Universidade de Évora

Mestrado em Engenharia Informática

Uma Metodologia para a construção automática de Ontologias e a sua aplicação em Sistemas de Recuperação de Informação

José Miguel Gomes Saias

orientador: *Prof. Doutor Paulo Quaresma*

Novembro de 2003

Esta dissertação não inclui as críticas e sugestões feitas pelo júri.

Prefácio

Este documento contém uma dissertação intitulada *"Uma Metodologia para a construção automática de Ontologias e a sua aplicação em Sistemas de Recuperação de Informação"*, um trabalho do aluno *José Miguel Gomes Saias*[1], estudante de Mestrado em Engenharia Informática na Universidade de Évora.

O orientador deste trabalho é o Professor Doutor Paulo Quaresma[2], do Departamento de Informática da Universidade de Évora.

O autor do trabalho é licenciado em Engenharia Informática, pela Universidade de Évora. A presente dissertação foi entregue em Novembro de 2003.

[1]jsaias@di.uevora.pt
[2]pq@di.uevora.pt

Agradecimentos

O trabalho de Mestrado constitui um eminente desafio que obriga o aluno a uma profunda dedicação. Na recta final do trabalho, dedico umas palavras a algumas pessoas, que de uma ou de outra forma, contribuiram para eu alcançar este desígnio.

Começando pela secção académica, quero agradecer ao meu orientador, o Professor Paulo Quaresma, pelo incentivo, acompanhamento e disponibilidade para as discussões sobre a evolução do trabalho.
Agradeço também à Professora Irene Rodrigues pela ajuda no campo de PLN, ao Professor Salvador Abreu pelas produtivas conversas sobre programação por restrições, e aos membros do DI que me ajudaram em determinadas questões.

Para a minha família, um grande agradecimento pelo apoio incondicional que me deram ao longo deste trabalho, como sempre!
Obrigado Mãe, Pai, Joaquim e Francisco.

Depois, gostava de referir um conjunto de colegas e amigos. Obrigado José Carlos, Miguel, David, Rui, e muitos outros que não poderia aqui nomear, pela camaradagem em projectos académicos e outros.
Obrigado também Artur, pelo teu apoio e compreensão!

Sumário

O exponencial aumento do volume de informação disponível, motivado pela evolução tecnológica, originou uma necessidade de meios automáticos de pesquisa e filtragem de informação, como os Sistemas de Recuperação de Informação (SRI).

Um SRI não deve limitar-se à pesquisa directa de palavras em documentos, deve conseguir extrair e processar a informação semântica de um texto. Os SRI podem representar a semântica dos documentos num ambiente Semantic Web, através de ontologias. O processo de construção de ontologias é complexo e moroso. É difícil saber se uma ontologia é suficientemente abrangente para a informação semântica dos documentos. Pretende-se a automatização deste processo, permitindo o ajuste dinâmico da ontologia à semântica encontrada nos documentos.

Este trabalho incide sobre a construção automática de ontologias a partir de um conjunto de documentos e a utilização dessas ontologias em processos de inferência. Foi construído um protótipo, que analisa a base de documentos em busca de elementos a incluir na ontologia que representará (parcialmente) a semântica expressa nesses documentos. A análise dos documentos envolve técnicas de PLN, Programação em Lógica e outras, descritas ao longo desta dissertação.

A Methodology to automatic Ontology construction and its application within Information Retrieval Systems

ABSTRACT

The exponential growth of the amount of available information, caused by technological evolution, created a need for automatic information lookup and filtering methods like Information Retrieval Systems (IRS).

IRS can perform direct word search in a document, but besides that they must be able to extract and process text semantic information. IRS can represent documents semantic content in a Semantic Web environment, through ontologies. The ontology construction process is somewhat complex and slow. It's difficult to know if an ontology is wide enough to cover the full documents semantic. The process must become automatic, to allow the dynamic ontology tuning and concordance with documents content.

This work is about automatic ontology construction from text documents and ontology-based inference process. A prototype has been made to analyse the documents text , looking for the elements to include in the new ontology which will partially represent the text semantics for those documents. The document analysis includes NLP techniques, Logic Programming and others described along this dissertation.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Neste primeiro capítulo faz-se uma introdução sobre a área do trabalho e a relevância que a temática aqui abordada tem na sociedade de informação.
O enquadramento e a motivação para o trabalho são apresentados na secção 1.1. Os objectivos definidos são enumerados na secção 1.2.

1.1 Enquadramento e Motivação

As Tecnologias de Informação desempenham um papel importante na sociedade actual. A tecnologia conquistou o seu lugar e, de forma mais ou menos evidente, estendeu-se a todos os utensílios e actividades da civilização moderna.
Com a Internet, a distância física deixou de ser um problema. A Web proporciona uma gigantesca base de dados em continuo crescimento e actualização, disponível em qualquer ponto do mundo, minimizando as diferenças sociais e económicas. A adaptação de pessoas e instituições às sucessivas evoluções tecnológicas tornou possível o acesso a informação em quantidades cada vez maiores, de um modo progressivamente mais rápido e mais fácil.

Perante um enorme volume de dados é imprescindível um processo de os organizar para se conseguir o melhor proveito. São necessárias técnicas automáticas de filtragem de informação, como Sistemas de Recuperação de Informação (SRI), para auxiliar o utilizador a encontrar aquilo que realmente procura.

O processo mais tradicional para pesquisar um documento, muito usado na maioria dos SRI, é a pesquisa por palavras-chave. A ideia é ter um vector de palavras associadas a cada documento, contendo uma série de conceitos identificados no

texto, por via manual ou automática. Tal abordagem conduz, em geral, a demasiados resultados, sendo que muitos deles são indesejáveis ou irrelevantes face ao pretendido pelo utilizador. Suponhamos que pretendemos encontrar uma página na web relativa a alguém que:

- o nome é Silva
- é funcionário da Universidade de Évora
- gosta de futebol

Se usarmos o mais popular motor de busca na internet e efectuarmos uma pesquisa típica com *Silva*, *Universidade de Évora* e *futebol*, a lista de resultados será muito extensa, incluindo:

- alunos da Universidade de Évora com apelido Silva
- calendário e resultados da equipa de futebol da Universidade de Évora
- pessoas cujo nome é Silva e mencionam terceiros ligados ao futebol

Figura 1.1: Excesso de resultados numa pesquisa

Não é suficiente ter os documentos indexados. Para conseguir um nível de eficácia superior, um SRI em bases de texto necessita obter, representar e processar a informação semântica dos textos. A verdadeira mensagem do texto só poder obter-se pelo significado expresso em cada frase ou expressão, pelo que um sistema não se deve limitar à pesquisa de palavras, isoladamente.

Para encontrar o resultado desejado na pesquisa anterior, não basta que o documento contenha aquelas palavras. É preciso deduzir do texto que existe uma relação laboral entre uma pessoa de nome *Silva* e uma instituição *Universidade de Évora*, bem como o gosto dessa pessoa por *futebol*.

Na área de Processamento de Língua Natural existem técnicas que permitem analisar um texto com o objectivo de extrair informação. Ao combinar algumas dessas técnicas, podemos obter relações entre palavras de cada expressão, chegando a fragmentos de informação que representam, parcialmente, a semântica do texto.

A Internet começa a reflectir a importância da semântica, com o surgimento e expansão da *Semantic Web*[1], que é alvo de crescente interesse, a nível global, tanto da parte de pessoas ligadas às novas tecnologias como de outros sectores, das ciências às letras.
Em todas as áreas de cada quadrante parece haver vontade de adoptar a versão Semantic Web dos documentos existentes e garantir que os novos respeitam este formato. Desde as famílias de felinos, répteis ou aves de rapina até às classes de palavras de uma gramática, os tipos de vinho ou meios de transporte, tudo, mesmo tudo, parece ser naturalmente representável em Semantic Web.

A possibilidade de ultrapassar a representação destinada a humanos, com linguagens capazes de lidar formalmente com a informação, num contexto favorável às máquinas e ao tratamento automático, leva a que esta área seja eleita para diversos fins.
Alguns serviços na Web, como a previsão meteorológica ou a cotação das bolsas, passaram a oferecer a informação num modo alternativo, invisível para nós, em que os dados são representados num formato predefinido e fixo, o que não acontece com a representação para leitura humana, onde se introduzem alterações periodicamente, quer seja no aspecto visual de letras ou tamanhos, na ordem de listagem ou agrupamento de categorias. Esta atitude facilita a construção de ferramentas para a consulta desses dados.

Para serem mais dinâmicos e abrangentes, os Sistemas de Recuperação de Informação devem estar preparados para lidar com a informação disponível, que pode ser proveniente de um meio exterior, nomeadamente em ambiente Semantic Web.
Por outro lado, é desejável que a interacção com estes sistemas não se resuma à obtenção de resposta a interrogações. Existe interesse em consultar a base de

[1] Um novo conceito de Web, descrito na secção 2.1.

conhecimento que tais sistemas possuem, tanto para consulta directa como para a preparação de novas ferramentas de inferência. Esta necessidade de cooperação justifica a publicação da informação de cada sistema.

A representação semântica dos documentos é a base para o processo de inferência. Esta informação pode servir para a construção de uma ontologia em Semantic Web, na medida em que envolve os conceitos referidos nos documentos.

Através de uma ontologia pode definir-se uma hierarquia de classes representativas de conceitos, objectos ou entidades, caracterizadas pelas respectivas propriedades. A formalidade inerente à ontologia e à linguagem usada garante a sua adequação à semântica dos documentos, resultando num meio ideal para publicação em ambiente Web e ao mesmo tempo possibilitando a inferência para responder a interrogações. A figura 1.2 mostra um esquema com uma ontologia e um sistema para construir e interagir com essa ontologia.

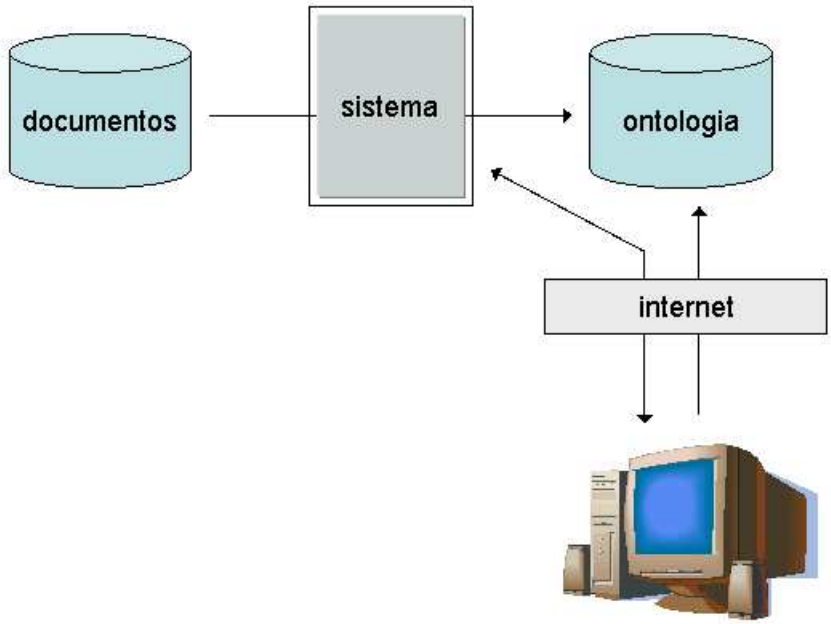

Figura 1.2: Sistema baseado em ontologia

As pessoas responsáveis pela elaboração da ontologia deparam-se com problemas que vão muito além das dificuldades técnicas. Na primeira tentativa para criar uma ontologia de conceitos, em geral, surgem dúvidas como:

- saber se a ontologia dever ser construída manualmente
- saber se a ontologia construída já é suficiente para o pretendido

Mesmo com alguma experiência, ao tentar construir uma nova ontologia relacionada com outro assunto, voltam a surgir as mesmas dificuldades. Não existe uma solução universal. Na prática existe sempre um risco de se obter uma ontologia demasiado simples, ou o contrário, uma ontologia tão elaborada que torna a sua utilização pouco viável.

O trabalho apresentado tem a ver com esta problemática, nomeadamente ao nível da automatização do processo de construção da ontologia.

1.2 Objectivos

O propósito do trabalho é mostrar uma possível aplicação de ontologias em Sistemas de Recuperação de Informação, apresentando uma metodologia para construção automática da ontologia e o modo como o sistema interage com a mesma.

Foi analisado o Sistema de Recuperação de Informação da Procuradoria Geral da República Portuguesa [QR01b], já existente (descrito mais adiante), e também a sua base documental, que consiste em pareceres de natureza jurídica, com o propósito de ser concebido um sistema paralelo, um modelo baseado em ontologias.

Um objectivo fundamental do trabalho é desenvolver um mecanismo para construir automaticamente uma ontologia, de maneira que esta represente, na medida do possível, a informação expressa no texto dos documentos. Tal mecanismo é procurado por muitos, e ainda que os seus resultados possam ser parcialmente incompletos, poderá contribuir significativamente para o melhoramento do processo. A ontologia construída pode ser usada para publicar os dados recolhidos do texto. Outra importante característica do sistema é a utilização da ontologia para processos de inferência para responder a interrogações sobre um documento, já que é a ontologia que representa a semântica parcial desse documento.

Resumindo, o que se descreve a seguir é uma metodologia para construir de modo automático uma ontologia a partir de um conjunto de documentos e a sua utilização em processos de inferência.

O próximo capítulo faz uma síntese dos conceitos e da área em que o trabalho se insere, abordando temas como Semantic Web e Recuperação de Informação. O capítulo 3 descreve alguns trabalhos relacionados com o tema desta dissertação, uns sobre a utilização de ontologias e outros relativos à construção de ontologias. O sistema proposto é apresentado no capítulo 4, e descrito em detalhe ao longo dos capítulos 5, 6 e 7. O capítulo 8 inclui alguns testes e exemplos de interacção com o sistema, e por último, a discussão do trabalho é apresentada no capítulo 9.

Capítulo 2

Conceitos Envolvidos

Antes de apresentar o trabalho realizado importa fazer uma introdução sobre a área em que se enquadra e os conceitos envolvidos.
A secção 2.1 descreve a área de Semantic Web, apresentando algumas definições, os seus objectivos (2.1.1) e princípios em que assenta (2.1.2). A subsecção 2.1.3 enumera as várias camadas lógicas que se identificam nesta área, seguindo-se o conceito de ontologia (2.1.4) e uma lista de linguagens utilizadas em Semantic Web (2.1.5).
A linguagem OWL, empregue neste trabalho, é descrita na secção 2.2, com informações sobre classes e propriedades (2.2.1), instâncias (2.2.2) e algumas ferramentas relacionadas com a linguagem (2.2.3).
A secção 2.3 apresenta uma breve descrição sobre a área de Recuperação de Informação, o objectivo dos Sistemas de Recuperação de Informação (2.3.1) e alguns métodos de Recuperação de Informação (2.3.2).

2.1 Semantic Web

A *World Wide Web* tornou-se num vasto e poderoso meio de comunicação, de pesquisa, palco para comércio e serviços. Contudo, o seu enorme potencial fica parcialmente limitado às capacidades do ser humano para navegar pelas diferentes fontes de informação disponíveis. Existe um volume dificilmente quantificável de documentos, com endereços URL e referindo outros documentos com *links* para os respectivos endereços, em que muitas vezes o utilizador se perde.

> *"The Semantic Web is a vision: the idea of having data on the Web defined and linked in such a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications."*

in Semantic Web Activity Statement[1]

> *"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."*

in [BLHL01]

As citações anteriores apresentam uma sucinta descrição do conceito de **Semantic Web** como uma extensão da web "tradicional" à qual se adicionam funcionalidades de carácter semântico.

Antes, os documentos estavam exclusivamente pensados para a leitura por pessoas, consistindo em textos em língua natural, normalmente incluindo algumas formatações sobre as fontes, as cores e os tamanhos. Em Semantic Web, os documentos são enriquecidos com uma informação complementar, representando a semântica que contêm, através de um código propício para o processamento automático, facilitando assim a consulta dos mesmos por parte de programas.

2.1.1 Objectivo

O objectivo da Semantic Web é o desenvolvimento de *standards* e tecnologias que permitam às máquinas e processos automáticos compreender mais informação expressa na Web, servindo de suporte para a descoberta de conhecimento, integração de dados e automatização de tarefas.

