INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Uma evolução do sistema ShRep: optimização, interface gráfica e integração de mais duas ferramentas

David José Gomes de Moura Rodrigues

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Júri

Presidente:	Professor Pedro Nuno Ferreira da Rosa da Cruz Diniz
Orientador:	Professor Nuno João Neves Mamede
Co-Orientadir:	Professor João Carlos Serrenho Dias Pereira
Vogais:	Professor José João Dias de Almeida

Setembro 2007

Agradecimentos

Após este longo trabalho gostaria de agradecer a todos aqueles que contribuíram quer directa, quer inderectamente para a sua concretização.

Um agradecimento especial para o orientador Nuno Mamede e para o co-orientador João Dias Pereira por me indicarem os caminhos a seguir e, com mestria, incentivar-me a explorar e aprimorar novas capacidades, por outras palavras, uma lição de como bem orientar.

Queria agradecer também a todos os membros do grupo L2f, em particular ao João Graça e ao Ricardo Ribeiro por todos os esclarecimentos acerca dos trabalhos por eles efectuados.

Um agradecimento também a David Matos por todo o apoio e sentido crítico apresentado com o decorrer da tese, que contribuíu para o seu crescimento.

Um agradecimento muito especial a toda a minha família, em particular aos meus pais, não só pelo apoio durante este ano, mas sobretudo, pelas condições que sempre me disponibilizaram de modo a ter hoje as capacidade de poder chegar até aqui.

Por fim, mas não menos importante, um agradecimento sentido a Margarida Campos, por ter sido revisora da tese e principalmente por todo o apoio sentimental que sempre demonstrou, estando sempre presente nos momentos mais difíceis.

Lisboa, November 24, 2007

David José Gomes de Moura Rodrigues

Resumo

Esta tese tem como ponto de partida o sistema ShRep desenvolvido por João Graça. Nesse sistema, João Graça apresenta uma solução para resolver alguns dos problemas existentes nas arquitecturas *pipes-and-filters*, normalmente usada em processamento de língua natural. O sistema apresentado por João Graça substitui a arquitectura *pipes-and-filters* por uma arquitectura cliente-servidor. Nesta arquitectura o servidor equipara-se a um repositório tendo a finalidade de conter os dados produzidos por todas as ferramentas e efectuar o processamento necessário para a gestão desses dados, enquanto o cliente é composto pelas ferramentas de processamento de língua natural que pedem dados ao repositório, efectuam o seu processamento, e deixam o seu resultado no repositório. Foram também desenvolvidas bibliotecas clientes que abstraem o protocolo de comunicação existente entre os clientes e o servidor.

A primeira fase desta tese foi o desenvolvimento de uma interface gráfica que permite a visualização da informação existente no repositório.

De seguida, dado o sistems ShRep apenas ter utilidade caso existam ferramentas integradas que possam ser utilizadas, foram adicionadas duas ferramentas àquelas que já existiam no sistema ShRep. A primeira ferramenta integrada foi a ferramenta MARv que é usada na desambiguação morfosintáctica e a segunda ferramenta integrada foi o Palavroso que atribui classes gramaticais a palavras.

Havendo a necessidade de efectuar modificações ao código fonte da ferramenta MARv, foram também efectuadas alterações de forma a melhorar o desempenho da ferramenta, quer em termos de tempo de processamento, quer em termos de memória utilizada, bem como na taxa de erro que a ferramenta apresenta ao efectuar a desambiguação.

Após a integração da ferramenta MARv verificou-se que o sistema ShRep apresentava um elevado tempo de processamento. Desta forma, foram efectuadas alterações que permitiram a optimização do sistema, nomeadamente alterações de forma a diminuir os impactos que o protocolo de comunicação apresentava entre o servidor e o cliente. Foram ainda alargados os elementos do modelo de domínio em que é possivel a execução em paralelo possibilitando às novas ferramentas integradas funcionarem desta forma com o repositório. Por fim, foi desenvolvido um módulo de código que permite a utilização dos sistema ShRep numa versão *StandAlone*, isto é, substituindo a utilização do servidor por ficheiros, sendo os dados carregados no início da aplicação, e descarregados no final do processamento da ferramenta.

Abstract

The starting point of this thesis was the ShRep system developed by João Graça. In it, João Graça presents a solution to solve some of the problems in the pipes-and-filters architecture, used mostly in natural language processing. The system replaces the pipes-and-filters architecture with a client-server architecture. In this architecture, the server acts as a repository in order to maintain all the data produced by the tools and to do all the processing necessary to the correct management of the data, while the client is composed by the natural language processing tools that request data from the repository, process it, and save the results in the repository.

The first step on this thesis was the development of a graphical interface that permits the visualization of the information stored in the repository.

Next, given that the ShRep system's usefulness relies on the presence of integrated tools, two additional tools were added to the system. The first one was MARv which is used in morphological disambiguation, while the other one, Palavroso, is used in the attribution of grammatical classes to words.

Because there was a need to modify the MARv tool's code, some changes were made in order to improve the tool's performance. these changes were the improvement of its processing time and memory management, and the reduction of its error rate when disambiguating.

After the MARv tool was fully integrated, the ShRep system presented a very high processing time. Because of this, some changes were made in order to optimize the system, namely by reducing the impact of the communication protocol between the client and the server. The elements in the domain model that permitted parallel execution were widened enabling the newly integrated tools to operate in this fashion with the repository. Finally a code model was developed in order to permit the use of the ShRep system in StandAlone mode; this was achieved by replacing the server with files, with the data being loaded at the beginning, and stored in the end of the tool processing cycle.

Palavras Chave
Keywords

Palavras Chave

Sistemas de processamento de Língua Natural

Integração de ferramentas de processamento de Língua Natural

Repositório / Ficheiros de dados

Optimização

Processamento Paralelo

Interface Gráfica

Keywords

Natural Language processing systems

Natural Language processing tools integration

Repository / Data Files

Optimization

Parallel processing

Graphic Interface

Índice

List of Figures

List of Tables

1 Introdução

1.1 Motivação

Esta tese centra-se na área dos sistemas de anotação. Um sistema de anotação caracteriza-se por efectuar anotações em sinais de dados, que podem ser, por exemplo, texto ou áudio. As anotações são comentários, notas, explicações, referências, exemplos, avisos, correcções ou qualquer outro tipo de informação que possa ser indexada ao sinal de dados original. Este tipo de informação é frequentemente designado por metadados, uma vez que adiciona informação sobre o sinal de dados já existente.

A principal motivação para esta tese é efectuar uma validação ao sistema ShRep (Almeida Varelas Graça, 2006) desenvolvido por João Graça, aquando da realização da sua Tese de Mestrado, designada *A Framework for Integrating Natural Language Tools* que tem como principal propósito a integração de várias ferramentas de anotação. A validação ao sistema ShRep pretende avaliar o desempenho do sistema utilizando várias ferramentas e utilizando sinais de dados com alguma dimensão, que permitam avaliar o comportamento do sistema. Esta validação permite identificar os problemas dos sistema ShRep e permite encontrar soluções para resolver esses mesmos problemas.

A avaliação de vários sistemas de anotação existentes na literatura já foi efectuada por João Graça. A principal motivação que levou João Graça a desenvolver este trabalho foi encontrar uma solução que resolva alguns problemas encontrados na arquitectura *pipes-and-filters*. Esta arquitectura é a arquitectura normalmente usada para integrar um conjunto de componentes de anotação. Nesta arquitectura, cada componente actua com os dados provenientes do componente anterior e entrega a informação resultante da sua análise para o próximo componente. Um problema existente nesta abordagem é o modo como a informação é passada de componente para componente. Caso um componente de anotação não passe toda a informação, descartando parte dela, essa informação pode ser necessária a um componente que venha a seguir na cadeia de processamento. Por outro lado, se cada componente juntar à informação que produz, toda a informação que lhe foi entregue, vai tornar cada vez mais pesado o processamento de cada componente de anotação, quando o número de ferramentas existentes no *pipeline* aumentar.

Outro problema encontrado nesta abordagem é a necessidade de existirem componentes de conversão entre os componentes de anotação, caso os componentes de anotação não utilizem o mesmo formato de representação de dados. Para n componentes de anotação trocarem informação entre si, são necessários, no limite, $n*n$ componentes de conversão.

Para solucionar estes dois problemas João Graça optou por uma arquitectura *blackboard* onde o resultado produzido por cada componente é guardado numa nova camada, num repositório centralizado. Esta solução resolve o primeiro problema descrito para a arquitectura *pipes-and-filters*, dado todos os resultados produzidos estarem disponíveis num servidor e cada componente de anotação ter apenas de aceder à informação que necessita. Relativamente ao problema da necessidade de existirem componentes de conversão ele não é resolvido na totalidade pela abordagem seguida por João Graça, sendo apenas reduzida a sua complexidade. Se houver necessidade de componentes de conversão apenas é necessário um entre o sistema ShRep e o sistema de anotação, e outro no sentido contrário. Neste caso, para *n* componentes de anotação são necessários, no limite, *2*n* componentes de conversão.

Uma outra motivação para esta tese é efectuar melhorias em algumas ferramentas de anotação, com o objectivo de melhorar o desempenho destas ferramentas nas cadeias em que se inserem. As melhorias efectuadas podem ser a nível da correcção de resultados ou a nível da redução do tempo do processamento. A correcção de resultados pode ser possível, dado algumas ferramentas usarem como base de trabalho modelos probabilísticos, que dificultam a produção de resultados totalmente correctos. A redução do tempo de processamento é importante para as cadeias de processamento em que as ferramentas se inserem, principalmente se a cadeia de processamento funcionar num modo *pipe-and-filters* dado cada ferramenta necessitar do término da ferramenta anterior para iniciar o seu processamento. Desta forma, a melhoria de uma ferramenta dessa cadeia tem influência em toda a cadeia de processamento.

1.2 Objectivos

Os objectivos desta tese são os seguintes:

- Actualizar o estado da arte sobre sistemas de anotação realizado na tese de mestrado de João Graça, efectuando uma comparação entre o sistema ShRep e o sistema *Unstructured Information Management Architecture* (UIMA), desenvolvido pela IBM (O. Suhre, 2004), que tem como principal função servir de interligação entre várias ferramentas, tal como acontece com o sistema ShRep;
- Analisar um conjunto de interfaces gráficas existentes para alguns sistemas de anotação, com a finalidade de perceber o que já existe nesta área. Uma interface gráfica é um mecanismo de interacção entre um sistema de computador e um utilizador, com vista a este poder seleccionar e alterar os componentes que formam esse mesmo sistema. A necessidade de uma interface gráfica surge com a integração de algumas ferramentas, sendo necessário encontrar uma solução para visualizar a informação existente e produzida por estas ferramentas;
- Aumentar o número de ferramentas integradas com o sistema ShRep e efectuar alterações nessas ferramentas que melhorem o seu funcionamento;

- Efectuar uma validação ao sistema ShRep, através de uma análise ao seu funcionamento, e efectuar algumas alterações para melhorar o seu desempenho.

1.3 Estrutura da dissertação

O capítulo 2 desta tese apresenta uma análise comparativa entre o sistema ShRep e o sistema UIMA desenvolvido pela IBM. O capítulo 3 descreve um conjunto de cinco interfaces gráficas usadas em sistemas de anotação bem como a interface gráfica desenvolvida para o sistema ShRep com o objectivo de facilitar a visualização da informação existente no repositório. O capítulo seguinte relata as melhorias que foram implementadas nalgumas ferramentas e o processo de integração destas no sistema ShRep. O capítulo 5 apresenta a análise efectuada ao desempenho do sistema ShRep, bem como as soluções encontradas para melhorar esse mesmo desempenho. O último capítulo descreve as conclusões e lista as prospostas de trabalho futuro.

2 Trabalho Relacionado

2.1 Introdução

Este capítulo efectua uma actualização do estado da arte presente na tese de mestrado elaborada por João Graça, através da comparação do sistema ShRep e o sistema UIMA desenvolvido pela IBM.

2.2 Shared Repository (ShRep)

O sistema ShRep é definido através do seu modelo conceptual e da sua arquitectura. O modelo conceptual é composto por um conjunto de entidades: repositório, dados, sinal de dados, índices, regiões, análises, segmentos, segmentações, relações, classificações, alternativas e relações cruzadas. De seguida é apresentada uma breve descrição de cada um destes conceitos:

- O repositório é formado por camadas que podem conter fontes de dados primárias ou dados provenientes das ferramentas de processamento de Língua Natural;
- Os dados permitem representar qualquer tipo de informação, por exemplo, texto ou áudio, e fornecem uma abstracção para a sua utilização;
- Os sinais de dados representam conjuntos de dados;
- Os índices apontam para posições dos sinais de dados;
- As regiões apontam para uma zona dos sinais de dados, sendo compostas por dois índices;
- Uma análise representa toda a informação produzida por uma ferramenta de língua natural;
- Um segmento representa um qualquer elemento linguístico;
- Um conjunto ordenado de segmentos é designado por segmentação;
- Uma relação representa uma interligação entre dois segmentos pertencentes à mesma camada;
- uma relação cruzada representa uma interligação entre dois segmentos de camadas distintas;
- As classificações servem para associar características a elementos que se consideram classificáveis, ou seja, segmentos e relações;

- As alternativas são usadas para representar ambiguidades.

A arquitectura é constituída por uma biblioteca cliente e por um servidor, que se encontram divididos em três camadas: camada de dados, camada de serviços e interface remota. A parte do cliente pode ser descrita em qualquer linguagem, desde que satisfaça os protocolos de comunicação com a interface remota do servidor.

2.2.1 Requisitos presentes no ShRep

De seguida enumeram-se os requisitos genéricos, os requisitos para o modelo conceptual e os requisitos para o sistema usados na construção do ShRep e que servem para caracterizar uma solução de Processamento de Língua Natural (PLN).

Os requisitos genéricos considerados foram:

1. Num sistema de PLN não deve ser perdida informação entre ferramentas;
2. As ferramentas devem apenas produzir informação directamente relacionada;
3. A solução deve simplificar a criação de novas ferramentas, fornecendo uma interface de Entrada/Saída, que gere a leitura e escrita dos dados usados e forneça um modelo de dados que possa ser usado por cada ferramenta para representar informação linguística;
4. A solução deve minimizar o número de componentes de conversão, quando a integração de uma ferramenta de PLN já existente não satisfaça o modelo do sistema;
5. A interface fornecida deve permitir navegar através da informação produzida pelas diferentes ferramentas.

Para o modelo conceptual (MC) foram considerados os seguintes requisitos:

1. Deve permitir representar muita informação linguística produzida pelas diferentes ferramentas;
2. Deve ser extensível, dado ser impossível prever todos os tipos de informação linguística que poderão aparecer no futuro;
3. Deve permitir representar informação proveniente de fontes primárias (texto, sinal de voz), mas também informação produzida pelas ferramentas de PLN;
4. As ferramentas de PLN devem permitir criar e alterar fontes de dados primárias;
5. Devem permitir identificar elementos de fontes de dados primárias (exemplo, a divisão de uma frase em palavras);

6. Devem permitir a criação de informação relacional entre elementos linguísticos (exemplo: relação entre um verbo e o sujeito);
7. Devem permitir associar propriedades a elementos linguísticos ou a relações (exemplo: propriedades morfológicas de uma palavra);
8. O MC deve permitir representar qualquer tipo de informação primária, como texto, voz ou vídeo.
9. Toda a informação linguística produzida por uma ferramenta de PLN deve estar associada à mesma camada;
10. O MC deve permitir obter informação linguística através da identificação da camada que contém essa informação;
11. Cada camada é associada à identificação da ferramenta que a produziu;
12. O MC deve permitir representar ambiguidades na identificação de elementos linguísticos;
13. o MC deve permitir representar elementos linguísticos em árvore (exemplo: árvores sintácticas);
14. o MC deve permitir criar relações entre elementos linguísticos de outras camadas;
15. o MC deve possibilitar representar ambiguidade na classificação, que corresponde a associar conjuntos distintos de características ao mesmo elemento linguístico;
16. o MC deve permitir a associação de características a elementos linguísticos ou relações de outras camadas;
17. o MC deve permitir representar relações entre elementos linguísticos de camadas distintas;
18. o MC deve permitir que os elementos linguísticos referenciem a fonte de dados primária, sem ter que copiar o seu valor, de modo a evitar a repetição ao longo das camadas;
19. o MC deve permitir representar dados que não existe na fonte de dados primária (exemplo: separação de contracções).

Por fim, foram considerados os seguintes requisitos de sistema:

1. O sistema deve facilitar a iteração de dados com o Repositório, dado que a iteração é a forma habitual de interacção entre uma ferramenta de PLN e os dados;
2. O sistema deve permitir a selecção de dados através da identificação da camada, de modo a que a ferramenta apenas trabalhe com os dados que precisa;
3. O sistema deve permitir aceder a dados que foram produzidos por uma Análise que ainda não terminou. Isto possibilita que uma outra ferramenta esteja já a trabalhar sobre esses dados;

4. O sistema deve guardar de forma persistente os dados;
5. O sistema deve permitir integrar ferramentas desenvolvidas em qualquer linguagem de programação.

2.3 Unstructured Information Management Architecture (UIMA)

Um sistema de gestão de informação não estruturada (Unstructured Information Management - UIM) pode ser caracterizado como um software que através da informação contida em texto, áudio, vídeo ou imagem, a organiza e encontra conhecimento útil para uma dada aplicação. Segundo os criadores do sistema UIMA (A. Lally, 2004) grande parte da informação útil encontra-se acessível de uma forma não estruturada.

Os resultados das análises produzidas pelos sistemas UIM podem ser organizados de forma estruturada, de modo a tornar mais eficientes os processos de pesquisa da informação que se procura. O sistema UIMA surge com o objectivo de facilitar a passagem de informação não estruturada para informação estruturada.

O sistema UIMA é composto por um conjunto de componentes, que apesar de relacionados, são definidos de forma independente. Estes componentes são: *Collection Reader*, *CAS Initializer*, *Analysis Engine*, *CAS Consumer* e *Collection Processing Engine*. A ligação entre estes componentes é efectuada através da partilha de objectos que podem representar informação sobre anotações ou informação sobre entidades existentes nas anotações, que são denominados de *Common Analysis Structure* (CAS). De seguida apresenta-se uma descrição mais detalhada dos componentes referidos e dos objectos Common Analysis Structure.

2.3.1 Collection Reader

O componente *Collection Reader* é usado para processar um corpus constituído por vários documentos e processar cada documento como sendo um CAS. Cada CAS pode conter apenas o documento, ou pode também conter informação adicional sobre o mesmo, designada *metadados*. Esta informação adicional pode ser aplicada a sequências de caracteres de texto, sendo cada sequência designada por *span*. Os dados existentes em cada documento podem ser texto, áudio ou vídeo.

2.3.2 CAS Initializer

Este componente serve para efectuar a operação de filtragem dos documentos provenientes do *Collection Reader* para a análise. Por exemplo, pode ser usado para remover as etiquetas de XML ou de HTML

existentes num ficheiro de texto, de modo ao anotador efectuar apenas a análise ao texto contido em algumas das etiquetas.

Na iniciação do CAS é-lhe atribuído um *Subject OF Analysis* (SOFA) de um documento. Um SOFA é a representação de um documento que pode ser analisado. Esta distinção entre documento e SOFA permite ter várias análises ao mesmo documento sem alteração da sua versão inicial.

2.3.3 Analysis Engine (AE)

Este componente serve para analisar documentos podendo inferir e guardar atributos descritivos acerca do mesmo, ou apenas de zonas que o constituem. Recebe como entrada um CAS, proveniente do processo de iniciação de documentos ou proveniente de uma análise efectuada anteriormente. O resultado da análise é um novo CAS contendo a informação que um dado anotador produziu, o qual se designa por *Analysis Result*.

O termo *document* refere-se a qualquer unidade que um AE pode produzir, tal como, um documento de texto ou um segmento de áudio. O sistema fornece um componente básico que incorpora os algoritmos que correm nas AE, sendo cada um deste componentes designado de *Annotator*.

Aggregate Analysis Engines Um AE pode ser constituído por vários AEs, aos quais se dá o nome de *Aggregate Analysis Engines*. Esta construção é transparente para quem constrói o Aggregate AE, necessitando apenas de saber o fluxo de execução.

O sistema UIMA está preparado para permitir diferentes execuções: correr cada um dos AE's no mesmo processo (*tightly-coupled*) ou correr cada AE em processos ou até máquinas diferentes (*loosely-coupled*). Permite também o uso de protocolos remotos, como por exemplo, SOAP, através de uma camada que ajuda a criação dos serviços remotos.

Descritor de Componentes O sistema UIMA apenas define a interface para os componentes mais básicos ao seu funcionamento. Anotadores e AE são dois desses componentes. A definição dos componentes está dividida em duas partes: a parte declarativa e a parte de código.

A parte declarativa, designada *Component Descriptor*, contém metadados que descreve o componente: a sua identidade, a sua estrutura e o seu comportamento, e é representada em XML. O sistema UIMA fornece uma ferramenta para criar e gerir os descritores dos componentes. Cada descritor contém campos básicos (por exemplo: nome do componente, autor, versão) e identifica o *Type System* que o componente usa e os tipos que necessita como entrada e os tipos que ele produz como saída.

A parte de código implementa o algoritmo que define o comportamento do componente, que pode ser desenvolvido em Java ou em C++.

2.3.4 CAS Consumer

O componente CAS Consumer funciona como fim do fluxo, isto é, trata a informação que os objectos CAS contêm após terem sido aplicados os anotadores. Por exemplo, pode ser usado para passar a informação existente nos objectos CAS para uma base de dados relacional. [1]

O sistema UIMA permite a construção de *Aggregate Analysis Engines*, isto é, sistemas de anotadores compostos por mais que um anotador, com a utilização de CAS Consumers, o que permite transferir a informação para uma base de dados após a aplicação de cada um dos anotadores.

2.3.5 Collection Processing Engine (CPE)

O componente Collection Processing Engine é utilizado para agregar os diversos componentes descritos, especificando o Collection Reader a utilizar, os anotadores a aplicar e o conjunto de CAS Consumers que tratam a informação final. A descrição deste componente pode ser efectuada em ficheiros XML.

2.3.6 Common Analysis Structure (CAS)

Conforme se pode depreender dos componentes acima descritos, o CAS desempenha um papel fundamental no funcionamento da arquitectura. O CAS pode ser usado pelos componentes e pelos vários anotadores para representar e partilhar os resultados entre si. O CAS (O. Suhre, 2004) é uma estrutura de dados baseada em objectos que permite representar objectos, propriedades e valores. Os objectos podem estar relacionados com outros objectos, através de hierarquias herditárias.

É possível ao utilizador definir um *Type System* que consiste em vários tipos de objectos que podem ser identificados no corpus em análise. Existe liberdade total de modo a tornar o *Type System* adequado a um domínio específico. Os tipos têm atributos, tais como idade ou ocupação. Estes *Type System* podem ainda ser organizados em taxonomias: por exemplo, Empresa pode ser um subtipo de Organização, e Frase pode ser um subtipo de Parse.

Annotation Type O *Annotation Type* é usado para etiquetar regiões, quer de texto quer de áudio. Tem duas propriedades, designadas *begin* e *end*, que representam as posições efectivas dos *spans*. No entanto, duas *Annotation Type* podem referenciar a mesma entidade. Para isso o CAS possui *Entity Types* que representam objectos no domínio que podem ser referenciados por diferentes expressões. Tem uma propriedade, *occurrences* que aponta para todas as *Annotation Type* daquele objecto.

Se se considerar a frase, "The span from position 101 to 112 in document D102 denotes a Person", como um possível resultado de uma análise, pode-se referir que o CAS contém um objecto do tipo

[1] Esta possibilidade satisfaz o requisito, apresentado na descrição do sistema ShRep, de guardar de forma persistente todos os dados.

Person. Num corpus, o AE cria um objecto do tipo Person distinto para cada pessoa identificada, e relaciona o objecto com o respectivo *span* no texto. [2]

O sistema UIMA suporta análises simultâneas de várias vistas de um documento, o que se torna útil para processar múltiplas modalidades (áudio e vídeo). Um AE analisa uma ou mais vistas de um documento, em que cada vista contém um *SOFA*.

Iteradores O sistema UIMA providencia uma implementação eficiente do CAS, onde é possível interagir com o documento, bem como ler e guardar o resultado das análises. Fornece um conjunto de métodos onde é possível indexar iteradores aos diferentes objectos existentes (por exemplo, ter um iterador para todas as Person existentes). [3]

Para Java, UIMA fornece JCas. Cada tipo declarado no *Type System* aparece como sendo uma classe Java.

2.3.7 Outros componentes e propriedades do sistema UIMA

O CAS e recursos externos É possível aceder a recursos externos através da interface *UIMA context* que assegura que diferentes anotações que estão a funcionar em simultâneo partilham a mesma instância de um ficheiro externo.

Pesquisa Semântica A procura semântica tenta tirar partido da informação existente no CAS para optimizar os resultados de uma pesquisa. Se ao efectuar uma procura semântica se pretende procurar pelas organizações que tenham "centro" no nome, é mais fácil referir que se trata de uma organização, do que procurar apenas pela palavra em causa.

