

Se, e volta-se ao exemplo deste relatório, no rodapé aparece algo diferente da numeração de página, provavelmente as palavras que aí aparecem são relevantes, aliás, tipicamente, identificam o assunto do documento.

Assim sendo, mais uma vez, está-se na presença de informação relevante para a detecção de termos relevantes no documento.

- ***Notas de Rodapé***

Muitas vezes, as notas de rodapé servem para explicar melhor algo, para darem indicações complementares, para se acrescentar uma nota menos importante àquilo que já se disse. Muitas vezes são usadas apenas para indicar bibliografia com mais informação sobre o assunto, para se indicar a fonte de uma afirmação ou para se remeter para outro documento. Noutros casos, dependendo do utilizador, introduzem assuntos ou melhoram uma explicação. Por isso, as palavras que aparecem nas notas de rodapé

[23] Em termos de etiqueta para a Internet (onde os textos tipicamente não são formatados), na chamada "netiqueta", considera-se que uma palavra em que todas as letras sejam maiúsculas é um grito ou uma chamada de atenção pelo que o recurso a esse tipo de escrita deve ser feito com algum cuidado.

terão maior ou menor importância relativamente ao corpo do texto dependendo da utilização que o autor do texto lhes dá. Esta pode ser mais uma opção configurável.

- ***Aspas***

Estes caracteres servem, por um lado, para indicar citações, nomes de obras referenciadas, mas, à semelhança do itálico, também servem como indicações de palavras estrangeiras, de estrangeirismos ou como indicação de uma palavra especial naquele contexto.

Seria interessante estudar se, quando uma palavra é desconhecida e aparece entre aspas, se isso pode ser interpretado como indicação de que essa palavra é um termo simples, ou não.

- ***Estilos***

Alguns editores de texto permitem a utilização de estilos de formatação que facilitam ao utilizador a manutenção da coerência global do texto em termos de forma visual e que permitem que as modificações de estilos em textos de tamanho mais elevado se façam mais rápida, eficiente e facilmente.

Exemplos disso são os estilos do HTML, a formatação usada no LaTex e os estilos que o Microsoft Word permite definir.

Se o sistema a desenhar poder incluir no seu conhecimento a informação que advém dos estilos, isto é, se for possível indicar ao sistema quais as marcas que permitem reconhecer um título ou qual a maneira de identificar o nome de um capítulo ou qual a maneira de identificar o nome de um capítulo, seria útil e relevante. O conhecimento assim extraído aumentaria o factor de certeza sobre a importância de uma palavra com base na sua formatação.

Esta seria mais uma forma de introduzir no sistema conhecimento do utilizador que é sempre útil e aumenta a fiabilidade dos resultados[24].

[24] Se o utilizador indicar quais os estilos que se podem encontrar, qual a sua hierarquia e qual as marcas que permitem reconhecer cada estilo, a fiabilidade das suposições que se fazem com base na formatação aumentaria substancialmente.

4. O PAsMo

Nesta secção apresenta-se o primeiro processo desenvolvido no âmbito do sistema ATA. A este processo deu-se o nome de PAsMo, Pós AnáliSe MOrfológica.

Esta secção está organizada do seguinte modo:

Terminologia:	dá uma breve explicação sobre os termos utilizados na descrição do sistema.
Arquitectura:	apresenta a arquitectura do sistema.
Dados de entrada:	descreve a sintaxe dos diferentes ficheiros que contêm a especificação das transformações que se pretendem efectuar. Também se refere a sintaxe e semântica do ficheiro que contém os dados a processar.
Dados de saída:	descreve a sintaxe dos resultados produzidos pelo PAsMo, e o conteúdo do ficheiro onde são apresentadas as estatísticas finais e as mensagens de erro.
Algoritmo:	inclui uma descrição do algoritmo usado para produzir as alterações desejadas. Atendendo ao facto do sistema PAsMo poder ser usado para o tratamento de corpora, é necessário ter em consideração a eficiência do algoritmo proposto.
Estruturas de dados:	contém a descrição das estruturas de dados que armazenam a informação fornecida ao PAsMo e a descrição das estruturas de dados necessárias a uma execução eficiente dos algoritmos utilizados. Também apresenta o diagrama UML das classes desenhadas.
Trabalho Futuro:	indica algumas das melhorias que se podem introduzir no sistema, de forma a torná-lo mais eficiente.
Avaliação:	propõe formas de avaliação do sistema que permitam a medição do desempenho do sistema e a futura comparação com o sistema resultante da introdução das melhorias propostas anteriormente.

4.1. Terminologia

Para evitar ambiguidades, esclarece-se desde já o significado de alguns termos:

Etiqueta – denomina um conjunto de pares atributo/valor usado para caracterizar morfologicamente uma palavra. Por exemplo, o par `CAT/v` pode ser usado para indicar que uma palavra pertence à categoria dos verbos.

Descrição – denomina a associação de um lema com uma etiqueta. Por exemplo: `[cantar CAT/v TMP/pres MOD/indicativo PES/2].`

Segmento – é um sinónimo para um símbolo (*token*), ou seja uma palavra simples ou composta. Por exemplo, `canto`, `lava-loiça` e `lua de mel`.

Segmento Etiquetado (ou forma etiquetada) – denomina a associação de um segmento com uma ou mais descrições. Por exemplo:

```
Canto     [cantar CAT/v TMP/pres MOD/indicativo PES/2]
          [canto  CAT/nc]
```

Segmento Ambíguo – denomina a associação de vários segmentos que representam vários significados para um conjunto de palavras.

Por exemplo: a expressão “cerca de” pode ter dois possíveis significados como se pode ver pela frase “uma cerca de madeira com cerca de 20 metros”.

O primeiro (“cerca de madeira”) corresponde a um nome seguido de uma preposição:

```
cerca     [cerca CAT/nc1 NUM/s GEN/f]
          [cerca CAT/adv]
          [cercar CAT/v NUM/_ GEN/_ PES/_ CAS/raiz]
          [cercar CAT/v NUM/s GEN/_ PES/3 CAS/ind TMP/pre]
          [cercar CAT/v NUM/s GEN/_ PES/2 CAS/itfp]
de        [de CAT/prep]
```

O segundo (“cerca de 20 metros”) corresponde a uma locução adverbial composta por duas palavras:

```
cerca de [cerca de, CAT/adv]
```

Ao nível morfológico não é possível resolver a ambiguidade, daí a existência de segmentos deste tipo.

Regra – é uma descrição de uma acção a fazer sobre um segmento ou sobre uma lista de segmentos. Representa uma substituição. Num caso (regras de recomposição) é a substituição de segmentos por outros, noutro (regras de correspondência) é a substituição de descrições por listas de *strings*.

Regra Activa – é uma regra que já emparelhou com os últimos *n* segmentos do texto e que ainda está à espera de segmentos para poder estar em condições de ser aplicada. Estas regras são utilizadas pelo algoritmo para isolar as substituições relativas a cada emparelhamento e aplicação de cada regra.

Regra Reduzida – é uma regra que está em condições morfológicas de ser aplicada, isto é, já encontrou segmentos com os quais emparelha e que só ainda não foi aplicada para garantir que as regras são aplicadas pela ordem que aparecem no ficheiro de regras e que os segmentos são substituídos de acordo com a ordem com que aparecem no ficheiro de texto.

Regra Disparada – é uma regra reduzida que está em condições de ser aplicada, isto é, todas as regras que lhe são anteriores no ficheiro de regras e/ou relativas a segmentos anteriores aos que foram utilizados para a reduzirem.

4.2. Arquitectura do PAsMo

O sistema é composto por vários processos que actuam sequencialmente sobre os dados de entrada para produzir o resultado desejado.

Comece-se por observar a Figura 3 onde se apresenta um diagrama do PAsMo.

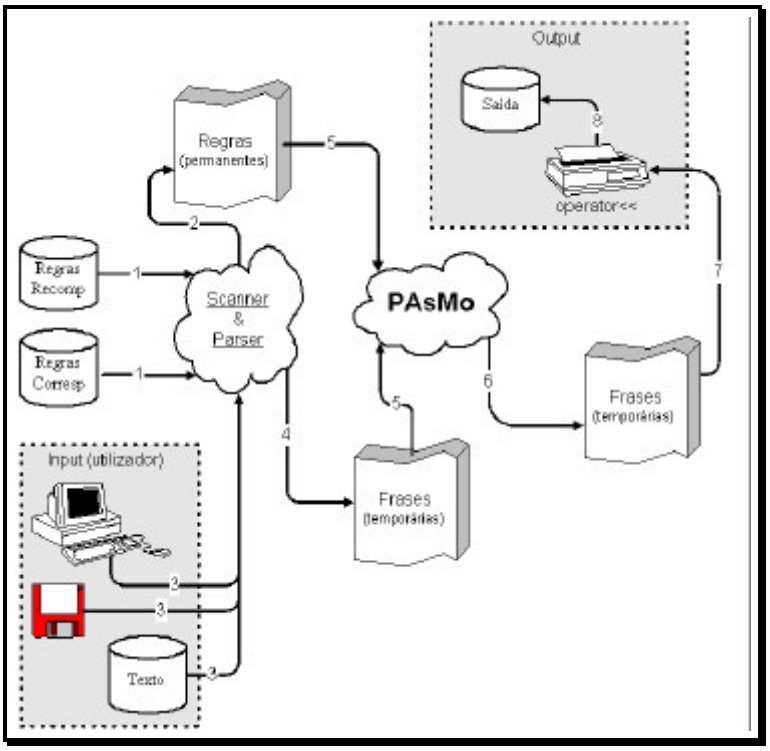


Figura 3. Arquitectura do PAsMo

O sistema começa por consultar o ficheiro com os parâmetros escolhidos pelo utilizador. Durante essa leitura, adquire informação relativa às opções que condicionam o seu funcionamento[25].