Com a adopção deste tipo de representação para documentos abre-se o caminho para novas formas de:

- publicação de informação
- pesquisa de informação
- integração de dados
- troca de informação

Ao efectuar uma pesquisa, os resultados obtidos serão mais precisos. Por outro lado, será mais simples e exequível integrar dados provenientes de diversas fontes, uma vez que é possível comparar e estabelecer uma relação entre os dados com um relativo rigor que antes não existia.

[1] http://www.w3.org/2001/sw/Activity

Actualmente, podem já encontrar-se bastantes documentos em formato Semantic Web, mas representam uma pequena parte dos documentos gerados no dia-a-dia. Depois existem ainda os documentos mais antigos, os quais poderiam vir a ser traduzidos para Semantic Web. A Web está em permanente actualização e é importante que em todo o mundo esteja esclarecido e consciente das vantagens da Semantic Web.

A ideia, a longo prazo, é ser mais que uma experiência bem sucedida para determinados domínios. É desejável que a Semantic Web atinja uma dimensão comparável à actual Web. Para isso é necessário aumentar o número de documentos anotados.
O impacto desta disseminação global da Semantic Web é enorme, na medida em que vai permitir tirar partido da capacidade de processamento das máquinas, de um modo mais eficaz e com uma eficiência jamais alcançada por um humano.

2.1.2 Princípios

Existem seis princípios que servem de base ao conceito de Semantic Web. Esses princípios são os seguintes:

1. Tudo pode ser identificado por URIs
2. Os recursos e as suas referências podem ter tipos
3. É aceitável ter apenas informação parcial
4. Não há necessidade de verdade absoluta
5. Há lugar para a evolução
6. Desenho minimalista

As pessoas, os animais, os lugares, os objectos e tudo o que existe no mundo físico e imaginário necessita de um identificador para que se possa referir inequivocamente. A Web já possui o conceito de identificador para um documento ou recurso, é conhecido como *Universal Resource Identifier*, ou abreviadamente, URI. Através da utilização deste tipo de identificador, único à escala global, é possível especificar um conjunto de características sobre uma entidade de um modo perceptível para todos, ultrapassando a barreira das designações regionais. Como exemplo para a identificação de um lugar, a cidade de Helsínquia pode ser referida em Semantic Web pelo URI da respectiva página Web oficial, tal como sugere a figura 2.1.

Figura 2.1: A cidade de Helsínquia identificada pelo URI da sua página Web

Os documentos que usualmente encontramos na Web estão pensados para a análise humana, sem meta-informação a explicar para que servem ou qual a sua relação com outros documentos. Ao encontrar um *link* no meio do texto, referindo outro documento, o utilizador pode usar a sua intuição e a partir do texto à volta do *link* deduzir algum tipo de relação entre os documentos, mas as máquinas não conseguem fazer o mesmo.

Em Semantic Web, os recursos relacionam-se com outros recursos, também através de *links* para determinados identificadores URIs. O que existe de novo relativamente à Web, é que os recursos e *links* podem ter tipos. Isto confere-lhes mais expressividade, associando-os a conceitos que poderão agora ser captados pelas máquinas.
É possível dizer que um recurso é uma especialização de outro, identificado pelo URI, ou até que o autor deste documento é um recurso Semantic Web do tipo pessoa (identificado também com um URI).

A gigantesca dimensão da Web foi um factor determinante para que as preocupações estivessem mais relacionadas com a escalabilidade que com a integridade das referências já existentes. Acontece muitas vezes que os documentos mudam de local e os *links* deixam de apontar para o recurso correcto. O mesmo pode acontecer em Semantic Web. É necessário que as aplicações estejam precavidas contra esta possibilidade e consigam funcionar com base nos dados disponíveis. Em geral, a informação disponível/útil é parcial, e é com isso que as aplicações terão de trabalhar.

Nem sempre o que se encontra num documento Web corresponde à verdade. Mais uma vez, a Semantic Web é igualmente afectada por casos de informações

incorrectas.
A fiabilidade de um documento deve ser avaliada pela aplicação que processa a informação desse mesmo documento. Este processo pode ser baseado na atribuição de um contexto às asserções encontradas. Cada contexto, com asserções, é etiquetado com o autor das afirmações, quando as fez e as credenciais ou nível de credibilidade que então lhe foi associado. Depois cabe à aplicação recolher apenas a informação de determinado grupo de contextos.

Quando um utilizador A indica ao seu programa para confiar nas informações do seu amigo B e este já fez o mesmo para um conjunto de pessoas, e assim por diante, cada um acaba por confiar nos que estão mais adiante na cadeia. É aqui que surge o conceito de *Chain of Trust*, pela cadeia de confiança estabelecida.

Um conceito pode ser definido de modo ligeiramente diferente por pessoas de lugares diferentes, ou inclusivamente pela mesma pessoa em momentos distintos. Em Semantic Web é possível utilizar dados relativos aos vários modos de definir o mesmo conceito usando vocabulários distintos.
Existe a possibilidade de adicionar dados, actualizando a informação disponível, sem que para isso seja necessário alterar os dados antigos. O princípio assenta em suportar a evolução sem que se alterem os dados já disponíveis. Como exemplo, consideremos os dados do passado sobre uma pessoa cujo nome é José, e que após concluir a sua licenciatura se inscreveu em mestrado.

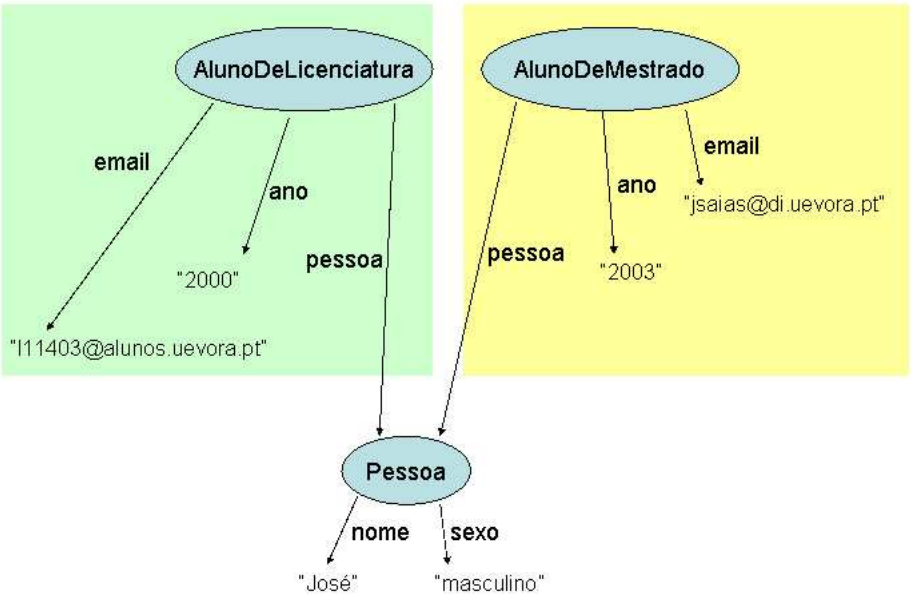

Figura 2.2: Evolução com novos dados, mantendo os antigos

De acordo com o apresentado na figura 2.2, a actualização do endereço de correio electrónico pode efectuar-se adicionando novos dados. Os dados antigos podem permanecer sem perigo de confusão e fazem sentido para o contexto em que foram inseridos. A data dos dados pode servir para identificar a informação actual.

Existe a preocupação de manter um desenho minimalista, de modo a preservar uma representação simples para os conceitos simples e tornar possível a representação de conceitos complexos.

2.1.3 As camadas lógicas de Semantic Web

Os princípios de Semantic Web são implementados em várias camadas de tecnologias Web e outros standards. Essas camadas são apresentadas na figura 2.3.

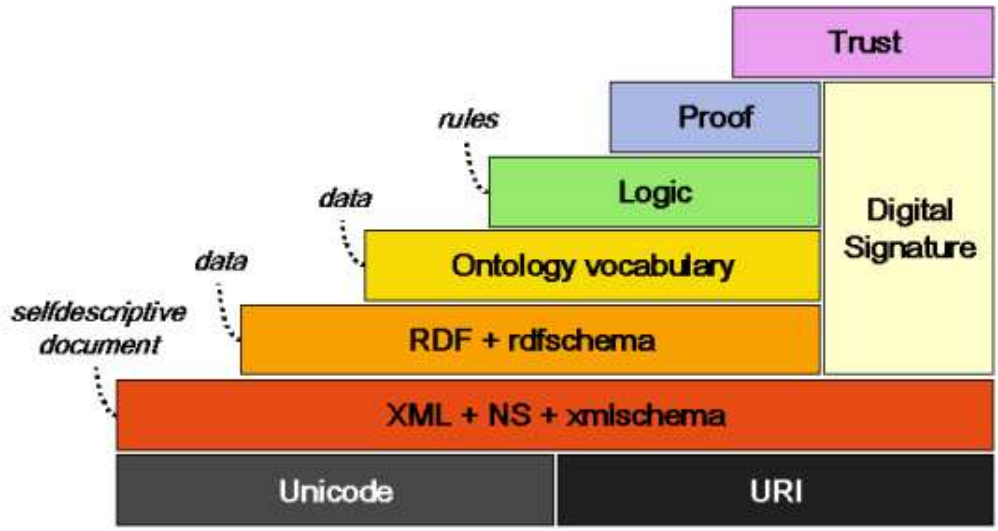

Figura 2.3: As camadas lógicas de Semantic Web

A camada de Unicode e URI tem a ver com a definição de *character sets* apropriados e mecanismos de referenciação de objectos (pelo respectivo endereço).
Ao nível da camada XML existe uma descrição baseada em standards XML, *namespaces* e *xmlschema* que torna possível a integração das definições Semantic Web noutras aplicações baseadas também em standards XML.

As camadas do topo: *Logic*, *Proof* e *Trust* (e de modo transversal as Assinaturas Digitais) são apresentadas apenas como demonstração. Actualmente está em preparação um conjunto de standards para este nível.
De baixo para cima, deve ser possível definir um conjunto de regras lógicas, avaliadas por aplicações da camada superior para provar alguma coisa. Na camada *Trust* existem mecanismos para confiar ou não em determinada prova. Prevê-se a utilização de assinaturas digitais como garantia da idoneidade da informação.

Finalmente, existem mais duas camadas intermédias, *RDF+rdfschema* e *Ontology vocabulary*. A primeira engloba as linguagens Semantic Web, definindo o vocabulário usado para a descrição dos objectos e dos seus respectivos tipos. Depois existe uma camada de estruturas hierárquicas de objectos, usualmente agrupados por classes, definida com uma linguagem da camada anterior.
As próximas secções são dedicadas a esta matéria.

2.1.4 Ontologia

Uma **ontologia** define os termos usados para descrever e representar uma área de conhecimento. As ontologias são usadas por pessoas e aplicações para a troca de informações sobre um determinado domínio ou área de conhecimento, como biologia ou medicina. Uma ontologia fornece definições de conceitos básicos de um domínio, apropriadas para o processamento automático.

Em Semantic Web, as ontologias são de grande utilidade, consistindo em estruturas com:

- classes para representar conceitos gerais de qualquer área
- relações identificadas entre os objectos
- propriedades ou atributos dos objectos descritos

As ontologias são efectivamente um meio de representar, com orientação para as máquinas e processamento automático, a semântica expressa em documentos Web.
As ontologias facilitam a pesquisa de informação e integração de dados de diferentes comunidades, porque fornecem uma base comum que garante a coerência dos dados (que assim estão devidamente categorizados e descritos, de um modo mais *standard*).

As ontologias podem ser classificadas em dois géneros. Numa ontologia horizontal procura-se atingir uma representação para todos os conceitos possíveis, sem uma descrição muito detalhada. No caso de uma ontologia vertical, só existem conceitos de determinada área, com uma descrição completa, de acordo com o domínio em que se inserem.

Exemplos de utilização de ontologias

Através da Semantic Web, as ontologias podem ser aplicadas para melhorar o funcionamento de aplicações Web já existentes e permitir a implementação de novas aplicações e serviços. Vejamos alguns casos em que se podem utilizar ontologias:

1. Portais Web
   Um portal na Web contem informação acerca de um determinado assunto, como o *site* de uma biblioteca ou as pautas de uma escola. Tipicamente os documentos do portal estão orientados para a leitura por pessoas, em que a preocupação se prende apenas com as fontes, cores e tamanhos das letras.
   Se um portal enriquecer os seus documentos com uma ontologia que represente a sua informação, estará aberto o caminho para a utilização de processos automáticos para a consulta desse portal.

2. Repositórios Multimédia
   Numa galeria virtual, por exemplo, seria muito útil ter uma ontologia para associar a cada imagem ou filme um conjunto de informações como o título, o ano, o autor e o tema. Se esta informação for incluída numa ontologia, para além de estar expressa textualmente, a pesquisa automática pode efectuar-se com superior eficiência por parte de aplicações ou agentes.

3. Serviços
   Uma ontologia facilita a troca de informação entre sistemas. Como tal, pode aplicar-se a vários serviços, da previsão meteorológica ao comércio electrónico.
   A computação ubíqua, por exemplo, envolve mobilidade e transparência da tecnologia para o utilizador. Os dispositivos devem automaticamente identificar-se perante outros, num ambiente de grande interoperabilidade. Esta é mais uma área em que se pode aplicar uma ontologia, para normalizar os conceitos e facilitar a troca de informação.

2.1.5 Linguagens Semantic Web

À medida que a Semantic Web foi crescendo, foram surgindo várias linguagens para codificar a semântica dos documentos. A natureza das linguagens criadas foi influenciada pela época e pela utilização que os seus criadores lhe pretendiam atribuir.
Em seguida é apresentada uma descrição das principais linguagens que marcaram a evolução da Semantic Web.

RDF

A linguagem *Resource Description Framework*[2] (RDF) foi pensada para representar informação sobre os recursos na World Wide Web. É particularmente útil para descrever meta-informação sobre os documentos, nomeadamente o título, o nome do autor ou data de actualização. Através da generalização de conceito de recurso (*web resource*), esta linguagem pode também ser empregue para representar informação sobre todas as coisas que são identificadas na Web.
A linguagem RDF fornece um meio universal para expressar a informação sobre recursos, que pode ser trocada entre aplicações sem perda de significado. O primeiro *draft* RDF foi publicado em Outubro de 1997 e chegou ao estatuto de *W3C Recommendation* em Fevereiro de 1999.

O RDF é baseado na identificação de recursos na Web através dos seus endereços URIs e na respectiva descrição. A descrição de um recurso faz-se com um conjunto de propriedades para as quais se atribui um valor.

A sintaxe RDF é baseada em XML, e como tal, propícia para aplicação em ambiente Web. Um *statement* RDF pode ser visto como um triplo que define o valor de uma propriedade para um objecto ou recurso.