Processamento Paralelo O sistema UIMA faz uso do processamento paralelo para aumentar os níveis de eficiência das aplicações. O processamento paralelo pode ser efectuado na mesma máquina através da adição de novas *threads*, mas também pode ser obtido usando várias máquinas distintas. Para tal, o sistema faz uso do protocolo VINCI (Agrawal & Robert J. Bayardo Jr., 2001) para obter o mesmo serviço distribuído pelas máquinas existentes.

No caso de se querer processar mais do que um documento em simultâneo é necessário ter mais do que uma instância do CAS que representam os documentos. De modo a reduzir a quantidade de memória usada, o sistema UIMA disponibiliza um sistema de gestão de CAS Pool de modo a fomentar a reutilização dos CAS, ao invés de criar um novo sempre que é necessário analisar um documento.

[2] Esta referenciação ao *span* no texto, cumpre o requisito, apresentado na descrição do sistema ShRep, dos elementos linguísticos referenciarem as fontes de dados primárias.

[3] A existência deste iteradores satisfaz o requisito de Fornecer Iteradores, apresentado na descrição do sistema ShRep.

Identificação de elementos linguísticos O sistema permite identificar elementos linguísticos a partir de fontes de dados primárias. Por exemplo, o próprio sistema já disponibiliza algumas ferramentas para efectuar a Tocanização de um texto, ou seja, a identificação de palavras no texto.

2.3.8 Exemplo de construção de um sistema de anotação

De seguida, descreve-se um exemplo para construir um sistema de anotação no sistema UIMA, com o objectivo de facilitar a percepção de cada um dos componentes que formam o sistema, e o modo como se efectua a ligação entre eles. O objectivo deste anotador será encontrar identificadores de números de quartos de edifícios que são construídos com base num determinado padrão.

O primeiro passo para a construção do anotador é a definição do seu *Type System*, identificando as entidades de domínio para as quais se pretende obter informação no corpus a analisar. No exemplo concreto, apenas será necessário um tipo que servirá para registar cada quarto encontrado. Desta forma, define-se o tipo *NumeroDeQuarto* e indica-se que o seu *Super Type*, isto é, o tipo do qual este tipo é estendido, é o tipo *Annotation*. Juntamente com a definição do tipo, é possível definir características relacionadas com o mesmo. No exemplo dado, define-se a característica *Edifício*, que indica qual o edifício onde o quarto se encontra. Pelo facto de o *Super Type* ser o tipo *Annotation*, o tipo criado tem mais duas características, designadas *begin* e *end*, que indicam as posições da anotação no texto original.

A definição do tipo é descrita num ficheiro XML, ou no caso do programador do sistema de anotação usar o ambiente de desenvolvimento Eclipse, o sistema UIMA possui *pluggins* para o programador ter uma interface gráfica para definição dos tipos.

O próximo passo é a geração do código correspondente aos tipos criados. Para esse efeito, o sistema UIMA possui uma ferramenta designada *JCasGen* que cria a classe Java correspondente a cada tipo. No exemplo apresentado, será criada a classe *NumeroDeQuarto*.

De seguida, é possível utilizar as classes Java criadas no passo anterior e começar a desenvolver o algoritmo que implementa o anotador desejado. De modo a tornar o anotador funcional no sistema UIMA, é necessário que a classe que define o funcionamento do anotador implemente o método *initialize*, o método *process* e o método *destroy*. O método *initialize* é chamado pelo sistema quando o anotador é criado. O método *process* é usado para começar o processamento sendo chamado para cada documento que é analisado. O método *destroy* é chamado quando o processamento dos documentos termina. No caso de definir a classe anotador estendendo a classe *JTextAnnotator_ImplBase* apenas é necessário implementar o método *process*, dado os restantes já se encontrarem definidos. No entanto, o programador tem a liberdade de redefinir estes dois métodos, bastando para isso estender a sua classe da classe *JTextAnnotator*.

Para o desenvolvimento do algoritmo é necessário criar padrões equivalentes aqueles que se pre-

tendem encontrar nos documentos. No caso de se desenvolver o anotador em Java, pode-se recorrer ao pacote de classes Regex, disponível a partir de Java 1.4, para a criação dos padrões. A definição do método *process* recebe como argumento o documento a ser analisado, representado por um JCas (a definição do objecto CAS para Java), e a referência para a especificação dos resultados. A codificação do método consiste em percorrer o documento que é fornecido como argumento e encontrar para cada edifício os padrões correspondentes. Para cada padrão encontrado correctamente, cria-se uma nova classe *NumeroDeQuarto* atribuindo-lhe os atributos *Edificio*, *begin* e *end* correctamente. Por fim, adiciona-se cada objecto criado a índices de modo a poder ser referenciado e utilizado por outros anotadores.

Após a criação do anotador é necessário definir o componente *Analysis Engine* que fará o processamento necessário. Como o exemplo descrito baseia-se num anotador simples, apenas serão referidos alguns passos que envolvem a criação deste componente. O exemplo em questão trata-se da definição de um *Analysis Engine Primitive*, dado ser composto por apenas um anotador. É necessário definir os *Type System* usados. No exemplo seguido, apenas é usado o *Type System* anteriormente especificado, isto é, a definição do número do quarto, e é definido como saída do anotador.

Tal como aconteceu para a definição dos tipos usados, este processo pode ser descrito num ficheiro XML ou pela utilização do *pluggin* apropriado no ambiente de desenvolvimento Eclipse.

Para visualizar os resultados do anotador, o sistema UIMA contem uma ferramenta para processar a análise aos documentos, designada *Document Analyzer*, sendo necessário indicar o directório onde se encontram as fontes de dados primárias, o directório onde serão escritos os resultados e a localização do *Analysis Engine* criado. É ainda possível indicar qual a linguagem e a codificação em que os ficheiros que contêm os documentos se encontram.

2.4 Análise comparativa

Esta secção efectua uma análise comparativa entre os dois sistemas apresentados.

2.4.1 Quadro comparativo de requisitos

De seguida apresentam-se duas tabelas comparativas que apresentam os requisitos, descritos na secção 2.2.1, que são cumpridos pelo sistema UIMA e os que são cumpridos pelo sistema ShRep. A tabela 2.1 compara os requisitos de sistema, enquanto a tabela 2.2 compara os requisitos do modelo conceptual. Nestas tabelas "V" significa que o requisito em causa é verificado, "NV" significa que o requisito não é verificado, "-" indica que o requisito não foi avaliado ou não foi possível obter informação que permitisse esclarecer se o requisito é ou não verificado, e "PI" indica que o requisito não se encontra verificado, mas é possível efectuar a sua implementação de modo ao requisito passar a ser verificado.

Requisito	UIMA	ShRep
Fornecer iteradores	V	V
Seleccionar dados baseados na identificação da ferramenta	PI	V
Processamento paralelo de dados	V	V
Guardar de forma persistente todos os dados	V	V
Haver independência da linguagem de programação	V	V

Table 2.1: Quadro comparativo dos requisitos de sistema

2.4.2 Comparação dos sistemas

Nesta secção pretende-se apresentar uma comparação entre o sistema UIMA e o sistema ShRep de modo a clarificar as semelhanças e as diferenças entre eles. A comparação encontra-se dividida em duas partes: modelo conceptual e arquitectura.

2.4.2.1 Modelo conceptual

No que se refere ao modelo conceptual ambos os sistemas têm a noção de camada que representa um conjunto de anotações produzidas por uma ferramenta. No sistema ShRep a noção de camada serve também para incluir as fontes de dados primárias no sistema. No sistema UIMA estes dados são geridos por um componente próprio, designado *Collection Reader*.

Relativamente aos dados, ambos os sistemas estão desenhados para poderem receber qualquer tipo de dados. Os dois sistemas usam índices para referenciar posições de dados e para delimitar zonas. No sistema ShRep esta delimitação designa-se por *Region*, enquanto no sistema UIMA designa-se por *Span*.

O sistema ShRep fornece ainda um conjunto de elementos que enriquecem o seu sistema conceptual, nomeadamente: a possibilidade de criar relações, quer entre elementos linguísticos da mesma camada, quer de camadas distintas, a possibilidade de representar ambiguidades através de elementos alternativos, a possibilidade de representar informação de forma hierárquica e a possibilidade de atribuir classificações a elementos linguísticos. A representação dos elementos linguísticos é efectuada através da noção de segmento. No sistema UIMA esta noção não se aplica. No entanto, o sistema possui a noção de tipos de entidades que têm como objectivo construir um modelo de dados adaptado a um domínio qualquer e que pode ser enriquecido através das várias anotações efectuadas. Deste modo, o sistema UIMA não implementa de forma directa alguns dos conceitos aqui descritos para o sistema ShRep, mas é possível construir um modelo de domínio adequado de modo ao sistema passar a aceitar este tipo de funcionalidades.

Uma semelhança entre os dois sistemas é a presença de iteradores. No sistema ShRep o iterador é um tipo abstracto que depois se encontra implementado para ter acesso aos vários elementos presentes

Requisito	UIMA	ShRep
Gerir várias fontes de dados	-	V
Gerir várias camadas de informação linguística	V	V
Usar vários tipos de fontes de dados	V	V
Permitir editar as fontes de dados	-	V
Toda a informação de uma ferramenta está associada à mesma camada	-	V
Camadas associadas à ferramenta que a produziram	PI	V
Identificação de elementos linguísticos	V	V
Representar ambiguidade nos elementos linguísticos	V	V
Representar elementos linguísticos em forma de árvore	-	V
Representar relações	V	V
Representar relações entre segmentos de outras camadas	V	V
Associar características a elementos linguísticos e a relações	V	V
Representar ambiguidade nas classificações	V	V
Associar características a elementos linguísticos ou a relações de outras camadas	V	V
Representar relações entre elementos linguísticos de camadas distintas	-	V
Elementos linguísticos referenciam fontes de dados primárias	V	V
Permitir representação de dados que não fazem parte de qualquer fonte de dados primária	-	V

Table 2.2: Quadro comparativo dos requisitos do Modelo Conceptual

no modelo conceptual. Por exemplo, existem iteradores para as análises, para as segmentações de uma análise, para os segmentos de uma segmentação. No sistema UIMA os iteradores têm a mesma funcionalidade, isto é, servem para navegar ao longo dos vários elementos existentes, nomeadamente, as entidades de domínios e as respectivas anotações.

Uma diferença clara entre os dois sistemas situa-se no modo de guardar os dados persistentemente. Enquanto no ShRep é responsabilidade do servidor efectuar esta tarefa, no UIMA é o programador ou o utilizador do sistema que tem a função de saber como quer guardar os dados de forma persistente.

2.4.2.2 Arquitectura

A arquitectura do sistema ShRep encontra-se dividida na biblioteca cliente e no servidor. O servidor tem a seu cargo a gestão de todas as camadas de dados já processados, sendo responsável por fornecer uma camada sempre que lhe é facultada, bem como guardar a informação proveniente da biblioteca cliente. Por seu lado a biblioteca cliente tem como função servir de intermediária entre o servidor e as ferramentas de língua natural. O sistema UIMA divide a sua arquitectura em componentes de modo a facilitar a utilização do sistema. Esses componentes são: Collection Reader, Cas Initializer, Analysis Engine, Cas Consumer e Collection Processing Engine. O sistema suporta uma descrição em XML de um fluxo de execução, de modo a executar um conjunto de anotadores e obter quer os resultados intermédios, quer

os resultados finais, como se de apenas um anotador se tratasse.

2.4.3 Conclusão

Ao avaliar os dois sistemas é possível verificar que ambos apresentam uma abordagem diferente dos sistemas de anotação tradicionais, no que se refere ao modo como os dados são partilhados entre as diferentes ferramentas. A criação destes dois sistemas foi claramente no sentido de facilitar este processo.

No entanto, o modo de resolução desta problemática por parte dos dois sistemas é distinto. Enquanto no sistema ShRep existe um conjunto de elementos que formam o modelo conceptual que é partilhado entre as várias ferramentas, no sistema UIMA a criação do modelo de dados que é partilhado é deixada ao critério do criador de cada ferramenta, existindo no entanto um elemento base. A abordagem seguida no ShRep pode ser mais limitativa que a abordagem existente no sistema UIMA, contudo torna mais fácil o processo de uniformização entre as várias ferramentas.

De referir também que a solução adoptada pelo sistema UIMA obriga à redefinição das entidades de domínio, cada vez que se muda a área de aplicação dos anotadores. No entanto, o facto de essa definição não ser em código facilita em parte o processo.

3 Interfaces Gráficas

3.1 Introdução

O desenvolvimento de uma interface gráfica para o sistema ShRep surge da necessidade de ser encontrado um modo que possibilite uma boa visualização dos dados existentes no repositório. O único modo existente até agora para visualizar esta informação era em formato textual organizado de forma hierárquica, tal como se mostra na figura A.1, o que dificultava uma correcta verificação acerca dos dados existentes.

Deste modo, este capítulo efectua um levantamento das interfaces gráficas de alguns dos sistemas de anotação comparados por João Graça e a interface gráfica do sistema UIMA da IBM. No final do capítulo é descrita a interface gráfica implementada para o sistema ShRep, tendo sido desenvolvida com o objectivo de ser usada para verificar o resultado produzido pelas ferramentas que se encontram integradas no sistema.

3.2 Trabalho relacionado

Foi efectuada a avaliação de sistemas que efectuam a análise sintáctica de texto (NLTK), de sistemas que efectuam transcrições de sinais de áudio para texto (Trancriber e AGTK), e de sistemas que servem de base para integrar várias ferramentas de anotação (GATE e UIMA).

3.2.1 Natural Language Toolkit (NLTK)

O sistema de anotação NLTK (Bird & Loper, 2002) surge para ajudar os alunos dos cursos de linguística computacional a criarem componentes de processamento de língua natural, de modo a compreenderem melhor os conceitos subjacentes. A sua aplicação está limitada a texto usando para isso uma classe designada *Token*. Esta classe representa uma unidade de texto que pode ser uma palavra, uma frase ou um documento. Contém um conjunto de atributos, tais como, o atributo TAG que permite adicionar informação ao *Token*, ou o atributo LOC que indica a localização do *Token* no texto original.

Existem ainda subclasses de *Token*. Um exemplo é *TreeToken* que contém uma lista que representa os descendentes de um *Token*, permitindo representar árvores. O sistema tem três ferramentas, contendo

cada uma a sua interface gráfica: RdParser, SrParser e Chart, que se apresentam de seguida.

3.2.1.1 RdParser

A ferramenta RdParser é uma ferramenta com ambiente gráfico que faz parte do *NLTK LITE* e é usada para visualizar a construção de árvores sintácticas, através do algoritmo *Recursive Descent Parser* (Burge, 1975). Na figura 3.1 é possível observar a visualização principal onde estão presentes as regras da gramática (à esquerda), o texto que é analisado e os comandos para efectuar as acções (em baixo), a área onde o desenvolvimento do algoritmo é efectuado (ao centro) e uma barra de estado que indica a última operação efectuada. No caso de o texto a analisar ou a construção da árvore ter uma dimensão superior à janela de visualização respectiva, são acrescentadas barras de deslocamento, quer verticais, quer horizontais. No entanto, na área de visualização da gramática tal não acontece. No caso de haver uma gramática de maior dimensão que a respectiva janela de visualização, apenas é possível visualizar as primeiras linhas que descrevem a gramática.

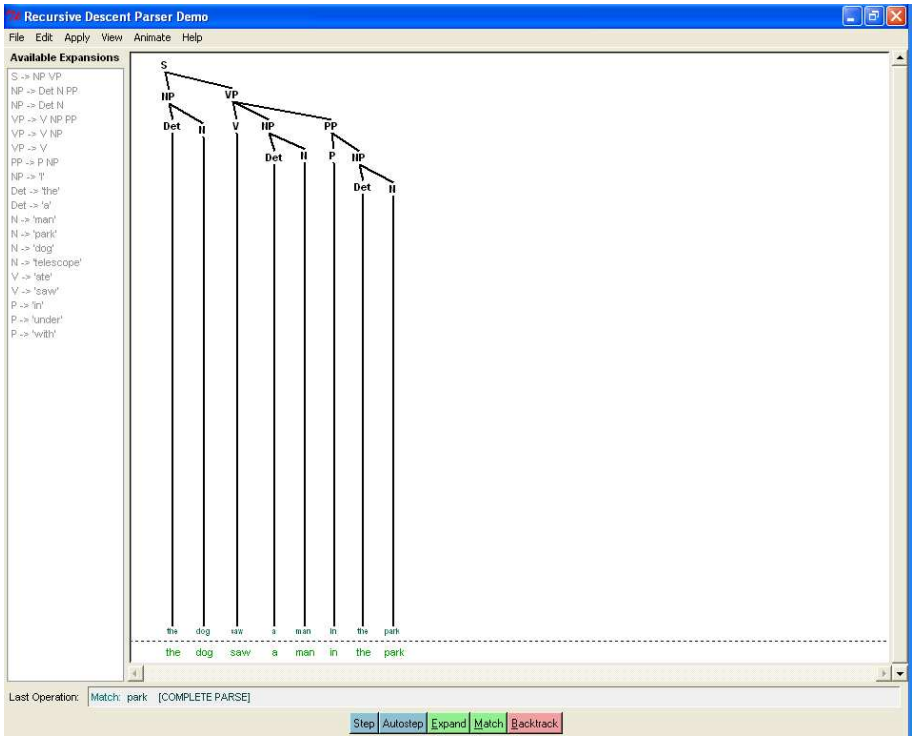

Figure 3.1: Visualização da interface gráfica da ferramenta RdParser.

A construção da árvore sintáctica é efectuada através de linhas e texto. O texto é usado para representar os nós que correspondem a símbolos da gramática. As linhas são utilizadas para ligar os nós entre si na construção da árvore, bem como para ligar os nós que representam símbolos terminais ao texto na análise correspondente. Durante a execução do algoritmo vão sendo fornecidas informações

gráficas que permitem acompanhar a evolução do mesmo. O texto da análise é diferenciado através de cores, sendo identificado o texto para o qual já foi efectuado correctamente o parser, do restante. Na gramática são salientadas as regras que naquele instante estão a ser analisadas, sendo ainda distinguidas as alternativas já analisadas das que faltam analisar, sendo as primeiras marcadas através da expressão *(TRIED)*. Na construção da árvore, a distinção entre os vários elementos também é feita através de esquemas de cores, sendo distinguidas: as ramificações que já foram bem sucedidas, a análise actual e as ramificações que ainda faltam analisar para completar o nó em causa.

A interacção é efectuada, ou através de botões, ou através de teclas específicas, que permitem ao próprio utilizador efectuar o algoritmo passo a passo, ou então que seja a própria ferramenta a efectuá-lo. Para além da utilização dos botões, na execução manual do algoritmo, o utilizador pode seleccionar uma regra da gramática que será, caso possível, imediatamente aplicada à situação actual. No caso do algoritmo ter sucesso, isto é, ter sido possível fazer corresponder o texto em causa à gramática, é mostrada essa informação na barra de estado das operações e é mantida a visualização da respectiva árvore. Caso o algoritmo falhe, é indicado esse resultado na barra de estado, e não é mostrada nenhuma árvore na área de visualização.

3.2.1.2 SrParser

A ferramenta SrParser também faz parte do *NTLK LITE* e usa o algoritmo *Shift Reduce Parser* (Schabes, 1991) para a construção de árvores sintácticas.

Conforme se mostra na figura 3.2, o sistema de visualização desta ferramenta é semelhante ao descrito para o sistema RdParser, relativamente à forma de apresentar a gramática, a barra de estado e os botões de acção. A diferença entre as duas visualizações consiste no modo como o texto é visualizado: enquanto na situação anterior o texto é estático, nesta visualização o texto é dinâmico. A partir do momento em que uma parte do texto é identificado como texto que já foi analisado e encontrada correspondência na gramática, esse texto passa da zona de *Remaining Text* para a zona de *Stack*. Desta forma, a ferramenta SrParser distingue o texto já analisado do texto que ainda falta analisar, através da zona de visualização em que este se encontra, enquanto no sistema RdParser era feito através de distinção de cores.

O algoritmo destaca as regras da gramática que podem ser aplicadas, cabendo ao utilizador escolher a regra que naquele momento pretende aplicar. De referir que nesta ferramenta não existe o modo automático para a execução do algoritmo.

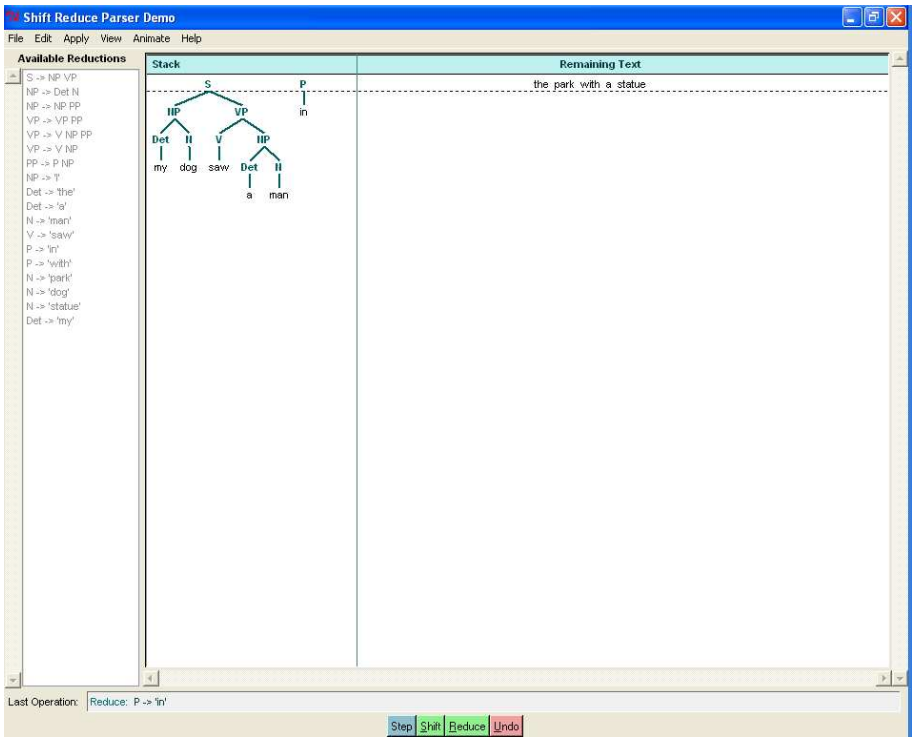

Figure 3.2: Visualização da interface gráfica da ferramenta SrParser.

3.2.1.3 Chart

A ferramenta Chart Parsing é usada para a construção de árvores sintácticas. É uma ferramenta destinada à aprendizagem, que permite a mistura de vários tipos de *parsing*, nomeadamente, uma estratégia *Top-Down*, uma estratégia *Bottum-up* ou uma estratégia baseada no algoritmo de *Earley* (Earley, 1970), sendo assim possível compreender as diferenças entre eles.

De acordo com a figura 3.3 é possível distinguir três secções no sistema de visualização: a parte superior mostra a árvore correspondente ao passo da análise que se encontra seleccionado, a parte central mostra o texto em análise e a parte inferior está reservada para a aplicação das estratégias de análise. A correspondência entre estas três secções é efectuada através de blocos verticais que dividem o texto a analisar em palavras.

A aplicação das diferentes estratégias pode ser efectuada passo a passo ou de forma automática. No final, é possível especificar um determinado passo da análise e observar o estado da árvore nesse instante, bem como a relação entre os elementos da árvore e o texto. Esta relação é representada através de linhas verticais, tal como se observa na figura 3.3.

No caso do passo do algoritmo seleccionado ter originado duas árvores, essa informação é indicada através de texto, sendo apenas mostrada de imediato uma das árvores. O utilizador dispõe de botões

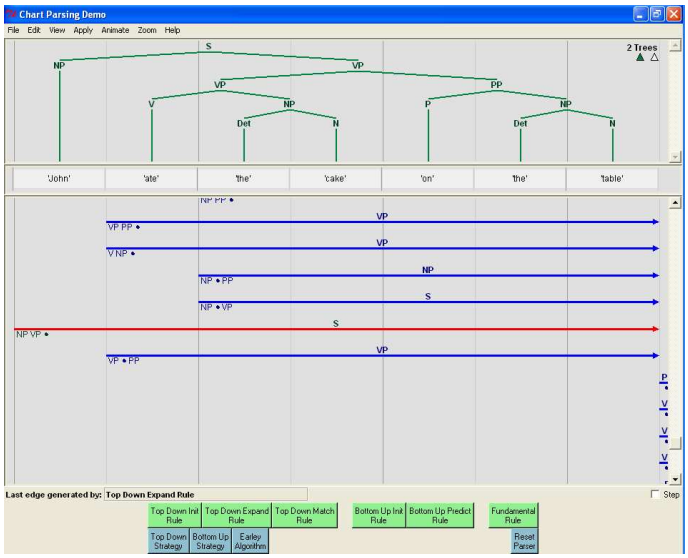

Figure 3.3: Visualização da interface gráfica da ferramenta Chart.

que lhe permitem alternar entre as diversas árvores. A indicação da árvore actual em visualização é obtida através da alteração da cor do botão respectivo.