Depois, ainda na fase de inicialização, são lidas, a partir dos ficheiros indicados pelo utilizador, as regras de recomposição de correspondência. O *scanner* e o *parser* encarregam-se de transformar os dados em estruturas de dados que, posteriormente, vão ser utilizadas pelo PAsMo. As regras são mantidas durante todo o funcionamento do sistema se bem que, na interface final do sistema, possam ser alteradas durante a sua utilização. É nesta fase que, posteriormente, serão feitos alguns cálculos que facilitem o processamento ao algoritmo. A maior parte dos ganhos de desempenho que o algoritmo pode ter no processamento de *corpus*, e portanto no processamento das frases, implicarão um aumento do processamento e do tempo dispendido nesta fase. As alterações possíveis e a forma como os ganhos podem ser medidos são descritas nas secções 4.7 e 4.8.

Uma vez inicializado o sistema, pode-se começar a processar o texto que o utilizador dá ao sistema. A entrada dos dados pode ser feita através dos meios usuais de inserção de dados. Para cada segmento de texto e enquanto não chega ao fim da frase, o sistema pede ao *scanner* e ao *parser* que lhe forneçam dados. Uma vez obtida uma frase o PAsMo utiliza as regras que conhece para modificar e processar a frase recebida.

Quando termina o processamento de uma frase, o sistema encarrega-se de escrever o resultado num formato adequado no ficheiro de saída indicado pelo utilizador.

[25] cf. a secção 4.3.1 deste documento para mais informações sobre o ficheiro de parametrização.

4.3. Dados de entrada

4.3.1. Ficheiro com parametrizações

Este ficheiro permite parametrizar o algoritmo de conversão:

- Número de passagens em que se tentam aplicar as regras de recomposição[26] ("*" = até que não haja regras disparadas; "n"= n passagens);
- Produção de estatísticas do processamento efectuado;
- Separadores a usar na saída (UNIX, Windows, MAC-OS, …);
- Língua (Português, Francês, Inglês) usada no ficheiro de saída;
- Só faz verificação sintáctica dos ficheiros de dados a processar;
- Só faz verificação sintáctica dos ficheiros de regras;
- Só faz verificação sintáctica do ficheiro de separadores;
- Pára logo que encontre um erro sintáctico num dos ficheiros de entrada;
- Símbolo a usar para agrupar pares atributo/valor nos ficheiros de regras;
- Símbolo a usar para agrupar pares atributo/valor no ficheiros de dados a processar;
- Faz divisão da saída em frases;
- Marcas sintácticas para início e fim de frases;

Neste documento assume-se que o carácter usado para agrupar pares atributo/valor é o "/" e que em todos os ficheiros são reconhecidos como comentários todas as linhas que têm na primeira coluna um dos seguintes símbolos "%", "#" e "//".

No entanto, no sistema PAsMo, alguns desses caracteres são configuráveis pelo utilizador. Podem definir-se os símbolos de separação e de agrupamento dos pares atributos/valor nos dados, nas regras de recomposição e nas regras de correspondência.

- ***Ficheiro de segmentos etiquetados a processar***

O ficheiro de segmentos etiquetados a processar é um ficheiro de caracteres e representa o resultado obtido após a análise morfológica. Para simplificar a linguagem, este ficheiro vai passar a ser denominado simplesmente por "ficheiro de entrada".

Quando for detectado um erro sintáctico essa informação deve ser adicionada ao ficheiro de log.[27]

- ***Sintaxe***

```
<Seg_etiquetado>+28
```

[26] cf. a secção 4.3.2 deste documento para mais informações sobre regras de recomposição.

[27] O ficheiro de log é uma das saída do PAsMo, ver a secção 4.4.2.

em que:

<Seg_etiquetado> ::= <Palavra> ”.” <NL> <Info_morfo> <NL> <NL>

<Info_morfo> ::= <Lista_descrições>+ | <Palavra_des>

<Lista_descrições> ::= ”[” <separador> <Palavra> <Att_val>+ ”].” <NL>

<Att_val> ::= ”,” <Separador> <Palavra> <SepAV> <Palavra>

<Palavra_des> ::= ”[” <Separador> <Palavra> ”,” <Separador> ”mi].” <NL>

<Palavra> ::= ”’” *string* ”’”

<Separador> ::= *espaço* | *tab*

<SepAV> ::= *carácter especificado na parametrização (normalmente ”/” ou ”,”)*

<NL> ::= *newline*

- ***Semântica***

Cada segmento etiquetado (**`Seg_etiquetado`**) representa um símbolo identificado pelo analisador morfológico. Na primeira linha é apresentado o segmento e nas linhas seguintes as possíveis descrições (uma por linha) se a palavra é conhecida, ou a indicação de que o segmento é desconhecido.

Por exemplo, o seguinte segmento etiquetado indica que `Mamede` é um segmento desconhecido do analisador morfológico (“`mi`” = *mot inconnu*):

```
'Mamede'.
[ 'Mamede', mi].
```

O próximo segmento etiquetado indica que ao segmento `quentes` estão associadas duas descrições contendo, respectivamente, os lemas `quentar1` e `quentar2`. Cada uma das descrições é constituída por um lema ao qual estão associados 5 atributos (`CAT`, `NUM`, `GEN`, `PES` e `MOD`) que têm o mesmo valor, excepto para o atributo `MOD` (valor `sub` e valor `itfn`). O valor do atributo `GEN` é o símbolo “`_`”, usado para representar um valor desconhecido (semântica idêntica à usada pela linguagem Prolog):

```
'quentes'.
[ 'quentar1', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'2', 'MOD'/'sub'].
[ 'quentar2', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'2', 'MOD'/'itfn'].
```

- ***Ambiguidades de segmentação***

É possível encontrar, no ficheiro de entrada, casos de ambiguidades de segmentação. Esta ocorrência indica que existem alternativas para a segmentação de uma sequência de palavras. Por exemplo, a sequência “`cerca de`” deve corresponder a dois segmentos em “`Uma` **`cerca de`** `madeira`” e a um só segmento em “`Ganho` **`cerca de`** `10 contos`”. Este tipo de ambiguidade é representada usando as sequências de caracteres:

`'###'. <inteiro>.`

para marcar as várias hipóteses de segmentação e a sequência:

`'###'. '/'.`

---

[28] Usa-se o símbolo “`+`” para representar a ocorrência de um ou mais elementos e o símbolo “`*`” para representar a ocorrência de zero ou mais elementos. Usam-se parêntesis (“`(`” e “`)`”) para indicar opcionalidade e usa-se o símbolo “`|`” para indicar alternativa.

para indicar o fim da ambiguidade de segmentação.

Exemplo de segmentos com ambiguidade

```
'###'. 1.

'cerca de'.
[ 'cerca_de', 'CAT'/'adv'].


'###'. 2.

'cerca'.
[ 'cerca', 'CAT'/'nc1', 'NUM'/'s', 'GEN'/'f'].
[ 'cerca', 'CAT'/'adv'].
[ 'cercar', 'CAT'/'v', 'NUM'/_, 'GEN'/_, 'PES'/_, 'CAS'/'raiz'].
[ 'cercar', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'3', 'CAS'/'ind', 'TMP'/'pre'].
[ 'cercar', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'2', 'CAS'/'itfp'].

'de'.
[ 'de', 'CAT'/'prep'].

'###'. '/'.
```

- ***Exemplos de segmentos etiquetados***

```
'<TITLE>'.
[ '<TITLE>', 'CAT'/'balise', 'TBAL'/'html', 'DBAL'/'tit_g'].

'Energia'.
[ 'energia', 'CAT'/'n', 'NUM'/'s', 'GEN'/'f', 'TAN'/'t2'].

'nuclear'.
[ 'nuclear', 'CAT'/'a', 'NUM'/'s', 'GEN'/_, 'TAN'/'t3'].
[ 'nuclear', 'CAT'/'v', 'MOD'/'inf'].
[ 'nuclear', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'1', 'MOD'/'inf'].
[ 'nuclear', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'3', 'MOD'/'inf'].
[ 'nuclear', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'1', 'MOD'/'sub', 'TMP'/'fut'].
[ 'nuclear', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'3', 'MOD'/'sub', 'TMP'/'fut'].
[ 'nuclear', 'CAT'/'v', 'NUM'/_, 'GEN'/_, 'PES'/_, 'MOD'/'raiz'].

'</TITLE>'.
[ '</TITLE>', 'CAT'/'balise', 'TBAL'/'html', 'DBAL'/'tit_d'].

'<P>'.
[ '<P>', 'CAT'/'balise', 'TPT'/'parag', 'TBAL'/'html'].

'Jacques'.
[ 'Jacques', mi].

'Outros'.
[ 'outro', 'CAT'/'pr_i', 'NUM'/'p', 'GEN'/'m', 'TPI'/'outro'].

'dois'.
[ 'dois', 'CAT'/'nm', 'NUM'/'p', 'GEN'/'m'].

'pontos'.
[ 'ponto', 'CAT'/'n', 'NUM'/'p', 'GEN'/'m', 'TAN'/'t1'].

'quentes'.
[ 'quentar', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'2', 'MOD'/'sub', 'TMP'/'pre'].
[ 'quentar', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'2', 'MOD'/'itfn'].
```

4.3.2. Ficheiro com regras de recomposição

As regras de recomposição têm como finalidade alterar a segmentação presente no ficheiro de entrada por agrupamento de segmentos etiquetados ou por transformação de um (ou mais) segmento(s) etiquetado(s) em vários segmentos etiquetados.