Como exemplo, podemos afirmar que "Susana tem 25 anos de idade" através da expressão:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:props="http://www.di.uevora.pt/~jsaias/sw/props#">
  <rdf:Description rdf:about=
              "http://www.di.uevora.pt/~jsaias/sw/pessoas#susana">
       <props:idade>25</props:idade>
  </rdf:Description>
</rdf:RDF>
```

[2] http://www.w3.org/TR/rdf-mt/

No início são definidos dois XML namespaces, *rdf* e *props*, usados para identificar o local onde são definidos: *Description*, *about*, *RDF* e *idade*. O recurso descrito é uma pessoa de nome "Susana", identificada pelo respectivo URI e caracterizada pela propriedade "idade".

RDFS

RDF Schema[3] (RDFS) estende a linguagem RDF com vocabulário para descrever classes de objectos, as suas propriedades e o tipo de dados para o valor dessas propriedades.
O vocabulário RDFS é normalmente usado com um *XML namespace* que surge no código com o prefixo `rdfs:`. Em RDFS, as propriedades de objectos ou recursos são caracterizadas através das propriedades da linguagem: `rdfs:domain` e `rdfs:range` onde se indica que classe tem a propriedade e qual o tipo para o respectivo valor da propriedade.

Voltando ao exemplo anterior, a propriedade *idade* encontra-se definida no URI indicado em `props`, da seguinte forma:

```
<rdf:Description rdf:ID="idade">
  <rdf:type rdf:resource=
          "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#Pessoa"/>
  <rdfs:range rdf:resource=
                      "http://www.w3.org/2001/XMLSchema#integer"/>
</rdf:Description>
```

O código descreve *idade*, atribuindo-lhe um tipo com o valor `Property` (trata-se de um propriedade em RDF). É também identificada a classe RDF que tem a propriedade, `Pessoa`, e o tipo para o seu valor (inteiro).

Topic Maps

Topic Maps[4] começou por ser um trabalho de *Conventions for the Application of HyTime*, até ser aceite como um tema de trabalho pelo *ISO's SGML working group*, em 1996. Foram necessários mais quatro anos até ser um standard aprovado, em Janeiro de 2000. A organização TopicMaps.Org, posteriormente, reformulou

[3] http://www.w3.org/TR/rdf-schema/
[4] http://www.topicmaps.org/

o Topic Maps dando-lhe uma sintaxe XML, de onde resultou o *XML Topic Maps* (XTM).

Entre as aplicações de Topic Maps, destaca-se a procura de informação em grandes repositórios de dados. A aplicação original da linguagem era a construção de índices, indexes e glossários para documentos, mas a sua aplicabilidade era mais vasta, nomeadamente na Web, para definição de relações entre entidades, constituindo uma estrutura que torna a pesquisa de dados mais eficiente.

SHOE

Simple HTML Ontology Extensions[5] (SHOE) é uma linguagem desenvolvida por estudantes e investigadores da Universidade de Maryland e é uma das primeiras *markup languages* baseadas em ontologias para a Web.
Esta linguagem é *SGML-compliant* desde a versão 1.0 e estende o HTML com *tags* necessárias para introduzir indicações semânticas.

SHOE pode ser usado para interrogações, directamente pelo software de um *browser* ou outras aplicações. Um dos seus propósitos é descobrir rapidamente a "não utilidade"de um documento [HH00].
Actualmente, a linguagem SHOE já não é mantida activamente pelos seus criadores mas teve um papel importante, tendo sido uma influência para linguagens posteriores.

OIL, DAML e DAML+OIL

Ontology Inference Layer[6] (OIL) resulta de um projecto de investigação da União Europeia para as *Information Society Technologies*. É uma tecnologia pensada para a Semantic Web e também procura resolver o *findability problem* de informação, facilitar a gestão do conhecimento e servir de suporte ao comércio electrónico.

DARPA Agent Markup Language[7] (DAML) nasceu num programa de investigação em Agosto de 2000, pela organização americana DARPA. A orientação base é o suporte à Semantic Web, mas também pode ter aplicações no campo do

[5] http://www.cs.umd.edu/projects/plus/SHOE/
[6] http://www.ontoknowledge.org/oil/
[7] http://www.daml.org/

conhecimento. O desenvolvimento da linguagem DAML demonstrou o interesse da comunidade de investigadores em IA nas possibilidades da Web.

DAML permite descrever melhor os dados RDF e assim adiciona mais semântica aos dados. Permite restringir os valores possíveis de uma propriedade, para além de contar com propriedades muito úteis para as aplicações, como `daml:samePropertyAs`, `daml:inverseOf` e `daml:TransitiveProperty`.
Pode dizer-se que DAML reforça o RDFS e acrescenta-lhe alguma semântica.
O que há de comum entre OIL e DAML:

- hierarquias de classes e propriedades baseadas em herança
- construção de classes a partir de outras, usando operadores como intersecção, união e negação
- restrição do domínio e cardinalidade das propriedades
- propriedades transitivas e inversas
- tipos de dados concretos, como inteiros e strings

DAML+OIL[8] surge já no final de 2000. Resulta da cooperação entre os investigadores americanos e europeus que desenvolveram o DAML e OIL, tendo sido aproveitado o melhor de cada linguagem, juntamente com alguma influência de SHOE.

OWL

A mais recente *Web Ontology Language*, conhecida como OWL[9], é uma *markup language* de caracter semântico para a publicação de informação na Web, usada para representar conceitos e relações entre conceitos.
A linguagem OWL é uma extensão de RDF e deriva da linguagem DAML+OIL, incorporando as actualizações resultantes da experiência de desenho e aplicação daquela linguagem.

Pode encontrar-se uma descrição mais completa sobre a linguagem OWL na secção 2.2.

[8] http://www.w3.org/TR/daml+oil-reference
[9] http://www.w3.org/TR/owl-ref/

2.2 A Linguagem OWL

A linguagem OWL[10], *Web Ontology Language*, apresenta-se como uma evolução das suas antecessoras, RDF e DAML+OIL, e encontra grande aceitação por parte dos utilizadores de linguagens Semantic Web. Em Agosto de 2003 encontra-se com o estatuto de *W3C Candidate Recommendation*.

OWL fornece três "sublinguagens", com diferente poder expressivo para utilização por distintas comunidades de utilizadores:

- **OWL Lite** - Para as necessidades básicas de uma hierarquia e restrições simples.
- **OWL DL** - Usado quando se pretende a máxima expressividade sem descorar a computabilidade da representação. (O nome DL deve-se à área *description logics*.)
- **OWL Full** - Quando se pretende a máxima expressividade e a liberdade sintáctica do RDF, sem preocupações computacionais.

Cada "sublinguagem" é uma extensão da anterior. OWL Full é uma extensão de RDF, enquanto que as outras são uma extensão de um subconjunto de RDF. Em cada aplicação escolhe-se a mais adequada, de acordo com os parâmetros expressividade e computabilidade.

2.2.1 Classes e Propriedades

Uma ontologia em OWL é composta pelas classes de objectos e pelas suas instâncias. Uma classe descreve uma categoria de objectos, com um conjunto de propriedades e relações com outros objectos.

O código com uma ontologia OWL começa normalmente com a definição de *XML namespaces*, incluindo um para a sintaxe da linguagem e que é usado em seguida com o prefixo `owl:`. Por defeito, todos os objectos criados são membros da classe `owl:Thing`, que é a classe raíz de qualquer hierarquia, ainda que implicitamente. Vejamos um exemplo onde se define a classe `Gato` como subclasse de `Felino`, em que é evidente a influência de RDF e RDFS:

[10] http://www.w3.org/TR/owl-ref/

```
<owl:Class rdf:ID="Gato">
  <rdfs:subClassOf rdf:resource="#Felino" />
</owl:Class>
```

Existem dois tipos de propriedades:

1. *datatype properties* - relações entre uma instância de uma classe e um literal ou tipo de dados XML Schema[11]
2. *object properties* - relações entre instâncias de duas classes

O código apresentado em seguida é a continuação da caracterização da classe criada, com as propriedades `progenitor` e `raça`.

```
<owl:ObjectProperty rdf:ID="progenitor">
  <rdfs:domain rdf:resource="#Gato" />
  <rdfs:range  rdf:resource="#Gato" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="raça">
  <rdfs:domain rdf:resource="#Gato" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

As propriedades têm tipos diferentes. Como se pode ver nos campos `rdfs:domain` e `rdfs:range`, a primeira relaciona duas instâncias da mesma classe, enquanto que a segunda tem um valor do tipo *string* para cada instância de `Gato`.

Em OWL Lite, as propriedades podem ter um tipo atribuído:

- inverseOf - Para expressar que uma propriedade é o inverso de outra. (Exemplo: progenitorDe, filhoDe).
- TransitiveProperty - Transitividade de uma propriedade P: Se $P(x, y)$ e $P(y, z)$ então $P(x, z)$ (Exemplo: antepassadoDe).
- SymmetricProperty - Propriedade simétrica: ao verificar-se $P(x, y)$ verifica-se também $P(y, x)$.
- FunctionalProperty - Propriedade que só pode tomar um único valor (por instância).
- InverseFunctionalProperty - Propriedade que só pode tomar um único valor e tem uma inversa que é *FunctionalProperty*.

[11] http://www.w3.org/XML/Schema

2.2.2 Instâncias

As instâncias de uma classe, por vezes chamados de *individuals*, são a representação de um individuo ou objecto particular daquela classe.
Consideremos o seguinte exemplo:

```
<Gato rdf:ID="garfield">
  <progenitor rdf:resource="#paiDoGarfield" />
  <raça rdf:datatype="&xsd;string">persa</raça>
</Gato>
```

Isto representa uma instância da classe `Gato`, identificada por `garfield`, cujo progenitor é igualmente uma instância desta classe, identificada por `paiDoGarfield`, e a sua raça está descrita com a *string* "persa".

Os exemplos apresentados ilustram algumas funcionalidades básicas da linguagem OWL, com as quais é possível desenvolver aplicações. Para uma descrição exaustiva da linguagem recomenda-se a leitura dos documentos referidos ao longo do texto.

2.2.3 Ferramentas

No contexto Semantic Web, surgiram vários tipos de ferramentas:

- agentes para recolher objectos de uma ontologia
- editores para facilitar a codificação de classes e propriedades
- visualizadores gráficos para pesquisa de uma ontologia
- browsers
- parsers
- ferramentas de interrogação

Existem várias ferramentas para a manipulação de ontologias que são dedicadas a esta linguagem, ou então já existiam e sofreram uma actualização para o suporte de OWL. Jena Toolkit é uma dessas ferramentas, integrando API para o processamento de ontologias RDF e Daml+OIL, e que recentemente apareceu renovada com uma versão 2.0 beta em que a API está pensada também para a

linguagem OWL. Esta ferramenta foi desenvolvida pelo *HP Labs Semantic Web research group*[12], e inclui funcionalidades como:

- API para construir ou fazer o parse a uma ontologia em RDF, DAML+OIL ou OWL
- RDFS rule reasoner
- RDF Query Language
- uso de bases de dados PostgreSQL ou Oracle para representar modelos persistentes

O primeiro contacto com a linguagem OWL faz-se normalmente através de experiências com pequenos blocos de código. Para verificar a correcção desse código, desde problemas de sintaxe nas tags a erros de tipos na referenciação de objectos, Dave Rager construiu uma ferramenta inspirada no DAML Validator[13] e no W3C RDF Validation Service[14] que usa uma versão modificada do Jena Toolkit para validação de código OWL. O OWL Validator[15] encontra-se ainda em desenvolvimento para suportar OWL Full, mas pode já utilizar-se através de:

- interface web: Através de um formulário, o utilizador indica o URL da ontologia ou bloco de código a validar.
- aplicação: A validação efectua-se localmente, com uma aplicação em Java.
- API: É fornecida uma API em Java que pode ser integrada noutra aplicação, para automatizar a verificação do código.

[12] http://www.hpl.hp.com/
[13] http://www.daml.org/validator/
[14] http://www.w3.org/RDF/Validator/
[15] Este utilitário OWL é também conhecido como vOWLidator e está disponível em: http://phoebus.cs.man.ac.uk:9999/OWL/Validator

2.3 Recuperação de Informação

Desde 1940 que a questão do armazenamento e necessidade de procurar informação tem merecido cada vez mais atenção dos investigadores. Numa abordagem muito simplista, o processo é simples de compreender.
Consideremos uma biblioteca, com livros relacionados com as mais variadas áreas de conhecimento, e um utilizador com necessidade de informação sobre um detalhe muito específico de determinado assunto. O utilizador poderia ler todos os livros da biblioteca, retendo os capítulos relacionados com o assunto desejado.
Este é um processo de RI, contudo é impraticável, porque o utilizador não aceitava e provavelmente não conseguia ler todos os livros.

Com a utilização dos computadores o processo passou a efectuar-se mais rapidamente e com maior eficiência. Por outro lado, a tecnologia levou ao crescimento exponencial do volume de dados disponível para consulta, de modo que a problemática da recuperação de informação persiste.

Tudo começa com uma interrogação. Uma interrogação é composta por um conjunto de palavras que caracterizam a informação que o utilizador pretende. Um dos conceitos fundamentais em RI é a relevância de um documento. Um documento é relevante se satisfaz a necessidade de informação do utilizador.
Relacionado com a relevância, existe o conceito de *recall*[16], definido como:

$$recall = \frac{n^{o}\ de\ documentos\ relevantes\ seleccionados}{n^{o}\ total\ de\ documentos\ relevantes} \tag{2.1}$$

O propósito na recuperação automática de informação é encontrar todos os documentos relevantes para determinada interrogação, ou maximizar o *recall*, evitando os erros, isto é, deve reduzir-se o número de documentos não relevantes seleccionados ao menor número possível, o que corresponde a aumentar a precisão.

$$precisão = \frac{n^{o}\ de\ documentos\ relevantes\ seleccionados}{n^{o}\ total\ de\ documentos\ seleccionados} \tag{2.2}$$

Um humano reúne as condições intelectuais para verificar a relevância de um documento relativamente a uma interrogação. Para um computador efectuar essa operação é necessário construir um modelo em que as decisões sobre a relevância se possam quantificar. Muito do trabalho de investigação em RI está relacionado

[16]Por vezes traduzido para português como *abrangência*.

com vários aspectos para a construção de tal modelo.

2.3.1 Sistemas de Recuperação de Informação

Um Sistema de Recuperação de Informação (SRI) pode ser visto como uma "caixa negra", em que existe um *input*, que dá lugar a um processamento, do qual resulta um *output*.

Relativamente aos elementos de *input*, eles são um conjunto de documentos e uma interrogação do utilizador.
É necessário encontrar uma representação, tanto para os documentos como para a interrogação, que seja adequada para o processamento. Esta representação é normalmente obtida pela aplicação de técnicas de processamento de língua natural, e outros processos de Inteligência Artificial, sobre o texto dos documentos e da interrogação. Por vezes aplicam-se métodos estatísticos sobre o texto, com o propósito de encontrar as palavras mais frequentes, pois entre elas podem estar as palavras-chave mais relevantes. Alguns destes processos são descritos na secção 2.3.2.

A parte do processamento de uma interrogação tem a ver com a procura de informação ao longo dos documentos. Este processo da recuperação de informação pode envolver vários algoritmos de integração e classificação dos dados, no sentido de melhorar a eficácia do sistema, tornando o cálculo da relevância dos dados tão preciso quanto possível.

O *output* de um SRI consiste num conjunto de citações ou documentos em que foi encontrada informação sobre a interrogação processada.

Os SRI podem ainda ser classificados como:

- convencionais: baseados numa pesquisa de informação precisa ou *exact-match*. Procuram documentos descritos exactamente pelos termos da interrogação.

- não convencionais: para encontrar os seus resultados procuram uma "boa aproximação" ou *best match*.

Para além desta classificação, os SRI podem utilizar diferentes modelos para representar e procurar informação sobre os documentos. A próxima secção descreve os principais métodos de RI.

2.3.2 Métodos de RI

Recuperação de informação é uma área complexa, onde nem sempre se conseguem obter resultados perfeitos. A eficácia do processo depende, por exemplo, da representação e estrutura escolhidas para os dados, dos algoritmos utilizados e de outros aspectos.
Existem alguns modelos em RI que são tomados como referência:

1. Modelo *String Search*
2. Modelo Booleano
3. Modelo *Vector Space*
4. Modelo Probabilístico

Em seguida apresentam-se as principais características destes modelos.

Modelo *String Search*

O primeiro modelo, tal como o nome indica, assenta na pesquisa de expressões. É também conhecido como Modelo de Pesquisa de Padrões, ou ainda por Modelo de Pesquisa Directa.
Neste modelo, a interrogação é uma expressão, com uma ou mais palavras, que o utilizador introduz. A identificação dos documentos relevantes é baseada na pesquisa de padrões ou expressões regulares, do seguinte modo:

> Dado um texto T e um padrão P, encontrar a posição da primeira ocorrência de P em T, ou indicar que não existe, se P não ocorre em T.

Aplicam-se algoritmos para efectuar a pesquisa da expressão, dada pelo utilizador, directamente sobre o texto original dos documentos.
Em cada documento haverá N ocorrências da expressão, ou nenhuma. Com o resultado da pesquisa, é possível seleccionar os documentos em cujo texto ocorre, com maior frequência, o padrão da interrogação do utilizador.

Este processo é muito utilizado em aplicações, como editores de texto e alguns utilitários de busca, para encontrar ocorrências de uma expressão ao longo de um texto.
Vejamos uma lista, não exaustiva, de vários algoritmos de pesquisa de *strings* (considerando a pesquisa de $s1$ em $s2$, com comprimento N e M, respectivamente):

1. *Naive*: efectua a pesquisa mais ingénua. No limite pode precisar de $(M - 1) * N$ verificações.

2. Knuth-Morris-Pratt: usa a informação dos caracteres em $s1$ para calcular quanto deve avançar depois de uma verificação sem sucesso. No pior caso requer $N + M$ comparações.

3. Boyer-Moore: um algoritmo mais eficiente, em que a pesquisa se efectua da direita para a esquerda em $s2$, e da esquerda para a direita em $s1$. No pior caso podem ser necessárias $N + M$ comparações, mas em média serão bastante menos.

Este modelo assenta em pesquisa de palavras em *exact-match*. Isto significa que um texto com a mínima diferença relativamente às palavras da interrogação não será seleccionado para o conjunto dos documentos tidos como relevantes.