No final da aplicação do algoritmo, a ferramenta permite ainda visualizar todas as árvores geradas, tal como se apresenta na figura 3.4.

3.2.2 Transcriber

A ferramenta *Transcriber* pode ser usada para anotação manual de sinais de áudio (Barras et al., 2001). A visualização principal desta ferramenta (Geoffrois et al., 2000) é composta por duas partes, de acordo com o que se observa na figura 3.5. A parte superior é o local onde é introduzido o texto transcrito e respectivas anotações. A parte inferior serve para visualizar o sinal áudio e para visualizar as anotações em camadas.

Na parte superior o texto é organizado em secções, segmentos e oradores. As secções servem para indicar o tópico ou o assunto de um conjunto de segmentos. O orador identifica a pessoa que naquele momento se exprimiu. A diferenciação entre secções e oradores é feita através da utilização de cores distintas. Um segmento representa uma parte da transcrição sendo representado visualmente por um pequeno círculo, seguido do texto que representa a transcrição. As anotações são introduzidas no meio do texto transcrito, distinguindo-se deste através da utilização de cores distintas.

É possível definir a resolução do sinal de áudio que está a ser representado. No caso da janela de visualização não ser suficiente para a representação do sinal, são acrescentadas barras de deslocamento

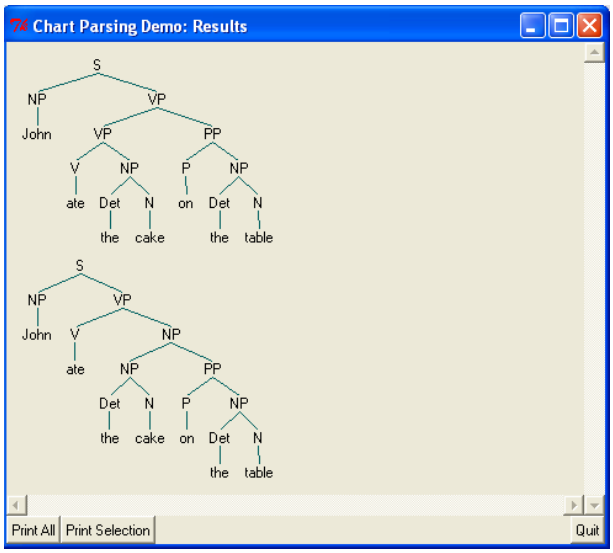

Figure 3.4: Visualização de resultados na ferramenta Chart.

de modo ao utilizador ter a possibilidade de navegar ao longo do mesmo.

A representação das camadas é feita através de cores e são alinhadas com o sinal de áudio. Encontram-se hierarquicamente organizadas em: segmentos, orador, tópico ou assunto e som de fundo.

No caso de haver sobreposição de sinais (por exemplo, quando existem dois oradores a expressarem-se em simultâneo), passa a haver dois sinais de áudio e a camada correspondente ao segmento que representa a transcrição é dividida em duas.

Existe uma interligação visual entre as várias partes da visualização, isto é, caso se seleccione uma parte do sinal de áudio, uma camada ou uma parte do texto transcrito é sempre destacado visualmente, e em simultâneo, a posição das três partes.

O utilizador tem a possibilidade de adicionar novos oradores, indicar características destes, como por exemplo o sexo, adicionar novas secções, ou adicionar diferentes categorias de anotação.

3.2.3 Annotation Graph Toolkit (AGTK)

O sistema de anotação AGTK (Bird & Liberman, 1999) serve de base para representar informação em grafos, constituídos por nós e arcos. Segundo Bird e Liberman (Bird & Liberman, 1999), é difícil conseguir transmitir a informação contida num grafo. As técnicas de visualização devem ser adaptadas a cada tipo de dados e a cada aplicação. No entanto, existem aspectos que podem ser generalizados. A informação temporal normalmente induzida por setas pode ser substituída por uma correcta representação dos arcos, e os diversos tipos de arcos podem ser representados por diferentes cores ou

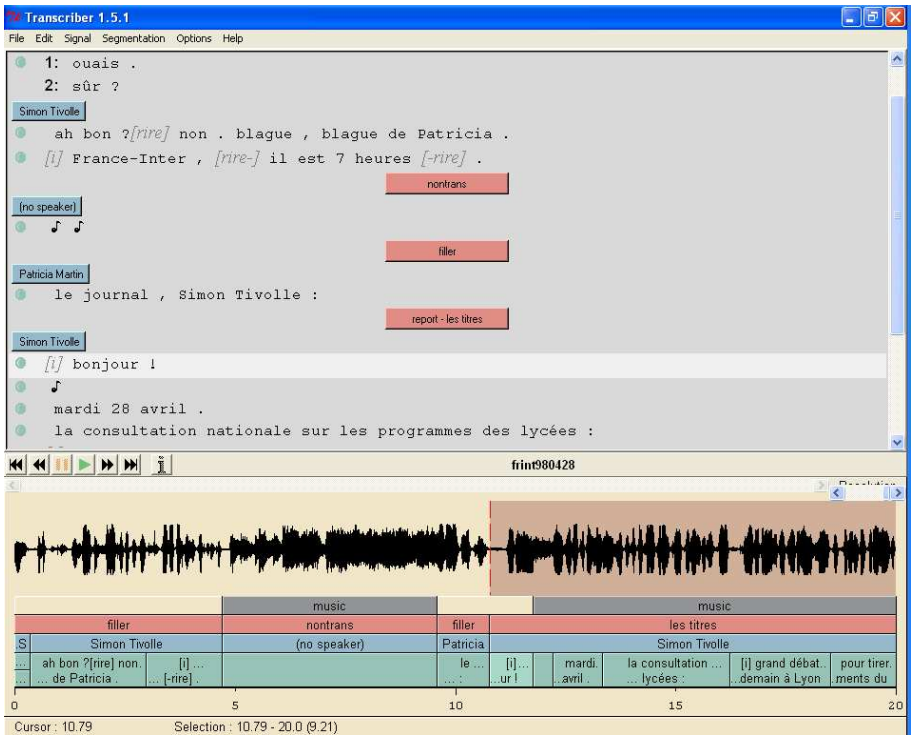

Figure 3.5: Visualização da interface gráfica da ferramenta Transcriber.

variação da posição vertical.

Na solução adoptada, a representação de arcos foi feita por rectângulos rotulado e o fim dos arcos e relações entre arcos representados por uma linha vertical.

3.2.3.1 TableTrans

A ferramenta TableTrans (*Manual Software, TableTrans*, 2004) pode ser usada para anotações linguísticas relacionada com sinais áudio, fundamentalmente para transcrição de sinal de voz para texto. Permite fazer anotações de regiões do sinal caracterizando-as com uma identificação do orador, da transcrição e das características da transcrição.

Visualmente (Bird et al., 2004) encontra-se dividido em duas partes, tal como se mostra na figura 3.6. A parte superior contém uma tabela, onde as linhas correspondem a uma região do sinal e cada coluna a uma dada característica de anotação, como por exemplo, identificação do orador, início de segmento, fim de segmento ou transcrição.

Na parte inferior encontra-se representado o sinal de áudio através da ferramenta WaveSurfer. Na existência de dois oradores é possível visualizar o espectro de som correspondente a cada um dos

oradores. É ainda possível acrescentar camadas de análise do sinal de áudio, como por exemplo espectogramas e indicadores temporais.

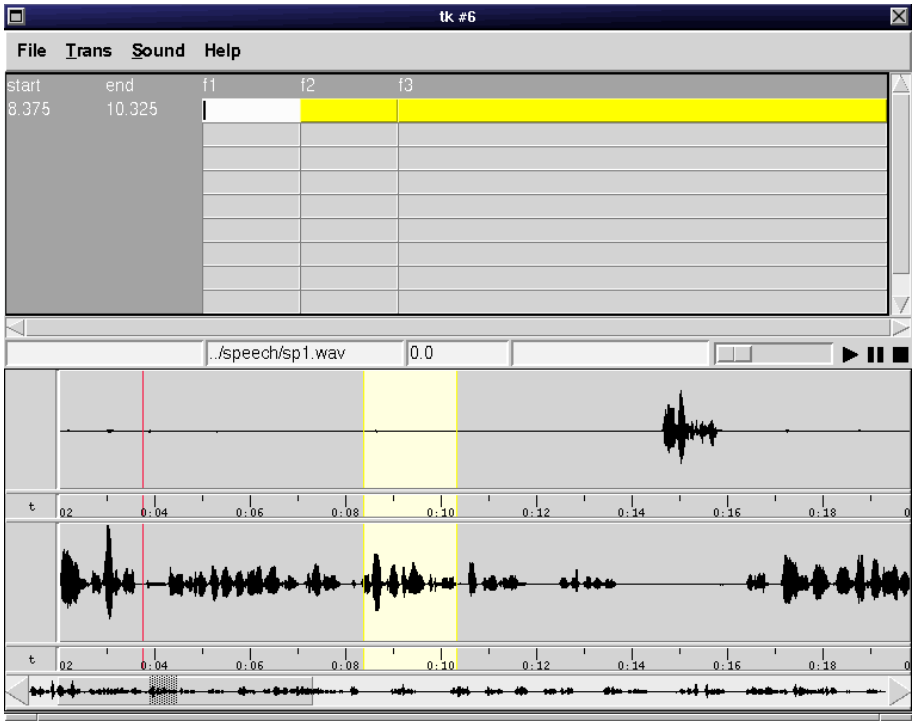

Figure 3.6: Visualização da interface gráfica da ferramenta TableTrans.

3.2.3.2 Multi-Channel Transcription

A ferramenta Multi-Channel Transcription (Bird et al., 2002) tem um aspecto gráfico muito semelhante com o sistema Transcriber. A principal diferença é a separação entre os discursos dos diversos oradores. No caso de haver vários oradores, cada orador tem uma janela própria onde é vista a sua transcrição.

3.2.4 General Architecture for Text Engineering (GATE)

O sistema *GATE Graphical Interface* (GGI) (Rodgers et al., 1997) pode ser usado para visualizar dados ou o resultado da execução de ferramentas integradas no sistema de anotação GATE. Está dividido em duas secções: construção de um grafo para controlar as interdependências das várias ferramentas e módulos de visualização para ver a informação proveniente das diversas ferramentas.

A construção do grafo de dependências pode ser obtida indicando os dados que têm que existir para o funcionamento de cada ferramenta, e os dados gerados por elas.

A visualização de informação pode ser classificada em dois tipos: texto com sobreposição de cores

ou quadros e visualizações mais complexas que englobam relações entre anotações no formato de grafo acíclico. Existem alguns tipos de visualização já implementados: visualizações *Single-Span* e *Text-Atribute* que associam diferentes cores a cada tipo de categorização ou visualização *Multiple-Span* que é usada para cadeias de anotações que consistem numa lista de anotações que inclui referências entre anotações.

Após a execução dos algoritmos (*Software GATE-General Architecture for Text Engineering 3.1*, 2006) é possível observar o seu resultado com base no corpus original. Na figura 3.7 é apresentado um exemplo.

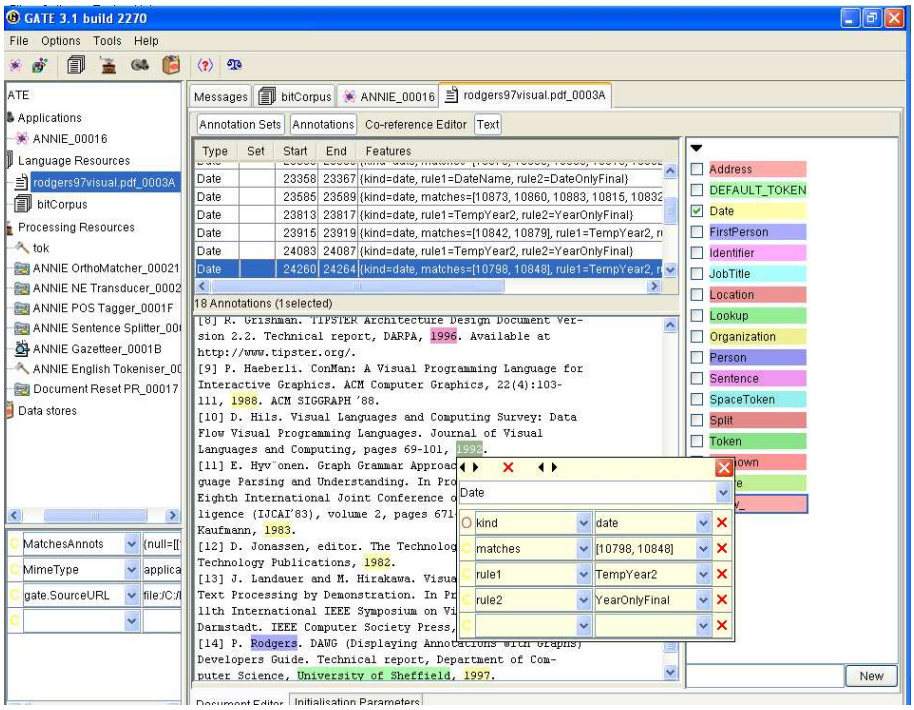

Figure 3.7: Visualização da interface gráfica do sistema GATE.

Para além do texto, o utilizador tem a possibilidade de visualizar as anotações e os conjuntos de anotações (*Developing Language Processing Components with GATE Version 4 (a User Guide)*, 2006). Os conjuntos de anotações servem para organizar as anotações por categorias. Algumas das categorias possíveis são: datas, localizações ou primeiro nome de pessoas. As categorias são diferenciadas através de cores, sendo possível ao utilizador seleccionar as categorias que deseja ver em cada momento. Ao seleccionar uma categoria, o texto existente no corpus anotado com essa categoria é destacado com a mesma cor. O utilizador tem ainda a possibilidade de gerir as cores que quer atribuir a cada categoria. A listagem das categorias encontra-se ordenada alfabeticamente para uma melhor identificação por parte do utilizador.

No caso do utilizador ter seleccionada a opção de ver as anotações, é adicionada uma tabela por

cima da zona onde se encontra o texto. Apenas são mostradas as anotações cujos conjuntos de anotações se encontrem seleccionados. As colunas da tabela são preenchidas com os seguintes campos: um para a identificação do conjunto, dois que delimitam em número de caracteres o local onde o texto se encontra e outro para as características da anotação. Cada linha corresponde a uma anotação.

Para cada anotação o utilizador tem ainda a possibilidade de visualizar e alterar as suas características. Para isso, basta fazer duplo clique na anotação que pretende visualizar ou alterar, e de seguida é mostrado um menu *pop-up* com as características já existentes.

No caso da janela de visualização não ser suficiente são adicionadas barras de deslocamento, à janela em causa.

3.2.5 Unstructured Information Management Architecture (UIMA)

O sistema UIMA desenvolvido pela IBM e amplamente apresentado na secção 2.3, é um sistema que tem como objectivo inferir informação a partir de dados não estruturados e organizar essa informação de forma estruturada.

O sistema contém um conjunto de aplicações para facilitar a interacção por parte do utilizador / programador de aplicações de Processamento de Língua Natural. No caso do utilizador usar o editor Eclipse, são disponibilizados *plugins* que permitem a configuração dos diversos componentes existentes através de uma interface. A alternativa é editar ou criar os ficheiros em XML através de um editor de texto.

O sistema tem disponível uma ferramenta para visualizar os resultados das anotações. Conforme se verifica na figura 3.8, essa ferramenta é composta por três áreas: o texto que foi anotado, as legendas e os detalhes da anotação.

As legendas referem os tipos de anotações que foram aplicados, sendo cada tipo identificado com uma cor distinta. Esta diferenciação permite uma melhor percepção das anotações existentes e onde elas se aplicam.

O texto original encontra-se presente, sendo salientadas as diferentes anotações que podem ser aplicadas ao texto que em cada instante se tem disponível na área de visualização. Cada tipo de anotação aparece com a mesma cor com que aparece na área das legendas, sendo possível seleccionar os tipos de anotação que se pretendem ver. No caso da área de visualização não ser suficiente para mostrar todo o texto é adicionada uma barra de deslocamento vertical.

Os detalhes de anotação são usados para fornecer mais informação acerca das anotações existentes. Quando é escolhida uma anotação na área do texto, é actualizada a área de detalhes. Esta área é constituída pelas anotações seleccionadas e pelos atributos que caracterizam essa anotação (normalmente

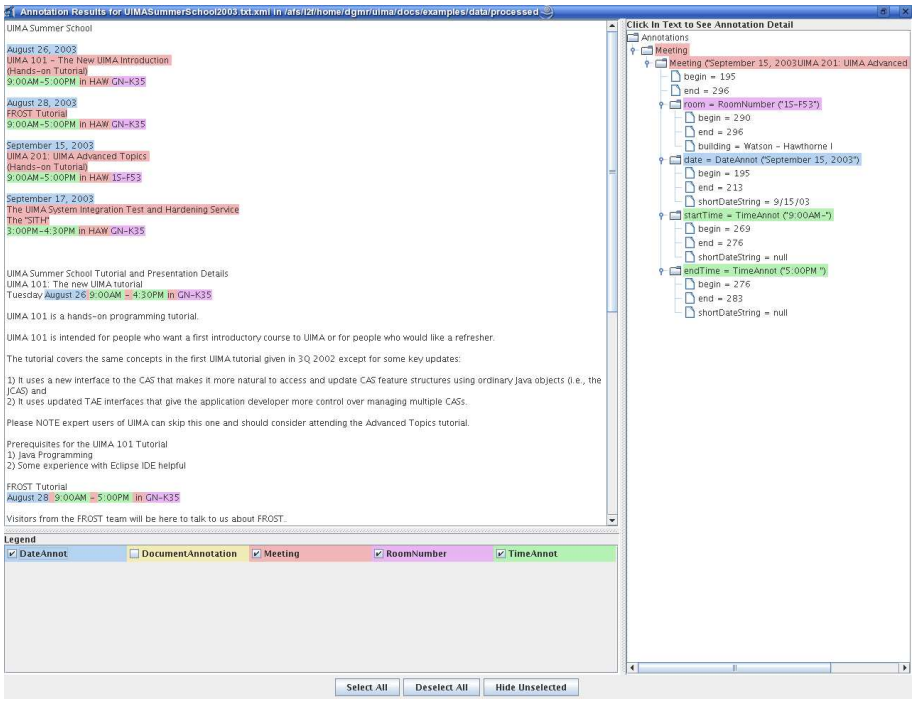

Figure 3.8: Visualização da interface gráfica do sistema UIMA.

begin e *end*, podendo existir outros). No caso de existir mais do que uma anotação e de estas serem de vários tipos, elas são organizadas hierarquicamente em árvore, sendo dada mais prioridade ao tipo de anotação, depois à anotação e por fim às propriedades.

Se a anotação escolhida for formada a partir de outras anotações, essa informação também fica disponível na zona de detalhes através de um esquema em árvore. Quando se selecciona a anotação principal é possível navegar na árvore de modo a obter as anotações que constituem essa agregação.

3.3 Interface gráfica do sistema ShRep

De seguida é descrita a interface gráfica que foi desenvolvida para o sistema ShRep.

O principal motivo que impossibilita a reutilização das interfaces gráficas das ferramentas RdParser, SrParser e Chart do sistema NLTK, e das ferramentas Transcriber e TableTrans (sistema AGTK) é o facto destas interfaces gráficas estarem orientadas para representar informação específica, ficando limitada a sua utilização. A interface gráfica do sistema NLTK está orientada para representar árvores sintácticas, enquanto que a interface gráfica das ferramentas Transcriber e TableTrans está orientada para representar transcrições e respectivas características, de sinais de áudio para texto.

Os sistemas GATE e UIMA são sistemas que se aproximam mais do sistema ShRep, no sentido

em que foram desenvolvidos com o objectivo de servir de base à integração de várias ferramentas de anotação. No entanto, as interfaces gráficas destes dois sistemas estão orientadas para o modelo de dados que cada uma suporta. A eventual utilização de uma das interfaces gráficas destes sistemas obrigaria a uma transformação dos dados, a partir do modelo utilizado no sistema ShRep, para o modelo de dados utilizado pelo sistema que fosse escolhido, efectuada através de um alinhamento dos modelos de dados. No entanto, verifica-se a existência de elementos no modelo de dados do sistema ShRep, como por exemplo a representação de alternativas, que dificulta a realização desse alinhamento.

Apesar de não ser utilizada nenhuma das interfaces gráficas avaliadas, foi analisada a informação que está presente e o modo como esta é visionada em cada um dos sistemas.

3.3.1 Descrição da interface gráfica do sistema ShRep

Para a implementação da interface gráfica havia três abordagens possíveis:

- Interface gráfica agregada ao servidor;
- Interface gráfica separada do servidor com um novo porto de acesso;
- Interface gráfica separada do servidor, utilizando o mesmo porto que as ferramentas de anotação.

A primeira abordagem tem a desvantagem de só permitir uma visualização da interface, tendo a vantagem de estar mais perto do servidor, eliminando, deste modo, problemas e ineficiências provenientes dos protocolos de comunicação.

A segunda abordagem cria um novo porto de acesso no servidor, distinto do utilizado pelas ferramentas de anotação. Tem a vantagem de poder utilizar um protocolo de acesso ao servidor distinto do usado pelas ferramentas, sendo possível expandir a interacção com o servidor, com o objectivo do servidor poder informar a interface das actualizações existentes nos dados que esta tem para visualizar. A principal desvantagem está relacionada com a necessidade do servidor ter conhecimento da informação existente na interface gráfica.

A terceira abordagem tem a vantagem da interface de programação, também designada API, já estar definida, tendo no entanto a desvantagem de necessitar de inquirir o servidor sobre eventuais actualizações dos dados que esta possui em visualização.

A abordagem escolhida foi a terceira, dado já ter a API definida, sendo apenas necessário expandi-la, considerando que a interface a implementar é apenas uma interface de depuração, isto é, para verificar a correcta integração das ferramentas de anotação no sistema ShRep.

Em termos de alteração da API apenas se estenderam os Iteradores de modo a estes se deslocarem quer para a frente, quer para trás, e de modo a colocá-los numa posição específica. Para este efeito,

foram adicionados os métodos *set*, *hasPrevious* e *previous* à interface que é disponibilizada pelo sistema ShRep.

Um dos problemas encontrados para implementar a interface gráfica do sistema ShRep é a complexidade do seu modelo conceptual. O facto do seu modelo conceptual ser constituído por vários elementos que se encontram interligados, faz com que exista muita informação a visualizar na sua interface gráfica. Deste modo, a interface gráfica implementada dá mais ênfase às características do modelo conceptual e às anotações produzidas pelas ferramentas, do que ao corpus que foi analisado.

Conforme se mostra na figura 3.9, a interface implementada apresenta pouca informação em cada contexto de visualização. Para suprir este problema, a interface gráfica possui uma grande capacidade de interacção, permitindo que o utilizador navega facilmente pela informação disponível com o objectivo de encontrar a informação que pretende visualizar.

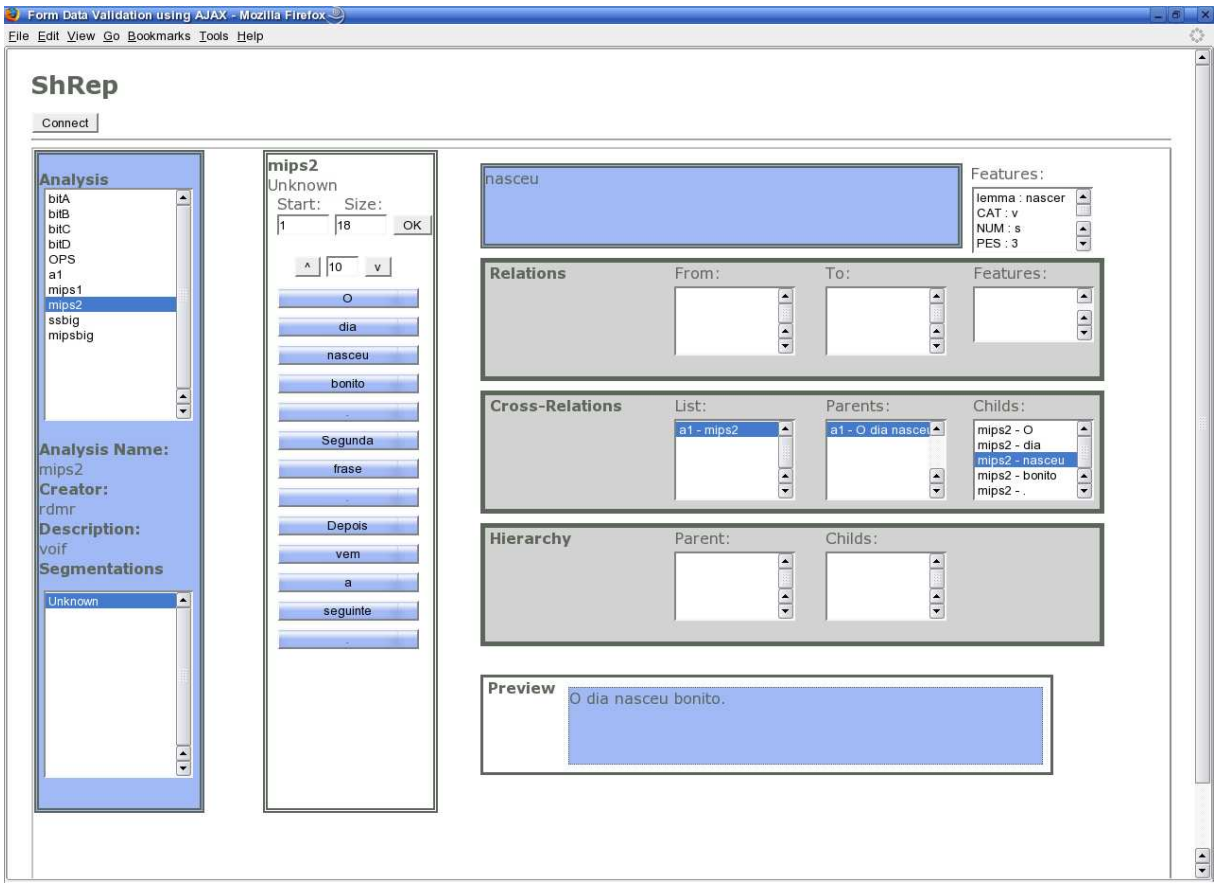

Figure 3.9: Interface gráfica para o sistema ShRep

A interface gráfica é composta por três secções, dispostas verticalmente da seguinte forma: a secção mais à esquerda contém a listagem das análises (zona superior) e a listagem das segmentações (zona inferior). A secção central contém a listagem dos segmentos. A secção mais à direita é composta pela

descrição pormenorizada de um segmento.