- ***Sintaxe***

```
<Regra_recomp>+
```

Em que:

```
<Regra_recomp> ::= <Regra_recomp_si> | <Regra_recomp_ci>
<Regra_recomp_si> ::= <descrip_morfo>+ "-->" <parte_direita> "."
<Regra_recomp_ci> ::= <descr_morf_iter> "-->" <part_dir_iter> "."
<descrip_morfo> ::= (<const_ou_var>) <info_morfo>
<info_morfo> ::= <lema_att_val> | <palavra_des>
<palavra_des> ::= "[" <Palavra> "," <separador> "mi]"
<lema_att_val> ::= "[" <const_ou_var> <att_val_ou_var>+ "]"
<att_val_ou_var> ::= "," <Palavra> <SepAV> <const_ou_var>
<descr_morf_iter> ::= (<descrip_morfo>) (<const_ou_var>) <info_morfo> <iter>
                      (<descrip_morfo>)
<iter> ::= número inteiro "+"
<parte_direita> ::= <segmento_result>+
<segmento_result> ::= <sig_res> <lema_att_val>
<sig_res> ::= <const_ou_var> | <concat_var>
<lema_att_val> ::= "["
                   <const_ou_var> | <concat_var>
                   <att_val_1_ou_mais>
                   "]"
<att_val_1_ou_mais> ::= <Par_AV> (<Par_AV> "," att_val_1_ou_mais)
<part_dir_iter> ::= <var_iter> <lema_att_val_iter>
<var_iter> ::= <variavel> <iterador>
<iterador> ::= "^" | "$" | "*" | "@*"
<lema_att_val_iter> ::= "["
                        <var_iter> | <Palavra>
                        <att_val_1_ou_mais>
                        "]"
<const_ou_var> ::= <Palavra> | <variavel>
<concat_var> ::= <variavel> ("+" <concat_var>)
<Par_AV> ::= <Palavra> <SepAV> <Palavra>
<Palavra> ::= "'" string "'"
<variavel> ::= string cuja primeira letra é maiúscula | underscore
<SepAV> ::= carácter especificado na parametrização (normalmente "/" ou ",")
```

As variáveis que ocorrem na parte direita de uma regra também devem ocorrer na parte esquerda dessa regra.

- ***Semântica***

As regras de recomposição (**Regra_recomp**) têm por finalidade modificar a segmentação feita pelo analisador morfológico. Estas regras são constituídas por duas partes, separadas por um símbolo separador "`-->`", e denominadas parte direita (à direita do símbolo separador) e parte esquerda (à esquerda do símbolo separador).

As regras de recomposição aplicam-se a segmentos etiquetados e criam novos segmentos etiquetados a partir dos primeiros. O lado esquerdo define as condições de aplicabilidade da regra: quando o padrão especificado pelo lado esquerdo emparelhar com uma sequência de segmentos etiquetados pode-se substituir essa sequência de segmentos etiquetados, pela sequência de segmentos etiquetados descritos no lado direito da regra.

Por exemplo, o segmento etiquetado cujo segmento é "`nas`" pode ser substituído por dois novos segmentos etiquetados associados aos segmentos "`em`" e "`as`" (recomposição de 1 para 2).

Se o analisador morfológico tiver delimitado três segmentos etiquetados cujos segmentos são respectivamente "`2`", "`de`" e "`Janeiro`", uma regra de recomposição pode agrupar estes três segmentos etiquetados num só, e associar ao segmento etiquetado obtido o segmento: "`2 de Janeiro`" (recomposição de 3 para 1).

A partir da sequência dos três segmentos etiquetados contendo os segmentos "`cantá`", "`-lo`", "`-ei`" (resultado da segmentação da cadeia de caracteres "`cantá-lo-ei`" pelo analisador morfológico) é possível obter, por recomposição, dois novos segmentos etiquetados contendo respectivamente os segmentos "`cantar-ei`" e "`-lo`", (recomposição de 3 para 2).

Embora não apresente nenhuma aplicação prática vantajosa, é possível imaginar regras de recomposição de 1 para 1, não havendo alteração na segmentação mas podendo, no entanto, a lista de descrições associadas ao segmento ser modificada.

É também importante notar que o resultado da aplicação das regras de recomposição não altera a sintaxe dos segmentos etiquetados (*i.e.*, cada segmento etiquetado vai continuar a ser constituído por um segmento acompanhado por uma ou várias descrições, e cada descrição composta por um lema associado a uma lista de pares atributo/valor).

Aplicabilidade das regras de recomposição

Uma regra de recomposição, cuja parte esquerda é constituída por *n* formas etiquetadas, é aplicável a uma sequência de *n* formas etiquetadas quando: existe um emparelhamento de cada uma das formas etiquetada pertencente à parte esquerda da regra com a correspondente forma etiquetada da sequência.

A palavra "correspondente" deve ser entendida como a especificação de que a primeira forma etiquetada do esquema emparelha com a primeira forma etiquetada da sequência, a segunda com a segunda, e assim por diante.

Por exemplo, a regra:

```
S1 [L1,'CAT'/'pref'] S2 [L2,'CAT'/'nc','NUM'/N,'GEN'/G] -->
      S1+S2 [L1+L2,'CAT'/'nc','NUM'/N,'GEN'/G].
```

é aplicável à seguinte sequência de formas etiquetadas:

```
'ex-'.
[ 'ex', 'CAT'/'pref', 'PREF'/'ex'].

'marido'.
[ 'marido', 'CAT'/'nc', 'NUM'/'s', 'GEN'/'m', 'TAN'/'t1'].
[ 'maridar', 'CAT'/'v', 'PES'/'1', 'TMP'/'pres', 'MOD'/'ind'].
```

Uso de variáveis no emparelhamento

Na expressão das regras de recomposição é possível usar variáveis (uma sequência de caracteres iniciado por uma maiúscula). No exemplo precedente, `S1`, `L1`, `S2`, `L2`, `N` e `G` representam variáveis. A interpretação que se deve dar a estas variáveis é idêntica à interpretação das variáveis em Prolog: a etiqueta `'NUM'/N` emparelha com qualquer etiqueta que tenha como atributo `'NUM'`, independentemente do valor que lhe esteja associado.

Aplicação de regras de recomposição

O resultado da aplicação de uma regra de recomposição a uma sequência de formas etiquetadas, é descrito na parte direita da regra. O valor das variáveis que possam ocorrer no lado direito das regras é calculado durante o emparelhamento de padrões associado à verificação das condições de aplicabilidade da regra.

Durante o emparelhamento entre uma parte esquerda de uma regra de recomposição e uma sequência de formas etiquetadas, as eventuais variáveis presentes na parte esquerda deverão tomar valores a serem usados no cálculo da parte direita.

Por exemplo, a regra:

```
'apesar dele' [L1, 'CAT'/'pper', 'NUM'/'s', 'GEN'/'m'] -->
    'apesar de' ['apesar_de','CAT'/'prep']
    'ele' ['ele','CAT'/'per','CAS'/'nom','NUM'/'s','GEN'/'m'].
```

é aplicável à seguinte sequência (unitária):

```
'apesar dele'.
[ 'apesar dele', 'CAT'/'pper', 'NUM'/'s', 'GEN'/'m'].
```

Dando origem às duas formas etiquetadas seguintes:

```
'apesar de'.
['apesar_de','CAT'/'prep'].

'ele'.
['ele','CAT'/'per','CAS'/'nom','NUM'/'s','GEN'/'m'].
```

Concatenação de segmentos ou lemas

A presença de um "`+`" entre variáveis da parte direita deve ser entendido como a concatenação das cadeias de caracteres que correspondem à instanciação destas variáveis.

Esta operação só pode ser usada para expressar concatenação de segmentos ou lemas. Por exemplo, a regra:

```
S1 [L1,'CAT'/'pref'] S2 [L2,'CAT'/'nc','NUM'/N,'GEN'/G] -->
    S1+S2 [L1+L2,'CAT'/'nc','NUM'/N,'GEN'/G].
```

é aplicável à sequência:

```
'ex-'.
[ 'ex', 'CAT'/'pref', 'PREF'/'ex'].

'marido'.
[ 'marido', 'CAT'/'nc', 'NUM'/'s', 'GEN'/'m', 'TAN'/'t1'].
[ 'maridar', 'CAT'/'v', 'PES'/'1', 'TMP'/'pres', 'MOD'/'ind'].
```

dando origem ao segmento etiquetado:

```
'ex-marido'.
['exmarido', 'CAT'/'nc', 'NUM'/'s', 'GEN'/'m'].
```

Quando se pretender adicionar um espaço entre cada duas cadeias de caracteres faz-se preceder o símbolo "+" do símbolo "@". Compare-se com atenção este exemplo com o anterior:

```
regra:  S1 [L1,'CAT'/'pref'] S2 [L2,'CAT'/'nc','NUM'/N,'GEN'/G] -->
            S1@+S2 [L1@+L2,'CAT'/'nc','NUM'/N,'GEN'/G].

sequência:    'ex-'.
              [ 'ex', 'CAT'/'pref', 'PREF'/'ex'].

              'marido'.
              [ 'marido', 'CAT'/'nc', 'NUM'/'s', 'GEN'/'m', 'TAN'/'t1'].
              [ 'maridar', 'CAT'/'v', 'PES'/'1', 'TMP'/'pres', 'MOD'/'ind'].

resultado:    'ex- marido'.
              ['exmarido', 'CAT'/'nc', 'NUM'/'s', 'GEN'/'m'].
```

- ***Recomposição com iteração***

É possível expressar na parte esquerda da regra que se pretendem agrupar vários segmentos etiquetados que tenham etiquetas idênticas. Este tipo de recomposição é designado por recomposição com iteração. Para indicar este desejo usa-se, à direita do(s) segmento(s) etiquetado(s) a iterar, um número inteiro seguido de um "+".