Modelo Booleano

No Modelo Booleano, um documento é visto como um conjunto de palavras, tal como uma interrogação. Sobre vários conjuntos de palavras é possível efectuar um conjunto de operações como a união e intersecção. Os conjuntos podem ser definidos com expressões booleanas AND, OR ou NOT (o que originou o nome do modelo).

Quando o utilizador define uma interrogação, é construído um conjunto cujos elementos são precisamente as palavras da interrogação. Então, inicia-se um processo de busca de documentos com intersecção não vazia com este conjunto. A lista de documentos recolhidos pode ser ordenada por ordem decrescente da cardinalidade da intersecção. Os primeiros documentos da lista serão os que mais têm a ver com a interrogação.

Como exemplo de interrogação, consideremos o caso em que o utilizador pretende documentos sobre exames de matemática ou física. A expressão booleana seria:

$$exame \text{ and } (matematica \text{ or } fisica)$$

Este modelo tem a vantagem de ser flexível ao ponto de ser o utilizador a controlar a pesquisa através da expressão que define. Contudo, uma interrogação mais complexa pode baralhar um utilizador menos habituado a expressões booleanas. Ao nível dos resultados deste processo, não é possível construir um *ranking* rigoroso e detalhado da relevância dos documentos, apenas pelos resultados da intersecção.
Por outro lado, como este é um processo convencional em que se procuram precisamente aquelas palavras em *exact-match*, podem existir documentos relevantes que não são seleccionados pelo processo por não terem a intersecção esperada. A propósito desta questão, existe uma variante conhecida como Modelo Booleano Estendido, que atribui diferente importância às palavras do texto, na tentativa de obter melhores resultados. Neste caso a ordenação final poderia ser por ordem decrescente sobre a soma do peso das palavras na intersecção.

Modelo *Vector Space*

O modelo *Vector Space* assenta no conceito de espaço, com N dimensões, onde existe informação. Cada documento é representado por um vector, no espaço, de acordo com as palavras encontradas no texto.
Primeiro, o texto é analisado, são eliminadas as *stopwords*[17]. As palavras restantes são normalizadas e, pela sua frequência no texto, é-lhes atribuído um peso que indica o seu grau de importância no texto. O peso varia entre 0 e 1, consoante a palavra seja menos ou mais importante para aquele documento.

Depois de aplicar o processo a todos os documentos, resulta um conjunto de vectores do tipo:

Doc1= { information(0), retrieval(0.3), system(0.4)... }

Doc2= { information(0), retrieval(0), system(0.8)... }

O processo é aplicado também às interrogações. O conjunto de todos os documentos e interrogações é então representado num espaço euclidiano com várias

[17]Palavras usadas para ligar frases ou expressões, como artigos, pronomes e conjunções (e, de, os, a, para...). Por serem demasiado frequentes têm pouco valor semântico e como tal não são consideradas em processos de pesquisa.

dimensões. Cada dimensão diz respeito a um conceito ou termo encontrado no texto dos documentos. O peso de cada termo é usado como coordenada para aquela dimensão. A figura 2.4 mostra uma representação dos documentos como vectores no espaço.

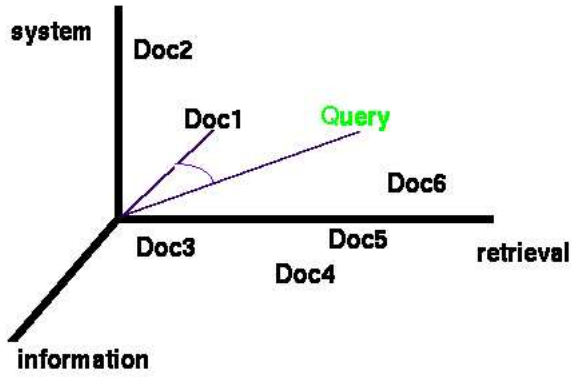

Figura 2.4: Documentos e Interrogação no Modelo *Vector Space*

A selecção dos documentos relacionados com a interrogação depende da proximidade, no espaço, dos vectores que os representam. Isto pode ser matematicamente calculado, por exemplo pelo ângulo entre os vectores.

Modelo Probabilístico

Este modelo foi apresentado inicialmente por *Maron and Kuhns* [MK60], em 1960. Como o nome sugere, este modelo lida com cálculo probabilístico, e a selecção de documentos faz-se calculando a probabilidade de cada documento ser relevante para a interrogação.

O cálculo envolve também alguma estatística, relacionada com a distribuição dos termos pelo documento. Usualmente é aplicado o modelo matemático *Bayesiano* para essas operações (o que torna este modelo também conhecido como *Bayesiano*).

À partida há várias assunções a considerar:

- um documento é encarado como um conjunto de termos
- a relevância é binária

- a relevância de um documento é independente da relevância dos restantes documentos
- os termos são independentes uns dos outros

Primeiro efectua-se uma análise estatística sobre todos os documentos, recolhendo informação sobre a frequência dos termos no texto. Estes dados são usados para calcular o grau de semelhança entre cada documentos D e a interrogação Q, de onde se conclui um dos casos:

- REL: o documento D é relevante para Q
- $notREL$: D não é relevante para Q

Assim, o objectivo é calcular $P(REL|D)$, a probabilidade de um documento ser relevante, dado o seu conteúdo. Esta probabilidade é calculada a partir da probabilidade de relevância de cada um dos termos de D, relativamente aos termos de Q. Este último cálculo depende dos resultados estatísticos antes calculados.

Os documentos podem ser ordenados por ordem decrescente da respectiva probabilidade de relevância face à interrogação. Esse será o resultado do sistema.

2.3.3 Ontologias e SRI

Como meio estruturado capaz de representar informação semântica, as ontologias podem ser utilizadas no âmbito de Sistemas de Recuperação de Informação com o objectivo de melhorar os resultados. A existência de relações hierárquicas, e não só, sobre a semântica da informação poderá potenciar uma melhoria da eficiência nos SRI.

Capítulo 3

Trabalho Relacionado

O presente capítulo faz uma síntese do trabalho relacionado com a área desta dissertação.
A secção 3.1 é dedicada a ontologias na Web, como podem ser usadas para responder a interrogações (3.1.1) e exemplos de ontologias verticais sobre domínios concretos (3.1.2).
Na secção 3.2 são apresentados dois géneros de processos, manuais (3.2.1) e semi-automáticos (3.2.2), para a construção de ontologias, incluindo alguns exemplos reais.

3.1 Ontologias na Web

Actualmente, a Web contem um gigantesco volume de dados, distribuídos por biliões de páginas. Isto constitui um meio de troca de informação, um pouco por todo o mundo.
A esmagadora maioria das páginas é formada apenas por texto em língua natural, o que conduz à procura de palavras-chave para encontrar resposta a uma interrogação. Na prática este é o processo adoptado, ainda que os resultados sejam considerados como pouco eficientes. A alternativa é desenvolver algum trabalho na análise dos textos.

Com a visão de Tim Berners-Lee, de uma nova Web [BLF99], a informação textual é complementada com meta-informação semântica. Neste paradigma, os serviços baseados na Web podem então ser redesenhados e automatizáveis. Em particular, a procura de informação entra numa nova época.

3.1.1 Interrogação baseada em Ontologias

Em [Ome01], Borys Omelayenko descreve um esquema para interrogação semântica em documentos Web, identificando dois tipos de ontologias e alguns algoritmos de aprendizagem.

Primeiro, o utilizador formula uma interrogação em linguagem natural. Essa expressão é então transformada para uma representação mais formal, através de dois tipos de ontologia:

1. *natural language ontology*: ontologia que representa as relações lexicais entre os conceitos da linguagem. É um tipo de ontologia horizontal, onde se procura representar todos os conceitos possíveis, sem uma descrição muito detalhada.

2. *domain ontology*: ontologia dedicada aos conceitos de uma área específica. Neste tipo de ontologia, os conceitos têm uma descrição completa, relativa ao contexto em que se inserem. Trata-se de uma ontologia vertical.

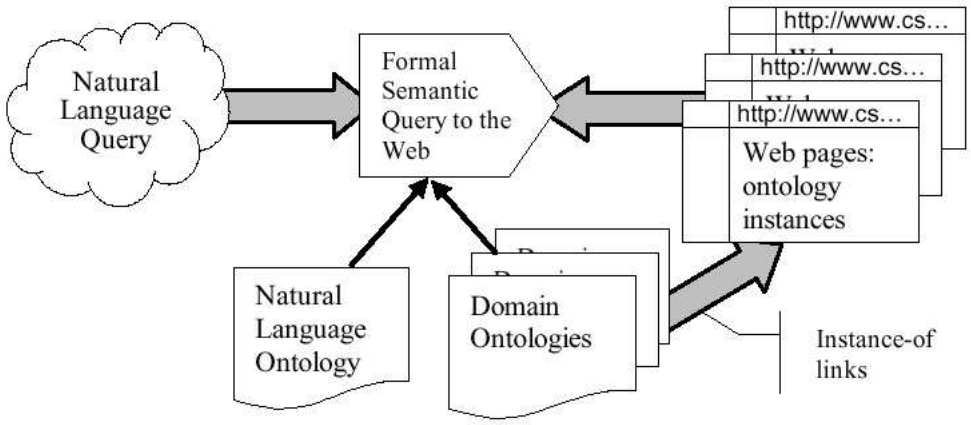

Figura 3.1: Interrogação semântica a documentos Web

Os documentos Web contém algumas instâncias, mais ou menos completas, da *domain ontology*. Para encontrar a resposta para o utilizador, o sistema deve mapear os conceitos da interrogação, expressos em instâncias da ontologia, em instâncias encontradas nos documentos. Este mapeamento não é trivial e pode envolver a inferência sobre a *domain ontology*. A figura 3.1 ilustra os passos do processo.

A ontologia de língua natural é utilizada para uma primeira interpretação da interrogação do utilizador, que terá de ser adaptada para a ontologia específica relacionada com o assunto em causa. Dentro desta *domain ontology*, usualmente construída à mão, torna-se possível estabelecer uma relação entre a interrogação e os documentos.
Constata-se que as ontologias horizontais estão mais desenvolvidas que as verticais, que possuem maior complexidade e relações entre os conceitos.

3.1.2 Desenvolvimento de Ontologias Verticais

Uma ontologia vertical é normalmente associada a uma área específica, descrevendo os conceitos desse domínio com algum detalhe. Existem ontologias para as mais variadas áreas, como biologia, vinicultura, medicina ou comércio electrónico. Em seguida são enunciados alguns casos de aplicação de ontologias em domínios concretos, com especial ênfase na área relacionada com esta dissertação.

Biologia

Nos últimos anos, a quantidade de informação no campo da biologia tem aumentado a um ritmo elevado. Com as novas tecnologias é agora possível a realização de mais experiências e todos os dias se fazem avanços. Por um lado, a base de conhecimento da área cresce bastante, por outro lado, uma só experiência pode envolver um enorme volume de dados, por exemplo sobre moléculas.

A necessidade de um sistema em que se pudesse modelar a base de conhecimento da área, para depois aplicar em tarefas como o tratamento de dados de novas experiências, foi a motivação de Baker et al. para a construção de uma ontologia na área de biologia [BGB+99].
Para representar a ontologia foi utilizada uma linguagem do tipo *Description Logics*, da área de representação de conhecimento, que permite asserções sobre classes de conceitos ou relações existentes, restrições e inferência.
A ontologia foi criada no âmbito do projecto *Transparent Access to Multiple Biological Information Sources*[1] (TAMBIS) e constitui uma plataforma sobre a qual podem existir vários serviços de análise e de resposta para complexas interrogações.

RiboWeb [ABC+99] é um sistema *online* baseado em ontologias para desenvolver trabalho colaborativo na área de biologia molecular. O sistema contem uma grande base de conhecimento, obtida designadamente a partir de publicações, que é

[1] http://img.cs.man.ac.uk/tambis

usada para testar hipóteses sobre a estrutura do *ribosome*. O *ribosome* é o "local" onde ocorre o fenómeno biológico de tradução do código genético em moléculas de proteína, essenciais à vida. É como um grande complexo molecular, formado por mais de 57.000 elementos e que usa o código genético DNA nos processos biológicos em estudo por este grupo de investigadores. Era necessário representar a informação de um modo estruturado. E nesse sentido foram criadas quatro ontologias:

- *physical-thing ontology*: para representar os componentes moleculares do *ribosome*
- *data ontology*: descreve o tipo de dados recolhidos em experiências no campo da biologia
- *reference ontology*: descreve as classes de referências onde o sistema recolhe informação
- *methods ontology*: sobre as tarefas que o RiboWeb pode desempenhar e os atributos de cada uma

A figura 3.2 mostra a relação entre as ontologias que servem de base ao funcionamento do sistema.

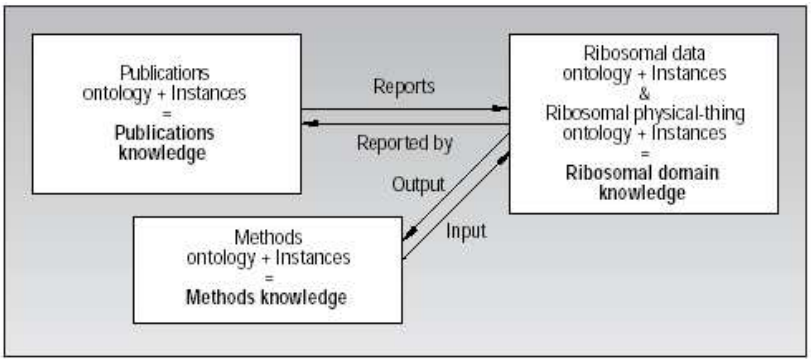

Figura 3.2: Utilização de ontologias no RiboWeb

Medicina

O artigo [GFH+03] mostra um caso de utilização de ontologias no ramo da medicina. O aumento da investigação na prevenção e tratamento do cancro tem levado ao surgimento de novas descobertas, envolvendo informação sobre novos elementos. O National Cancer Institute, na América, efectua actualizações mensais ao seu léxico. Em Fevereiro de 2003 existiam 26.000 conceitos, divididos por 24

taxonomias.
Cada taxonomia, definida por um grupo de especialistas, é depois convertida para OWL Lite. A ontologia final inclui conceitos e relações semânticas no campo da medicina, incluindo genes, doenças, químicos, anatomia, organismos e proteínas.

Ontologias na área jurídica

O trabalho descrito nesta dissertação é aplicado a um conjunto de documentos da área jurídica, uma área bastante complexa, onde também existem trabalhos no sentido da construção de ontologias.
Em trabalho recente de Benjamins et al. [BCC+03], é apresentada uma aplicação de ontologias na área jurídica. O objectivo do trabalho é auxiliar jovens juízes, em Espanha, na tomada de decisões através de um sistema de apoio baseado em ontologias.

Primeiro foi recolhida informação acerca das dificuldades encontradas por alguns juízes, a partir da qual especialistas da área elaboraram um conjunto de perguntas para uma lista de *frequently asked questions* (FAQ). Através desta FAQ, os autores pretendem representar um género de informação muito específico, que designam de *professional legal knowledge* (PLK). O PLK está associado a este grupo de profissionais, depende da experiência e da formação.
A ontologia para o PLK é baseada no conhecimento dos juízes e de documentação sobre casos e interpretações. O conceito mais geral é *proceso*, patente na figura 3.3, juntamente com conceitos relacionados, de acordo com o sistema legal espanhol.

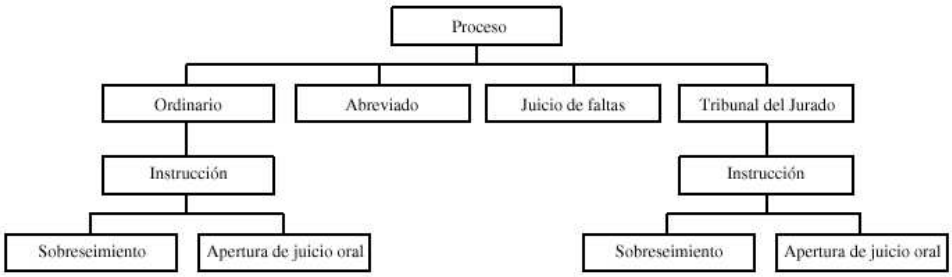

Figura 3.3: Parte da ontologia do PLK

A ontologia, definida por especialistas depois de consultarem diversas fontes, foi tecnicamente construída através da ferramenta Protégé (a utilização desta e outras ferramentas será descrita na secção 3.2.1).
Para interacção com o utilizador, foi criada uma aplicação Web, que recebe interrogações em língua natural e responde com um conjunto de pares pergunta/resposta

relacionados, obtidos da FAQ, que está representada numa ontologia.