Inicialmente as três secções encontram-se sem informação. A interacção entre a interface gráfica e o utilizador começa no momento em que o utilizador carrega no botão *Connect* para efectuar a ligação com o servidor. Como resposta, obtém a listagem das análises existentes. Ao seleccionar uma análise, o utilizador tem disponível informação genérica sobre a análise e a listagem das segmentações que constituem a análise escolhida. Após escolher uma segmentação, é possível visualizar os primeiros segmentos da mesma. Devido à escassez de espaço, apenas é possível visualizar 18 segmentos de cada vez, havendo no entanto a possibilidade de navegar ao longo dos segmentos através dos botões existentes na parte superior da secção que contem a listagem dos segmentos. É possível indicar a partir de que segmento se pretende iniciar a visualização, a quantidade de segmentos a visualizar e indicar o número de segmentos que se pretende deslocar de cada vez.

Ao seleccionar um dos segmentos é mostrada a informação relacionada com o mesmo na secção de especificação do segmento. Na parte superior é mostrado o texto correspondente à região do sinal de dados do segmento. No canto superior direito são mostradas as características do segmento. Na parte central são mostradas as relações, sendo listado os segmentos que formam relação com o segmento seleccionado, quer onde o segmento seleccionado participa como origem, quer como destino. É também mostrada uma listagem das relações cruzadas e um nível de hierarquia, indicando, caso exista, o segmento ascendente e os segmentos descendentes do segmento escolhido. Ao seleccionar um dos elementos que compõe a relação, são mostradas as características que compõe a relação entre o segmento activo e o segmento seleccionado na relação. Ao seleccionar uma relação cruzada são mostrados os segmentos que compõe essa relação. De modo a facilitar a interacção com a ferramenta, é possível seleccionar, na terceira secção, os segmentos existentes nas relações, nas relações cruzadas e na hierarquia, apresentando-se uma visualização prévia do segmento. Ao efectuar-se duplo clique no segmento que se encontra na visualização prévia, passa esse segmento a ser o primeiro segmento visível na segunda secção, sendo mostrada a informação associada a esse segmento na terceira secção.

Em termos de tecnologia foi usado *Web Services* para efectuar a interacção entre o browser e a biblioteca cliente de acesso ao servidor, e usada a tecnologia *Ajax* (Garrett, 2005), tendo sido usado a implementação proposta por *Open Rico* (Dave Crane, 2005) , para facilitar a actualização da informação.

3.4 Análise comparativa

A tabela 3.1 compara as características encontradas nos sistemas de visualização analisados com a interface gráfica desenvolvida para o sistema ShRep. Nesta tabela, "V" indica que a característica em causa é representada pela ferramenta, enquanto "NV" indica que a característica em causa não é verificada pela ferramenta.

Característica	Rd Parser	Sr Parser	Chart	Trans-criber	Table Trans	GATE	UIMA	ShRep
Representa àrvores sintácticas	V	V	V	NV	NV	NV	NV	V
Representa gramáticas	V	V	NV	NV	NV	NV	NV	NV
Representa sinais de áudio	NV	NV	NV	V	V	NV	NV	NV
Tem barra de estado	V	V	V	NV	NV	NV	NV	NV
Tem barra de deslocamento quando existe muita informação	V	V	V	V	V	V	V	V
Permite interacção com a ferramenta	V	V	V	V	V	V	V	V
Permite edição das anotações efectuadas	NV	NV	NV	V	V	V	NV	NV
Usa diferenciação de texto através de cores	V	NV	NV	V	NV	V	V	NV
Usa diferenciação de texto através da sua posição	NV	V	NV	NV	NV	NV	NV	NV
Usa diferenciação dos diversos elementos linguísticos através de cores	V	V	V	V	NV	V	V	V
Tem ligação entre os diferentes elementos presentes	V	NV	V	V	V	V	V	V
Existe uma listagem das anotações	NV	NV	NV	NV	V	V	V	V

Table 3.1: Quadro comparativo das características dos sistemas de visualização.

Da análise da tabela 3.1 é possível concluir:

- todos os sistemas analisados contêm barras de deslocamento para a eventualidade de a informação a mostrar ser superior ao tamanho do espaço reservado para essa informação;
- todos os sistemas analisados permitem interagir com o sistema de visualização, permitindo explorar as várias anotações efectuadas;
- a maioria dos sistemas apresenta relações gráficas entre os vários elementos de visualização;
- é bastantes frequente o uso de diferenciação através de cores, quer para o texto que foi anotado, quer para os vários elementos linguísticos existentes;
- alguns sistemas permitem editar e alterar as anotações existentes de modo a efectuar pequenas correcções.

Destas características descritas a interface gráfica do sistema ShRep apenas não satisfaz as duas últimas.

4 Integração de ferramentas

4.1 Introdução

O objectivo de integrar mais ferramentas no sistema ShRep é permitir criar cadeias de processamento, com um mínimo de funcionalidade. O sistema ShRep isoladamente tem apenas como função servir de repositório de dados às várias ferramentas que se encontram integradas no sistema. São as ferramentas que têm como função produzir resultados, tendo por base os dados existentes no repositório.

As ferramentas até agora integradas no sistema ShRep são as seguintes:

- *InitialTextCreator*. Esta ferramenta coloca no repositório um texto que se encontre num ficheiro;
- *SentenceSplit*. Esta ferramenta segmenta um sinal de dados em frases, criando os respectivos segmentos;
- MIPS. Esta ferramenta atribui para cada palavra as classes gramaticais a que essa palavra pode pertencer.

No decorrer desta tese foram integradas mais duas ferramentas, sendo a primeira o MARv (Ribeiro et al., 2003) que é uma ferramenta de desambiguação morfosintáctica, e a segunda o Palavroso (Medeiros, 1995) que é uma ferramenta de anotação que tem por finalidade a atribuição de classes gramaticais a palavras. De seguida é apresentada uma descrição aprofundada destas duas ferramentas e do processo de integração efectuado.

4.2 MARv

Esta secção efectua uma apresentação da ferramenta, uma descrição da análise feita e das melhorias implementadas, e por fim, apresenta o processo de integração da ferramenta no sistema ShRep.

4.2.1 Descrição da ferramenta

MARv é uma ferramenta de desambiguação desenvolvida por Ricardo Ribeiro, no contexto da realização da sua Tese de Mestrado, tendo por objectivo a desambiguação morfosintáctica. No decorrer

de uma análise pode verificar-se a existência de uma palavra à qual pode ser atribuída mais do que uma classe gramatical, tal como se pode observar na figura 4.1.

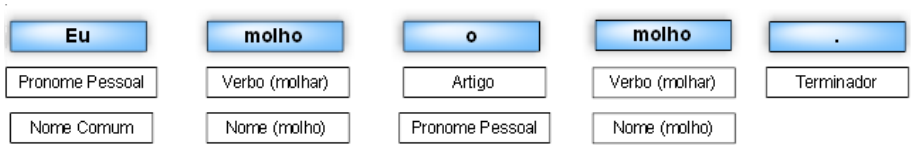

Figure 4.1: Exemplo de frase classificada com algumas classes gramaticais, contendo ambiguidade.

Para solucionar este problema, o MARv implementa o Algoritmo de Viterbi (Viterbi, 1967), que se baseia em modelos probabilísticos de classes gramaticais. A arquitectura do MARv encontra-se representada na figura 4.2, podendo ser descrita em quatro componentes:

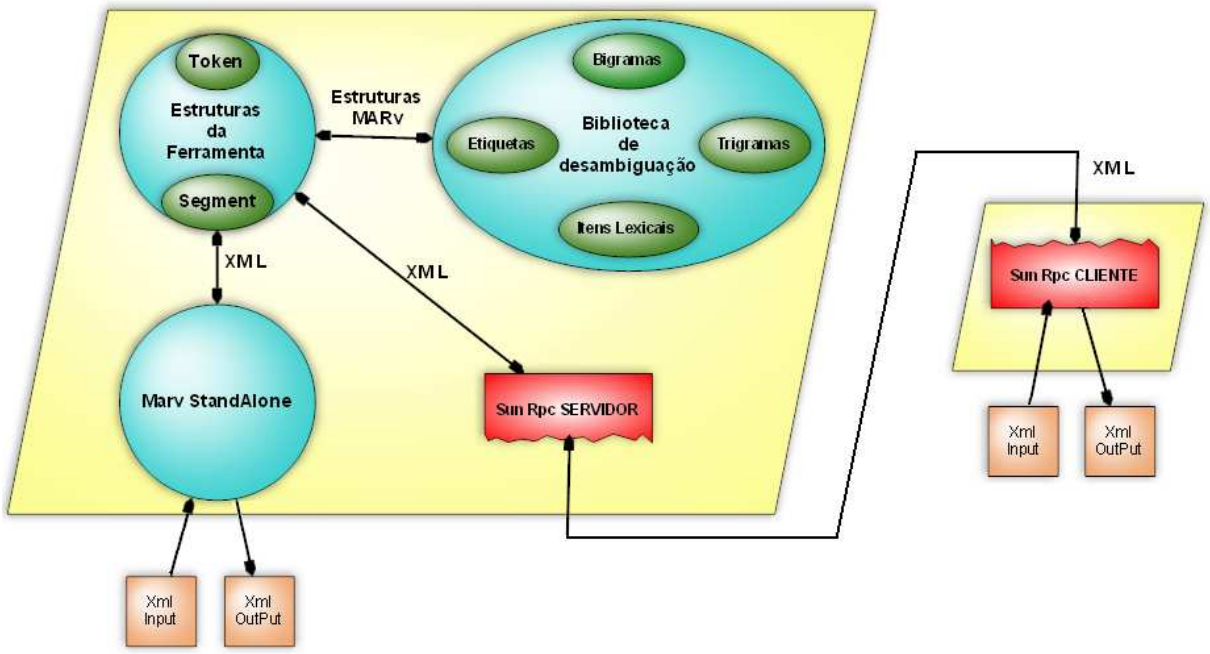

Figure 4.2: Arquitectura do MARv.

- Um componente que é responsável por efectuar a desambiguação através da implementação do Algoritmo de Viterbi, contendo os dicionários utilizados. Estes são constituídos pelas etiquetas; pelas probabilidades dos bigramas e dos trigramas formados através da combinação das várias etiquetas; e pelos itens lexicais que indicam para uma dada palavra a probabilidade dela aparecer num texto com uma dada classe gramatical;
- Um componente que contém as estruturas de dados utilizadas pela biblioteca de desambiguação, sendo constituído pelas classes *Segment* e *Token*, onde um *Segment* representa um conjunto de To-

kens, e um *Token* representa uma palavra que se pretende desambiguar juntamente com as várias classes gramaticais a que a palavra pode pertencer;

- Um componente de aplicação para o funcionamento no formato *standalone*;
- Um componente de aplicação para o funcionamento no formato cliente-servidor.

Todos os componentes estão implementados na linguagem C++, sendo usado Sun-Rpc (Srinivasan, 1995) para efectuar a comunicação no formato cliente-servidor. Os dados de entrada, correspondendo ao texto que se pretende desambiguar, e os dados de saída, correspondendo ao texto já desambiguado, são representados em formato XML de acordo com a definição de tipos, vulgarmente designado por DTD, representado na figura 4.3.

```
<?xml version='1.0' encoding='iso-8859-1' ?>
<!--dtd for MARV's input and output -->

<!ELEMENT MARv (sentence)*>

<!ELEMENT sentence (word)+>

<!ELEMENT word (class)+>
<!ATTLIST word name CDATA #REQUIRED>

<!ELEMENT class EMPTY>
<!ATTLIST class root CDATA "">
<!ATTLIST class pos CDATA #REQUIRED>
<!ATTLIST class tag CDATA #REQUIRED>
<!ATTLIST class clitic CDATA #IMPLIED>
<!ATTLIST class marv_selection CDATA #IMPLIED>
```

Figure 4.3: DTD usado inicialmente pela ferramenta.

4.2.2 Alterações implementadas

Após uma análise prévia ao funcionamento da ferramenta MARv verificou-se a necessidade de proceder a algumas alterações, com o objectivo de melhorar o tempo de processamento, a quantidade de memória utilizada e diminuir a taxa de erro que a ferramenta produz ao efectuar a desambiguação.

Para melhor analisar o efeito de cada alteração em cada um dos três parâmetros anteriormente referidos foi utilizado um texto de análise constituído por 57 498 palavras e por 102 776 etiquetas, correspondendo, em média, a cerca de 1,78 etiquetas por palavra. Existe ainda o texto pré-analisado manualmente que contém para cada palavra a classe gramatical correctamente atribuída, o que permite avaliar a taxa de erros que a ferramenta produz.

A análise dos três parâmetros foi efectuada para a leitura dos dicionários, para o funcionamento da ferramenta na versão *standalone* e na versão cliente-servidor. A análise à leitura dos dicionários foi efectuada de uma forma independente do resto da aplicação para permitir avaliar o impacto das diversas alterações apenas neste componente da ferramenta. Esta funcionalidade é significativamente relevante dado a leitura dos dicionários ser uma das principais diferenças entre as duas versões existentes, uma vez que na versão cliente-servidor é efectuada apenas uma vez, enquanto na versão *standalone* é efectuada sempre que se pretende desambiguar um texto.

De seguida é apresentada, de uma forma aprofundada, a análise às diversas alterações implementadas, explicando a razão que levou a cada alteração assim como o impacto que teve nos três parâmetros analisados, ou seja, em termos de tempo de processamento, de quantidade de memória utilizada, e de número de erros produzidos. Foram considerados os seguintes pontos:

- A análise efectuada a cada alteração tem sempre como base de comparação os resultados produzidos pela última alteração descrita;
- Os tempos apresentados correspondem à média de executar a aplicação 30 vezes;
- A comparação entre as várias alterações foi efectuada tendo em conta o tempo indicado pelo parâmetro *Real*, dado este corresponder ao tempo que o utilizador da ferramenta espera efectivamente pela desambiguação;
- A quantificação de memória utilizada foi calculada através do auxílio da ferramenta *Valgrind* (Nicholas Nethercote, 2003) tendo sido considerada a informação que resulta do cálculo de alocação de memória que é indicado por esta ferramenta. Apenas foi efectuada a medição de memória até ao fim da leitura dos dicionários utilizados pelo MARv, dado que se pretende medir, em termos de memória, o impacto que as várias alterações aos dicionários provocaram no uso da mesma.

Todos os testes da ferramenta MARv foram executados numa máquina com um único processador *Intel Pentium 4 CPU 3,20 GHz*, tendo 1 GB de memória, um disco de 32 GB, tendo sio utilizado o sistema operativo Suse Linux 10.2.

4.2.2.1 Estado inicial

Efectuando-se a análise descrita na secção anterior à ferramenta sem ter sido efectuada qualquer alteração obtém-se os resultados apresentados na tabela 4.1, que servem como referência inicial.

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	6.7s	68.1s	69.3s
	User	6.6s	0.2s	69.0s
	Sys	0.03s	0.1s	0.4s
Memória		757 627 MB		
Número Erros		3928 [Taxa erro global : 6,83%]		

Table 4.1: Resultados iniciais.

4.2.2.2 Suspensão das probabilidades lexicais

Após uma breve análise à implementação do Algoritmo de Viterbi verificou-se que também estava implementada uma extensão no que diz respeito às probabilidades lexicais. O Algoritmo de Viterbi utiliza as probabilidades lexicais das classes existentes nas últimas palavras, juntamente com a probabilidade de ocorrer a palavra que está a ser análisada e as classes que lhe estão associadas. Nesta extensão, em vez de ser considerada a probabilidade de dada palavra ocorrer com determinada classe gramatical, é considerada a sequência das últimas três palavras que formam a frase em análise, considerando que se trabalha a nível de trigramas.

Após analisar o desempenho do funcionamento do algoritmo com e sem a utilização desta extensão, verificou-se, tal como se apresenta na tabela 4.2, que o algoritmo tem um desempenho melhor, em termos de número de erros, quando usado na sua versão original, isto é, sem a extensão. Em termos de taxa de erro, esta passa dos 6,83% iniciais para 5,91%.

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	6.5s (-2.44%)	44.9s (-34.02%)	52.7s (-24.01%)
	User	6.5s (-2.48%)	0.2s (-5.98%)	52.3s (-24.19%)
	Sys	0.04s (+4.09%)	0.1s (-3.23%)	0.4s (+9.47%)
Memória		757 627 MB (+0.00%)		
Número Erros		3396 (-15.66%) [Taxa erro global : 5,91%]		

Table 4.2: Resultados sem probabilidades lexicais, e comparação com resultados inicais.

Do ponto de vista teórico, a existência da extensão ao algoritmo de Viterbi resultaria na introdução de mais informação para a desambiguação, o que deveria traduzir-se numa menor taxa de erro. Neste caso, tal facto não se verifica devido ao texto que foi usado para treino não ter tamanho significativo e não conter informação significativa para o nível de exigência necessário para captar este tipo de informação.

Relativamente ao tempo de processamento verifica-se que a leitura dos dicionários manteve-se praticamente inalterada, devido ao facto de os mesmos não terem sofrido qualquer tipo de modificação. No entanto, o tempo de processamento da ferramenta para efectuar a desambiguação do texto de análise

diminui significativamente justificado pelo facto de se ter retirado a extensão já referida, que requeria algum tempo de processamento.

Em relação à memória utilizada para a leitura dos dicionários não se verifica qualquer modificação, uma vez que os dicionários são os mesmos.

4.2.2.3 Mudança de estruturas de entrada de dados

A ferramenta MARv estava apenas habilitada a funcionar com entradas de dados no formato XML, sendo posteriormente convertidas para estruturas de dados existentes na ferramenta MARv. No entanto, um dos propósitos desta tese é a integração de ferramentas no sistema ShRep, implicando assim que MARv terá que suportar um novo formato de entrada e saída de dados. Com o objectivo de evitar a tradução de tipos de dados do sistema ShRep para XML e novamente deste para estruturas usadas pela ferramenta MARv, houve a necessidade de desenvolver no MARv uma nova funcionalidade que facilite esta integração, e que, simultaneamente, também simplifique a integração de outros sistemas ou a introdução de outro tipo de entrada ou de saída de dados.

Desta forma, foi criada uma abstracção de dados que possibilita que a ferramenta MARv funcione com as suas estruturas de dados internas, sendo possível iniciá-las através de diversos tipos de fontes de dados. Como se mostra na figura 4.4, foi criada uma classe abstracta designada *iomanager* que estabelece as várias fases do processamento, sendo cada uma das classes concretas responsável por transformar os dados no formato que sabe interpretar, para estruturas de dados usadas pela ferramenta MARv.

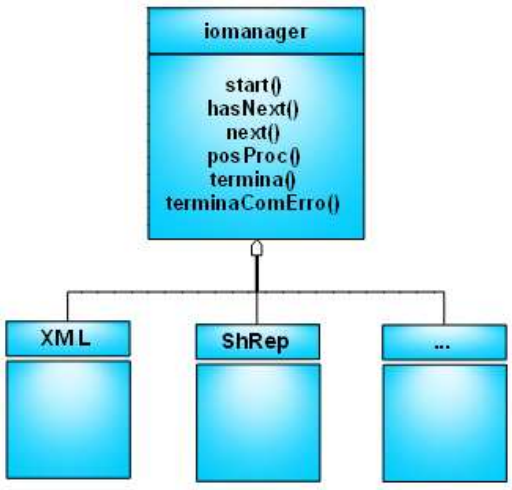

Figure 4.4: Diagrama de classes que gere as fontes de entrada de dados.

Tal como se verifica na tabela 4.3 esta alteração provocou um aumento no tempo do processamento da ferramenta, justificado pela criação de novas classes, que levam a que estas se encontrem mais in-

terligadas e menos dependentes, o que torna mais fácil o seu entendimento, simplificando o processo de desenvolvimento.

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	6.8s (+4.39%)	60.3s (+34.22%)	67.2s (+27.55%)
	User	6.8s (+4.35%)	0.2s (+6.59%)	66.8s (+27.66%)
	Sys	0.03s (-3.57%)	0.1s (+8.99%)	0.4s (+12.18%)
Memória		757 714 MB(+0.01%)		
Número Erros		3396 (+0.00%) [Taxa erro global : 5,91%]		

Table 4.3: Resultados após efectuar alteração às estruturas de entrada de dados, e comparação com última alteração efectuada.

4.2.2.4 Conversão de probabilidades em logaritmos

O MARv estava implementado sob um modelo probabilístico, que tinha a condicionante de perda de precisão, que provém do facto do modelo probabilístico assentar sobre a multiplicação de valores que se encontram compreendidos no intervalo entre 0 e 1, o que resulta em valores tão pequenos que torna difícil representá-los computacionalmente. Esta limitação implica que só se possam efectuar análises a pequenos conjuntos de segmentos, tendo sido designado o valor 7 como valor óptimo, de acordo com os testes efectuados na altura do desenvolvimento do MARv. No caso de se desejar efectuar a análise a um texto com um elevado número de palavras, só é possível analisar conjuntos de 7 segmentos de cada vez, o que implica que se tenha alguma perda de precisão nos resultados alcançados, dado que de 7 em 7 segmentos o processo é reiniciado. Cada vez que o processo é reiniciado o primeiro segmento é analisado por si só, o segundo segmento já tem em conta o primeiro, mas só a partir do terceiro segmento é que é tida em conta o máximo de informação disponível, considerando que se trabalha ao nível de trigramas.

Uma possível solução para resolver o problema da precisão é manter a desambiguação com probabilidades, usando uma janela deslizante de 7 segmentos. No entanto, esta abordagem tem o problema de ser necessário ter informação de contexto dos 7 segmentos.

Uma solução alternativa, que não requer tanta informação de contexto é a alteração do modelo probabilístico para um modelo de logaritmos das probabilidades. Esta alteração implica a passagem das multiplicações e divisões existentes para somas e subtracções, respectivamente. A conversão de probabilidades em logaritmos torna-se possível dado que a função logarítmica é contínua e crescente, o que permite manter a coerência de resultados entre os dois modelos.

Uma das alterações efectuadas no MARv foi então a conversão de probabilidades, utilizadas nos Dicionários, em logaritmos, como se pode observar na figura 4.5, passando a aplicar a operação aritmética da soma e subtracção em vez da operação aritmética do produto e da divisão. Com esta alteração, surge

a possibilidade de serem analisadas frases, independentemente da sua dimensão, em vez de conjuntos limitados de segmentos, dado que a operação aritmética da soma entre dois valores fraccionários não altera o nível de precisão do resultado da mesma.

```
2809                                    2809
Nc+Nc Nc+Nc 0.0000027318500088    Nc+Nc Nc+Nc -12.81053151937342
Nc+Nc A.    0.0755084797270865    Nc+Nc A.     -2.58351031476788
Nc+Nc Xy+Xy 0.0000004202846167    Nc+Nc Xy+Xy -14.68233369636653
Nc+Nc Xf+Xf 0.0000008405692335    Nc+Nc Xf+Xf -13.98918651568761
Nc+Nc V.+Pi 0.0000210142308371    Nc+Nc V.+Pi -10.77031069083845
```

Figure 4.5: Pequeno extracto de bigramas com probabilidades e com logaritmos.