Por exemplo a parte esquerda da regra:

```
S1 [L1,'CAT'/'nm']1+ S2 [L2,'TIPO'/'medida'] -->
  S1@*+S2 [L1@*+L2,'CAT'/'nm_med','NUM'/'p'].
```

deve ser interpretada como uma sequência de uma ou mais formas etiquetadas, seja qual for o segmento associado mas que contêm uma descrição que emparelha com `[L1,'CAT','nm']` seguida por uma forma etiquetada que contém uma descrição que emparelha com `['km', 'CAT'/'n']`.

Assim, a parte esquerda desta regra emparelha com a seguinte sequência de 4 formas etiquetadas:

```
'5'.
['5', 'CAT'/'nm'].

'3'.
['3', 'CAT'/'nm'].

'7'.
['7', 'CAT'/'nm'].

'km'.
['km', 'CAT'/'n', 'TIPO'/'medida'].
```

O cálculo do resultado da recomposição com iteração introduz novos símbolos que podem aparecer associados a variáveis na parte direita das regras e que permitem

especificar a concatenação de variáveis pretendida. Os três novos símbolos são "^", "$" e "*" e têm a seguinte interpretação (V é uma variável):

V^: Instancia V^ com o valor do primeiro emparelhamento de V (ocorrido durante o emparelhamento da sequência de formas etiquetadas com a parte esquerda da regra).

V$: Instancia V$ com o valor do primeiro emparelhamento de V.

V*: Instancia V* com a concatenação (sem espaço intermédio) de todas as instanciações de V, conservando a ordem de instanciação.

Tal como anteriormente, o símbolo "@" deve ser usado sempre que se pretender a introdução de um espaço entre as cadeias de caracteres a concatenar:

V@*: Instancia V@* com a concatenação (com um espaço intermédio) de todas as instanciações de V conservando a ordem de instanciação.

Assim, considerando a regra de recomposição com iteração e a sequência de formas etiquetadas acima mencionadas, o resultado da aplicação da regra é:

```
'537 km'.
['537 km', 'CAT'/'nm'].
```

Com efeito, a expressão "S1@*" tem o valor '537', *i.e.*, a concatenação (sem espaço) de todas as instanciações da variável "S1" que ocorreram durante o emparelhamento da sequência de formas etiquetadas com a parte esquerda da regra. As expressões "S1@*+S2" e "L1@*+L2" depois de avaliadas foram associadas com '537 km'.

- ***Exemplos de regras de recomposição***

```
S1 [L1, mi] S2 [L2, mi] S3 [L3, mi] S4 [L4, mi] -->
   S1+S2+S3+S4 [L1+L2+L3+L4, 'CAT'/'npr', 'TAN'/'t4'].

S1 [L1,'CAT'/'pref'] S2 [L2,'CAT'/'v','NUM'/N,'GEN'/G,'PES'/P,'MOD'/M,'TMP'/T]
   --> S1+S2
   [L1+L2,'CAT'/'v','NUM'/N,'GEN'/G,'PES'/P,'MOD'/M,'TMP'/T,'TRACE'/'prefixe'].

S1 [L1,'CAT'/'nm']1+ S2 [L2,'TIPO'/'medida'] -->
   S1*+S2 [L1*+L2,'CAT'/'nm_med','NUM'/'p'].

S1 [L1,'CAT'/'nm']1+ --> S1* [L1*, 'CAT'/'nm','NUM'/'p'].

S1 [L1,'CAT'/'v_cli'] -lo [L2,'CAT'/C,'CAS'/C2]
   S3 [L3,'CAT'/'termv','NUM'/N,'GEN'/G,'PES'/P,'MOD'/M,'TMP'/T] -->
   S1+S3 [L1, 'CAT'/'v','NUM'/N,'GEN'/G,'PES'/P,'MOD'/M,'TMP'/T]
   –lo [L2,'CAT'/C,'CAS'/C2].

S1 [L1,'CAT'/'nm'] S2 ['milhar','CAT'/'n'] -->
   S1+S2 ['num milhares', 'CAT'/'nm','TRACE'/'milhar','NUM'/'p'].

S1 ['ter','CAT'/'v','NUM'/N,'GEN'/G,'PES'/P,'MOD'/M,'TMP'/T]
   S2 ['ter', 'CAT'/'v','NUM'/'s','GEN'/'m','MOD'/'par','TMP'/'pas']
   S3 [L3, 'CAT'/'n','NUM'/'s','TAN'/'t2'] -->
   S1+S2+S3 ['ter
   N','CAT'/'v','NUM'/N,'GEN'/G,'PES'/P,'MOD'/M,'TMP'/T,'TRACE'/'ter'].

S1 [L1,'CAT'/'pref'] S2 [L2,'CAT'/'v','NUM'/N,'GEN'/G,'PES'/P,'MOD'/M,'TMP'/T]
   S3 [L3,'CAT'/'v','NUM'/N,'GEN'/G,'PES'/P,'MOD'/M,'TMP'/T] -->
   S1+S3
```

```
   [L1+L3,'CAT'/'v','NUM',N/'GEN',G/'PES',P/'MOD',M/'TMP',T/'TRACE','prefixe']
   S2 [L2, 'CAT'/'n'].

['apesar_de','CAT'/'partd','NUM'/'p','GEN'/'m'] -->
   'apesar de' ['apesar_de','CAT'/'prep']
   'os' ['o','CAT'/'artd','NUM'/'p','GEN'/'m'].
```

4.3.3. Ficheiro com regras de correspondência

As regras de correspondência têm como finalidade permitir alterar descrições associadas a segmentos etiquetados.

- ***Sintaxe***

<Regra_corresp>+

Em que:

<Regra_corresp> ::= <lista_att_val> "-->" <Lista_*strings*> "."

<lista_att_val> ::= "[" (<traço_lema>) <att_val>+ "]"

<att_val> ::= "," <Palavra> <SepAV> <Palavra>

<traço_lema> ::= "'LEM'" <SepAV> <Palavra>

<Palavra> ::= "'" *string* "'"

<SepAV> ::= *carácter especificado na parametrização (normalmente "/" ou ",")*

<Lista_*strings*> ::= *string* | <Lista_*strings*> *string*

- ***Semântica***

As regras de correspondência (**`Regra_corresp`**) permitem transformar etiquetas com o formato conhecido pelo analisador morfológico, noutras etiquetas. Estas regras são constituídas por duas partes, separadas por um símbolo separador "`-->`", e denominadas parte direita (à direita do símbolo separador) e parte esquerda (à esquerda do símbolo separador).

O lado esquerdo destas regras pode ser uma descrição ou uma etiqueta e define as condições de aplicabilidade da regra: quando o padrão especificado pelo lado esquerdo de uma regra emparelhar com uma determinada descrição pode-se substituir a etiqueta dessa descrição, pelo lado direito da regra (uma cadeia de caracteres).

A representação de uma descrição no lado esquerdo de uma regra faz-se através de uma sintaxe própria: o lema tem de ser precedido do símbolo `'LEM'`. Por exemplo, o lado esquerdo da regra:

```
['LEM'/'ser','MOD'/'inf'] --> 'copinf'.
```

representa uma descrição que tem como lema `'ser'` e como etiqueta um único par atributo/valor (`'MOD'/'inf'`).

Lado esquerdo é uma etiqueta

Quando o lado esquerdo de uma regra é uma etiqueta, essa regra é aplicável a uma descrição se a descrição incluir essa etiqueta, ou seja, incluir todos os pares atributo/valor presentes na parte esquerda da regra. Por exemplo, a regra:

```
['CAT'/'v','NUM'/'s','MOD'/'inf'] --> 'CAT'/'v','MOD'/'inf'.
```

quando aplicada ao segmento etiquetado:

```
[ 'ser', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'2', 'MOD'/'inf'].
```

produz o segmento etiquetado:

```
[ 'ser', 'CAT'/'v', 'MOD'/'inf'].
```

Lado esquerdo é uma descrição

Quando o lado esquerdo de uma regra é uma descrição, essa regra é aplicável se existir uma descrição que inclua esse lema e inclua a etiqueta descrita no lado esquerdo da regra, ou seja, quando existir uma descrição que tenha o mesmo lema do especificado no lado esquerdo e que simultaneamente inclua todos os pares atributo/valor especificados na parte esquerda da regra. Por exemplo, a regra:

```
['LEM'/'ser','MOD'/'inf'] --> 'copinf'.
```

pode ser aplicada à descrição:

```
[ 'ser', 'CAT'/'v', 'NUM'/'s', 'GEN'/_, 'PES'/'2', 'MOD'/'inf'].
```

dando origem à nova descrição:

```
[ 'ser', 'copinf'].
```

- ***Exemplos de regras de correspondência***

```
['CAT'/'n','TAN'/'t1','NUM'/'s'] --> ‘@macro_nome’.
['CAT'/'n','TAN'/'t3','NUM'/'s'] --> 'CAT' 'n' 'NUM' 's'.
['LEM'/'ser','MOD'/'inf'] --> 'copinf'.
```

4.3.4. Ficheiro de separadores

- ***Sintaxe***

Um separador pode ser descrito por uma cadeia de caracteres ou uma sequência de inteiros (separados por um espaço) representando uma sequência de códigos ASCII.