Guiraude Lame já antes apresentara um trabalho de construção de uma ontologia para a área jurídica [Lam01]. Este trabalho está associado a um *site* francês de legislação com documentos do *Journal officiel de la République française édition lois et décrets*.
O objectivo é conseguir uma ontologia com termos jurídicos para apoio a RI, em dois sentidos:

1. expansão textual: através de uma ferramenta de consulta, quando um utilizador introduz uma palavra é-lhe sugerida uma lista de termos associados, permitindo ao utilizador refinar a interrogação.

2. classificação: identificando os termos de um texto na ontologia é possível efectuar uma classificação temática do documento.

A ontologia foi desenhada numa abordagem *bottom-up*. Para seleccionar os termos da ontologia foram utilizadas técnicas de PLN. As relações entre os termos foram inferidas através do contexto em que se encontram. A ideia é que dois termos similares não têm que surgir necessariamente juntos, mas antes em contextos idênticos, por exemplo, rodeados pelas mesmas palavras. Os resultados obtidos carecem de avaliação por pessoas especializadas.
Posteriormente seria ainda necessário materializar a ontologia, passando do modelo teórico para uma efectiva representação numa linguagem Semantic Web, que pode então ser consultada por aplicações.

Um dos objectivos do *E-Government* é dotar os cidadãos de meios de acesso a serviços e informação governamental. Este processo depende muito da legislação vigente, e como tal, obriga a uma rápida capacidade de adaptação face à alteração ou introdução de novas leis.
Neste contexto, em 1999 surgiu o *Program for an Ontology-based Working Environment for Rules and regulations*, também conhecido como POWER. O programa envolve vários países e desde Setembro de 2001 é apoiado pela União Europeia, altura em que passou a designar-se E-POWER[2]. Nos últimos anos, os países da União Europeia têm vindo a adoptar medidas de convergência para uma legislação comum. O programa E-POWER, envolvendo vários países membros, apresenta-se como uma plataforma que pode também contribuir para esse objectivo.

[2]http://lri.jur.uva.nl/ epower/index.html

Uma das fases do projecto, designada *POWER-method*, tem a ver com a formalização da captura e tradução de informação jurídica, das fontes em que se encontra até aos modelos formais ou *POWER-models*. Estes modelos formais permitem identificar problemas na fonte de informação (inconsistências ou circularidades), simular o efeito de uma nova lei e, de um modo geral, suportar o processo de desenvolvimento de sistemas, ferramentas ou componentes que assentam numa base de conhecimento jurídica. A figura 3.4 ilustra o papel central do *POWER-model*.

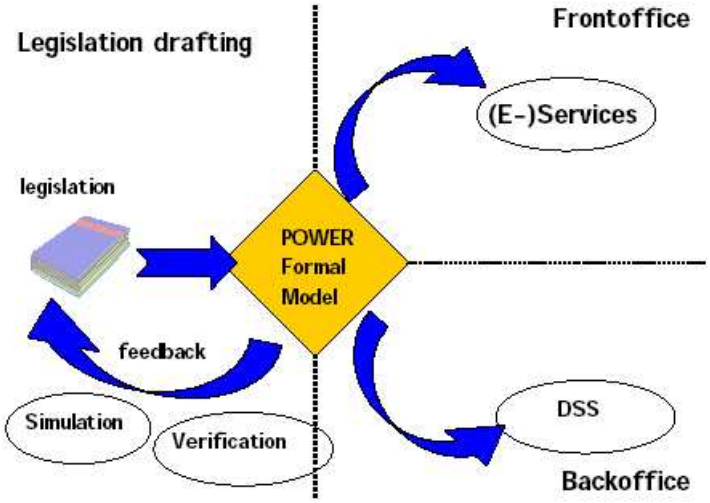

Figura 3.4: E-POWER: o modelo formal

Apesar dos benefícios para a qualidade do trabalho nesta área, a utilização do *POWER-method* pode requerer mais tempo que o disponível por uma entidade legisladora, ou outro *POWER-user* (a designação dos utilizadores do método), quando urge a tomada de uma decisão.
Em 2002, Tom van Engers e Radboud Vanlerberghe apresentaram uma versão simplificada para o método, o *POWER-light version* [vEV02]. Esta abordagem compreende dois passos:

1. *Structure analysis*: é o primeiro passo efectuado e consiste numa análise à estrutura dos documentos jurídicos. Tipicamente, este tipo de documentos possui uma estrutura fixa, formada por campos de diferente significado. A análise pode detectar atempadamente a falta de algum elemento, por lapso. Por outro lado, é usual encontrar referências a outros documentos ou secções. Estas referências são também analisadas.
   Deste modo é possível efectuar uma rápida verificação à estrutura do documento, ajudando a prevenir erros que só mais tarde seriam detectados.

2. *Domain modeling*: análise ao conteúdo do texto jurídico. Nos casos em que, por exemplo, várias pessoas trabalham partes individuais de uma nova legislação, o responsável pela compilação final é obrigado a um trabalho de edição para evitar, por exemplo, repetições.
   Esta verificação é um precioso auxílio para identificar uma eventual inconsistência ou redundância no documento que está em preparação.

Apesar de não contemplar algumas funcionalidades mais complexas do modelo original (como a simulação e suporte a outros serviços sobre a base de conhecimento jurídica), o *POWER-light version* constitui um mecanismo de apoio indicado sobretudo para os casos em que existe pouco tempo disponível.

O crescente movimento de cidadãos entre os países da Europa e o aumento das empresas que oferecem produtos e serviços em várias zonas jurisdicionais, para além de justificar uma política de convergência da União Europeia, requer a análise e comparação da legislação de cada país.
No caso de uma empresa, tem de haver garantias de que os seus produtos ou serviços estão em conformidade com a lei vigente em cada país. Ao desenvolver um novo produto, a empresa pode ter interesse em obter uma lista de países com legislação semelhante (na área fiscal, de segurança ou outra), para efectuar aí o seu negócio.

Em [BvEW03], os autores apresentam uma proposta de utilização do modelo E-POWER no sentido de comparar a legislação de vários países, detectando semelhanças. É apresentado o caso concreto da *Dutch Tax and Customs Administration* (DTCA), onde legislação e processos de negócio são modelados com *Unified Modeling Language* (UML) e ontologias, através das quais foi já possível conceber uma plataforma de análise eficiente à legislação (sobretudo na fase de preparação de novas leis). Neste trabalho, os autores estudam a possibilidade de utilizar as ontologias existentes para detectar a similaridade de legislação em diferentes jurisdições.

Os problemas encontrados têm a ver com pequenas diferenças na "cultura jurídica" dos países, que tornam difícil o relacionamento entre as ontologias jurídicas de cada um. Para ultrapassar esta barreira foram definidas algumas normas que sugerem a resolução para alguns destes casos.

A ideia está representada no esquema da figura 3.5. Para além da utilidade, já mencionada, de encontrar as semelhanças legais, a identificação de diferenças de

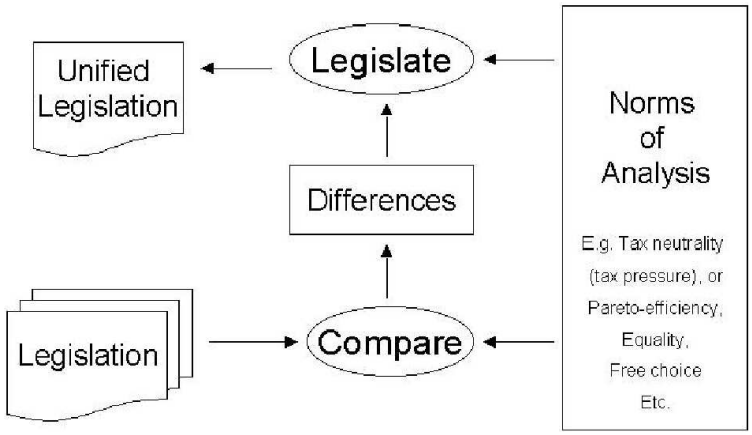

Figura 3.5: E-POWER: comparação e convergência de legislação

fundo na legislação de cada país membro pode ser um primeiro passo, identificando o que carece de actualização e assim contribuindo para o objectivo inicial de uma legislação comum.
Foram efectuadas algumas experiências, onde se verificou que a quantidade de informação necessária para explicar a diferença entre dois regulamentos é potencialmente enorme. O trabalho de análise comparativa baseada em ontologias continua, com o propósito de aperfeiçoar todo o processo.

3.2 Construção de Ontologias

Até aqui vimos alguns exemplos de utilização de ontologias como meio de representar informação de um modo estruturado e com alguma semântica, através de propriedades e relações que caracterizam os conceitos representados.
Esta secção é dedicada ao modo como efectivamente se pode construir a ontologia usando uma linguagem Semantic Web.

3.2.1 Processos Manuais

Na maioria dos casos, o responsável pela construção da ontologia tem os dados, previamente preparados, sobre um conjunto de conceitos e suas características, organizados hierarquicamente.
A preparação prévia consiste na selecção dos conceitos a incluir, como os descrever e quais as relações que se estabelecem entre eles. Esta fase é usualmente baseada

em estudos e/ou longas reuniões com especialistas (linguistas ou especialistas no assunto em causa).

Para dar corpo à ontologia, em Semantic Web, basta utilizar uma linguagem, como DAML+OIL ou OWL para, representar aqueles conceitos. Esta tarefa pode efectuar-se à mão, usando apenas um editor de texto, quando se conhece bem a sintaxe da linguagem. Por este processo ser repetitivo (e cansativo), começaram a surgir ferramentas para desenhar a ontologia, que facilitam a obtenção da versão final, em Semantic Web.

Em 2002, Huynh et al. apresentaram a ferramenta Haystack [HKQ02]. O Haystack foi pensado para ajudar os utilizadores a descrever os seus documentos com meta-informação, incluindo propriedades como o tema e o autor. Para além disso, permite expressar explicitamente que vários documentos estão relacionados.
Esta ferramenta é baseada em *views* sobre os documentos e possui uma interface gráfica que esconde a complexidade da sintaxe RDF.

SMORE [KPHG03] é outra ferramenta para gerar *markup* semântico e ontologias, que permite, por exemplo, anotar uma página *web* ou um *e-mail*. Inclui uma interface gráfica, através da qual é possível construir, alterar e estender ontologias em RDF.
Outra interessante funcionalidade presente é a inferência de factos a partir das asserções introduzidas pelo utilizador. Se existir uma propriedade `joga`, definida da classe `Atleta` para a classe `Desporto`, e o utilizador introduzir o triplo (`Sandra - joga - ténis`), o programa vai automaticamente deduzir que `Sandra` e `ténis` são instâncias de `Atleta` e `Desporto`, respectivamente.
Esta ferramenta inclui ainda um "portal virtual semântico" onde o utilizador pode efectuar pesquisas acerca da semântica dos documentos.

Uma das aplicações mais utilizadas em projectos relacionados com Semantic Web (por exemplo no trabalho já citado em [BCC+03]) é o Protégé[3]. Esta ferramenta apresenta uma vasta gama de funcionalidades, disponibilizadas por uma interface gráfica sobre Java. Trata-se de uma ferramenta bastante completa, incluindo opções mais avançadas que vão para além do necessário para um utilizador comum. Um dos pontos fortes do Protégé é a possibilidade de trabalhar com várias linguagens, incluindo RDF e OWL.

[3]http://protege.stanford.edu/

Para mostrar como se pode construir uma ontologia com o auxílio de uma destas ferramentas, vamos aplicar o Protégé a um pequeno exemplo.
Suponhamos que pretendemos representar a classe `Cidadão`, tendo como subclasses `Civil` e `Militar`. Para criar a classe `Cidadão` basta utilizar um dos botões do painel, que conduz a uma janela onde se pode definir o nome da classe e eventuais propriedades. A figura 3.6 mostra a a definição de uma propriedade `idade`, do tipo inteiro, que pode ser definida uma só vez para cada instância e cujo valor mínimo é zero. As classes `Civil` e `Militar` são construídas a partir da sua superclasse,

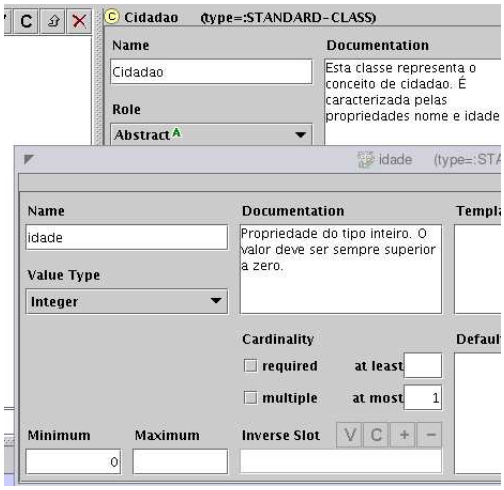

Figura 3.6: Protégé: definição de uma propriedade da classe

como mostra a figura 3.7.

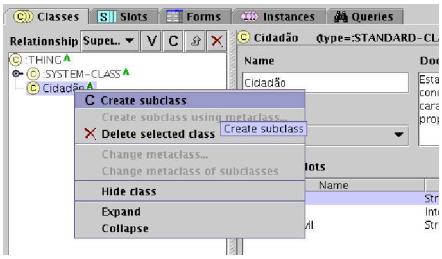

Figura 3.7: Protégé: definição de subclasse

Na classe `Militar` podemos acrescentar as propriedades `ramo` e `posto`, para indicar qual o ramos das Forças Armadas e a patente, respectivamente, de cada instância desta classe. A listagem das propriedades desta classe está na figura 3.8, onde se podem distinguir as que são herdadas (a branco). Note-se a restrição sobre

a propriedade `ramo`, que deve estar associada a uma subclasse de `RamoMilitar` (classes já criadas).

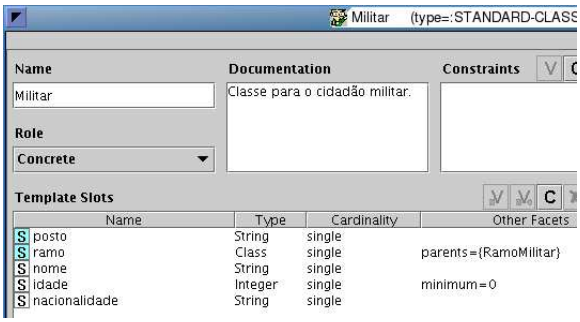

Figura 3.8: Protégé: listagem das propriedades de uma classe

Definida a hierarquia classes, o Protégé permite ainda adicionar as instâncias das classes, através de um formulário com os campos correspondentes às propriedades antes criadas. A figura 3.9 mostra uma instância da classe `Militar`.

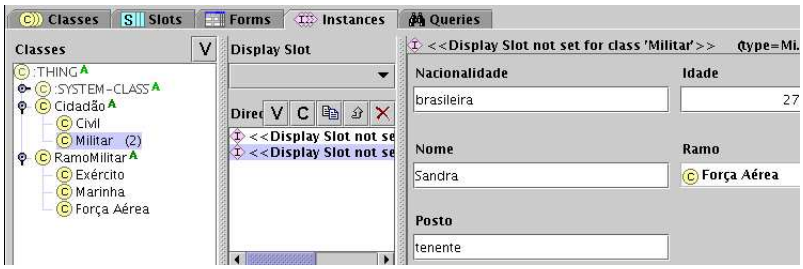

Figura 3.9: Protégé: instâncias para a classe Militar

Sobre os dados inseridos é ainda possível efectuar interrogações, com a ajuda da interface gráfica. A figura 3.10 apresenta uma interrogação em que se procuram todas as instâncias de `Cidadão` com `idade` superior a 18 anos. O resultado aparece do lado direito, com duas ocorrências encontradas. Para terminar a descrição da ferramenta, importa salientar que o utilizador pode guardar classes e instâncias numa das várias linguagens suportadas, incluindo RDF e OWL.

3.2.2 Processos Semi-Automáticos

Esta secção é dedicada aos processos que minimizam a intervenção humana na construção de ontologias. A construção, desde o zero, de uma ontologia de classes é

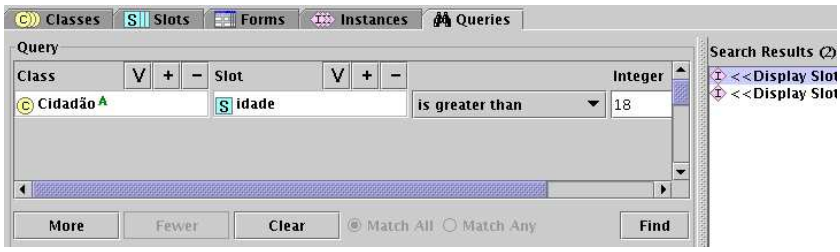

Figura 3.10: Protégé: interrogação sobre as instâncias

um processo complicado que dificilmente dispensa a ajuda do utilizador. Veremos alguns trabalhos com diferente grau de automatização, dependendo do objectivo e dos dados auxiliares.