Foi aplicado o texto de análise referido anteriormente, tendo sido utilizados conjuntos limitados de sete segmentos, de forma a evidenciar apenas o impacto da alteração do modelo probabilístico para o modelo de logaritmos. Não foram analisados, nesta fase, os impactos da passagem de sete segmentos para um conjunto maior, dado ter-se como objectivo a desambiguação orientada à frase e o texto de análise não conter informação indicativa de início das mesmas. Os resultados provenientes da alteração do uso de probabilidades para logaritmos encontram-se representados na tabela 4.4, dos quais se retiram as seguintes conclusões:

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	6.9s (+1.15%)	60.8s (+0.89%)	70.4s (+4.79%)
	User	6.8s (+1.24%)	0.2s (+3.09%)	70.0s (+4.77%)
	Sys	0.03s (-0.74%)	0.1s (+0%)	0.4s (4.17%)
Memória		757 714 MB (+0.001%)		
Número Erros		3396 (+0.00%) [Taxa erro global : 5,91%]		

Table 4.4: Resultados utilizando logaritmos, e comparação com última alteração efectuada.

- Relativamente à última alteração verifica-se que a passagem dos dicionários de probabilidades para logaritmos não provocou grandes alterações na leitura dos mesmos. Relativamente ao tempo de processamento verificado pela aplicação do texto de análise verifica-se um aumento, embora não muito significativo ao utilizar logaritmos em vez de probabilidades;
- Relativamente à taxa de erros conclui-se, tal como era esperado, que não há variação da taxa aquando da aplicação de logaritmos em vez de probabilidades, mantendo a análise a 7 segmentos. Mais tarde irá efectuar-se uma análise que permite verificar o impacto da extensão da análise a mais que 7 segmentos;
- Em termos de memória utilizada até ao momento da leitura dos dicionários verifica-se uma subida, embora não significativa.

4.2.2.5 Conversão dos dicionários do formato textual para o formato numérico

Nos dicionários utilizados pelo MARv, as etiquetas e as palavras necessárias à análise eram representadas com uma descrição textual. Após uma avaliação ao modo como o tempo estava a ser usado pela aplicação foi constatado que 50% do tempo utilizado na análise servia para o processamento de operações relacionadas com cadeias de caracteres. Sabendo que o tempo de processamento de valores numéricos é mais rápido que o de cadeias de caracteres, converteu-se a descrição textual das etiquetas e palavras, usadas nos dicionários, em valores numéricos.

Para auxiliar esta conversão, foi desenvolvido um módulo de código que gera os dicionários no novo formato, atribuindo a cada etiqueta e a cada palavra um valor numérico distinto. Na figura 4.6 encontra-se uma parte dos dicionários representados em formato textual à esquerda e em formato numérico à direita. Esta modificação deu origem a mais um ficheiro e a uma estrutura de dados para os dicionários constituída pelas palavras que podem ocorrer nos itens lexicais, que agora se encontram numericamente codificadas. A única alteração verificada na fase de desambiguação, prende-se com a necessidade de conversão das palavras que se pretendem desambiguar e das respectivas classes gramaticais do formato texto com que vêm da entrada de dados, para o formato numérico. No entanto como as palavras existentes e as classes gramaticais se encontram carregadas com o programa, esta transformação torna-se fácil de efectuar.

```
2809                                2809
Nc+Nc Nc+Nc -12.81053151937342      1 1 -12.81053151937342
Nc+Nc A.     -2.58351031476788      1 0  -2.58351031476788
Nc+Nc Xy+Xy -14.68233369636653      1 2 -14.68233369636653
Nc+Nc Xf+Xf -13.98918651568761      1 3 -13.98918651568761
Nc+Nc V.+Pi -10.77031069083845      1 4 -10.77031069083845
```

Figure 4.6: Pequeno extracto de bigramas em formato texto e em formato numérico.

Foi aplicando o texto de análise já descrito anteriormente tendo sido utilizados os resultados conseguidos com a última alteração efectuada como termo de comparação. Na tabela 4.5 são apresentados os resultados deste estudo e deles se retiram as seguintes conclusões:

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	0.8s (-87.80%)	8.3s (-86.42%)	9.2s (-86.94%)
	User	0.8s (-88.00%)	0.2s (-16.86%)	8.9s (-87.31%)
	Sys	0.02s (-48.13%)	0.1s (+0.85%)	0.3s (-30.92%)
Memória		26 223 MB (-96.54%)		
Número Erros		3396 (+0.00%) [Taxa erro global : 5,91%]		

Table 4.5: Resultados utilizando dicionários em formato numérico, e comparação com última alteração efectuada.

- Em relação ao tempo de processamento da análise conclui-se que para todas as dimensões analisadas existe uma redução bastante significativa do tempo utilizado, sendo que, para a leitura de dicionários se verificou uma redução de 87,80%, para a versão cliente-servidor verificou-se uma redução de 86,42% e para a versão *standalone* verificou-se uma redução de 86,94%. Esta redução do tempo de processamento deve-se ao facto de o formato numérico dos dicionários permitir a sua representação através de um tipo de dados primitivo, o que se traduz numa diminuição do custo de processamento associado às operações efectuadas, isto é, tanto no carregamento dos dados uma vez que há menos informação a ler, bem como na utilização da informação através das operações de pesquisa e comparação;

- Relativamente à taxa de erros conclui-se, tal como era esperado, que não há variação da taxa aquando da conversão dos dicionários do formato textual para o formato numérico;

- Em termos de memória utilizada até ao momento da leitura dos dicionários verifica-se uma diminuição bastante acentuada (96.54%), resultante da substituição da representação dos elementos que compõe os dicionários de cadeias de caracteres para valores numéricos inteiros. A representação de cadeias de caracteres é efectuada por um objecto composto o que resulta na ocupação de mais espaço de memória quando comparada com a simples utilização de um valor inteiro.

4.2.2.6 Mudança de estruturas do tipo *Map* para estruturas do tipo Vector Multidimensional

Inicialmente, o MARv guardava os bigramas, os trigramas e os itens lexicais, que constituem os dicionários, em estruturas de dados do tipo *Map*. A conversão dos dicionários em formato numérico facilita a representação destes dicionários em estruturas de dados do tipo Vector multidimencional, pois, sendo as etiquetas e os índices do vector de formato numérico, permitem uma indexação directa entre eles. Por exemplo, a uma etiqueta à qual tenha sido atribuída o valor inteiro 1, será inserida directamente na posição do vector cujo índice é 1. Esta alteração permitirá um aumento do desempenho tanto a nível de carregamento como a nível de pesquisa dos valores que se pretendem obter.

Na situação da representação dos dicionários usando um formato numérico, os bigramas passam a ser representados por um vector bidimensional correspondendo cada dimensão às várias etiquetas existentes. Da mesma forma, os trigramas podem ser representados por um vector tridimensional. Os itens lexicais podem ser representados por um vector bidimensional, em que uma das dimensões corresponde às palavras e a outra dimensão corresponde às etiquetas.

Para os bigramas e trigramas a taxa de ocupação de informação útil dos vectores é elevada (96,36% e 94,55%, respectivamente), pelo que se justifica a substituição do tipo de estruturas de dados usado na sua representação. Para os itens lexicais verifica-se que a taxa de ocupação de informação útil no vector

é baixa (2.93%), já que, por norma, uma palavra apresenta um número reduzido de classes gramaticais associadas, o que conduz a uma utilização inapropriada de memória, caso sejam representados por vectores, dado os vectores alocarem espaço para todas as combinações dos seus índices, enquanto que as estruturas do tipo *Map* só alocam espaço para os elementos que efectivamente contêm. Ou seja, os potenciais ganhos em tempo acabam por não compensar as potenciais perdas em memória, pelo que, os itens lexicais se mantêm representados por estruturas de dados do tipo *Map*.

Desta forma, os resultados conseguidos com a alteração descrita, encontram-se representados na tabela 4.6, através dos quais se retiram as seguintes conclusões:

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	0.6s (-32.66%)	7.7s (-6.42%)	8.2s (-11.24%)
	User	0.6s (-32.59%)	0.2s (+1.55%)	7.8s (-12.01%)
	Sys	0.01s (-36.69%)	0.1s (-2.10%)	0.3s (+8.53%)
Memória		12 712 MB (-51.52%)		
Número Erros		3396 (+0.00%) [Taxa erro global : 5,91%]		

Table 4.6: Resultados utilizando vectores multidimensionais, e comparação com última alteração efectuada

- Em relação ao tempo de processamento da análise conclui-se que para todas as dimensões analisadas existe uma redução do tempo utilizado, sendo que, para a leitura de dicionários se verificou uma redução de 32,66%, para a versão *standalone* verificou-se uma redução de 11,24% e para a versão cliente-servidor verificou-se uma redução de 6,42%. A redução do tempo de processamento da leitura dos dicionários deve-se ao facto de a inserção dos elementos em formato vectorial se realizar por indexação directa, enquanto que para o formato tipo *Map* é necessário calcular o local para a sua inserção. A diferença percentual verificada entre as duas versões referidas pode ser explicada pelo facto de a versão *StandAlone* efectuar a leitura dos dicionários, contendo assim o ganho obtido na leitura dos mesmos. Na versão Cliente-Servidor apenas é ganho o tempo de acesso aos dicionários, que também acontece na versão *StandAlone*, dado a leitura dos dicionários já se encontrar efectuada no momento em que é feita a desambiguação;

- Relativamente à taxa de erros conclui-se, tal como era esperado, que não há variação da taxa aquando da substituição de estruturas do tipo *Map* por estruturas do tipo Vector multidimensional;

- Relativamente à memória utilizada até ao momento da leitura dos dicionários verifica-se uma redução para metade, quando se passa do uso de estruturas do tipo *Map* para estruturas do tipo Vector multidimensional. Esta diminuição é justificada pelo facto de a utilização de estruturas do tipo *Map* requerer mais informação adicional do que as estruturas do tipo Vector. Pode variar com a implementação, mas, por norma, as estruturas do tipo Vector apenas precisam da indicação do

número de elementos e da próxima posição a ser preenchida enquanto as estruturas do tipo *Map*, no caso de, por exemplo, estarem implementadas sobre a forma de *Red-Black-Trees* precisam de guardar para cada elemento informação acerca dos descendentes que possuem;

4.2.2.7 Alteração dos dicionários

Outra alteração efectuada no MARv consistiu em efectuar a desambiguação dos segmentos tendo em conta o início das frases, já que, em teoria, é possível que determinada classe gramatical seja mais frequente no início de frase. Para implementar esta alteração no MARv foi necessário efectuar duas outras alterações:

- Recriar os dicionários tendo em conta a informação de início de frase;
- Efectuar a desambiguação de frases completas em vez de conjuntos de sete segmentos.

Os dicionários utilizados até esta fase do trabalho não tinham em conta o início de frase, ou seja, a análise dos segmentos era efectuada tendo apenas em conta as classes gramaticais atribuídas a cada um, não considerando a possibilidade de dada classe do primeiro segmento ocorrer, com maior probabilidade, no início de frase. Tendo por objectivo fornecer mais informação para a desambiguação dos segmentos, foi necessário recriar os dicionários, incorporando esta nova informação. Outra alteração efectuada nestes novos dicionários prende-se com o facto de os segmentos não serem apenas palavras mas também poderem ser conjuntos de palavras que detêm significado próprio quando se encontram juntas (como por exemplo, ”em vão”).

A geração destes novos dicionários alterou o número de elementos presentes nestes. Os bigramas passaram de 2 809 entradas para 1 255, e os trigramas passaram de 148 877 para 42 875 entradas. Esta diminuição de 55.32% e 71.20% para bigramas e trigramas, respectivamente, é justificada principalmente com o desaparecimento de etiquetas compostas, como por exemplo *Nc+Nc*, que reduz de forma significativa o número de combinações existentes. Não houve necessidade de incluir este tipo de etiquetas nestes dicionários dado as actuais cadeias de processamento onde MARv se encontra a ser utilizado não conterem estes tipo de etiquetas o que faz com que não seja requisitado ao MARv a desambiguação de segmentos que contenham etiquetas deste tipo. Relativamente ao número de itens lexicais verificou-se um ligeiro aumento de 24 849 para 28 466 pelo facto de o texto utilizado para gerar os dicionários conter um número maior de combinações palavra / etiqueta que o texto utilizado para gerar os dicionários anteriores.

Na tabela 4.7 encontram-se representados os resultados comparativos entre os dicionários usados anteriormente e os utilizados após esta alteração. Destes resultados é possível concluir que:

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	0.3s (-45.27%)	7.7s (-0.91%)	8.1s (-1.34%)
	User	0.3s (-45.75%)	0.2s (+0.93%)	7.7s (-1.02%)
	Sys	0.01s (-26.14%)	0.1s (-4.18%)	0.3s (-7.47%)
Memória		7 816 MB (-38.51%)		
Número Erros		3109 (-8.45%) [Taxa erro global : 5,41%]		

Table 4.7: Resultados de alterar dicionários, e comparação com última alteração efectuada.

- O tempo de processamento para a leitura dos dicionários diminuiu 45,27%, justificado através da redução do número de entradas já apresentado, sobretudo a nível de trigramas. Em termos de desambiguação verifica-se também uma diminuição, mas não tão relevante. No processamento da versão cliente-servidor verifica-se uma diminuição de 0.91% e na versão *standalone* verifica-se uma diminuição de 1.34%;
- Relativamente ao número de erros verifica-se uma diminuição de 8,45%, colocando a taxa de erro em 5,41%. É natural que a mudança de dicionários por si só provoque alterações na taxa de erro, dado ser através destes que a desambiguação é feita. O decréscimo na taxa de erro aqui verificado deve-se à diminuição do número de entradas dos dicionários o que provoca uma menor ambiguidade na resolução das várias hipóteses existentes;
- Em relação à memória utilizada até à leitura dos dicionários verifica-se uma diminuição, justificada pela redução do número de entradas existente nos dicionários.

4.2.2.8 Aumento da janela de análise para mais de 7 segmentos

Aquando da exposição sobre a alteração do modelo probabilístico para o modelo de logaritmos foi referida a possibilidade de analisar frases em vez de conjuntos de sete segmentos. Até este ponto ainda não tinha sido avaliado o efeito desta alteração. Esta análise é apresentada neste momento pelo facto de só agora o texto e os dicionários utilizados conterem informação respeitante ao início de frase, o que permite a separação eficaz das mesmas.

A tabela 4.8 representa a descrição comparativa entre a utilização de análises de conjuntos de sete segmentos e análises de frases completas, de onde é possível concluir o seguinte:

- Relativamente às alterações verificadas no tempo de processamento, estas podem ser consideradas irrelevantes, considerando as variações observadas;
- Em termos de número de erros verifica-se uma diminuição de 3,53% o que permite obter uma taxa de erro global de 5,22%. O texto analisado encontra-se dividido por 1999 frases, o que corresponde,

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	0.3s (+0.28%)	7.7s (+0.44%)	7.9s (-1.69%)
	User	0.3s (+0.09%)	0.2s (-1.93%)	7.6s (-1.74%)
	Sys	0.01s (+4.62%)	0.1s (-0.34%)	0.3s (+0.08%)
Memória		7 816 MB (+0.001%)		
Número Erros		2999 (-3.53%) [Taxa erro global : 5.22%]		

Table 4.8: Resultados de analisar frase completa em vez de 7 segmentos, e comparação com última alteração efectuada.

em média, a cerca de 28 palavras por frase. A principal vantagem deste sistema de análise orientada à frase não reduz, ao contrário do que se poderia supor, de forma significativa a taxa de erro. O que este sistema permite é a possibilidade de analisar a frase toda de uma só vez, contrapondo com a análise de 7 em 7 segmentos usada até aqui. Neste ponto verifica-se também a principal vantagem de usar logaritmos em vez de probabilidades. Se fossem utilizadas probabilidades num sistema orientado à frase iria acontecer que do sétimo segmento para a frente a desambiguação iria resultar em valores muito pequenos através dos quais seria difícil distinguir de forma clara qual a classificação mais apropriada para dado segmento. Com a utilização de logaritmos é possível aumentar a janela de análise e manter, ou mesmo melhorar, a taxa de erro existente;

- Uma vez que os dicionários utilizados são os mesmos, a variação de memória utilizada até à leitura dos mesmos é insignificante sendo provocada por pequenas alterações no código.

4.2.2.9 Considerar informação de início de frase na desambiguação

Com este novo texto já é possível analisar o impacto nos resultados finais de incluir a informação de início de frase. Considerando a informação de início de frase é possível suprimir a utilização de bigramas, dados estes terem sido utilizados no início da desambiguação de um conjunto de segmentos, pelo facto de ainda não haver informação suficiente para a construção de trigramas. Assim, para a primeira palavra de uma frase eram apenas consideradas as etiquetas que esta continha, para a segunda palavra eram consideradas as etiquetas desta e as etiquetas da primeira palavra, e só a partir da terceira palavra é que era aproveitada o máximo de informação disponível.

Com esta nova abordagem a utilização dos bigramas pode ser substituída por entradas especiais nos dicionários que indicam o início de frase, tendo sido inserida a etiqueta *SS* nos dicionários usados pelo MARv. Desta forma, utilizando-se sempre trigramas, a desambiguação para a primeira palavra de uma frase tem em conta as etiquetas da palavra bem como o facto delas surgirem no início de frase, através da entrada de trigrama composta por (*SS*, *SS*, etiquetas) e para a segunda palavra é considerado o trigrama composto por (*SS*, etiquetas da primeira palavra, etiquetas da segunda palavra), tendo assim em conta, não só as classes da primeira palavra, mas também o facto das etiquetas desta palavra surgirem em

segundo lugar numa frase.

Na tabela 4.9 são apresentados os resultados das alterações produzidas pela introdução da informação de início de frase, podendo concluir-se que:

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	0.3s (+0.08%)	7.6s (-1.27%)	8.0s (+1.12%)
	User	0.3s (-0.22%)	0.2s (+1.11%)	7.7s (+1.19%)
	Sys	0.01s (+16.18%)	0.1s (-3.25%)	0.3s (-1.55%)
Memória		7 815 MB(+0.001%)		
Número Erros		2995 (-0.13%) [Taxa erro global : 5.21%]		

Table 4.9: Resultados de considerar informação de início de frase, e comparação com última alteração efectuada.

- Da mesma forma que na alteração anterior, a introdução de informação de início de frase na desambiguação não se traduz em alterações relevantes, no que diz respeito a tempo de processamento;
- Com a introdução da informação de início de frase verificou-se uma diminuição da taxa de erro global para 5,21%. No entanto, o número de alterações verificadas relativamente ao último teste foi de 61 segmentos classificados de forma distinta, o que significa que houve segmentos que estariam bem classificados mas passaram a ser classificados de forma errada mas que houve outros que passaram a ficar bem classificados. Desta forma podemos concluir que, com os dicionários existentes, a informação de início de frase não traz nenhum acréscimo significativo à desambiguação;
- Uma vez que os dicionários utilizados são os mesmos, a variação de memória utilizada até à leitura dos mesmos é insignificante sendo provocada por pequenas alterações no código.

4.2.2.10 Alteração do DTD

Numa das cadeias de processamento em que o MARv está inserido, houve necessidade de alterar o DTD usado pelos dados que são transmitidos nessa mesma cadeia. No caso de se manter o MARv a funcionar com o DTD inicial, havia necessidade de fazer pré-processamento e pós-processamento fora da aplicação MARv, de modo a este poder ser utilizado. No entanto, com a nova arquitectura desenvolvida na ferramenta MARv tornou-se mais fácil introduzir diferentes fontes de dados, podendo ser considerada a existência de um formato de DTD diferente como uma nova fonte de dados. Para implementar este novo DTD, basta estender a classe já existente que processa XML, bastando alterar a leitura dos dados do formato XML para estruturas MARv, e posteriormente efectuar a escrita destas estruturas para o formato XML de acordo com o novo DTD. Na figura 4.7 encontra-se representada o

novo diagrama de classes de entrada de dados na ferramenta MARv, e na figura 4.8 a especificação do novo DTD introduzido.

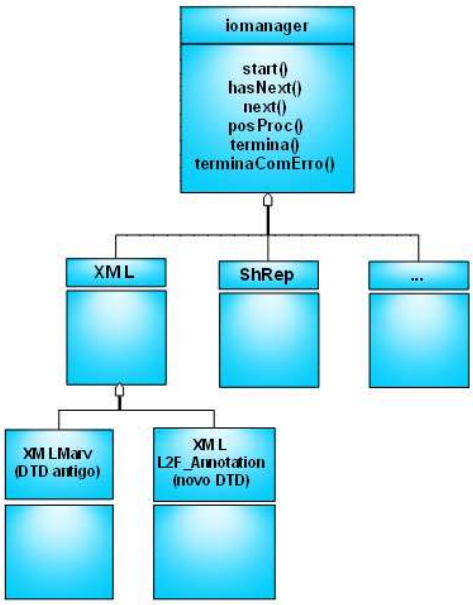

Figure 4.7: Diagrama de classes que gere as fontes de entrada de dados.

O resultado produzido pela ferramenta neste novo DTD, isto é, a indicação da classe escolhida na desambiguação encontra-se no campo HMM do DTD. No DTD inicialmente utilizado pelo MARv era adicionado um boleano a cada classificação, indicando se essa classificação tinha sido ou não a escolhida.

Para dar a possibilidade ao utilizador de poder indicar em qualquer momento o DTD que pretende usar, foi criada mais uma *flag* a ser preenchida aquando da execução da ferramenta. Aplicou-se os dados no novo formato do DTD de modo a verificar o impacto deste nas três vertentes que têm sido analisadas, tal como se apresenta na tabela 4.10, de onde se conclui o seguinte:

Parâmetro		Leitura Dicionários	Cliente-Servidor	*StandAlone*
Tempo	Real	0.3s (+3.92%)	13.0s (+71.70%)	13.5s (+68.78%)
	User	0.3s (+3.52%)	0.3s (+112.76%)	12.8s (+66.85%)
	Sys	0.01s (+0.00%)	0.2s (+101.51%)	0.6s (+100.64%)
Memória		7 818 MB (+0.03%)		
Número Erros		2995 (+0.00%) [Taxa erro global : 5.21%]		

Table 4.10: Resultados de alterar o DTD utilizado, e comparação com última alteração efectuada.

- Relativamente à leitura dos dicionários verifica-se uma pequena alteração, pouco significativa, dado que os dicionários utilizados são os mesmos. No entanto, no que se refere ao tempo de processamento na desambiguação verifica-se um aumento significativo, de cerca de 71.70% para a

```
<?xml version='1.0' encoding='iso-8859-15' ?>

<!ELEMENT l2f_annotation (sentence)*>
<!-- the main input is a sequence of sentences -->

<!ELEMENT sentence (word)*>
<!-- each sentence is a sequence of words (those in the sentence) -->

<!ELEMENT word (class)*>
<!ATTLIST word name CDATA #REQUIRED>
<!-- each word has a name (the string that appears in the text)
     and a morphological classification (class)                -->

<!ELEMENT class (id)*>
<!ATTLIST class root CDATA #REQUIRED>
<!-- each classification corresponds to a root (of the word)
     and a sequence of pairs attribute/value (id's)          -->

<!ELEMENT id EMPTY>
<!ATTLIST id atrib CDATA #REQUIRED
             value CDATA #REQUIRED>
<!-- each id is a pair attibute/value -->
```

Figure 4.8: Segundo DTD introduzido na ferramenta.

versão cliente servidor e de 68.78% para a versão *standalone*. Verifica-se este aumento dado que o novo DTD obriga à introdução de mais nós no ficheiro XML, isto é, para o mesmo texto descrito inicialmente passa-se de um ficheiro com 162 274 nós para um ficheiro com 367 826 o que se traduz num ficheiro maior a ser processado;

- Relativamente à taxa de erros conclui-se, tal como era esperado, que não há variação da taxa aquando da utilização de um DTD diferente no formato dos dados de entrada;
- Constata-se de novo, tal como era esperado, que não houve alterações relevantes no que se refere à memória utilizada até ao momento de efectuar a leitura dos dicionários.

4.2.2.11 Outras alterações

Para além das alterações já referidas foram ainda criadas algumas opções na execução do MARv, com o objectivo de tornar a aplicação mais versátil de modo a que o utilizador tenha a capacidade de moldar a execução da mesma, de acordo com as suas necessidades. Desta forma, foi criada uma opção que permite ao utilizador indicar a codificação que pretende aplicar aos dados de saída de XML. Foi dada ainda a possibilidade ao utilizador de poder executar a desambiguação com utilização das probabilidades lexicais. Apesar de se ter verificado que a utilização destas probabilidades lexicais produz piores resultados, pode haver casos em que esta situação não se verifique e os resultados produzidos possam

ser melhores.

Uma outra alteração introduzida nesta nova versão do MARv está relacionada com as demais etiquetas que fazem parte de um *Token*. Na versão inicial era considerado que um *Token* era sempre constituído por uma palavra e pela definição dos campos *pos*, *tag* e *root*, sendo que apenas o campo *pos* era usado na desambiguação. Nesta versão do MARv foi introduzida uma maior liberdade, sendo possível o *Token* ser constituído por um qualquer número de campos, continuando apenas o campo *pos* a ter interesse a nível de desambiguação. Os restantes campos são ignorados aquando da leitura dos dados, sendo posteriormente mantidos inalterados no momento de escrita dos resultados produzidos pela desambiguação.