- ***Semântica***

Os segmentos etiquetados descritos no ficheiro de separadores indicam os segmentos que vão delimitar o texto inicial em “pedaços” que serão depois fornecidos a outro programa para serem processados (tipicamente um analisador sintáctico). Estes pedaços são habitualmente denominados frases. Entre estes separadores é usual considerar os símbolos de pontuação que delimitam fins de frases (como o ponto final e o ponto de exclamação) e os delimitadores de início e fim de títulos.

A delimitação em frases faz-se usando os separadores de frases, ou seja, sempre que se identifica um separador no ficheiro de entrada, termina-se uma frase e dá-se início a uma nova frase.

- ***Exemplo de um ficheiro de separadores***

```
'.'
'?'
'!'
';'
046 046 046
'</TITLE>'
'</H1>'
'<BR>'
```

4.4. Dados de saída

4.4.1. Ficheiro de saída

Existem várias alternativas para a apresentação da computação efectuada:

- uma sequência de segmentos etiquetados com uma sintaxe semelhante à do ficheiro de dados de entrada;
- uma sequência de segmentos etiquetados com uma sintaxe alterada: cada segmento etiquetado é compactado num única linha;
- uma sequência de segmentos etiquetados com uma sintaxe semelhante à do ficheiro de dados de entrada, mas onde são introduzidas marcas sintácticas no início e no fim de cada fase (definida através do ficheiro de separadores);
- uma sequência de segmentos etiquetados com uma sintaxe alterada (cada segmento etiquetado é compactado num única linha) e em que são introduzidas marcas sintácticas no início e no fim de cada fase.

De seguida apresenta-se a sintaxe (AF) para a apresentação dos segmentos etiquetados calculados onde se pretende conhecer o início e fim de frases.

- ***Sintaxe identificação de frases***

<Saida_frase>+

Em que:

<Saida_frase> ::= "[" <NL> hipotese_fr>+ <NL> "]"

<hipotese_fr> ::= <Separador> "[" <NL> <Varios_seg_etiquet> "]"

<Varios_seg_etiquet> ::= string contendo a descrição de segmentos etiquetados

<Separador> ::= *string a definir no ficheiro de parametrização*

<NL> ::= *newline*

Para cada frase são apresentadas todas as combinações de eventuais ambiguidades de segmentação que possam ter sido consideradas.

Exemplo do formato AF (com separação de frases)

```
[
 [
 'Energia' , ['energia', 'nc_tit'],
 'nuclear' , ['nuclear', 'adj3_s', 'nuclear', 'vinf_s', 'nuclear', 'eliminer'],
 '</TITLE>' , ['</TITLE>', 'eliminer']
 ]
] % fim da frase número 1
[
 [
 '<P>' , ['<P>', 'eliminer'],
 'Outros' , ['outro', 'outro'],
 'dois' , ['dois', 'card_p'],
 'pontos' , ['ponto', 'nc1_p'],
 'quentes' , ['quentar', 'vc12','quentar', 'eliminer'],
 'de' , ['de', 'prep'],
 'a' , ['o', 'artd_s'],
```

```
'agenda' , ['agenda', 'nc1_s','agendar', 'vc','agendar', 'eliminer'],
'são' , ['são', 'adj3_s','ser', 'copc'],
'a' , ['o', 'artd_s','a', 'prep','a', 'cli_ac'],
'construção' , ['construção', 'nc1_s'],
'aeronaútica' , ['aeronaútica', 'adj1_s','aeronaútica', 'nc1_s'],
'europeia' , ['europeu', 'adj3_s'],
'e' , ['e', 'coord'],
'a' , ['o', 'artd_s','a', 'prep','a', 'cli_ac'],
'questão' , ['questão', 'nc1_s'],
'de' , ['de', 'prep'],
'o' , ['o', 'artd_s'],
'uso' , ['uso', 'nc1_s','usar', 'vc12'],
'de' , ['de', 'prep'],
'a' , ['o', 'artd_s'],
'energia' , ['energia', 'nc2_s'],
'nuclear' , ['nuclear', 'adj3_s','nuclear', 'vinf_s','nuclear', 'eliminer'],
'para' , ['para', 'prep','parar', 'eliminer'],
'aplicações' , ['aplicação', 'nc1_p'],
'civis' , ['civil', 'adj3_p'],
'.' , ['.', 'eliminer']
]
] % fim da frase número 2
```

4.4.2. Ficheiro de log

Sempre que for detectado um erro num dos ficheiros de entrada, uma descição do erro deve ser registada:

- Nome do ficheiro onde foi encontrado o erro;
- Número da linha do ficheiro onde foi encontrado o erro;
- Tipo de erro encontrado.

Quando for pedida uma estatística do processamento efectuado, deve ser apresentado:

- Nome dos ficheiros lidos (com a indicação do número de linhas lidas);
- Relativamente aos ficheiros de entrada e saída:
  - Número de formas etiquetadas;
  - Número médio de lemas por formas etiquetadas;
  - Número de ambiguidades de segmentação;
  - Número médio de alternativas por ambiguidade de segmentação;
  - Número máximo e alternativas nas ambiguidade de segmentação;
- Relativamente aos ficheiros de regras:
  - Número de regras definidas;
  - Número de regras por tipo (1 para 1, 1 para 2, 1 para 3, 2 para 1, …);
  - Número de regras não usadas;
  - Número de utilizações por regra;

- Relativamente ao desempenho do algoritmo:
    - Tempo gasto (elapsed time e CPU time);
    - Número de ciclos pretendidos e efectuados;
    - Número de regras colocadas na agenda; (por ciclo);
    - Número de regras disparadas (por ciclo);
    - Número de concatenações com e sem espaço;
    - Número de regras que poderiam ter disparado mas não o foram por haver outra regra prioritária (por ciclo);

4.5. Algoritmo

O princípio subjacente a este algoritmo é o emparelhamento de padrões, assumindo a existência de variáveis. O algoritmo pode ser esquematizado através da aplicação sucessiva de oito acções:

1. leitura da parametrização
2. leitura das regras de recomposição
3. repete (número de vezes especificadas na parametrização)

   aplicação das regras de recomposição
4. leitura das regras de correspondência
5. aplica regras de correspondência
6. elimina descrições repetidas (num mesmo segmento etiquetado)
7. faz divisão em frases
8. apresenta resultado no formato pedido

A aplicação das regras de recomposição necessita de ser efectuada com bastantes cautelas, enquanto que a aplicação das regras de correspondência não necessita de cuidados especiais.

4.5.1. Aplicação das regras de recomposição

Cuidados a ter

Ter em conta as seguintes recomendações:

- As regras têm de ser aplicadas pela ordem que aparecem no ficheiro (de regras de recomposição).
- É necessário manter uma agenda com as regras que ficam activas.
- A conclusão de uma regra pode implicar a desactivação de regras que estejam na agenda
- A utilização do carácter "+" nas regras de recomposição implica que as regras na agenda podem ser continuadas caso se verifique uma de duas condições.

Exemplo: `a1+ b --> c`, depois do primeiro `a`, a regra pode ser continuada com outro `a` ou com um `b`.

- Para aumentar a eficiência, pode ser necessário indexar as regas por vários parâmetros, como por exemplo: segmento, lema, atributo, valor, atributo/valor.
- Talvez não valha a pena fazer a mesma coisa para as regras que estão na agenda.
- Como o ficheiro de entrada pode ser muito grande, é preferível ir escrevendo, num ficheiro, a saída que for sendo calculada (para evitar o consumo de memória)

4.5.2. Algoritmo de aplicação das Regras de Recomposição

1. Limpa a agenda
2. Lê do ficheiro de entrada uma forma etiquetada F
3. Avançar as regras da agenda, eliminando da agenda as regras que não estão à espera da forma etiquetada F
4. Se a primeira regra da agenda poder ser disparada:
   (i) aplica a regra (faz a substituição correspondente)
   (ii) retira da agenda todas as regras que foram introduzidas na agenda num momento igual ou anterior ao da última forma etiquetada F usada pela regra que foi disparada
5. Se a regra disparada usa F vai para **2**
6. Procura novas regras (não activas) que emparelhem com F
7. Ordena as regras anteriores pela ordem do ficheiro de entrada
8. Adiciona as regras ordenadas ao fim da agenda
9. Vai para **2**.

4.6. Estruturas de dados

Uma das maiores preocupações no desenvolvimento do código foi a sua estruturação, a criação de dados suficientemente abstractos e genéricos e a facilidade de evolução do sistema. Isto porque, o sistema que agora se apresenta ainda não está totalmente completo, isto é, ainda não implementa todas as funcionalidades desejadas, no entanto, quer-se facilitar esse trabalho de aperfeiçoamento que, aliás, nunca se pode dar por terminado.

Assim sendo, as classes foram desenhadas tendo em conta o requisito de eficiência e velocidade do sistema final obtido, mas, ao mesmo tempo, foi dada particular atenção ao seu desenho, preferindo-se um desenho claro e sem ambiguidades dum sistema um pouco mais lento a um sistema veloz onde a evolução fosse dificultada pela falta de estruturação do código.