Alani et al. desenvolveram um trabalho de pesquisa de informação sobre artistas na Web [AKM+02], que envolve ontologias. O objectivo é mostrar ao utilizador uma biografia, gerada automaticamente, sobre o artista escolhido.
Para gerar a biografia, o sistema desenvolvido baseia-se em dados recolhidos na Web. São escolhidos alguns documentos para o processo de extracção de conhecimento. Este processo começa com uma análise sintáctica, após a qual se efectua uma selecção de termos relevantes para a base de conhecimento.
A selecção dos termos é baseada numa ontologia (de classes), existente à partida, que dá informação sobre os conceitos e tipo de dados a extrair. O resultado da análise sintáctica é relacionado com a ontologia, com recurso à base de dados lexical WordNet[4], por exemplo para identificar um termo no texto que é sinónimo de um conceito na ontologia.
Os termos seleccionados, se estiverem de acordo com alguma relação na ontologia podem ser usados para criar instâncias, que constituem uma parte da biografia pretendida. Neste caso, a automatização encontra-se ao nível da instanciação da ontologia, o que os autores qualificaram de *Automatic Ontology Population*.

Kietz et al. desenvolveram um método semi-automático para obtenção de uma ontologia a partir da intranet de uma empresa [KMV00]. No caso apresentado pretendia-se construir uma ontologia para o universo de uma companhia de seguros.
O ponto de partida é uma ontologia genérica e predefinida. Em seguida essa ontologia é estendida com novos conceitos, oriundos de um processo de busca nos documentos em língua natural da intranet e com recurso a um dicionário. A ontologia resultante é então alterada através de um processo de *pruning*, eliminando

[4]http://www.cogsci.princeton.edu/ wn

os conceitos que não pertencem ao domínio (seguros, no caso).

Já em 2003, Guiraude Lame apresentou um método para identificar conceitos e relações para construir ontologias jurídicas de raíz, através de técnicas de análise a textos [Lam03].
O método começa com uma análise sintáctica ao texto de um conjunto de documentos com legislação francesa (*Codes Napoléon*). Estes documentos estão estruturados logicamente e possuem definições de conceitos jurídicos, mais ou menos explícitas. Os termos encontrados no texto são tratados como etiquetas de conceitos jurídicos.
Através das relações sintácticas entre os termos deduzem-se algumas relações semânticas entre os respectivos conceitos.
Depois da análise sintáctica, o resultado é uma lista de termos, sem *stopwords*. Em seguida recolhe-se informação sobre o grupo sintáctico (sujeito ou complemento) em que surge cada substantivo, relativamente ao verbo da frase.
Após esta recolha de contexto sintáctico, as relações entre termos podem ser deduzidas pela similaridade entre os respectivos contextos. A similaridade pode ser avaliada por métodos estatísticos.
Os resultados obtidos carecem de validação por parte de especialistas. O processo descrito ainda não substitui o processo manual de construção da ontologia, funciona antes como uma sugestão, através da identificação (automática) de conceitos e relações a representar na ontologia.

No ano anterior havia sido apresentado um método parecido para a dedução automática de ontologias [KW02]. Latifur Khan e Lei Wang propuseram a utilização do *vector space model* para avaliar a similaridade entre os termos. Em seguida era aplicado um algoritmo de *clustering* para definir as relações da hierarquia, numa abordagem *bottom-up*.

Em suma, podem identificar-se dois géneros de procedimentos para elaborar uma ontologia em Semantic Web:

1. directamente por especialistas: o modelo da ontologia é definido por pessoas e passado posteriormente para uma linguagem Semantic Web

2. por processos semi-automáticos: através de sistemas (com maior ou menor autonomia) que sugerem a ontologia. Este processo requer normalmente uma validação final, porque existe o risco de ter elementos (conceitos ou relações) em excesso ou em falta.

Capítulo 4

O Sistema Proposto

Este capítulo faz uma descrição geral do sistema proposto para atingir os objectivos traçados.
A secção 4.1 descreve o que se pretende com o trabalho (4.1.1) e qual a metodologia a seguir (4.1.2). A arquitectura é apresentada na secção 4.2, identificando os módulos do sistema (4.2.1) e a plataforma em que o trabalho foi desenvolvido (4.2.2).
Os capítulos seguintes explicam detalhadamente os procedimentos adoptados em cada fase do trabalho.

4.1 Apresentação

Para serem mais dinâmicos e abrangentes, os Sistemas de Recuperação de Informação devem estar preparados para aceitar e raciocinar com a informação disponível, que pode ser proveniente de um meio exterior.
Por outro lado, é desejável que a interacção com estes sistemas não se resuma à obtenção de resposta a interrogações. Existe interesse em consultar a base de conhecimento que tais sistemas possuem, tanto para consulta directa como para a preparação de novas ferramentas de inferência. Esta necessidade de cooperação justifica a publicação da informação de cada sistema.

A recente área *Semantic Web* é alvo de crescente interesse, a nível global, tanto da parte de pessoas ligadas às novas tecnologias como de outros sectores, das ciências ou das letras. A possibilidade de ultrapassar a representação destinada a humanos, com linguagens capazes de lidar formalmente com a informação, num contexto favorável às máquinas e ao tratamento automático, leva a que esta área seja eleita para os mais variados fins.

A publicação de qualquer género de informação deve ser efectuada de um modo consistente. As **ontologias** surgem como um meio eficaz para esse propósito e são usadas neste trabalho como a estrutura em que se representa a informação.

4.1.1 O que se pretende

No contacto inicial com ontologias surge uma dificuldade comum à grande maioria dos utilizadores. Há necessidade de elaborar uma ontologia de conceitos e esse processo levanta dúvidas como:

- saber se a ontologia dever ser construída manualmente
- saber se a ontologia construída já é suficiente para o pretendido

Acontece que, ao tentar construir uma nova ontologia relacionada com outro assunto, voltam a surgir as mesmas dificuldades. Não existe uma solução universal. Na prática existe sempre um risco de se obter uma ontologia demasiado simples, ou o contrário, uma ontologia tão elaborada que torna a sua utilização pouco viável.

O trabalho apresentado tem a ver com esta problemática, nomeadamente ao nível da automatização do processo de construção da ontologia.
Tomando como ponto de partida o Sistema de Recuperação de Informação da Procuradoria Geral da República Portuguesa, já existente, e também a sua base documental, vamos tentar conceber um sistema paralelo baseado em ontologias.

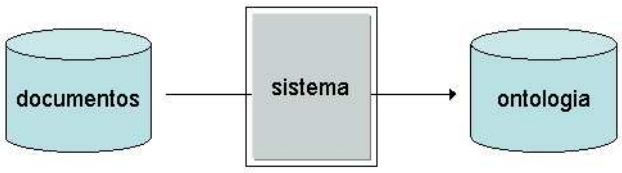

Figura 4.1: Sistema para construir ontologias

Um dos aspectos principais do trabalho é um mecanismo para construir automaticamente uma ontologia que represente, na medida do possível, a informação expressa no texto dos documentos. A figura 4.1 ilustra a aplicação do sistema aos documentos.

A ontologia construída pode ser usada para publicar os dados recolhidos do texto. Outra importante característica que o sistema deve oferecer é a utilização da ontologia para processos de inferência, como ilustrado pela figura 4.2.

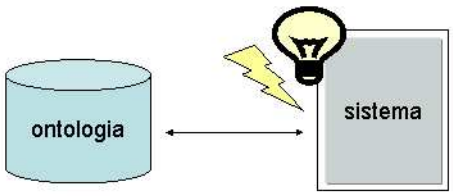

Figura 4.2: Sistema capaz de inferir sobre a ontologia

4.1.2 Metodologia

O funcionamento do sistema envolve quatro passos:

1. extracção de informação dos documentos
2. inferência de uma ontologia de classes
3. inferência de instâncias para representação dos documentos
4. uso da ontologia em processos de inferência

Em primeiro lugar faz-se uma análise ao texto dos documentos a fim de se extrair informação sobre a qual se vai trabalhar em seguida. Na prática o que se procura é uma representação formal de informação semântica para cada frase. Neste sentido, os textos são sujeitos a uma análise sintáctica seguida de análise semântica. O resultado é uma representação semântica parcial de cada frase, usando uma estrutura DRS [KR93], que é descrita na secção 5.3.1.

A segunda fase, como a própria designação indica, faz a inferência de uma ontologia de classes em **OWL**, a linguagem Semantic Web adoptada. Para isso, recebe como input um vasto conjunto de estruturas que representam frases em documentos, e para cada uma retira o sujeito, o verbo e o complemento, com eventuais expressões associadas. Os elementos recolhidos são usados para a construção das classes.

A etapa seguinte infere instâncias OWL para as classes já existentes. A partir da representação DRS das frases, de uma ontologia de classes OWL (que pode ser a gerada na fase anterior ou uma outra ontologia externa ao sistema) e recorrendo a um mecanismo de inferência por abdução, as instâncias OWL para as classes são geradas. Este processo é usualmente designado de interpretação pragmática de frases em língua natural.

Por último, as classes e instâncias OWL, automaticamente geradas pelo sistema, são usadas como uma base de conhecimento onde é possível inferir resposta para interrogações efectuadas por utilizadores. A inferência faz-se num contexto de programação em lógica, para onde todos os dados OWL são replicados.
No decorrer do processo, a informação dos documentos vai sendo filtrada e transformada, de acordo com a figura 4.3.

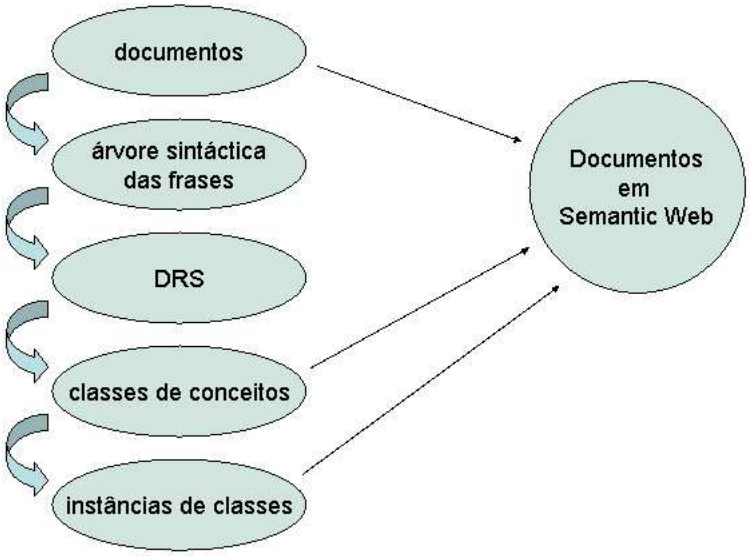

Figura 4.3: Etapas na representação da informação

4.2 Arquitectura

O trabalho realizado junta a utilização de ferramentas já disponíveis com outras desenvolvidas inteiramente de raíz. Trata-se de um protótipo sobre o qual se fizeram alguns testes, descritos ao longo deste documento, não devendo ser considerado um produto exaustivamente aperfeiçoado.

4.2.1 Módulos do Sistema

A arquitectura do sistema é constituída por módulos complementares mas independentes, cada um tem a sua função particular, contribuindo para o sistema global.
A figura 4.4 apresenta um esboço elucidativo da arquitectura, identificando os módulos relacionados com cada fase da informação mostrada na figura anterior.

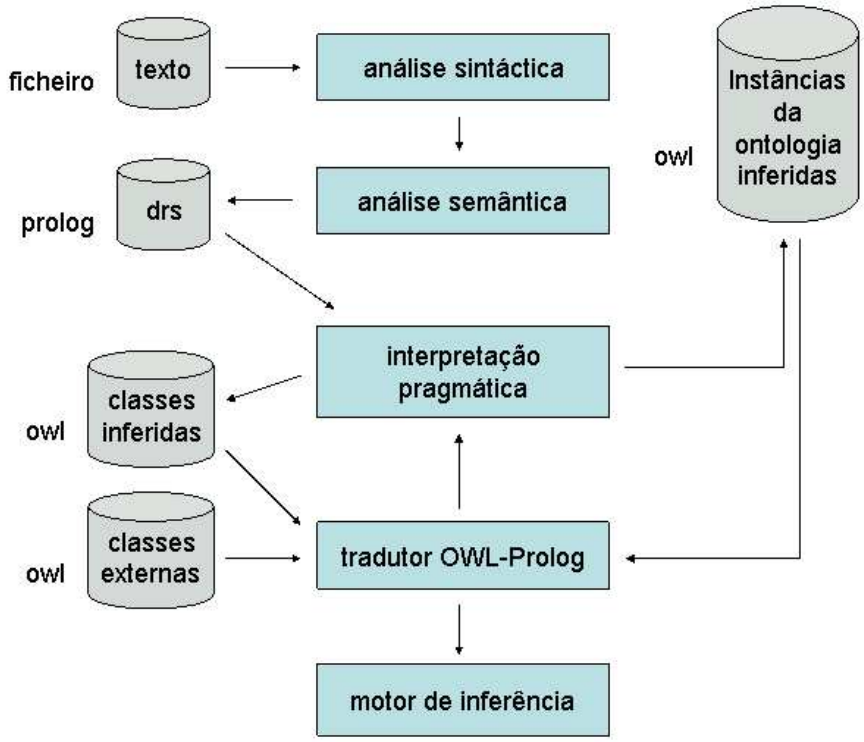

Figura 4.4: Arquitectura do sistema

Existe um módulo para a análise sintáctica e outro para a análise semântica, descritos nas secções 5.2 e 5.3, respectivamente. Este bloco faz a passagem de frases em língua natural para uma estrutura lógica, DRS, em Prolog.
O tradutor OWL-Prolog, descrito na secção 6.3.1, faz a replicação dos dados semânticos na ontologia para um contexto de programação em lógica. É nesse contexto lógico que o motor de inferência funciona, respondendo a interrogações do utilizador. A interpretação pragmática das frases é usada para gerar novos

elementos da ontologia.

4.2.2 Plataforma

Todo o trabalho foi desenvolvido utilizando o sistema operativo Linux. A produção de código OWL é feita com uma aplicação concebida na linguagem Java. O motor de inferência, apresentado no capítulo 7, é baseado na linguagem de programação em lógica Prolog, mais especificamente GNU Prolog[1].

Ao gerar código OWL foram feitas validações com a ferramenta OWL Validator, implementada em Java e também disponível para utilização via web[2].

A ontologia é publicada (por exemplo para o tradutor OWL-Prolog) através de um *web server*, que no caso é Apache. Associado ao servidor web, foi instalado o módulo JServ para suportar Java Servlets, usadas para uma interface web de testes, descrita na secção 8.1.
Na manipulação da ontologia, a aplicação Java faz uso de algumas ferramentas já disponíveis, como a API Jena[3] (para o parse ao OWL) ou CUP.

[1] http://gnu-prolog.inria.fr/
[2] http://owl.bbn.com/validator/
[3] http://www.hpl.hp.com/semweb/

Capítulo 5

Extracção de Informação dos Textos

Este capítulo descreve a primeira parte do trabalho, relacionada com extracção de informação.
A base documental do sistema é descrita na secção 5.1, onde se pode ver a natureza dos textos (5.1.1) e o Sistema de Recuperação de Informação a que estão associados (5.1.2). A secção 5.2 relata o modo como se efectua a análise sintáctica aos documentos e a secção 5.3 é dedicada à análise semântica e às estruturas usadas para representar essa semântica (5.3.1).

5.1 A base de documentos em análise

Os textos em língua natural são a matéria prima para o sistema. Antes de delinear qualquer solução é necessário verificar a natureza e formato dos documentos.

5.1.1 Os documentos jurídicos

O conjunto de documentos utilizados está relacionado com o Sistema de Recuperação de Informação da Procuradoria Geral da República Portuguesa [QR01b], descrito na próxima secção. Cada documento é um parecer jurídico da PGR. No total, são cerca de 7,000 documentos jurídicos desde 1940, totalizando aproximadamente 10,000,000 de palavras.

Os documentos têm uma estrutura específica, definida num formato compatível com XML. Os campos são marcados por tags ou etiquetas ao longo do documento.