No início das alterações efectuadas ao MARv efectuou-se uma alteração à sua arquitectura com o objectivo de facilitar a integração com novas fontes de dados. Contudo, verifica-se que cada fonte de dados poderá ter um tipo de parametrização diferente (por exemplo, no XML pode ser útil indicar a codificação dos dados de saída, enquanto para o ShRep é necessário definir a localização do servidor). Deste modo houve necessidade de criar uma estrutura de dados que abstraísse as diferentes parametrizações, sendo para tal criada a classe abstracta *CommandLineInterpreter*, sendo esta parametrização efectuada com base nos dados fornecidos na linha de comandos, aquando da execução da ferramenta MARv. No caso da ferramenta ser executada na versão cliente-servidor, esta parametrização é efectuada pelo cliente, permitindo assim diferentes parametrizações tendo apenas um único servidor.

A solução adoptada inicialmente tinha a desvantagem de toda a aplicação ser carregada sempre que se executa a ferramenta. Isto implica que se um utilizador estiver a processar um ficheiro de XML, o código respectivo ao tratamento de dados provenientes do sistema ShRep é também carregado, embora não seja utilizado. A solução encontrada para a resolução deste problema foi a criação de bibliotecas que possam ser carregadas dinamicamente, tendo sido utilizado o padrão de desenho *Abstract Factory* (Gamma et al., 1994). A utilização deste padrão possibilita a identificação e posterior carregamento do código necessário para efectuar a desambiguação para um determinado tipo de dados, sendo esta identificação efectuada em tempo de execução.

Com estas simplificações ainda foi possível reduzir o tempo de processamento, bem como a memória utilizada, tendo se mantido fixa a taxa de erro em ambas as definições de DTD existente, tal como se apresenta nas tabela 4.11 e 4.12.

4.2.2.12 Síntese das alterações

Na tabela 4.13 efectua-se uma comparação entre o estado inicial e o estado final da ferramenta, tendo sido utilizado o DTD que a ferramenta já usava inicialmente. Verificam-se ganhos bastante relevantes

DTD usado inicialmente				
Parâmetro		Leitura Dicionários	Cliente-Servidor	*standalone*
Tempo	Real	0.3s (-2.25%)	7.3s (-3.35%)	7.5s (-6.21%)
	User	0.3s (-2.14%)	0.2s (-2.2%)	7.3s (-5.67%)
	SYS	0.01s (-8.62%)	0.1s (-6.72%)	0.3s (-11.85%)
Memória		7 706 MB (-1.44%)		
Número Erros		2995 (+0.00%) [Taxa erro global : 5.21%]		

Table 4.11: Resultados de efectuar pequenas alterações, e comparação com a tabela 4.9

Novo DTD inserido na ferramenta				
Parametro		Leitura Dicionários	Cliente-Servidor	*Standalone*
Tempo	Real	0.3s (-5.30%)	12.1s (-7.16%)	12.2s (-9.97%)
	User	0.3s (-4.73%)	0.3s (-1.55%)	11.7s (-8.75%)
	Sys	0.01s (-8.86%)	0.2s (-7.94%)	0.6s (-27.36%)
Memória		7 706 MB (-1.44%)		
Número Erros		2995 (+0.00%) [Taxa erro global : 5.21%]		

Table 4.12: Resultados de efectuar pequenas alterações, e comparação com a tabela 4.10

a nível de tempo de processamento (95.46%, 89.22% e 89.17%, respectivamente para leitura dos dicionários, versão Cliente Servidor e versão *StandAlone*) e a nível de memória utilizada até ao fim da leitura dos dicionários (98.98%), verificando-se ainda um ganho relativo de 23.72% na taxa de erro, ficando assim a taxa de erro global em 5.21%, para o texto analisado.

4.2.3 Análise à performance da ferramenta

Esta subsecção efectua duas análises ao comportamento do algoritmo.

4.2.3.1 Ordens de grandeza do comportamento do algoritmo

Foi efectuada uma análise ao comportamento do algoritmo através do aumento do tamanho do ficheiro de entrada, provocando o aumento quer do número de palavras, quer do número de classificações existentes. Efectuou-se a medição para o dobro, o triplo e o quádruplo do ficheiro inicialmente usado, onde se apresenta os resultados na tabela 4.14.

Nesta análise verifica-se que o tempo de processamento acompanha o tamanho do ficheiro de entrada, sendo possível considerar que existe uma relação directa entre o número de palavras e classificações existentes no ficheiro de entrada, e o tempo de processamento da ferramenta.

Parâmetro	Versão Inicial	Versão Final	Variação Percentual
Prob. Lexicais	Sim	Não	
Dicionários			
Modelo	Probabilístico	Logarítmico	
Formato	Textual	Indices Numéricos	
Estruturas de dados	*Map*	Vectores Multidimensionais	
Número Entradas	176 535	72 566	
Análise			
Número Segmentos	7	Orientado à Frase	
Inf. Início Frase	Não	Sim	
Tempos			
Leitura Dicionários	6.7 s	0.3 s	-95.46%
Cliente-Servidor	68.1 s	7.3 s	-89.22%
StandAlone	69.3 s	7.5 s	-89.17%
Memória Utilizada	757 627 MB	7 706 MB	-98.98%
Taxa de erro	6.83%	5.21%	-23.72%

Table 4.13: Avaliação entre versão inicial e versão final.

Versão	Versão Inicial	Dobro	Triplo	Quádruplo
StandAlone	7.508s	14.955s	22.507s	30.401s
Cliente-Servidor	7.339s	15.042s	22.504s	31.278s

Table 4.14: Avaliação após aumento do ficheiro de entrada.

4.2.3.2 Cliente-servidor VS *StandAlone*

Da tabela 4.14 é ainda possível verificar que a partir de uma dada dimensão do ficheiro de entrada a ferramenta produz melhores resultados de tempo de processamento quando executada na versão *StandAlone* que na versão Cliente-Servidor.

Em termos de funcionamento, a versão Cliente-Servidor distingue-se da versão *StandAlone* pelo facto de não necessitar de carregar os dicionários. Contudo, tem como trabalho extra passar a informação, via RPC, do lado Cliente para o lado Servidor no início da execução, e o Servidor tem que enviar os dados para o lado do Cliente quando já se encontram desambiguados.

Pela análise da referida tabela é possível estimar que só a partir de dimensões do ficheiro com pouco mais de três vez do usado inicialmente, é que compensa, em termos de tempo de processamento, usar a ferramenta na versão *StandAlone*. Isto verifica-se porque só a partir desse ponto é que a quantidade de informação que é transmitida entre Cliente e Servidor demora mais tempo que o carregamento dos dicionários. O tempo de transmissão da informação é substancialmente pequeno, pelo facto de a execução do Cliente e do Servidor ocorrerem na mesma máquina e pelo facto de toda a informação ser transmitida numa única chamada do lado Cliente para o lado Servidor.

4.2.4 Integração no sistema ShRep

Tendo a abstracção já efectuada em termos de entrada e saída de dados do sistema MARv, a integração com o sistema ShRep ficou facilitada. Desta forma apenas foi necessário criar a classe *inputShRep* e a classe *commandShRepInterpreter*, tendo sido para a primeira necessário definir os métodos que se encontravam definidos na classe abstracta *iomanager*.

De início é estabelecida a comunicação com o servidor ShRep, através dos dados fornecidos na linha de comandos, que já se encontram tratados pela classe *commandShRepInterpreter*. Como o ShRep ainda não possui uma camada semântica foi necessário assumir que algumas condições se verificam, das quais se destacam:

- Os segmentos que correspondem a segmentos de início ou segmentos de fim de frase, contêm em si uma classificação a indicar esse tipo de informação;
- Os segmentos contêm em si uma classificação que inclui uma *feature* que corresponde ao atributo *pos* necessário para efectuar a desambiguação.

Desta forma, apenas foi necessário efectuar um alinhamento entre as estruturas usadas pelo sistema ShRep, e as estruturas utilizadas pela ferramenta MARv. Na entrada de dados foi considerado que cada *Segment* de uma dada *Segmentation* do sistema ShRep corresponde a um *Token* na ferramenta MARv, e que cada *Classification* de um *Segment* no sistema ShRep corresponde a uma *Classification* na ferramenta MARv, retirando-se da primeira a informação *pos* que interessa para a desambiguação. A informação de início e fim de frase existente nas *Classification* de alguns *Segment* foi considerada para indicar à ferramenta MARv que já se encontra disponível uma frase completa para desambiguar.

Para a saída de dados é criada uma nova *Analysis* no sistema ShRep, onde são criadas novas *Classification* contendo a etiqueta escolhida pela ferramenta MARv. Na figura 4.9 é mostrado um exemplo de uma pequena transformação, onde as duas primeiras camadas representam, respectivamente, o sinal de dados e a análise que servirá de entrada à ferramenta MARv, e a terceira camada corresponde à análise produzida por essa ferramenta.

O processamento continua a ser efectuado frase a frase. Os resultados obtidos, usando a interacção com o sistema ShRep, encontram-se na tabela 4.15, de onde é possível retirar as seguintes conclusões:

- Em termos de tempo de processamento verifica-se uma grande discrepância entre a ferramenta ser executada tendo como fonte de dados um ficheiro XML, ou tendo como fonte de dados o sistema ShRep. Esta variação é provocada por ineficiências do sistema ShRep que se encontram analisadas no capítulo 5;

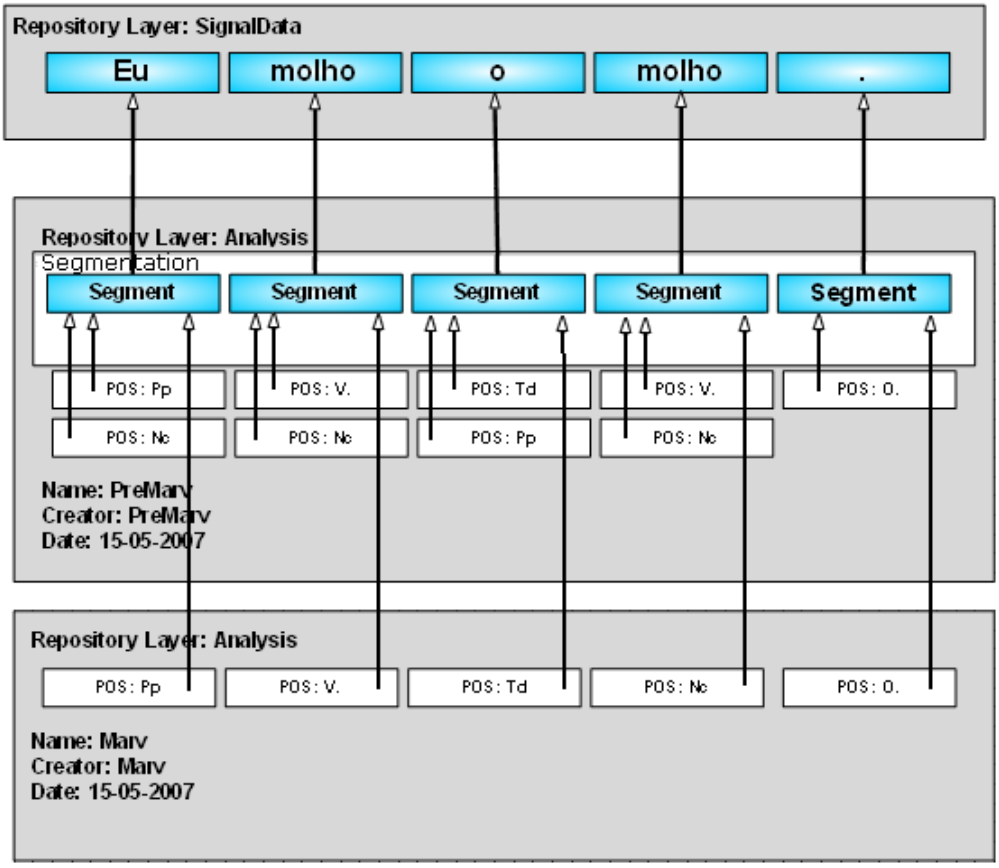

Figure 4.9: Criação de uma análise após a ferramenta MARv ter desambiguado um texto.

- Em termos de número de erros verifica-se que não há alteração, uma vez que a execução do algoritmo é independente da fonte de dados existente no momento de execução da ferramenta.

Umas das limitações encontradas no integração com o sistema ShRep é o facto de a actual arquitectura do MARv apenas possibilitar a entrada de dados provenientes do ShRep no funcionamento *Standalone*. Caso se pretenda correr a ferramenta MARv, em formato cliente-servidor, com o sistema ShRep, é necessário desenvolver um cliente e um servidor que funcionem com dados fornecidos pelo ShRep.

Parâmetro		ShRep
Tempo	Real	22m43s
	User	5m46s
	SYS	0m45s
Número Erros		2995 [Taxa erro global : 5.21%]

Table 4.15: Execução da ferramenta MARv, usando o Sistema ShRep como fonte de dados.

4.3 Palavroso

4.3.1 Descrição da ferramenta

O Palavroso é uma ferramenta de anotação que atribui a cada palavra classes gramaticais que lhe correspondem.

Apesar de já se encontrar integrada no sistema ShRep a ferramenta MIPS, que tem o mesmo objectivo que a ferramenta Palavroso, isto é, indicar classes gramaticais possíveis de uma dada palavra, foi integrada a ferramenta Palavroso, criando-se desta forma a possibilidade da criação de cadeias de processamento com maior variedade nas ferramentas utilizadas.

4.3.2 Integração no sistema ShRep

A integração desta ferramenta no Sistema ShRep foi efectuada de uma forma diferente de todas as ferramentas integradas até aqui. O processo de integração de ferramentas tem sido alcançado através da alteração do código das mesmas. No entanto, pode haver ferramentas que se pretendam integrar nas quais não possa ser possível alterar o código existente, por exemplo, pelo facto de o código não estar disponível, tornando-se assim necessário encontrar outro modo de efectuar a integração. No caso da ferramenta Palavroso, a razão que leva à escolha de outro modo de integração não é o facto de o código não estar disponível, mas sim o facto de tornar o processo de integração mais rápido.

Desta forma, o processo de integração considera a ferramenta a integrar como uma *blackbox*, no sentido em que não se preocupa com o funcionamento da ferramenta, nem o modo como se encontram codificados os algoritmos, tendo apenas em consideração a forma de executar a ferramenta, de perceber como lhe fornecer os dados de entrada e de como esta produz os dados de saída.

Assim, foi desenvolvido um módulo que funciona como intermediário entre a Biblioteca Cliente do Sistema ShRep e a ferramenta Palavroso. Este módulo tem como função solicitar os dados à Biblioteca Cliente, transformá-los em dados que a ferramenta Palavroso sabe interpretar e fornecê-los a esta ferramenta. Após a ferramenta ter executado o seu processo, este módulo tem como função interpretar o resultado da execução e inserir os dados necessário na Biblioteca Cliente.

A ferramenta Palavroso está preparada para receber quer frases quer palavras unitárias, podendo ser acedidas através de ficheiro ou através do *standard input*. Assim, o módulo de código desenvolvido coloca cada *Segment* existente numa determinada *Analysis* do sistema ShRep numa *String* e executa a ferramenta Palavroso. O resultado é direccionado para um ficheiro, sendo posteriormente analisado pelo módulo de código. Um exemplo de um ficheiro de resultados produzido pela ferramenta Palavroso encontra-se no figura 4.10, a partir do qual o módulo desenvolvido insere na *Analisys* que se está a formar um *Segment* com as respectivas *Classifications*, para cada palavra que se encontra nesse ficheiro.

```
O (Pd..=sm.== Td...sm... Pp..3sm.as Nc...sm...)
bonito (Nc...sm... A....smp..) dia (Nc...sm...) nasceu (V.is3s=...)
bonito (Nc...sm... A....smp..) . (O.........)
```

Figure 4.10: Exemplo do formato de uma frase após ter sido processada pela ferramenta Palavroso.

4.3.3 Desempenho da ferramenta

O facto de não ser utilizado qualquer código pertencente à ferramenta, sendo apenas efectuada uma chamada ao sistema operativo para executar a ferramenta, faz com que o tempo de processamento seja maior do que utilizar código da ferramenta. Este custo tem um impacto maior mediante o número de execuções da ferramenta aumente. Tal como se apresenta na tabela 4.16, executar a ferramenta frase a frase para um conjunto de 3000 frases demora cerca de 13 vezes mais do que executar a ferramenta com 100 frases, fazendo *buffering* das mesmas. Se ainda assim for aumentado o tamanho do buffer para 500 frases, o desempenho passa para cerca de 42,13%, quando comparado com 100 frases.

Parâmetro		1 Frase cada vez	100 Frases de cada vez	500 Frases de cada vez
Tempo	Real	3m57s	18.535s	7.810s
	User	26.762s	3.192s	3.074s
	SYS	22.180s	0.404s	0.217s

Table 4.16: Execução da ferramenta Palavroso, utilizando 1, 100 e 500 frases de cada vez que é feita a chamada à ferramenta.

5 Alterações ao Sistema ShRep

5.1 Introdução

Ao longo do desenvolvimento da presente tese verificou-se a necessidade de efectuar uma validação ao Sistema ShRep. Durante a integração da Ferramenta MARv verificou-se que o tempo de processamento utilizado por esta ferramenta, usando como fonte de dados o sistema ShRep, era extremamente elevado, quando comparado com a execução da ferramenta utilizando como fonte de dados ficheiros em formato XML, tal como se pode constatar na tabela 5.1.

Parâmetro		Ficheiro XML	ShRep
Tempo	Real	8.005s	22m43s
	User	7.688s	5m46s
	SYS	0.314s	0m45s

Table 5.1: Comparação da execução da ferramenta MARv, usando o Sistema ShRep ou ficheiros XML como entrada e saída de dados.

Foi efectuado um conjunto de experiências no sistema ShRep com o objectivo de identificar os problemas existentes no sistema, sendo também apresentadas soluções que resolvem esses mesmos problemas.

A medição do tempo para o Sistema ShRep foi efectuada tendo por base a mesma informação que é usada na ferramenta MARv executada através de ficheiros XML, sendo esta informação colocada no sistema ShRep no momento em que o Servidor se inicia. As experiências foram efectuadas numa máquina com 2 processadores 2.74 GHz, e com 3 GB de memória.

5.2 Análise ao desempenho do sistema ShRep

Analisando o Sistema ShRep verifica-se que os principais motivos que levam a uma grande utilização do tempo de processamento se prendem com:

- Duplicação da representação dos objectos provenientes do Servidor na Biblioteca Cliente, não existindo o cuidado de verificar se determinado objecto já se encontra no lado da Biblioteca Cliente e se encontra actualizado, antes de ser efectuado o pedido ao Servidor.

- Elevado número de pedidos individuais, efectuando para cada pedido de informação e de actualização uma chamada remota a partir da Biblioteca Cliente ao Servidor;
- Elevada quantidade de informação trocada entre a Biblioteca Cliente e o Servidor, e vice-versa.

Para dar solução a estas problemáticas foram desenvolvidas alterações ao Sistema ShRep que se prendem com:

- Representação única de cada objecto na Biblioteca Cliente;
- Diminuição do número de pedidos de informação e actualização de dados da Biblioteca Cliente ao Servidor;
- Diminuição da informação trocada entre a Biblioteca Cliente e o Servidor, e vice-versa.

5.2.1 Eliminação da duplicação de informação

De acordo com o funcionamento inicial do Sistema ShRep a responsabilidade de manutenção da coerência dos dados era do Servidor, ou seja, a Biblioteca Cliente tinha apenas um papel de intermediário entre uma aplicação que utilize o Sistema ShRep e o Servidor do mesmo, com o objectivo de tratar as questões relacionadas com o protocolo de comunicação. Este facto implica, por exemplo, que caso uma ferramenta pretenda aceder ao mesmo segmento por mais que uma vez, fazendo o pedido à Biblioteca Cliente, esta terá sempre que aceder ao Servidor para dar resposta a cada pedido, efectuando deste modo comunicações desnecessária entre o servidor e a aplicação, dado que o segmento pode ser guardado no biblioteca cliente para posteriormente voltar a ser fornecido à ferramenta.

Ao efectuar um pedido de um determinado elemento do domínio, definido no modelo conceptual, a informação contida na resposta a esse pedido é mais complexa do que o próprio pedido, ou seja, contém mais informação do que a que era pedida. Por exemplo, quando se pede informação sobre um segmento, é incluída a informação da segmentação a que este pertence, bem como as relações e as classificações que estão associadas ao segmento. Simultaneamente, verifica-se que a Biblioteca Cliente replica os objectos existentes no Servidor, através da informação XML que é fornecida por este. Contudo, verifica-se que esta replicação conduz a uma duplicação desnecessária da informação. Por exemplo, se a análise a que um dado segmento pertence for a mesma que criou as classificações existentes no segmento irá duplicar-se a representação dessa mesma análise.

Para solucionar os problemas anteriormente identificados procedeu-se à introdução de *caches* na Biblioteca Cliente que permitem guardar os elementos do domínio que já foram recebidos do Servidor. As caches foram concretizadas usando tabelas de dispersão.

Esta solução apresenta a vantagem de se poder aceder ao mesmo elemento do domínio tantas vezes quantas forem necessário, efectuando o pedido ao Servidor apenas uma vez, já que é mantida na *cache* da Biblioteca Cliente uma réplica desse objecto de domínio, sendo verificada a existência do elemento pretendido sempre que é pedido um objecto de domínio à Biblioteca Cliente. Da mesma forma, a introdução de caches também elimina a duplicação desnecessária de informação verificada na Biblioteca Cliente através da criação dos objectos provenientes na informação XML fornecida pelo Servidor, já que, antes da criação de qualquer objecto de domínio é verificada a sua existência na respectiva *cache*. Por exemplo, se a resposta dada pelo Servidor a um pedido de um dado elemento traz juntamente a informação relativa a uma dada análise, esta análise só é criada caso não exista na respectiva *cache*.

A próxima alteração efectuada foi com o objectivo de tentar reduzir a informação desnecessária que é passada da Biblioteca Cliente para o Servidor, substituindo a passagem de informação completa por informação de identificação. Por exemplo, se se pretender adicionar uma classificação a um segmento, não é necessário passar toda a informação que o segmento contém (identificação das relações, identificação das classificação que já contém, entre outros), bastando apenas enviar a identificação do segmento, que se pode resumir, por exemplo, a três números inteiros, correspondendo à identificação do segmento, da segmentação a que este pertence, e da análise da qual a segmentação faz parte.

Com as alterações descritas anteriormente já foram focados os três objectivos de melhoria traçados inicialmente, tendo-se verificado uma redução de 21.20% do tempo de processamento que se encontra representado na tabela 5.2.

Parâmetro		ShRep
Tempo	Real	17m541s (-21.20%)
	User	4m49s (-16.39%)
	SYS	0m39s (-15.00%)

Table 5.2: Execução da ferramenta MARv, usando o Sistema ShRep com *caches* na Biblioteca Cliente, e comparação com estado inicial.

5.2.2 Redução de pedidos individuais

Apesar de se ter verificado, com a alteração anterior, uma redução no número de pedidos por parte da aplicação ao Servidor, ainda se observa um elevado número de chamadas por parte da Biblioteca Cliente ao Servidor. Desta forma, a próxima alteração teve por objectivo reduzir o número de chamadas efectuadas ao Servidor, aquando do pedido dos dados necessários para a execução de uma ferramenta, mais concretamente, quando a aplicação está a iterar sobre os dados do domínio. A navegação sobre os elementos do repositório é feita normalmente utilizando iteradores. Na concretização inicial, cada iteração implica um pedido ao servidor. Por forma a reduzir o número de pedidos, foi efectuada a

alteração dos iteradores na Biblioteca Cliente, através da criação de estruturas de dados que permitem trazer do Servidor um conjunto de elementos do mesmo tipo, numa única chamada. Se essa estrutura já tem o elemento pedido pela ferramenta, esse elemento é fornecido à mesma. Caso contrário é pedido um conjunto de elementos ao servidor. Por exemplo, é possível invocar o método *getNSegments* que permite trazer um conjunto de segmentos da mesma segmentação do lado do Servidor para a Biblioteca Cliente, num único pedido.

Esta alteração torna a utilização dos iteradores existentes mais eficiente. No entanto, é necessário definir no lado do cliente estruturas de dados capazes de suportar os vários segmentos e fornecê-los correctamente quando solicitados. Para este efeito foi criado nos elementos de dominio da Biblioteca Cliente estruturas de dados que guardam os vários objectos de domínio que foram pedidos ao Servidor, mesmo sem terem ainda sido requisitados pelo utilizador da Biblioteca Cliente. Por exemplo, no caso dos segmentos, foi criada um lista que se encontra na classe *Segmentation* da Biblioteca Cliente, e sempre que são pedidos segmentos através do iterador da segmentação é verificado se o elemento pedido já se encontra nessa lista. Caso uma ferramenta solicite, através de um iterador, um elemento que ainda não se encontra na Bilbioteca Cliente, é efectuado um pedido por esta ao Servidor. A deslocação de vários elementos de domínio do Servidor para a Biblioteca Cliente é transparente para o utilizador da mesma, uma vez que ele continua a pedir um elemento de cada vez.