Para se ter em atenção a clareza do código e a eficiência do sistema, utilizou-se uma linguagem imperativa orientada às classes, o C++.

Comece-se por observar o diagrama UML das classes.

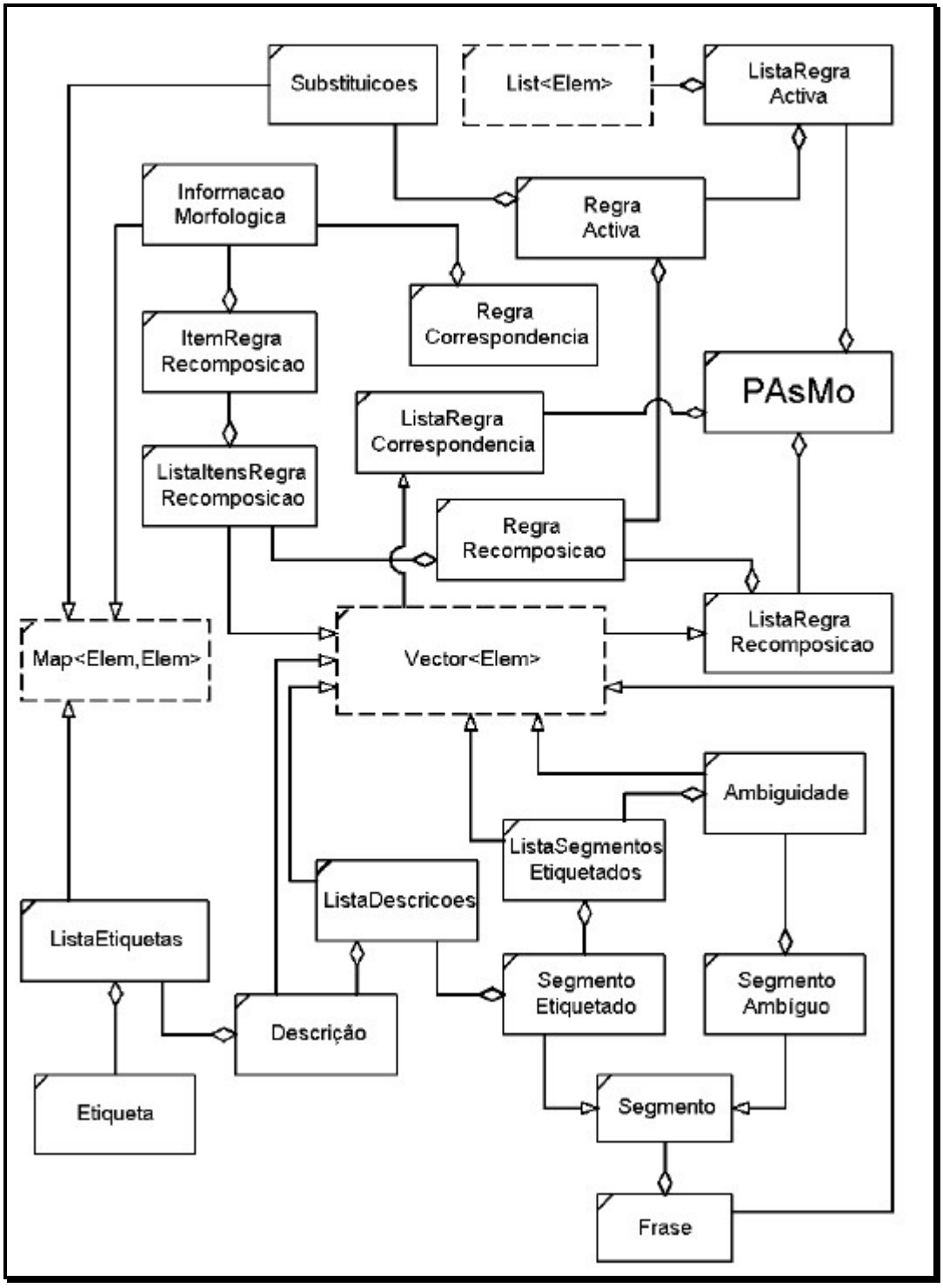


Figura 4. Diagrama UML das classes desenhadas

Os nomes dados às classes foram escolhidos de acordo com os elementos que representam. Assim, a classe “etiqueta” representa um par atributo/valor, a classe “descricao” representa uma descrição e assim sucessivamente.

Sempre que possível utilizaram-se as classes standard da linguagem C++, o "vector", o "map" e o "list".

Além disso utilizou-se como base a classe "Word", já desenhada e implementada anteriormente para outros sistemas que, por ser a base de todas as classes e ser utilizada em todas, se omitiu do diagrama UML para facilitar a sua leitura.

Neste documento não se estende a descrição das estruturas de dados criadas, deixando-se a análise pormenorizada para o relatório técnico com o código comentado que se apresenta em anexo.

4.6.1. Segmentos

Os segmentos são pequenas porções de texto, a estrutura mais pequena com significado completo dentro da frase.

Para facilitar o processamento das frases, e porque numa frase podemos ter segmentos de dois tipos – etiquetados e ambíguos – criou-se uma super classe Segmento da qual as outras duas herdam.

A super classe é totalmente abstracta e, de acordo com a definição da linguagem, obrigam as classes que herdam dela a implementar métodos de impressão ("print" e "show") de expansão (que convertam a classe numa lista de segmentos etiquetados).

Optou-se por representar as etiquetas por "maps" porque, apesar deste tipo de estrutura ser criado e destruído com bastante frequência, aquilo que se vai percorrer sequencialmente é o antecedente e, por isso, é mais simples de utilizar um map.

Um dos trabalhos futuros será medir as variações de desempenho obtidas quando se utiliza "maps" ou "vectors" na implementação de cada uma das classes.

4.6.2. Regras de recomposição

As regras de recomposição são criadas na fase de inicialização e são utilizadas pelo sistema para reescrever os segmentos, reagrupando-os na forma desejada pelo utilizador.

Estas regras vão ser aplicadas sucessivamente aos segmentos pelo que o seu desenho é crucial para o desempenho do sistema.

Também aqui se farão testes de verificação de qual a melhor estrutura base a utilizar: "maps" ou "vectors".

4.6.3. Regras de correspondência

As regras de correspondência só são aplicadas uma vez, utilizam-se na fase de escrita dos resultados e representam a forma como se quer ver agrupada a informação das descrições dos segmentos. Optou-se por desenhar estas regras de forma semelhante às anteriores por serem em tudo semelhantes e para facilitar a correcção do código final.

O desempenho do algoritmo de aplicação destas regras não é crucial para o desempenho final do sistema dada a utilização reduzida que se faz delas. Além disso, por só ser aplicada uma vez a cada segmento, é o próprio PAsMo o responsável pela sua aplicação e pela produção e utilização dos resultados por ela gerados.

4.6.4. Estruturas de apoio ao algoritmo

Para se construírem os dados necessários ao sistema e para se proceder ao processamento dos dados recorre-se a quatro grandes classes auxiliares: o *scanner*, o *parser*, a regra activa e a agenda que se descrevem de seguida.

- **Scanner *e os* Parsers**

Para ler os dados vindos dos ficheiros com as regras de recomposição e de correspondência e com o texto, recorreu-se às classes de objectos que se utilizam tipicamente para esse género de funções: o *scanner* e os *parsers*.

Como o código e os dados a ler têm uma estrutura clara, bem definida, pôde definir-se um scanner genérico para os três tipos de dados, tendo bastado a criação posterior de *parsers* que o utilizem. Para isso utilizou-se o programa "*flex*" que gera automaticamente o código de leitura do dados, a partir de um ficheiro onde se descreva a máquina de estados a utilizar.

As classes dos *parsers* tinham vários métodos em comum devido às suas semelhanças estruturais e conceptuais. Por isso, optou-se por se criar a super classe *parser* genérico e virtual que encapsula os métodos e as estruturas de dados comuns. A geração do código dos *parsers* foi feita recorrendo ao programa "*bison*" que, dada uma gramática regular e a descrição das acções a executar na presença de cada elemento reconhecido, produz o código que constrói as estruturas de dados a partir dos dados lidos pelo scanner. Dessa classe fez-se herdar os *parser* responsáveis pela leitura de cada tipo de dados em particular.

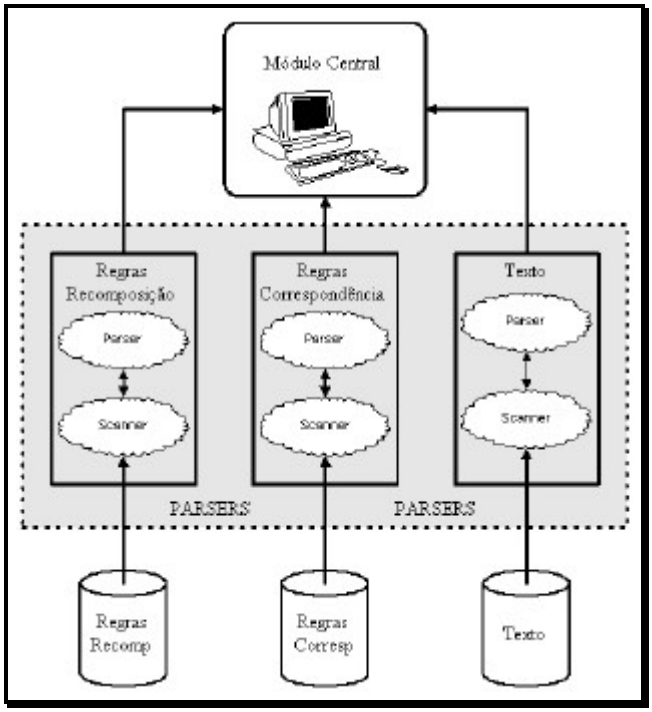


Figura 5. Inserção dos *Scanners* e dos *Parsers* na arquitectura do PAsMo

A figura 5 mostra como é feita a leitura dos dados utilizando estes objectos.