Eis alguns dos campos:

- número
- título
- descritores
- relator
- conclusões
- texto integral

O campo *descritores* tem um conjunto de conceitos referidos no documento e que caracterizam o assunto do mesmo. Em *texto integral* está todo o parecer em língua natural (DOC_T_INTEGRAL). Exemplo de um documento PGR:

```
<!-- sino section NUMDOC -->
doc1
<!-- end section -->

<!-- sino section RELATOR -->
ANTÓNIO EXPERIÊNCIA
<!-- end section -->

<!-- sino section DESCRITORES -->
COMBATE AOS INCÊNDIOS
SUBSÍDIO DE RISCO
...
<!-- end section -->

<!-- sino section DOC_T_INTEGRAL -->
Senhor Ministro do Ambiente,
Excelência:

O Gabinete de Vossa Excelência dignou-se enviar o...
Tal cooperação subordina-se aos interesses da...
O bombeiro da localidade salvou a criança indefesa.
...
<!-- end section -->
```

Figura 5.1: Formato inicial do documento PGR

O texto destes documentos serviu de base para o trabalho apresentado, mas note-se que é possível aplicar o sistema a qualquer texto em língua portuguesa, como veremos mais adiante.

5.1.2 O Sistema de Recuperação de Informação da PGR

Como já foi referido, este sistema aplica-se sobre a base de textos jurídicos da PGR. Trata-se de um sistema de recuperação de informação cooperativo. A sua arquitectura baseia-se em agentes e foi implementada sobre Linux e Prolog.

O sistema recorre a várias áreas da Inteligência Artificial:

- **processamento de língua natural** - para lidar com a informação linguística através de dicionários e *part-of-speech taggers*
- **clustering** - usa algoritmos de clustering para agrupar documentos pelo assunto
- **programação em lógica e agentes** - paradigma de programação em lógica dinâmica como base para a construção de agentes autónomos [PQ98, APP+00]

A figura 5.2 mostra as duas camadas do SRI.

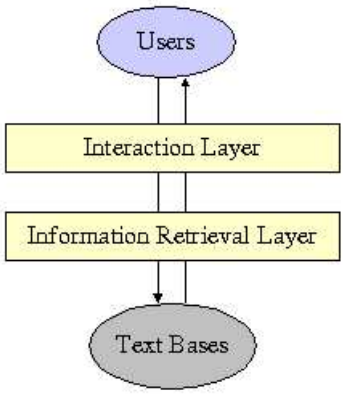

Figura 5.2: Camadas no SRI da PGR

A *interaction layer* é uma arquitectura de agentes, implementados em programação em lógica, que se encarrega da interacção com o utilizador. Cada utilizador tem um agente associado que é responsável por representar um modelo desse utilizador, tentar inferir os seus objectivos e se possível concretizá-los, respondendo a interrogações. As interrogações podem ser introduzidas graficamente ou através de frases.

A *information retrieval layer* trata do acesso aos documentos. Esta camada é formada por agentes que recebem interrogações da camada anterior e aplicam várias técnicas de processamento de língua natural aos textos para obter a resposta.

5.2 Análise Sintáctica

A procura de informação no texto do documento implica uma análise às frases. Assim, o primeiro passo nesse sentido é uma análise sintáctica. Para esta operação, utiliza-se um analisador sintáctico desenvolvido por Eckhard Bick no âmbito do projecto VISL[Bic00].

5.2.1 O Projecto VISL

O nome VISL vem de *Visual Interactive Syntax Learning*[1], é um projecto de investigação no *Institute of Language and Communication*, *University of Southern Denmark*, desde Setembro de 1996, envolve investigadores e alunos e o seu objectivo é o desenho e implementação de ferramentas gramaticais baseadas na Internet, com propósito educacional e de investigação.
No início, o projecto aplicava-se a quatro línguas: Inglês, Francês, Alemão e Português. Desde então tem vindo a estender-se e a lista já contabiliza mais de vinte línguas, entre as quais Russo, Japonês e Esperanto.

Quando vistas de modo isolado, as palavras podem revelar-se ambíguas em termos do tipo de palavra, inflexão, função sintáctica e conteúdo semântico. Fundamentalmente, é o contexto da(s) frase(s) que determina como uma palavra deve ser compreendida (juntamente com o "conhecimento" que o leitor tem do mundo). *Constraint Grammar* é um formalismo gramatical destinado à resolução de ambiguidades. Este formalismo baseia-se na definição de regras para decidir qual das

[1] http://visl.hum.sdu.dk/visl/

possíveis leituras de uma palavra deve ser escolhida, e quais as leituras que devem ser descartadas, num dado contexto textual.
Estas regras podem ser compiladas num programa, que recebe como *input* um texto processado morfologicamente, mas que inclui ambiguidades, e cujo resultado deve ser a leitura correcta[2] para cada palavra. Vejamos um exemplo de *input* para tal aplicação, com as várias leituras possíveis[3] para cada palavra da frase "*Nunca como peixe*":

```
<nunca>
   nunca  ADV (advérbio)
<como>
   como  <rel> ADV (advérbio relativo)
   como  <interr> ADV (advérbio interrogativo)
   como  KS (conjunção)
   como  <vt> V PR 1S VFIN (verbo, presente, 1ª pessoa do singular)
<peixe>
   peixe  N M S (nome, masculino, singular)
```

A palavra *como* apresenta quatro leituras possíveis. Para resolver a ambiguidade pode recorrer-se a uma regra como[4]:

```
select for a given word form the VFIN reading (finite verb)
           if there is no (neither to the left nor the right)
           other word that can be VFIN
```

Deste modo, seria seleccionada a última leitura, que corresponde à forma verbal *como*.

O *parser* para o Português resulta da investigação de Bick na área de anotação gramatical automática e análise do vocabulário da Língua Portuguesa. O funcionamento desta ferramenta é baseado no formalismo *Constraint Grammar*. Este *parser* abrange uma grande percentagem da Língua Portuguesa.

[2]Em casos de "verdadeira" ambiguidade podem existir duas ou mais leituras possíveis, que serão preservadas e passadas à fase seguinte (outra verificação).
[3]As leituras possíveis de cada palavra no exemplo estão alinhadas à direita.
[4]Regra apresentada numa versão simplificada.

5.2.2 Aplicação do *Parser* ao Texto

Tomemos como exemplo a seguinte frase:

O bombeiro salvou a criança.

Esta frase apresenta uma estrutura simples, contendo um sujeito, um verbo e um complemento. Aplicando o *parser* a esta frase obtemos o resultado da figura 5.3.

```
C-1 O bombeiro salvou a criança.
A1
STA:fcl
SUBJ:np
=>N:art('o' M S)        O
=H:n('bombeiro' M S)    bombeiro
P:v-fin('salvar' PS 3S IND)     salvou
ACC:np
=>N:art('o' F S)        a
=H:n('criança' F S)     criança
.
```

Figura 5.3: Output do *parser* VISL aplicado à frase

Ao longo das várias linhas de código, podemos encontrar as etiquetas:

SUBJ - representa o sujeito

P - representa o predicado

ACC - representa o complemento

Como se pode ver, o analisador sintáctico identificou correctamente as secções da frase, representando-as através de uma árvore detalhada e com etiquetas sintácticas.

O resultado da análise do *parser* encontra-se num formato próprio, não *standard*, pelo que foi necessário introduzir uma ferramenta de normalização para um formato adequado às fases seguintes. A ferramenta usada é descrita em [GVGQ03] e está disponível para os utilizadores VISL. O seu resultado é uma representação estruturada, em XML ou Prolog.

O código VISL da figura 5.3 seria codificado em três ficheiros XML:

1. O primeiro ficheiro associa cada palavra a uma etiqueta que funciona como identificador único.

```
<!DOCTYPE words SYSTEM "words.dtd">
<words>
<word id="word_1">O</word>
<word id="word_2">bombeiro</word>
<word id="word_3">salvou</word>
<word id="word_4">a</word>
<word id="word_5">criança</word>
<word id="word_6">.</word>
</words>
```

2. O segundo ficheiro representa a informação gramatical de cada palavra (*part-of-speech info*), como o tempo, modo e pessoa, no caso dos verbos. Cada palavra é identificada pela respectiva etiqueta.

```
<!DOCTYPE words SYSTEM "wordsPOS.dtd">
<words>
<word id="word_1">
<art canon="o" gender="M" number="S"/>
</word>
<word id="word_2">
<n canon="bombeiro" gender="M" number="S"/>
</word>
<word id="word_3">
<v canon="salvar">
<fin tense="PS" person="3S" mode="IND"/>
</v>
</word>
<word id="word_4">
<art canon="o" gender="F" number="S"/>
</word>
<word id="word_5">
<n canon="criança" gender="F" number="S"/>
</word>
</words>
```

3. O último ficheiro representa a estrutura da frase. É possível identificar cada pedaço (*chunk*) da frase, com as palavras que o formam, e ainda se diz respeito a um nome, a um artigo ou outros.

```
<!DOCTYPE text SYSTEM "text_ext.dtd">
<text>
<paragraph id="paragraph_1">
<sentence id="sentence_1" span="word_1..word_6">
<chunk id="chunk_1" ext="subj" form="np" span="word_1..word_2">
<chunk id="chunk_2" ext="n" form="art" span="word_1">
</chunk>
<chunk id="chunk_3" ext="h" form="n" span="word_2">
</chunk>
</chunk>
<chunk id="chunk_4" ext="p" form="v_fin" span="word_3">
</chunk>
<chunk id="chunk_5" ext="acc" form="np" span="word_4..word_5">
<chunk id="chunk_6" ext="n" form="art" span="word_4">
</chunk>
<chunk id="chunk_7" ext="h" form="n" span="word_5">
</chunk>
</chunk>
</sentence>
</paragraph>
</text>
```

Outra funcionalidade que a ferramenta apresenta é a conversão VISL para Prolog. O código VISL da figura 5.3 seria representado em Prolog tal como mostra a figura 5.4.

```
sentence(syn(sta(fcl),
             subj(np, n(art('o','M','S'),'O'),
                    h(n('bombeiro','M','S'),'bombeiro')),
             p(v_fin('salvar','PS','3S','IND'),'salvou'),
             acc(np, n(art('o','F','S'),'a'),
                    h(n('criança','F','S'),'criança', '.'))
)).
```

Figura 5.4: Resultado do *parser* VISL em Prolog

A árvore sintáctica que antes estava em VISL foi convertida para factos Prolog,

através de vários predicados que representam as palavras e indicações sintácticas. Esta tipo representação é próprio de um paradigma de programação em lógica.

5.3 Análise Semântica

Para cada documento vamos ter um conjunto de frases descritas numa estrutura, criada no passo anterior, com a informação lexical das palavras e elementos sintácticos que constituem a frase. Essa estrutura é analisada com base no formalismo de *Discourse Representation Theory* [KR93].

5.3.1 Discourse Representation Structure

Discourse Representation Theory (DRT) é uma das teorias de *dynamic semantics* que surgiram nos últimos vinte anos. Estas teorias incidem sobre a semântica e a sua dependência do contexto. A DRT define um modo de representar a semântica do discurso através da sua lógica e não apenas pela elocução[5] com que surge. Em DRT o discurso é representado por *Discourse Representation Structure* (DRS).

Uma DRS é um par com um conjunto de referentes do discurso *d1,d2... ,dn* e um conjunto de condições DRS *Cond(d1,d2... ,dn)*.
Os referentes do discurso identificam sobre quem é o discurso. As condições DRS são asserções, sobre os referentes, que são expressas no discurso.

A semântica de uma DRS é definida pela expressão:

$$\exists\, d1, d2..., dn : Cond\,(d1, d2..., dn)$$

A possibilidade de guardar e manipular os referentes do discurso, relativos a todo um texto, permite o conceito de histórico, e deste modo, a resolução de anáforas e elipses. A aplicação de processos de abdução é também possível a partir desta representação, como veremos mais adiante.

O carácter lógico desta teoria conduz à utilização de ferramentas lógicas, nomeadamente a linguagem de programação em lógica Prolog. Esta teoria para representação do discurso é também utilizada e descrita num trabalho posterior de Quaresma e Rodrigues [QR03].

[5] Forma de exprimir por meio de palavra.

5.3.2 Informação Semântica Parcial

A partir da estrutura sintáctica descrita em XML ou em Prolog é possível identificar as asserções e respectivos referentes em cada frase. Essa estrutura é analisada e convertida para expressões em Lógica de Primeira Ordem.

A frase de exemplo, cuja análise sintáctica em formato Prolog se encontra na figura 5.4, é representada por uma DRS com a estrutura da figura 5.5

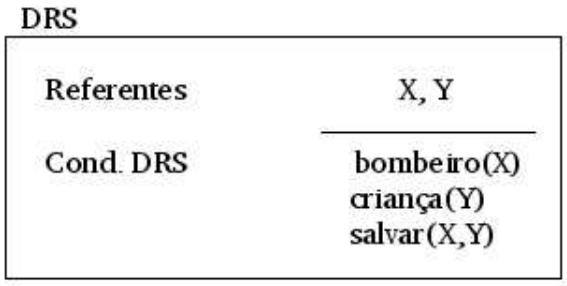

Figura 5.5: Elementos de uma DRS para o exemplo da frase.

O resultado obtido é uma representação semântica parcial da frase, consistindo numa DRS com duas listas, uma com a frase reescrita e outra com os referentes do discurso.

Na prática a DRS extraída para a frase vai ser representada em Prolog, pela expressão:

```
sentence(doc1, [ bombeiro(X), criança(Y), salvar(X,Y) ],
               [ ref(X), ref(Y) ] ).
```

Figura 5.6: Exemplo da DRS em Prolog

Isto significa que existe uma instância de *bombeiro* X e uma instância de *criança* Y que estão relacionadas pela acção de *salvar*.

Vejamos o caso em que a frase é um pouco mais complexa:

O bombeiro da localidade salvou a criança indefesa.

Agora existem mais elementos gramaticais, e o output do analisador sintáctico será o seguinte:

```
C-1 O bombeiro da localidade salvou a criança indefesa.
A1
STA:fcl
SUBJ:np
=>N:art('o' M S)        O
=H:n('bombeiro' M S)    bombeiro
=N<:pp
==H:prp('de' <sam->)    de
==P<:np
===>N:art('o' F S <-sam>)       a
===H:n('localidade' F S)        localidade
P:v-fin('salvar' PS 3S IND)     salvou
ACC:np
=>N:art('o' F S)        a
=H:n('criança' F S)     criança
=N<:adj('indefeso' F S) indefesa
.
```

Figura 5.7: Output do analisador VISL no segundo exemplo

Neste caso, a DRS seria representada pelo código da figura 5.8.

```
sentence(doc1, [ bombeiro(X), localidade(X1), rel(X,X1),
                    criança(Y), indefeso(Y),
                    salvar(X,Y) ],
                [ ref(X), ref(Y), ref(X1) ] ).
```

Figura 5.8: DRS em Prolog, segundo caso

Relativamente ao caso anterior, na figura 5.6, agora surge uma instância de *localidade*, X1, com a qual X está relacionado (*rel*). Também a instância Y aparece agora com mais informação, sabe-se que é *indefeso*.

De momento, o sistema em causa efectua uma análise semântica muito restrita. A sintaxe em Prolog dos documentos, como mostra a figura 5.4, será usada como *input* para o seguinte código:

```
singleSentenceAnalysis(InFile,OutFile):-
   see(InFile),
   repeat, read(Sen), processSentence(Sen,OutFile), !.

processSentence(sentence(S),OutFile):-
   S=..SL, allTriplos(SL,OutFile), fail.
processSentence(end_of_file,_):- seen.

allTriplos([],_).
allTriplos(L,OutFile):-
   getASubj(L,Subj,SAdj,SND,SPPL,L2),
   getAVerb(L2,Verb,L3),
   getADirObj(L3,Obj,OAdj,OPPL,L4),
   reportTriplo(Subj,SAdj,SND,SPPL, Verb, Obj,OAdj,OPPL, OutFile),
   !,
   allTriplos(L4,OutFile).
```

Para já, o interesse recai apenas nos predicados, com respectivo sujeito e complemento, caracterizados por eventuais adjectivos e sintagma preposicional. Os predicados representam acções mencionadas no discurso. Este módulo lê a sintaxe do documento e em cada frase vai procurar um ou mais triplos com sujeito, predicado e complemento, a partir dos quais será construída a DRS.