O facto de trazer vários elementos do domínio do mesmo tipo não implica uma redução muito significativa no número de chamadas, se apenas for enviada, por parte do Servidor, a informação respeitante ao elemento de domínio pedido. Por exemplo, trazer só os segmentos e não trazer juntamente as classificações, pode implicar novamente um conjunto elevado de chamadas para trazer as respectivas classificações.

Para suprir este problema é possível associar aos segmentos as classificações dos mesmos. No entanto, esta abordagem tem alguns problemas, nomeadamente, é necessário decidir que informação trazer em cada momento. Ao analisarem-se algumas ferramentas que podem trabalhar com o sistema ShRep, verifica-se que a informação que elas precisam é variável, sendo difícil encontrar algum padrão. Desta forma, a melhor solução que minimiza o número de chamadas efectuadas para obter a informação desejada, e por outro lado, não corre o risco de trazer todo o repositório existente no Servidor para o lado da Biblioteca Cliente, é dar a possibilidade ao utilizador da biblioteca de indicar que informação quer que venha associada a cada um dos seus pedidos. Por exemplo, na integração da ferramenta MARv é possível indicar à Biblioteca Cliente, que juntamente com os segmentos se pretende também obter as classificações, e dar indicação de que não se precisa das relações associadas aos segmentos.

Para evitar problemas a nível de protocolo na chamada remota de procedimentos (RPC), relativos à quantidade de informação passada de uma só vez, foi limitado o número de elementos a passar num único pedido para 20 000.

Efectuando uma medição do tempo de processamento com esta nova abordagem, isto é, trazendo de uma só vez 20 000 segmentos e trazendo também as classificações associadas aos mesmos, verifica-se uma redução bastante significativa (59.12%), quando comparado com a última medição efectuada. Os novos tempos de processamento podem ser observados na tabela 5.3.

Parâmetro		ShRep
Tempo	Real	7m19s (-59.12%)
	User	1m31s (-68.34%)
	SYS	0m14s (-62.79%)

Table 5.3: Execução da ferramenta MARv, usando o Sistema ShRep efectuando um o pedido de um conjunto de elementos de domínio, e comparação com última alteração.

5.2.3 Concentração de pedidos no lado da Biblioteca Cliente

Comparando-se o tempo de processamento verificado na alteração anterior com a execução da ferramenta MARv tendo como dados de entrada e de saída ficheiros XML, ainda se verifica uma diferença de 7 minutos e 11.142 segundos. Desta forma, continuando a análise ao sistema ShRep, e tendo em conta as alterações já efectuadas, verifica-se que uma grande parte do tempo de processamento agora utilizado é gasto na comunicação entre a Biblioteca Cliente e o Servidor, para efectuar as actualizações de informação.

A estratégia usada para resolver este problema foi concentrar um conjunto de pedidos, no lado da Biblioteca Cliente, num *Buffer*, e só depois enviá-los, todos juntos, para o Servidor. Esta solução diminui o número de chamadas efectuadas ao Servidor, por parte da Biblioteca Cliente, no entanto implica algumas mudanças filosóficas no funcionamento do sistema ShRep. Esta alteração impõe que a Biblioteca Cliente tenha responsabilidades que até agora pertenciam exclusivamente ao Servidor, das quais se destacam:

- Actualização dos objectos que foram alvo de modificação. Até aqui, como as actualizações eram efectuadas uma a uma no Servidor, este continha sempre a informação mais actual, o que fazia com que, sempre que uma ferramenta que utilizasse a Biblioteca Cliente pedisse um elemento pela primeira vez este era pedido ao Servidor e era devolvida a informação mais actual; nesta nova abordagem, como os pedidos de actualização não são logo efectuados no Servidor é necessário fazer incidir essas actualizações nos objectos de domínio existentes do lado da Biblioteca Cliente de modo a assegurar que o utilizador da Biblioteca Cliente tem sempre a informação mais actualizada, e que, simultaneamente, consegue ver de imediato o resultado das suas actualizações. Por exemplo, se é criada uma classificação, esta deve ser logo associada ao respectivo segmento.

- Verificação de operações. A Biblioteca Cliente tem que garantir que as operações que estão a ser pedidas são possíveis de ser efectuadas. Antes estas alterações eram efectuadas no Servidor, contudo, nesta nova abordagem é necessário que sejam feitas na Biblioteca Cliente, como por exemplo, verificar que a análise onde se está a proceder às alterações se encontra no estado *Aberto*.

- Atribuição de identificadores aos elementos criados. Dado os objectos serem agora criados na Biblioteca Cliente e haver necessidade de se efectuar referenciação aos mesmos para comunicar com o Servidor, é mais fácil criar a identificação dos objectos de domínio no lado da Biblioteca Cliente. Passar esta responsabilidade do lado do Servidor para a Biblioteca Cliente não acarreta problemas para a maioria dos objectos de domínio, uma vez que uma análise apenas pode ser alterada por uma ferramenta de cada vez, o que impossibilita, por exemplo, que duas ferramentas criem duas segmentações para a mesma análise, atribuindo-lhe identificadores iguais. No entanto, para a criação de objectos mais gerais, como é o caso dos sinais de dados e das análises, é necessário ser o Servidor a indicar o identificador, uma vez que a criação simultânea destes objectos de domínio não é restrita a uma única ferramenta, ao contrário do que acontece com os restantes.

Com esta nova abordagem o Servidor passa principalmente a ter a função de manter dados que podem ser acedidos por várias aplicações.

Para encapsular os pedidos de criação de objectos foi utilizado o padrão *Command* (Gamma et al., 1994) que se encontra representado na figura 5.1.

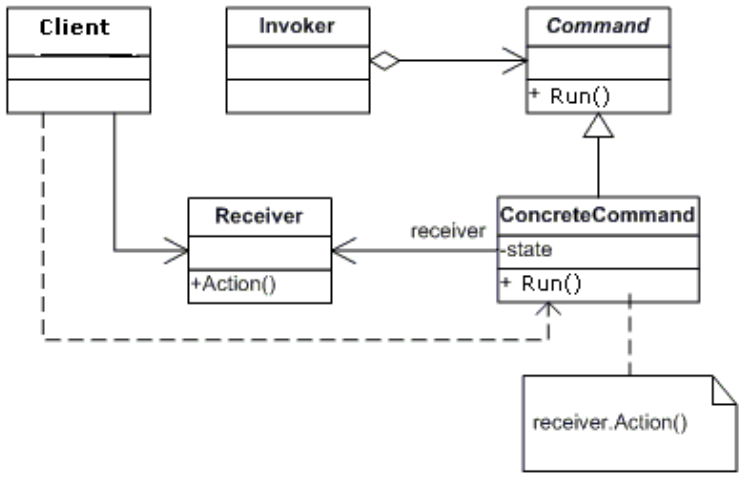

Figure 5.1: Diagrama de classes do padrão de desenho *Command*.

Para cada tipo de elemento do domínio existe uma classe *ConcreteCommand*, na qual são criados os vários pedidos por parte da Biblioteca Cliente, sendo depois descodificados no lado do Servidor

onde os dados ficam actualizados. Se os pedidos de actualização forem efectuados de 20 000 em 20 000 verifica-se uma redução de 32.57%, tal como se apresenta na tabela 5.4.

Parâmetro		ShRep
Tempo	Real	4m56s (-32.57%)
	User	0m22s (-75.39%)
	SYS	0m0s (-93.30%)

Table 5.4: Execução da ferramenta MARv, usando o Sistema ShRep com *buffers* para armazenar pedidos de actualização, e comparação com última alteração.

5.2.4 Compactação da informação usada na comunicação Biblioteca Cliente - Servidor

Analisando-se os dados transmitidos entre a Biblioteca Cliente e o Servidor, verifica-se que o formato utilizado para representar os dados transmitidos é muito descritivo. Isto tem um custo elevado na quantidade de informação adicional a transmitir, relativamente aos elemento do domínio que se pretendem transmitir. A quantidade de informação a transmitir tem influência no tempo de geração dessa informação, no tempo de transmissão e no tempo de processamento para interpretar o que essa informação representa. A utilização de uma linuagem mais descritiva tem apenas a vantagem de facilitar a depuração das mensagens trocadas entre a Biblioteca Cliente e o Servidor.

Assim, para resolver este problema foi efectuada uma compactação da informação trocada entre a Biblioteca Cliente e o Servidor nos pedidos de actualização, tornando as mensagens trocadas num formato menos legível ao ser humano. Um exemplo da versão anterior e da nova versão é apresentado na imagem 5.2.

```
A) VERSÃO ANTIGA
<create-segment-command id="1" type="" description="">
  <segmentation-identification id="1" analysis-id="1" />
  <region signal-data-id="1">
     <start-index><index type="0" value="1"></start-index>
     <end-index><index type="0" value="7"></end-index>
  </region>
</create-segment-command>

B) VERSÂO NOVA
<create-segment-command s="1 0 0 1 1 1 1 0 1 0 7 0 0">
```

Figure 5.2: Exemplo de compactação da informação transmitida entre a Biblioteca Cliente e o Servidor.

Com esta abordagem verifica-se uma diminuição de 44.09% do tempo de processamento, que se encontra representado na tabela 5.5.

Parâmetro		ShRep
Tempo	Real	2m45s (-44.09%)
	User	0m17s (-20.56%)
	SYS	0m1s (+7.00%)

Table 5.5: Execução da ferramenta MARv, usando o Sistema ShRep com simplificação da informação trocada da Biblioteca Cliente para o Servidor, e comparação com última alteração.

5.2.5 Optimização da procura de elementos no Servidor

Efectuando uma análise apenas ao Servidor, com o objectivo de verificar onde este utiliza o tempo de processamento, verifica-se que grande parte deste tempo gasto é usado na procura de elementos de domínio, tempo este que aumenta quando o número de elementos também aumenta. Tal facto resulta do armazenamento dos elementos de domínio ser efectuado apenas em estruturas de dados do tipo lista, o que, por um lado, simplifica a inserção de novos elementos, mas por outro, dificulta a pesquisa dos mesmos. Este tipo de estruturas facilita a procura no caso desta ser efectuada por índice numérico, como por exemplo nos iteradores, mas dificulta a pesquisa quando se pretende procurar um elemento da lista por um determinada campo.

Verificando-se a existência de um grande número de pesquisas de elementos pela sua identificação, surge a necessidade de alterar ou introduzir estruturas de dados que melhorem o processo de pesquisa dos elementos de domínio, pelo que foram introduzidas tabelas de dispersão no lado do Servidor. Dado o número de pesquisas efectuadas pela identificação dos elementos ser elevada, verifica-se, tal como era esperado, uma redução do tempo de processamento de 67.46%, como se apresenta na tabela 5.6.

Parâmetro		ShRep
Tempo	Real	0m53s (-67.46%)
	User	0m18s (+0.01%)
	SYS	0m1s (-2.69%)

Table 5.6: Execução da ferramenta MARv, usando o Sistema ShRep usando tabelas de dispersão no Servidor, e comparação com última alteração.

5.2.6 Compactação da informação usada na comunicação Servidor - Biblioteca Cliente

Da mesma forma que se reduziu a informação que é transmitida nos comandos da Biblioteca Cliente para o Servidor, é também possível efectuar o mesmo processo na informação que é transmitida do Servidor para a Biblioteca Cliente. Nas figuras B.1 e B.2 encontra-se representado um exemplo de

um segmento em formato antigo e um segmento em versão simplificada, de onde se constata uma significativa redução.

Analisando o desempenho da ferramenta, verifica-se uma redução de cerca de 50.13%, tal como se apresenta na tabela 5.7.

Parâmetro		ShRep
Tempo	Real	0m26s (-50.13%)
	User	0m9s (-48.84%)
	SYS	0m0s (-56.52%)

Table 5.7: Execução da ferramenta MARv, usando o Sistema ShRep com simplificação da informação trocada da Biblioteca Cliente para o Servidor, e comparação com última alteração.

Na tabela 5.8 é apresentada uma comparação dos tempos de processamento. Em percentagem encontra-se a diferença entre o sistema ShRep na versão inicial e na versão final. O novo tempo verificado já pode ser considerado um tempo aceitável o que possibilita a utilização do Sistema ShRep. Efectuando uma comparação com o tempo de processamento da ferramenta MARv tendo como entrada e saída de dados ficheiros XML, verifica-se que o tempo ainda não é próximo, no entanto, as vantagens que o sistema apresenta minimiza a desvantagem proporcionada pela diferença temporal que ainda se verifica.

Parâmetro		Ficheiro XML	ShRep Inicial	ShRep Final
Tempo	Real	8.0s	22m43s	26.862s (-98.03%)
	User	7.7s	5m46s	9.265s (-97.33%)
	SYS	0.314s	0m45s	0.440s (-99.04%)

Table 5.8: Comparação da execução da ferramenta MARv, usando o Sistema ShRep antes e após as alterações.

5.3 Execução de ferramentas em paralelo

O tratamento de grandes quantidades de informação em cadeias de processamento torna-se lento, se a execução de cada ferramenta necessitar de aguardar pelo fim de execução da ferramenta precedente na referida cadeia. Não obstante, verifica-se a existência de ferramentas cuja produção de resultados de determinadas partes se considera independente das restantes. Desta forma, existem situações em que uma ferramenta não precisa de esperar que a ferramenta anterior termine a sua análise, podendo ser processada a informação à medida que a ferramenta anterior vá produzindo resultados. Por exemplo, os resultados produzidos pela ferramenta Palavroso são independentes de palavra para palavra, o que significa que quando a ferramenta está a atribuir as categorias gramaticais a uma palavra, todas as

palavras antecedentes já se encontram com as categorias atribuídas, estando disponíveis para a execução de outra ferramenta.

Desta forma, é possível eliminar algum tempo de espera através da criação de funcionalidades no sistema ShRep, que permitam a execução em paralelo de várias ferramentas. A criação deste paralelismo de execução não causa problemas, dado que isto já estava previsto no sistema ShRep. Em cada momento apenas uma ferramenta pode alterar uma análise, mas o sistema ShRep possibilita o acesso aos dados já produzidos por uma ferramenta a outras ferramentas.

Devido ao facto do tempo de execução das várias ferramentas ser, à partida, distinto, faz com que exista a possibilidade de uma ferramenta necessitar de consumir dados que ainda não foram produzidos por determinada ferramenta. Isto obriga a que a ferramenta consumidora fique à espera desses dados. Aquando da execução do pedido, existe a possibilidade da ferramenta ficar bloqueada na Biblioteca Cliente ou no Servidor. Ficar à espera no lado da Biblioteca Cliente leva a que o pedido seja efectuado mais que uma vez, ou a que o Servidor tenha de indicar à Biblioteca Cliente que os dados que esta precisa já se encontram disponíveis. Isto implicaria um acréscimo do número de pedidos realizados por parte da Biblioteca Cliente ao Servidor, ou uma mudança de arquitectura no funcionamento do sistema ShRep, que possibilitasse o Servidor de comunicar com a Biblioteca Cliente sem ser como resposta a um pedido desta. Assim, a implementação dos mecanismos de bloqueio no lado do Servidor não gera estes dois problemas descritos, uma vez que o pedido continua a ser efectuado apenas uma vez, e que o Servidor tem conhecimento do momento em que os dados estão disponíveis para dar resposta a esse mesmo pedido da Biblioteca Cliente.

Este algoritmo que permite a partilha de dados enquanto a ferramenta ainda está a processar outras partes de dados já existia no sistema ShRep, mas apenas para o elemento do domínio StringData. Desta forma foi alargado esse algoritmo para outros elementos do domínio, nomeadamente os segmentos e as classificações.

O algoritmo tem em consideração se a análise se encontra fechada. Em caso afirmativo devolve o máximo de informação existente no repositório que satisfaça o pedido efectuado. Caso a análise se encontre em aberto, significa que existe uma ferramenta que está a processar dados, de onde pode resultar informação para ser introduzida nessa análise, sendo no entanto possível fornecer a informação que já se encontra no Servidor. Desta forma, caso já exista informação no Servidor que satisfaça o pedido, esta é de imediato devolvida. Em caso contrário, é esperado um período de tempo de 3 segundos, sendo posteriormente verificado se já há informação disponível.

A implementação desta funcionalidade põe em causa algumas decisões tomadas aquando da melhoria do desempenho do sistema ShRep, nomeadamente, a retenção de informação de actualização dos dados por parte da Biblioteca Cliente, com o objectivo de aglomerar um número significativo de pedidos, reduzindo assim o número de pedidos efectuados ao Servidor. Com a utilização de paralelismo,

poderá ser mais vantajoso efectuar mais pedidos e ter a informação mais actualizada no Servidor, disponível para ser consumida por outra ferramenta. Deste modo, foi estendida a interface disponibilizada pela Biblioteca Cliente às ferramentas, passando estas a poder indicar o momento em que pretendem que a informação seja actualizada no Servidor, podendo fazê-lo sempre que a ferramenta considere que a informação produzida até um dado momento já está disponível para outra ferramenta. A funcionalidade de indicar o momento em que é efectuada a actualização dos dados por parte da Biblioteca Cliente no Servidor é assim dividida entre a ferramenta e a Biblioteca Cliente, não existindo uma obrigatoriedade por parte da ferramenta em indicar que pretende efectuar a actualização de dados no Servidor. A ferramenta tem disponível um comando explícito, enquanto a Biblioteca Cliente efectua essa actualização sempre que considere que tem um número suficiente de pedidos, estando actualmente definido o limite de 20 000 pedidos.

Apesar da solução apresentada pode ocorrer que uma dada ferramenta esteja à espera de dados que já foram produzidos por outra ferramenta, mas que ainda não foram enviados por parte da Biblioteca Cliente dessa ferramenta ao Servidor. Neste caso, ocorre um período de espera por parte da ferramenta que aguarda por dados para processar que pode ser minimizado. Uma solução possível seria a criação de mecanismos que permitam ao servidor indicar às Bibliotecas Clientes que determinada ferramenta está à espera de dados para processar, sendo nesse momento devolvidos os elementos que já se encontrem processados.

Um dos problemas encontrados foi a possibilidade de uma ferramenta consumidora trazer informação do Servidor que ainda não se encontra completa. Por exemplo, caso uma ferramenta solicite ao Servidor os segmentos de uma segmentação e pedir juntamente as classificações, pode ocorrer que no momento do pedido existam segmentos, no lado do Servidor, que ainda não têm as classificações associadas. Para evitar esta situação foi considerado que caso uma análise se encontre no estado aberto, não é possível fornecer à Biblioteca Cliente o último elemento de cada tipo de objecto de domínio. Esta solução pressupõe que os elementos de domínio são inseridos no Servidor por uma certa ordem. Contudo, pode haver ferramentas em que este princípio não se verifique, sendo assim esta condição uma imposição e, simultaneamente, uma limitação do sistema ShRep. Para o caso prático descrito, é considerado que apenas o último segmento não tem as classificações associadas, sendo possível devolver todos os restantes segmentos.

Uma outra solução que poderia ter sido adoptada era cada ferramenta ter a possibilidade de indicar quais os elementos que ela criou que estão disponíveis para a utilização de outras ferramentas. Esta informação só seria considerada enquanto a análise se encontrasse no estado aberto, assumindo-se que uma análise fechada tem todos os elementos de domínio acessíveis.

Foi efectuada uma análise que permite verificar que impacto a introdução de paralelismo pode causar, em termos de desempenho das ferramentas integradas no sistema ShRep. Na tabela 5.9 encontra-

se o tempo de processamento de uma pequena cadeia de processamento composta pelas ferramentas *InitialTextCreator*, *SentenceSplit*, Palavroso e MARv, onde se apresenta o tempo de execução das ferramentas na situação em que cada uma aguarda a conclusão da anterior, e a situação de executar as mesmas ferramentas em paralelo.

Tamanho *Corpus*	Sem Paralelismo	Com Paralelismo
3 000 Frases	21.5s	16.9s (-21.66%)
12 000 Frases	69.6s	54.8s (-21.25%)

Table 5.9: Execução de uma cadeia de processamento, com e sem a introdução de paralelismo na sua execução.

A medição do tempo foi efectuada no Servidor do sistema ShRep, tendo início no momento em que este começa a sua execução, e termina no momento em que a ferramenta MARv indica o fim do seu processamento, através do fecho da análise que esta mesma ferramenta produz.

Verifica-se um ganho de cerca de 21% com a introdução de paralelismo, num corpus composto por 3000 Frases. Verifica-se também que com o aumento do corpus em análise o ganho em usar paralelismo mantêm-se quase constante, em percentagem. No entanto, apenas com estas ferramentas integradas torna-se difícil encontrar uma boa configuração para o ambiente de paralelismo, nomeadamente, determinar o tempo que uma ferramenta deve esperar por resultados de outra ferramenta, determinar se é útil definir um conjunto mínimo de elementos a devolver e determinar com que frequência deve uma dada ferramenta pedir que os seus dados sejam actualizados no Servidor.

5.4 Sistema ShRep na versão StandAlone

Após as várias optimizações implementadas no sistema ShRep tem interesse verificar qual o comportamento do sistema na versão StandAlone, em que a ferramenta e o sistema ShRep correm no contexto do mesmo processo. O interesse surge pelo facto de se ter detectado que a principal razão pelo elevado tempo de processamento está relacionado com o protocolo de comunicação existente entre as Bibliotecas Cliente e o Servidor.

A principal diferença desta versão é o facto de não haver Servidor, o que obriga a que o sistema ShRep seja capaz de ler os dados que necessita bem como guardar os resultados produzidos pelas ferramentas através de ficheiros. A não existência de um Servidor impossibilita a execução de ferramentas em paralelo. No entanto, a utilização na versão StandAlone não deixa de aproveitar outras vantagens existentes no sistema ShRep, nomeadamente, a facilidade de criar ou integrar ferramentas tendo como base o modelo que o sistema ShRep oferece.

Esta nova funcionalidade foi implementada na Biblioteca Cliente do sistema, tendo sido

aproveitado o modelo de domínio que possui e que, derivado das alterações efectuadas e já descritas, tem as funcionalidades necessárias para a gestão dos objectos de domínio.

Desta forma foi criada uma classe que abstrai os objectos de domínio da versão que estão a usar. A nova classe que gere a fonte de dados de ficheiros tem a responsabilidade de saber interpretar os dados existentes nos ficheiros e converter essa informação em objectos de domínio do sistema ShRep, bem como fazer a operação inversa aquando do término da execução da ferramenta. Os dados são todos carregados inicialmente, sendo formado todo o modelo de domínio existente. Uma ferramenta que utilize o sistema ShRep acede a este através dos objectos de domínio disponibilizados pela Biblioteca Cliente. Desta forma, como os dados são lidos inicialmente e transformados em objectos do domínio, a ferramenta tem sempre disponível os dados existentes. Na versão Cliente Servidor isto já não se aplica, dado que a Biblioteca Cliente necessita de ir pedir dados ao servidor várias vezes durante a execução de uma ferramenta.

Para representar os dados em ficheiros foi usado o mesmo tipo de estruturas que foi usado para enviar a informação entre a Biblioteca Cliente e o Servidor, isto é, informação maioritariamente em formato numérico e codificado, com o objectivo de não perder demasiado tempo a efectuar a leitura dos dados.

Para as ferramentas executarem com esta nova versão apenas têm que indicar a localização do ficheiro de entrada e a localização do ficheiro de saída, em vez de indicar a localização do Servidor.

Efectuando-se uma análise ao desempenho desta versão, através da execução da ferramenta MARv com o *corpus* que tem sido utilizado, verificam-se os resultados apresentados na tabela 5.10, dos quais é possível verificar que a execução da ferramenta MARv utilizando o sistema ShRep na versão *StandAlone* tem um ganho de 66,61% quando comparado com a utilização do sistema ShRep utilizando o repositório como contentor de dados. Esta diferença permite constatar que o uso de tecnologia RPC para comunicação entre o cliente e o Servidor tem um custo significativo em todo o sistema ShRep usado com repositório.

Parâmetro		XML	ShRep c/Repositório	ShRep *StandAlone*
Tempo	Real	7.5s	26.9s	9.0s
	User	7.3s	9.3s	8.8s
	SYS	0.3s	0.4s	0.1s

Table 5.10: Execução da ferramenta MARv com o sistema ShRep em verão *StandAlone*

.

Verifica-se ainda, que a diferença existente entre a utilização do sistema ShRep nesta nova versão e a utilização de ficheiros XML é de apenas 16,27%, o que corresponde a 1.459 segundos para o corpus analisado.

Um dos problemas encontrados, com influência no tempo de processamento, está relacionado com a necessidade do sistema ShRep na versão *StandAlone* necessitar de carregar todo o conteúdo do ficheiro no início da sua execução, e efectuar o descarregamento de toda a informação para um outro ficheiro no fim do processamento ter sido efectuado. Isto implica que em cada execução de uma ferramenta tenha que ser guardada não só a informação criada por essa ferramenta, mas toda a informação que já existia anteriormente.