Os parsers desenhados têm, a diferenciá-los de parsers comuns, a possibilidade de recuperação de erros: sempre que uma regra é mal escrita e não se consegue reconstruir aquilo que o utilizador pretendia, é escrito no ficheiro de log o ficheiro e a linha onde o erro foi encontrado e uma breve descrição do erro. Isto permite que o utilizador se engane sem que o sistema tenha que abortar o funcionamento.

Esta característica é particularmente importante se considerarmos a utilização final da ferramenta: em *corpus* linguísticos a probabilidade de haver um erro é elevada e não se quer um sistema frágil ou que não seja robusto a erros, pois não se está em condições de abortar todo o processamento e perder informação apenas porque, por exemplo, o linguista se enganou no carácter que separe os pares atributo/valor numa determinada regra.

- ***Regra Activa***

De cada vez que se tenta aplicar uma regra, é necessário encontrar as substituições que permitem à regra emparelhar com um determinado segmento. Por isso, e para encapsular a informação relativa a cada regra activa, criou-se uma classe especializada que sabe qual a regra que lhe deu origem, quais os segmentos que já foram utilizados pela regra para se desenvolver, quais as substituições feitas e o que falta para se disparar a regra. Como a informação relativa às regras originais é constante e as regras originais permanecem inalteradas durante todo o algoritmo, optou-se por guardar na regra activa apenas um ponteiro para a regra que lhe deu origem.

São estas regras que sabem qual é a evolução das regras à medida que é lido mais texto. São elas que são responsáveis pelo emparelhamento.

- ***Agenda***

A agenda junta todas as regras activas e é responsável pelo algoritmo que determina quais, dentro as regras do conjunto inicial se vão tentar emparelhar com o novo segmento. Nesta versão do sistema, limita-se a criar regras activas com todas as regras de recomposição existentes, invocando sobre cada uma o método que lhes adicionam segmentos que, além de fazer evoluir a regra informa a classe que invoca o método do resultado obtido. Sempre que a regra não emparelha é ignorada, se ficar activa depois da inserção do segmento é guardada para poder ser processada com os segmentos seguintes.

É também a Agenda a responsável pela decisão sobre quando e como aplicar uma regra de recomposição sobre o texto. Dados os requisitos do algoritmo aplica uma regra sempre que for a primeira regra e estiver reduzida, isto corresponde a disparar as regras de acordo com a ordem do ficheiro inicial e apenas se as regras relativas aos segmentos anteriores não poderem ser aplicadas antes.

4.7. Trabalho Futuro

Durante a primeira fase de trabalho que agora termina (correspondente ao último ano do curso e ao trabalho final) desenvolveu-se um sistema inicial que, devido aos cuidados tidos no desenho e implementação das estruturas de dados pode, com bastante facilidade, ser facilmente desenvolvido.

Há ainda algumas alterações a fazer e algumas melhorias a serem introduzidas, algumas teorias a serem testadas e alguns requisitos a serem satisfeitos. Deste ponto de vista há ainda algum trabalho a desenvolver. Nesta sub-secção apresenta-se algum trabalho que ainda falta fazer e indicam-se caminhos a tomar no futuro para complementar e optimizar o código. Sugerem-se também algumas melhorias cuja eficiência precisa de ser testada.

- Implementar indexes que facilitem a procura e a introdução de novas regras na agenda: indexar pela palavra, pelos lemas e pelas categorias das etiquetas.
- Criar um sistema que possa ser associado numa "*shell*" de Unix (ou Linux) ao SMorph e ao analisador sintáctico, criando-se pipelines entre os vários sistemas.
- Testar a possibilidade de criar um sistema que encapsule o SMorph e o PAsMo, ou o PAsMo e o analisador sintáctico de superfície (ou os três) evitando a escrita e a leitura de ficheiros intermédios.
- Fazer variar os vários parâmetros que influenciam o funcionamento do sistema para testar exaustivamente o seu comportamento diante de variações (para verificar a robustez) e medir tempos gastos (para se poderem fazer previsões realistas sobre os desempenhos esperados).
- Testar implicações a nível de desempenho e de complexidade de código quando se trocam "*maps*" por "*vectors*" (ou vice-versa) nas listas de etiquetas dos segmentos, nos itens das regras de recomposição e nas informações morfológicas das regras de correspondência.
- Fortalecer os testes feitos às classes e ao sistema. Enriquecer a bateria de testes que já se começou a desenhar.

4.8. Avaliação

Para avaliar este sistema será necessário, antes de mais, medir os ganhos de desempenho em termos de tempo quando comparados com o sistema MPS. Além disso deve-se medir a facilidade que se aumenta na escrita de regras e o enriquecimento que se opera sobre a gramática admitida. Além disso, deve-se ter em conta a facilidade de correcção, manutenção e evolução do código devido à sua estruturação e ao uso de abstracções que isolam as responsabilidades de cada objecto.

No âmbito deste trabalho só interessa fazer uma avaliação em termos computacionais, pelo que a adequação das regras os ganhos de acordo com a ordem das regras ou de acordo com as categorias encontradas não são relevantes, até porque a ordem das regras introduz uma hierarquia que é preciso respeitar.

No entanto, e porque o objectivo final é processar *corpus* linguísticos, interessa considerar, medir e avaliar o crescimento dos tempos e do desempenho com o aumento da complexidade das frases, com a variação do número de segmentos por frase, com o crescimento do número de regras, com a variação do número de vezes que se processa cada frase, com o aumento de ambiguidade na frase, etc.

Em princípio, supõe-se que a variação do número de regras deve influenciar linearmente os tempos do sistema. Espera-se um crescimento linear do tempo em

função do número de regras consideradas e do número de frases a processar. Além disso, espera-se que com a criação de indexes internos à agenda o sistema possa diminuir o número de testes necessários antes de encontrar regras que possam ser aplicadas sobre a frase, o que pode, de certa forma, diminuir os tempos gastos e fazer baixar a influência do número de regras de linear para logarítmica.

Supõe-se que número de vezes que se processa a frase deva influenciar os tempos linearmente até se atingir um determinado patamar, espera-se que, a uma determinada altura o crescimento seja limitado (mais uma vez, crescimento logarítmico), o que significa que, a partir de um determinado valor, aplicar as regras *n* vezes ou *n+1* produz o mesmo resultado uma vez que deixa de haver regras que se possam aplicar.

O aumento da ambiguidade existente nos segmentos, da forma como se faz o processamento, corresponde a um aumento do número de frases. Espera-se que estes aumentos façam os tempos crescer linearmente. Nalguns testes feitos os tempos médios por frase eram os mesmos para textos com muitas ou com poucas frases. O tempo global crescia, portanto, linearmente.

Com a introdução das melhorias descritas na sub-secção anterior (cf. 4.7) esperam-se melhorias significativas nos tempos e nas dependências relativas aos dados introduzidos.

5. Metodologia Utilizada

Nesta secção referem-se brevemente dois aspectos relevantes da metodologia utilizada no desenvolvimento do sistema.

Começa-se por se explicar a utilização de programação literária e as vantagens encontradas na ferramenta que se utilizou: o Fweb.

De seguida explicam-se os testes desenvolvidos que permitem a quem quiser corrigir, melhorar ou desenvolver o sistema fazer uma verificação rápida de que as funcionalidade anteriormente implementadas não foram danificadas.

5.1. Ferramenta de Programação Literária: FWeb

Para facilitar a escrita do código e do manual técnico que o documenta utilizou-se uma ferramenta que permite ao programador descrever os algoritmos do sistema primeiro em pseudo-código, de forma resumida e num alto nível de abstracção e, depois, à medida que avança no sistema, ir especificando o significado de cada item que surge no pseudo-código escrito "traduzindo-o" em pseudo-código mais específico ou em código C++.

A figura seguinte mostra o funcionamento da ferramenta.

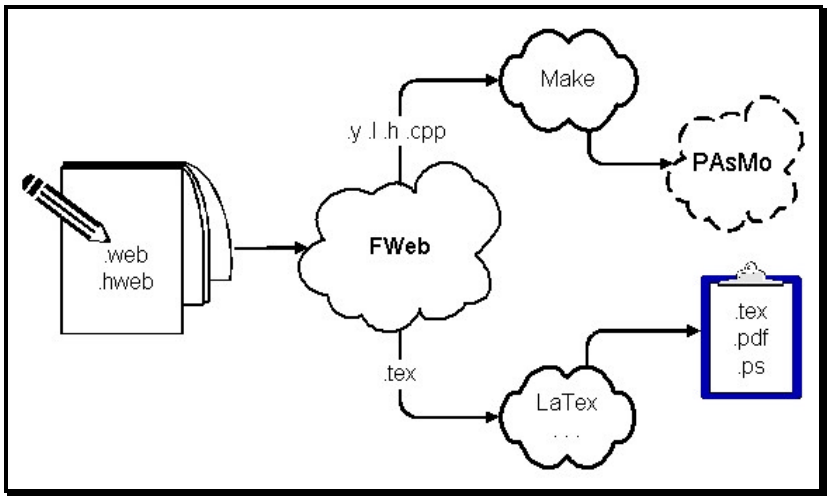


Figura 6. Funcionamento da ferramenta utilizada na escrita do código, o FWeb

O programador escreve ficheiros ".web" e ".hweb" onde descreve as classes os algoritmos, etc., em pseudo-código ou em código fonte, conforme o que for mais adequado e vai introduzindo explicitações e explicações daquilo que o sistema faz, das opções tomadas, etc.