É desta forma possível criar um modelo, que permita ao sistema carregar apenas a informação que a ferramenta precisa e guardar apenas a informação que foi criada por essa ferramenta. Esta nova abordagem requer um gestor de fontes de dados com capacidade para identificar em que ficheiros se encontra a informação disponível, sendo eventualmente necessária informação de controlo de informação. Por exemplo, se se optasse por uma solução em que cada análise fosse guardada num ficheiro distinto, seria necessário ter informação que permitisse identificar todas as análises que tivessem informação acerca de um segmento, ou então cada ferramenta saber as análises de que necessita. Caso houvesse um pedido de todas as classificações de um dado segmento, teria que ser possível identificar todas as análises que criaram classificações para esse segmento.

6 Conclusão e Trabalho Futuro

6.1 Sumário

Este trabalho teve como ponto de partida a tese de mestrado de João Graça (Almeida Varelas Graça, 2006), que desenvolveu o sistema ShRep que pretende servir de base a ferramentas de processamento de língua natural. O objectivo principal deste trabalho foi a validação deste sistema na qual são identificadas as principais razões que levam o sistema a apresentar um mau desempenho, sendo também feita a exposição de soluções que visem a resolução dessas problemáticas. Houve ainda dois resultados importantes. O primeiro foi a criação de uma interface gráfica que permita uma visualização da informação existente no repositório, e o segundo foi a integração de duas ferramentas no sistema ShRep, tendo sido, previamente, optimizado o funcionamento de uma delas.

No capítulo 3 é descrita a interface gráfica implementada, servindo apenas para depuração do resultado produzido pelas diversas ferramentas de anotação. O desenvolvimento de uma interface gráfica para o sistema ShRep surge da necessidade de ser encontrado um modo que possibilite uma boa visualização dos dados existentes no repositório. A interface desenvolvida necessita de alterações, nomeadamente, a passagem para uma interface gráfica que permita não só visualizar a informação existente no repositório, mas também criar novas anotações.

No capítulo 4 é descrita a integração da ferramenta MARv e da ferramenta Palavroso no sistema ShRep. Na primeira ferramenta foram efectuadas alterações que permitem melhorar o desempenho, quer em termos de tempo de processamento quer em termos de taxa de erro. O facto da ferramenta implementar o Algoritmo de Viterbi, que se baseia em modelos probabilísticos, resulta na existência de algumas dificuldades em produzir resultados 100% correctos. Desta forma, é possível equacionar soluções que tentem reduzir a taxa de erro. Nas alterações efectuadas destacam-se a eliminação da utilização das probabilidades lexicais, a alteração dos dicionários e a passagem do uso de probabilidades para o uso de logaritmos, possibilitando uma análise frase a frase, o que permitiu diminuir de 6.83% para 5.21% a taxa de erro no corpus analisado.

Relativamente ao tempo de processamento da ferramenta destaca-se a passagem da representação dos dicionários do formato textual para o formato numérico como a grande causa para a redução do tempo verificado, sendo que a passagem de estruturas do tipo *Map* para estruturas do tipo Vector Multidimensional também tenha dado um pequeno contributo. Verifica-se também que o tempo de proces-

samento da ferramenta não tem grande variação quando usada na plataforma Cliente Servidor ou na plataforma *StandAlone*, resultante do facto da transmissão de dados efectuada entre o cliente e o servidor não acrescentar grande quantidade de processamento, uma vez que os dados são passados numa única chamada.

O processo de integração da ferramenta MARv no sistema ShRep foi efectuado de forma semelhante às ferramentas que já tinham sido integradas, isto é, através da alteração do código da ferramenta. No entanto, para a ferramenta Palavroso foi apresentado um processo diferente de integração, baseado numa chamada ao sistema para executar a ferramenta e posteriormente, processar os resultados produzidos por esta. Desta forma, não é necessário efectuar qualquer alteração à ferramenta, ou mesmo, ter conhecimento do código da ferramenta.

No capítulo 5 são descritos os problemas encontrados no sistema ShRep e as soluções apresentadas para a sua resolução. Verifica-se que quase todos os problemas encontrados estão relacionados com situações que envolvem o protocolo de comunicação utilizado entre o Servidor e a Biblioteca Cliente. Desta forma, as alterações implementadas tiveram por finalidade a redução de interacção do Servidor com os clientes. A introdução de *caches* na Biblioteca Cliente, o pedido de um conjunto de elementos e a concentração de pedidos de actualização permitiu reduzir o número de chamadas efectuadas entre a Biblioteca Cliente e o Servidor, enquanto que a alteração do detalhe da informação que é transmitida entre a Biblioteca Cliente e o Servidor permitiu reduzir a quantidade de informação transmitida. Com estas alterações passou-se de uma situação em que o sistema demorava a processar o texto de teste de 22 minutos, para uma situação em que demora apenas 26,862 segundos.

Dado os problemas de desempenho estarem relacionados com o protocolo de comunicação usado entre a Biblioteca Cliente e o Servidor, foi desenvolvido um módulo que permite executar as ferramentas de processamento de língua natural usando o sistema ShRep numa plataforma *StandAlone*. Nesta plataforma, deixa de haver necessidade da existência de Servidor passando os dados a ser fornecidos e guardados através de ficheiros. Esta utilização dificulta a usabilidade do sistema enquanto repositório de dados, com o propósito de ter os dados partilhados a várias ferramentas, mas por outro lado melhora o seu tempo de processamento para valores próximos daqueles que algumas ferramentas apresentam, quando usadas sem o sistema ShRep. De constar ainda, que é possível efectuar melhorias nesta versão *StandAlone* que permita ao sistema ShRep produzir resultados em termos de tempo de processamento muito próximos dos alcançados pelas ferramentas isoladamente, ou mesmo ter melhores resultados.

Foi ainda alargado o processo de execução de ferramentas em paralelo para as ferramentas introduzidas. Apesar de se verificar alguns ganhos, é necessário ter mais ferramentas integradas que permitam ter uma maior visibilidade dos requisitos necessários às ferramentas, com o objectivo de determinar de forma correcta alguns parâmetros utilizados.

6.2 Trabalho Futuro

Nesta secção são apresentados os pontos que não ficaram totalmente implementados e os pontos que merecem melhor análise de vantagens e desvantagens.

6.2.1 Interface Gráfica

Apesar de ter sido desenvolvida uma interface gráfica, ainda é possível melhorá-la, nomeadamente, passar de uma interface gráfica de depuração para uma interface gráfica que permita efectuar alterações quer aos sinais de dados quer às anotações existentes. Para este efeito, era útil desenvolver um conjunto de funcionalidades, das quais se destacam:

- Permitir efectuar alterações ao repositório através da interface gráfica. Esta funcionalidade permite, caso seja necessário, efectuar pequenas alterações às análises existentes, com o objectivo de corrigir pequenos detalhes que não foram bem introduzidos pelas ferramentas de anotação;
- Visualizar em simultâneo mais que uma segmentação, minimizando a necessidade de interacção entre o utilizador e a ferramenta;
- Representar no repositório e na interface gráfica outra fonte de dados para além de texto, como por exemplo, sinais de áudio;
- Representar alternativas aos segmentos, dado ser o único elemento do modelo conceptual que não se encontra representado na interface gráfica;
- Melhorar a visualização das hierarquias de um segmento, através da criação de um esquema em árvore que permita uma navegação ao longo da hierarquia, dado o sistema de visualização actual apenas mostrar um nível ascendente e um nível descendente.

6.2.2 Ferramenta MARv

Apesar das alterações que foram implementadas, seria ainda possível melhorar alguns aspectos, que se apresentam de seguida.

- Separação das classes que gerem as entradas e as saídas de dados: nesta versão é assumido que o formato dos dados de saída é definido com base no formato dos dados de entrada. A única excepção é a possibilidade de definir a codificação usada nos dados de saída de XML. Com a separação de responsabilidades de entrada e saída é possível ter, sem grande custo computacional,

a tradução entre diferentes tipos de dados. Por exemplo, seria possível ter dados de entrada provenientes do sistema ShRep e serem escritos num ficheiro XML, ou poderia ter-se dados de entrada com um determinado formato de DTD e ter dados de saída num formato de DTD diferente. Estas funcionalidades tornam-se mais rápidas se executadas pelo programa, quando comparadas com a execução de pós processamento, através de conversores, dado a ferramenta MARv não ter um grande custo computacional para executar esta tarefa, dado que os dados de entrada são transformados em estruturas da própria ferramenta, bastando depois do processamento indicar qual o novo formato desejado;

- Utilização da mesma árvore XML: quando os dados de entrada e de saída se encontram formatados com o mesmo DTD, poderia-se usar a mesma árvore de XML, criada no início do parser, fazendo nela as alterações de nós necessárias e efectuando no fim a escrita dessa mesma árvore. Esta possibilidade poderia, eventualmente, tornar a aplicação mais rápida, dado que a árvore inicial encontra-se a ser percorrida duas vezes: uma para captar os *Tokens* e o campo *pos* usado na desambiguação, e outra para colocar na árvore de saída os campos que se encontram na árvore de entrada e que não foram utilizados na desambiguação;

- Mudança de arquitectura: é necessário efectuar uma análise mais aprofundada acerca das vantagens e desvantagens que uma mudança de arquitectura poderia trazer. Seria útil efectuar alterações que facilitassem o funcionamento em cliente-servidor com qualquer tipo de dados. De acordo com o que se verifica na figura 6.1 a principal diferença desta arquitectura, quando comparada com a existente, é o facto da biblioteca de desambiguação se encontrar no lado servidor e a informação que navega do cliente para o servidor serem estruturas internas usadas pelo MARv. Recorde-se que actualmente o funcionamento cliente-servidor apenas está habilitado a trabalhar com XML, sendo informação textual que navega entre o cliente e o servidor. Para além de facilitar o funcionamento em modo cliente-servidor para qualquer tipo de fonte de dados, esta arquitectura tem também a vantagem de permitir instanciar de uma forma mais fácil vários servidores contendo bibliotecas de desambiguação, o que possibilitará, no futuro, efectuar a desambiguação de uma forma distribuída. No entanto, é necessário analisar o custo de serializar e desserializar as estruturas. Na arquitectura actual esse custo é baixo, dado a informação que navega entre cliente e servidor ser em formato XML, sendo que o servidor já se encontra desenvolvido para funcionar com este formato de dados.

- Analisar a utilização de palavras compostas nos dicionários: como já foi referido, a introdução dos novos dicionários permitiu a introdução de palavras compostas, isto é, a possibilidade de ter conjuntos de palavras que eventualmente possam ter melhor resultados quando usadas em conjunto, como por exemplo, ”em vão”. No entanto ainda não foi avaliado se a desambiguação tendo em conta a possibilidade da existência destas expressões produz melhores resultados, em

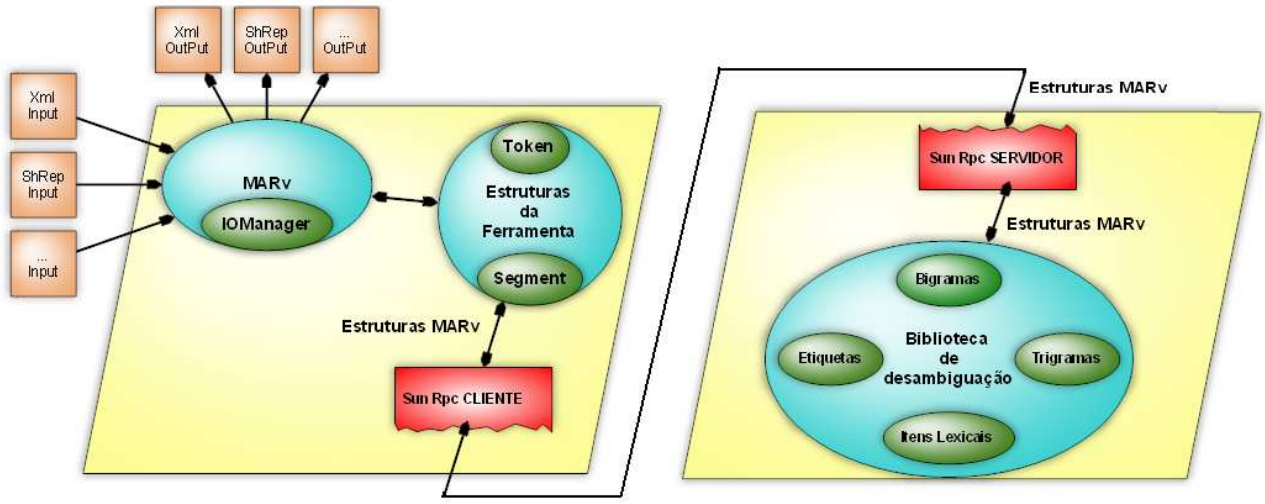

Figure 6.1: Arquitectura do MARv.

termos de taxa de erro, e em termos de tempo de processamento. Não foi possível realizar esta análise dado o *corpus* usado para teste não incluir uma divisão que tenha em conta esta nova característica.

6.2.3 Sistema ShRep

Apesar do trabalho efectuado no sistema ShRep continua a haver necessidade de o fazer evoluir. Para tal, é apresentado de seguida um conjunto de questões que necessitam de ser analisadas:

- Substituir a implementação escolhida para o protocolo de comunicação, com o objectivo de eliminar ineficiências na opção tomada, aquando da escolha de XML-RPC;
- Efectuar uma análise ao processamento em paralelo de modo às ferramentas tirarem mais proveitos em termos de tempo de processamento deste mecanismo;
- Criar um módulo de gestão de ficheiros eficaz para a plataforma *StandAlone*, que evite a leitura e a escrita integral de toda a informação existente.

Ficam ainda por desenvolver soluções para alguns problemas relatados na tese de João Graça, de onde se destaca:

- Integrar mais ferramentas;
- Implementar outros tipos de fontes de dados, como por exemplo, áudio;

- Permitir a remoção de dados, controlando o impacto provocado nos dados que as várias ferramentas possuem em processamento;
- Desenvolver um módulo de persistência no Servidor, evitando que todo o conteúdo do repositório se encontre em memória.

6.3 Contributo

Esta tese tem como contributo as melhorias efectuadas ao sistema ShRep e a criação de um módulo para o funcionamento em *StandAlone*, que tornam o sistema usável, tornando-se assim uma alternativa a alguns sistemas existentes. Com um modelo conceptual bem definido e suficientemente abrangente, e com tempos de processamento aceitáveis, o sistema ShRep apresenta-se como uma boa base de trabalho para o desenvolvimento de ferramentas de processamento de língua natural, evitando preocupações relacionadas com o modelo de dados a utilizar. Foram dados alguns passos para o desenvolvimento de mecanismos que permitam a utilização de ferramentas em paralelo, necessitanto no entanto de uma análise mais aprofundadada.

Esta tese contribuíu também para o desenvolvimento de uma interface gráfica que permita visualizar a informação existente no repositório, tendo sido apresentadas as dificuldades em apresentar grandes quantidades de informação.

Um outro contributo foi a integração de duas ferramentas no sistema ShRep, que aumenta a capacidade de criação de cadeias de processamento mais diversificadas. Foi ainda apresentado de um novo processo de integração de ferramentas, sem necessitar interferir com o código da mesma.

Um último contributo desta tese é a melhoria significativa em termos de tempo de processamento, em termos de memória utilizada e em termos de taxa de erro de uma ferramenta de desambiguação, melhorando assim o desempenho das cadeias de processamento onde essa ferramenta é usada.

Bibliography

Agrawal, R., & Robert J. Bayardo Jr., S. P., Daniel Gruhl. (2001, May). Vinci: A service-oriented architecture for rapid development of web applications. In *Acm 1-58113-348-0/01/0005.* ACM.

A. Lally, D. F. e. (2004). Building an example application with the unstructured information management architecture. *IBM SYSTEMS JOURNAL*, *43*(3).

Almeida Varelas Graça, J. de. (2006). *A framework for integrating natural language tools*. Unpublished master's thesis, Universidade Técnica de Lisboa, Instituto Superior Técnico.

Barras, C., Geoffrois, E., Wu, Z., & Liberman, M. (2001). Transcriber: development and use of a tool for assisting speech corpora production. *Speech Communication*, *33*(1–2), xx–xx. (accepted for publication)

Bird, S., & Liberman, M. (1999). *A formal framework for linguistic annotation* (Tech. Rep. No. MS-CIS-99-01). Philadelphia, Pennsylvania.

Bird, S., & Loper, E. (2002). Nltk: The natural language toolkit. *Corr*, *cs.CL/0205028*.

Bird, S., Ma, X., Lee, H., Maeda, K., Randall, B., Zayat, S., et al. (2004). *Software tabletrans.* (`http://agtk.sourceforge.net/`)

Bird, S., Maeda, K., Ma, X., Lee, H., Randall, B., & Zayat, S. (2002). Tabletrans, multitrans, intertrans and treetrans: Diverse tools built on the annotation graph toolkit. *arXiv:cs.CL/0204006 v1*.

Burge, W. H. (1975). *Recursive programming techniques*. Addison-Wesley.

Dave Crane, D. J., Eric Pascarello. (2005). *Ajax in action* (M. Publications, Ed.). Manning Publications.

Developing language processing components with gate version 4 (a user guide). (2006, October). (`http://gate.ac.uk/sale/tao/index.html`)

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, *13*, 94-102.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software* (1.ed ed.; Addison-Wesley, Ed.). Addison-Wesley.

Garrett, J. J. (2005, Fevereiro 18). *Ajax: A new approach to web applications* (Tech. Rep.).

Geoffrois, E., Barras, C., Bird, S., & Wu, Z. (2000). Transcribing with annotation graphs. In (pp. 1517–1521).

Manual software, tabletrans. (2004). (`http://sourceforge.net/projects/agtk/`)

Medeiros, J. C. (1995). *Processamento morfológico e correcção ortográfica do português.* Portugal.

Nicholas Nethercote, J. S. (2003). Valgrind: A program supervision framework. In *Electronic notes in theoretical computer science 89.*

O. Suhre, T. G. e. (2004). Design and implementation of the uima common analysis system. *IBM SYSTEMS JOURNAL*, *43*(3).

Ribeiro, R., Mamede, N. J., , & Trancoso. (2003). Using morphossyntactic information in tts systems: comparing strategies for european portuguese. In *Computacional processing of the portuguese language:: 6 th international workshop, propor 2003, faro, portugal, june 26-27, 2003. proceedings* (Vol. 2721). Spinger.

Rodgers, P., Gaizauskas, R., Humphreys, K., & Cunningham, H. (1997). Visual execution and data visualization in natural language processing. *IEEE Symposium on Visual Languages*, *97*.

Schabes, Y. (1991). Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. In *Proceedings of the 29th annual meeting on association for computational linguistics* (pp. 106–113). Morristown, NJ, USA: Association for Computational Linguistics.

Software gate-general architecture for text engineering 3.1. (2006, April). (`http://gate.ac.uk/download/index.html`)

Srinivasan, R. (1995, August). *Rpc: Remote procedure call protocol specification version 2* (Tech. Rep.). Sun Microsystems.

Viterbi, A. J. (1967, April). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In *Ieee transactions on information theory.* (The Viterbi decoding algorithm is described in section IV)

Author Index

I Appendices

A Interface gráfica do sistema ShRep

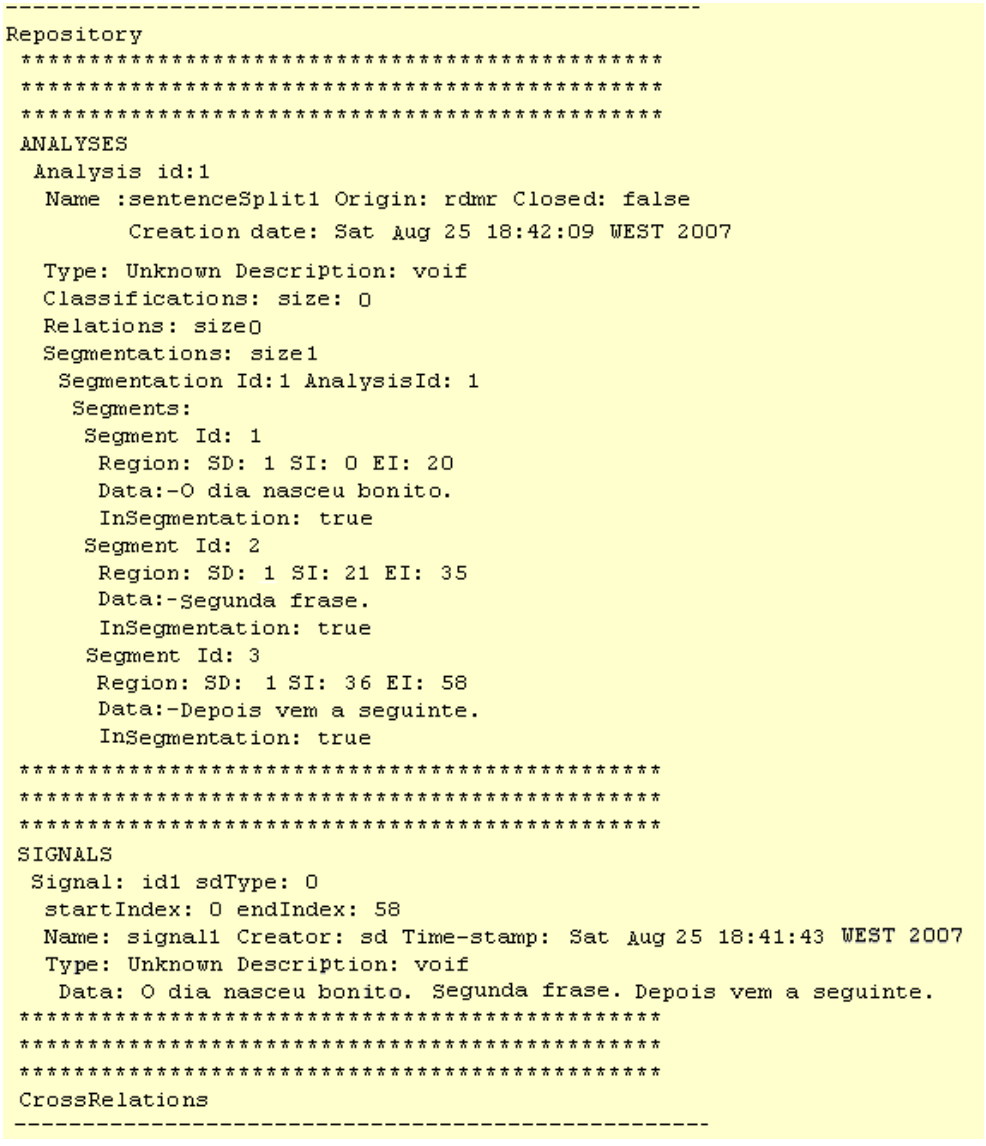

Figure A.1: Interface gráfica existente inicialmente para o sistema ShRep

B Exemplo da informação trocado entre o Servidor e a Biblioteca Cliente

```
A) VERSÃO ANTIGA
<segment id="11" type="" description="">
  <segmentation-identification id="1" analysis-id="1" />
  <region signal-data-id="1">
    <start-index><index type="0" value="57" /></start-index>
    <end-index><index type="0" value="58" /></end-index>
  </region>
  <original-data><data data="O" data-type="0" /></original-data>
  <classifications>
     <classification id="14" type="Unknown" description="voif">
       <analysis-identification id="1" />
       <segment-identification id="11" segmentation-id’="1" analysis-id="1" />
       <features>
         <feature key="root" value=" " />
         <feature key="pos" value="Pd" />
         <feature key="tag" value=" " />
       </features>
     </classification>
     <classification id="15" type="Unknown" description="voif">
       <analysis-identification id="1" />
       <segment-identification id="11" segmentation-id’="1" analysis-id="1" />
       <features>
         <feature key="root" value=" " />
         <feature key="pos" value="Td" />
         <feature key="tag" value=" " />
       </features>
     </classification>
     <classification id="16" type="Unknown" description="voif">
       <analysis-identification id="1" />
       <segment-identification id="11" segmentation-id’="1" analysis-id="1" />
       <features>
         <feature key="root" value=" " />
         <feature key="pos" value="Pp" />
         <feature key="tag" value=" " />
       </features>
     </classification>
     <classification id="17" type="Unknown" description="voif">
       <analysis-identification id="1" />
       <segment-identification id="11" segmentation-id’="1" analysis-id="1" />
       <features>
         <feature key="root" value=" " />
         <feature key="pos" value="Nc" />
         <feature key="tag" value=" " />
       </features>
     </classification>
   </classifications>
   </relations />
</segment>
```

Figure B.1: Exemplo de versão antiga da informação transmitida entre o Servidor e as Bibliotecas Cliente.

```
B) VERSÃO NOVA
<segment s="11 0 0
            1 1
            1 1 0 57 0 58
            0
            1 1 O 0
            4
            14 1 Unknown 1 voif 1 0 11 1 1
            3
            1 root 0 pos 1 Pd 1 tag 0
            1 root 0 pos 1 Td 1 tag 0
            1 root 0 pos 1 Pp 1 tag 0
            1 root 0 pos 1 Nc 1 tag 0
            0">
```

Figure B.2: Exemplo da nova versão (compactada) da informação transmitida entre o Servidor e as Bibliotecas Cliente.