No final, o FWeb gera com um comando (*ftangle*) os ficheiros ".h" e ".cpp" que se podem compilar recorrendo a uma "Makefile"; com outro (*fweave*) o ficheiro ".tex" que uma vez compilado nos permite gerar um *postscript* que com a listagem do código devidamente comentada. Aos ficheiros com o código fonte acrescenta informação que

permite ao compilador do C++ indicar a origem dos erros referindo-se aos ficheiros ".hweb" e ".web" e não aos ficheiros gerados com o código fonte o que permite e facilita a correcção de erros.

A vantagem encontrada nesta ferramenta foi a facilidade que introduz na escrita do código. Em vez de se escrever o pseudo-código num documento e o código noutro e se ter a necessidade de verificar constantemente a coerência entre os vários ficheiros, garante-se que sempre que se altera o código se está em condições de gerar o documento correspondente. Por exemplo, se se encontra uma falha no algoritmo principal do sistema e se quer corrigir, na abordagem tradicional, é necessário corrigir dois ficheiros: o do relatório e o relativo à implementação do algoritmo. Usando esta ferramenta, basta corrigir um ficheiro.

5.2. Testes Desenvolvidos

De acordo com as metodologias de desenvolvimento de código aprendidas ao longo do curso, foram implementados testes para todas as classes, para os seus construtores e destrutores e para os seus métodos.

Nesta secção descreve-se sumariamente os testes feitos para validação do funcionamento do *scanner* e dos *parsers*. A Figura 7, mostra um esquema do processo utilizado.

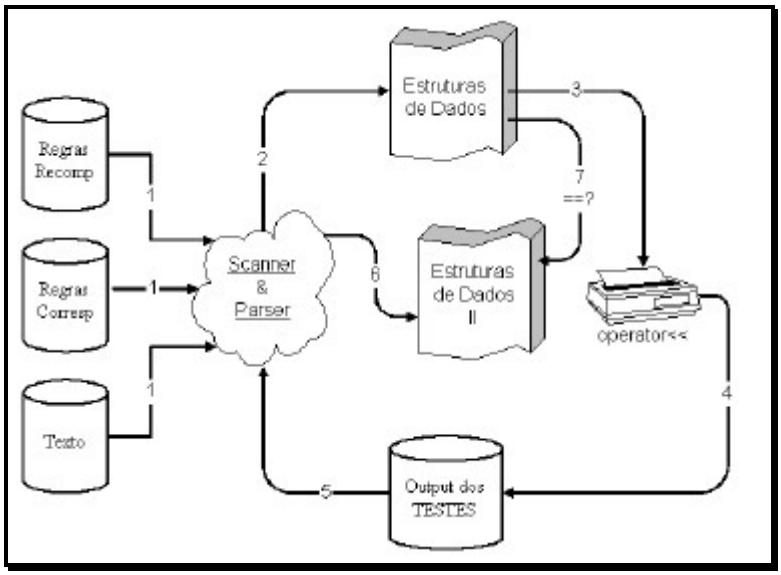


Figura 7. Arquitectura dos testes desenvolvidos para as classes *Scanner* e *Parser*

Começou-se por, na definição das classes, implementar o `operator<<` de tal forma que, a impressão de uma estrutura de dados correspondesse àquilo que o parser e o scanner lêem dos ficheiros de entrada. Para testar os objectos procede-se à leitura de todas as estruturas de dados para memória [1]. Isso cria a estrutura de dados na memória do sistema [2], recorrendo ao operador definido [3] imprimem-se as estruturas criadas para um ficheiro[4]. Faz-se a leitura desse ficheiro [6] e comparam-se as estruturas de dados assim obtidas [7]. Se, além disso se escrever para outro ficheiro de texto as novas estruturas [II] pode-se reforçar o teste comparando os ficheiros criados.

6. Trabalho Futuro

Uma vez bem definida a arquitectura do sistema, uma vez compreendidos os objectivos do trabalho há que começar a implementar um sistema que se comporte como até agora se descreveu.

Para isso, a primeira coisa a fazer será estabelecer quais as estruturas de dados necessárias, quais os tipos de dados abstractos a utilizar e quais as funcionalidades específicas de cada um.

Uma vez terminada esta tarefa, pode, então, passar-se à implementação do sistema. Para isso usar-se-ão, tanto quanto possível, as técnicas de programação e de desenvolvimento de software aprendidas ao longo do curso.

No final, como não podia deixar de ser, será necessário testar exaustiva e rigorosamente o sistema desenvolvido e avaliar os resultados obtidos a nível de desempenho. Para isso será necessário ter algumas métricas com que se consiga caracterizar a qualidade dos resultados obtidos. Posteriormente poderá, ainda, ser feito um estudo das variações dos resultados devidas às variações de propriedades do sistema. Por exemplo, qual o melhor valor para a razão entre a frequência esperada de uma palavra e a frequência obtida.

Procurar-se-á saber quais as medidas usuais de avaliação e comparar-se-ão os resultados obtidos no sistema implementado com resultados obtidos noutros sistemas descritos nos artigos dos quais se retiraram algumas ideias.

7. Conclusão

Ao longo do relatório que agora termina, quis explicitar-se de forma detalhada qual o trabalho desenvolvido no âmbito do Trabalho Final de Curso e quais os objectivos da Tese de dissertação para a obtenção do grau de Mestre da aluna Joana Lúcio Paulo, nº 44 048 do Curso de Engenharia Informática e de Computadores do Instituto Superior Técnico da Universidade Técnica de Lisboa, trabalho e tese desenvolvidos sob a orientação do Prof. Nuno J. Mamede.

Antes de mais, indicou-se resumidamente qual o trabalho final a desenvolver e quais os objectivos finais a cumprir. Depois, introduziram-se alguns conceitos mais relevantes para a compreensão do tema abordado e da exposição que se seguia; localizou-se o trabalho no actual contexto de investigação na área em que se insere e nas áreas mais importantes com ela relacionadas.

No capítulo seguinte, fez-se um esboço inicial da arquitectura do sistema tendo-se aprofundado com maior detalhe o subsistema mais relevante para este trabalho, aquele que se vai implementar.

Apresentou-se, então, o trabalho já desenvolvido, o sistema de análise pós-morfológica, PAsMo. Nesse capítulo explicou-se a terminologia relevante para a área, descreveu-se a arquitectura do sistema, apresentou-se a especificação dos dados parametrização, de entrada de dados, de saída do processamento, e de registo de log. De seguida apresentou-se as estruturas de dados desenhadas, indicou-se o trabalho futuro que se vai desenvolver e a forma de avaliação do sistema.

Finalmente apresentou-se a metodologia utilizada no desenvolvimento do sistema e um possível planeamento para o trabalho que ainda falta desenvolver.

Neste capítulo fez-se esta breve conclusão remissiva.

8. Bibliografia

[Anscombre 91] Anscombre, J-C. (1991) "L'article zéro sous préposition", *Langue Française*, Septembre 91 p. 24-39

[Bourigault 92] Bourigault, D. (1992) "Surface grammatical analysis for the extraction of terminological noun phrases", *Proceedings of the 15th International Conference on Computational Linguistics, COLING'92*, p. 977-981, Nantes

[Clarkson & Rosenfeld 1997] P.R. Clarkson and R. Rosenfeld (1997); "Statistical Language Modeling Using the CMU-Cambridge Toolkit"; *Proceedings ESCA Eurospeech*

[Faiza 99] Abbaci Faiza, Développement du Module Post-SMorph. Mémoire de DEA de linguistique et informatique, GRIL, Université Blaise Pascal, Clermont-Ferrand, 1999

[Jacquemin & Bourigault 2000] Jacquemin, C., et Bourigault, D. (2000), "Term Extraction and Automatic Indexing", R. Mitkov, editor, *Handbook of Computational Linguistics*, Oxford University Press, Oxford

[Justeson 95] Justeson, J. S., Katz S. M. (1995) "Technical terminology: some linguistic properties and an algorithm for identification in text", Natural Language Engineering, p. 9-27.

[Kennedy 98] Kennedy G. (1998), *An introduction to Corpus Linguistics*, London: Addison Wesley Longman Limited

[Neto 96] Marquez Neto, A. P. (1996) "Terminologia e *Corpus* Linguístico", *Revista Internacional de Língua Portuguesa* – RILP nº 15, p. 100-108

[Salah 98] Aït-Mokhtar Salah, L'analyse présyntaxique en une seule étape, tese de doutoramento, GRIL, Laboratoire de recherche sur le Langage, Université Blaise Pascal, Clermont Ferrand, France, 1998

[Silva 99] Ferreira da Silva, J., Pereira Lopes, G. (1999) "A local maxima method and a fair dispersion normalization for extracting multi-words units from corpora", *International Conference on Mathematics of Language*, Orlando, July 99

9. Anexo