Foi elaborada uma ferramenta de uso genérico para construir representações semânticas parciais deste tipo, relativas às frases de um documento, e foi aplicada a todos os documentos jurídicos da Procuradoria Geral da República Portuguesa. O número de triplos (e consequentemente estruturas DRS) extraídos para cada documento varia em função do número e tipo de frases que o documento contém (já que nem todas possuem complemento). Em média, o número de triplos para um documento é 27% do número de frases desse documento.
O capítulo 8 inclui exemplos da aplicação deste processo.

Capítulo 6

Construção de Ontologias

Este capítulo relata o processo automático de construção da ontologia relacionada com os documentos.
Na secção 6.1 indica-se qual a linguagem Semantic Web escolhida (6.1.1), bem como a ontologia base (6.1.2), que será estendida ao longo do processo. A geração automática de classes e instâncias é descrita na secção 6.2. Na secção 6.3 é dedicada à produção de instâncias de classes de uma ontologia externa ao sistema.

6.1 Linguagem Semantic Web e Ontologia Base

A passagem de um documento para Semantic Web visa apenas adicionar anotações semânticas ao formato inicial, de modo a facilitar o processamento automático da mesma informação. Não se adiciona mais informação ao documento, apenas se procura representar a mesma informação num formato que as máquinas facilmente entendam e possam, a partir daí, inferir algo.

6.1.1 A escolha da Linguagem

De certo modo, a Semantic Web pode ser encarada como uma enorme base de dados distribuída. Para que os dados possam ser representados é necessário haver regras e garantias de que o formato usado é o mesmo. Em suma, é necessário escolher uma linguagem.

Esta é uma área recente mas na qual existem várias linguagens, conforme já foi dito no capítulo 2. Numa fase inicial, a linguagem escolhida foi DAML+OIL. Esta linguagem surgiu em 2000 e resulta da cooperação entre investigadores americanos

e europeus, responsáveis pelas linguagens antecessoras DAML e OIL, respectivamente.
Com esta linguagem foi elaborada a primeira versão da Ontologia base, cuja versão actual é apresentada na próxima secção, e efectuadas algumas experiências.

Web Ontology Language, ou abreviadamente OWL, é uma extensão de RDF e deriva da linguagem DAML+OIL. A linguagem OWL foi adoptada pelo sistema em Março de 2003, quando ainda se encontrava em versão *Working Draft* no W3C [1], por estar amplamente difundida a nível mundial.
Todo o conteúdo existente até então foi traduzido para OWL, que na verdade possui bastantes semelhanças com a linguagem anterior.

6.1.2 A Ontologia Base

Depois da selecção da linguagem, era preciso criar uma ontologia. A ontologia é uma especificação que garante a coerência semântica ao conteúdo Semantic Web. Em OWL, uma ontologia consiste num conjunto de classes, com propriedades de vários tipos, e as instâncias dessas mesmas classes.

O passo inicial é obter uma representação do documento em Semantic Web. Esta primeira transformação diz respeito apenas aos campos do documento, já identificados por *tags* no formato inicial. É uma conversão directa que não implica ainda técnicas de PLN.

Para criar a ontologia é preciso ter em conta que se pretende descrever. Para isso estuda-se o formato dos documentos para encontrar um conjunto de propriedades a incluir na classe que os vai representar.
Do documento da PGR mostrado antes, na figura 5.1, podemos identificar os campos:

- número
- relator
- descritores
- texto integral

[1] Durante a escrita desta dissertação, em Agosto de 2003, a linguagem encontra-se na fase *W3C Candidate Recommendation*

Nota: *Na realidade um documento tem mais campos. Os campos relevantes são aqui apresentados, os restantes são tratados do mesmo modo e não trariam qualquer vantagem, antes pelo contrário, tornariam o exemplo muito vasto e menos apetecível.*

É necessário criar uma classe `Document` que contenha propriedades para representar os campos do documento. Assim, com a linguagem OWL, a classe define-se do seguinte modo:

```
<owl:Class rdf:ID="Document">
  <rdfs:comment>a PGR document</rdfs:comment>
</owl:Class>

        <owl:DatatypeProperty rdf:ID="numero">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:DatatypeProperty rdf:ID="relator">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:ObjectProperty rdf:ID="descritor">
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="#Concept" />
        </owl:ObjectProperty>

        <owl:DatatypeProperty rdf:ID="textoIntegral">
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>
```

Antes de mais, temos propriedades *DatatypeProperty* e *ObjectProperty*. As primeiras relacionam a instância de uma classe com tipos de dados XML Schema[2] (string, integer). As segundas relacionam instâncias de duas classes.

[2]http://www.w3.org/XML/Schema

Algumas das propriedades estão marcadas com o tipo *FunctionalProperty*. Tomando o exemplo de *relator*, isto significa que cada instância de `Document` só pode ter um relator. O mesmo já não acontece com a propriedade *descritor*, já que podem existir vários para o mesmo documento.

As expressões que se encontram entre aspas iniciadas por # representam URIs que referem objectos no próprio ficheiro OWL em que a classe é definida. Os iniciados por & representam um URI abreviado em que se usa a XML ENTITY[3]. Os *headers* com a respectiva definição podem ser consultados na versão integral da ontologia base, que consta no *Apêndice A* deste documento (página 123).

A propriedade *descritor* relaciona uma instância de `Document` com uma (ou mais, já que não é funcional) instância de `Concept`. Esta classe representa as palavras-chave ou conceitos principais do conteúdo do documento e já estão identificadas em cada um dos documentos.
Conforme se pode ver no respectivo código, existem propriedades que relacionam estas instâncias com outras do mesmo tipo. Essas relações permitem registar se um conceito é mais específico que outro, ou se é mais geral, ou relacionado com algum outro, ou ainda equivalente.

Este é o código OWL para Concept:

```
<owl:Class rdf:ID="Concept">
  <rdfs:comment>to represent a concept in field descritor
  </rdfs:comment>
</owl:Class>

        <owl:DatatypeProperty rdf:ID="conceptName">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Concept" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:ObjectProperty rdf:ID="moreAbstractThan">
          <rdfs:domain rdf:resource="#Concept" />
          <rdfs:range rdf:resource="#Concept" />
        </owl:ObjectProperty>
```

[3]No exemplo, `&xsd;` vai ser equivalente a `http://www.w3.org/2001/XMLSchema#`

```
<owl:ObjectProperty rdf:ID="moreSpecificThan">
  <rdfs:domain rdf:resource="#Concept" />
  <rdfs:range rdf:resource="#Concept" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="relatedWith">
  <rdfs:domain rdf:resource="#Concept" />
  <rdfs:range rdf:resource="#Concept" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="equivalentWith">
  <rdfs:domain rdf:resource="#Concept" />
  <rdfs:range rdf:resource="#Concept" />
</owl:ObjectProperty>
```

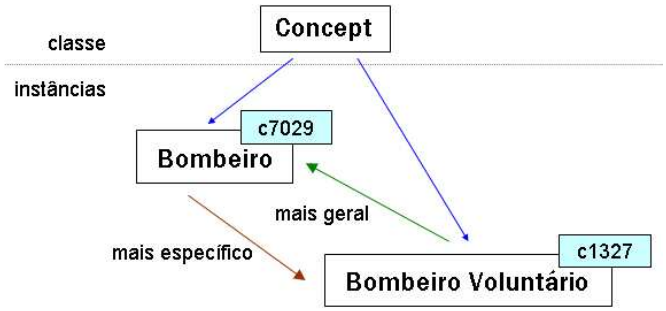

Figura 6.1: Excerto da ontologia sobre os descritores do documento

A figura 6.1 mostra graficamente duas instâncias de `Concept`, para representar *bombeiro* e *bombeiro voluntário*, conceitos que pertencem ao campo descritor de pelo menos um documento. O código OWL para essas instâncias é listado em seguida. Note-se o uso das propriedades para expressar uma relação entre as instâncias: o conceito de *bombeiro* é mais geral que o conceito de *bombeiro voluntário*.

```
<pgr:Concept rdf:ID="c7029">
  <pgr:conceptName>bombeiro</pgr:conceptName>
  <pgr:moreAbstractThan rdf:resource="#c1327"/>
</pgr:Concept>
```

```
<pgr:Concept rdf:ID="c1327">
  <pgr:conceptName>bombeiro voluntário</pgr:conceptName>
  <pgr:moreSpecificThan rdf:resource="#c7029"/>
</pgr:Concept>
```

Finalmente, a instância em OWL que representa o documento PGR apresentado na figura 5.1 (página 52) tem o código:

```
<pgr:Document rdf:ID="doc1">
   <pgr:docNumber>doc1</pgr:docNumber>

   <pgr:relator>ANTÓNIO EXPERIÊNCIA</pgr:relator>

   <pgr:descritor rdf:resource=
      "http://abc.di.uevora.pt/~jsaias/owl/concept.owl#c1327"/>
   <pgr:descritor rdf:resource=
      "http://abc.di.uevora.pt/~jsaias/owl/concept.owl#c4892"/>

   <pgr:docTIntegral>
   Senhor Ministro do Ambiente,
   Excelência:

   O Gabinete de Vossa Excelência dignou-se enviar o...
   Tal cooperação subordina-se aos interesses da...
   O bombeiro da localidade salvou a criança indefesa.
   ...
   </pgr:docTIntegral>
</pgr:Document>
```

O prefixo *pgr*, por exemplo em `pgr:Document`, é um XML namespace definido no *header* do ficheiro e que identifica o URL da ontologia a que a classe `Document` pertence.

Importa salientar que está presente toda a informação que estava no formato inicial. Não se acrescentou nenhum dado, apenas houve uma transformação para uma linguagem mais formal e estruturada.

Até agora, construimos uma versão Semantic Web com os documentos do sistema. Em seguida vamos enriquecer essa representação com mais dados sobre os documentos.

6.2 Ontologia de Conceitos com Dois Níveis

Nesta secção apresenta-se um processo automático para criação de ontologias a partir de textos.

Para que se conseguisse lidar com documentos dos mais diversos domínios foi necessário recorrer a uma ontologia básica, que representasse conceitos de um modo generalista. A ideia é ter uma representação de conceitos relevantes ao documento com poucas relações hierárquicas entre eles.

Numa fase posterior à construção da ontologia, pode recorrer-se a uma reorganização da hierarquia de conceitos que a constitui, com intervenção humana por parte de especialistas na área dos documentos, com vista à obtenção de uma ontologia mais elaborada.

6.2.1 Classes Entity e Action

A primeira preocupação na elaboração deste processo automático tem a ver com a identificação de algo para procurar no texto e que faça sentido representar. Tanto no caso particular dos documentos jurídicos como para outras áreas temáticas, seria útil identificar acções referidas no texto, bem como os seus autores ou terceiras partes envolvidas.

De uma forma geral, vamos assumir que cada conceito, pessoa ou objecto de uma acção é uma entidade. Essas entidades encontram-se relacionadas pela acção. De acordo com isto, a ontologia base vai ser estendida com as classes:

```
<owl:Class rdf:ID="Entity">
  <rdfs:comment>to represent an entity found
                in the document</rdfs:comment>
</owl:Class>

        <owl:DatatypeProperty rdf:ID="entName">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Entity" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:ObjectProperty rdf:ID="entDoc">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
```

```
          <rdfs:domain rdf:resource="#Entity" />
          <rdfs:range rdf:resource="#Document" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="entModifier">
          <rdfs:domain rdf:resource="#Entity" />
          <rdfs:range rdf:resource="#Modifier" />
        </owl:ObjectProperty>


<owl:Class rdf:ID="Action">
  <rdfs:comment>some action retrieved from a
                sentence in the document</rdfs:comment>
</owl:Class>

        <owl:ObjectProperty rdf:ID="subject">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action" />
          <rdfs:range rdf:resource="#Entity" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="verb">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action" />
          <rdfs:range rdf:resource="#Entity" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="object">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action" />
          <rdfs:range rdf:resource="#Entity" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="actionDoc">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action" />
          <rdfs:range rdf:resource="#Document" />
        </owl:ObjectProperty>


<owl:Class rdf:ID="Modifier">
  <rdfs:comment>to add info to collected entities</rdfs:comment>
</owl:Class>
```

```
<owl:DatatypeProperty rdf:ID="modType">
  <rdfs:domain rdf:resource="#Modifier" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="modValue">
  <rdfs:domain rdf:resource="#Modifier" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="mod">
  <rdfs:domain rdf:resource="#Modifier" />
  <rdfs:range rdf:resource="#Modifier" />
</owl:ObjectProperty>
```

A classe `Entity` tem propriedades para registar o nome da entidade, o documento em que surge e uma terceira propriedade `entModifier`. Através desta propriedade, do tipo `Modifier`, é possível registar informação adicional sobre a entidade, informação essa que é retirada da frase, do local onde que se identificou a entidade.
As propriedades da classe `Action` incluem o triplo de conceitos ou entidades que forma uma acção, tipicamente o sujeito, o verbo e o complemento, e ainda o documento em que a acção é relatada. A figura 6.2 ilustra as classes que formam a ontologia base, ainda sem relações hierárquicas.

Figura 6.2: Resumo da ontologia base

A partir da representação semântica parcial das frases do documento vamos procurar acções e respectivas entidades.
`Entity` é a classe de topo para as entidades e está predefinida pelo sistema, as restantes são criadas à medida que se analisam os documentos. Sempre que surge um novo conceito, isto é, uma ocorrência de uma entidade nunca antes vista, cria-se uma subclasse de `Entity`.

Voltando ao exemplo da frase:

O bombeiro salvou a criança.

As entidades extraídas são as seguintes:

- bombeiro
- salvar
- criança

Cada entidade leva à introdução de uma nova classe. A sua disposição na hierarquia está representada na figura 6.3.

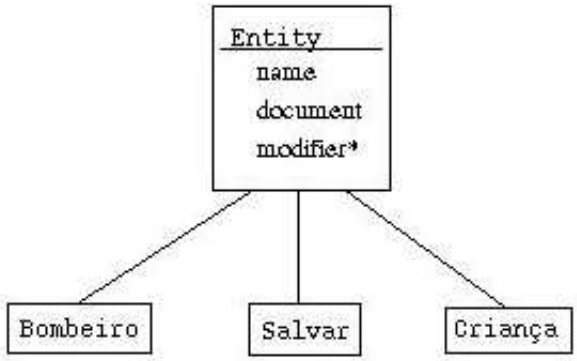

Figura 6.3: Disposição das classes na hierarquia de entidades

O código OWL, gerado automaticamente, para representar a hierarquia da figura 6.3 é o seguinte:

```
<owl:Class rdf:ID="bombeiro">
  <rdfs:subClassOf rdf:resource="&pgr;Entity" />
</owl:Class>

<owl:Class rdf:ID="salvar">
  <rdfs:subClassOf rdf:resource="&pgr;Entity" />
</owl:Class>
```

```
<owl:Class rdf:ID="criança">
  <rdfs:subClassOf rdf:resource="&pgr;Entity" />
</owl:Class>
```

6.2.2 Produção de Instâncias

Depois de gerar as classes da ontologia, é necessário extrair as instâncias dessas classes que são referidas no texto dos documentos. Cada conceito vai corresponder a uma instância de uma subclasse de `Entity` e a associação de conceitos na mesma acção faz-se com uma instância de `Action`.

Pela análise da representação semântica da frase que leva à construção das subclasses descritas identificam-se simultaneamente as respectivas instâncias. Para o caso apresentado, o código automaticamente gerado é:

```
<pgre:bombeiro rdf:ID="e13">
    <pgr:entName>bombeiro</pgr:entName>
    <pgr:entDoc rdf:resource=
             "http://abc.di.uevora.pt/~jsaias/owl/docs/doc1#doc1" />
</pgre:bombeiro>

<pgre:salvar rdf:ID="e14">
    <pgr:entName>salvar</pgr:entName>
    <pgr:entDoc rdf:resource=
             "http://abc.di.uevora.pt/~jsaias/owl/docs/doc1#doc1" />
</pgre:salvar>

<pgre:criança rdf:ID="e15">
    <pgr:entName>criança</pgr:entName>
    <pgr:entDoc rdf:resource=
             "http://abc.di.uevora.pt/~jsaias/owl/docs/doc1#doc1" />
</pgre:criança>
```

O prefixo *pgre* é um XML namespace definido no *header* do ficheiro do código e que identifica o URL onde estão as classes `bombeiro`, `salvar` e `criança`. A propriedade `entDoc` identifica o documento do qual foram extraídas as instâncias. Vistas as instâncias das entidades, a instância de `Action` que representa a acção do salvamento é:

```
<pgr:Action rdf:ID="action-doc1_1">
   <pgr:subject rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docentities/doc1#e13" />
   <pgr:verb rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docentities/doc1#e14" />
   <pgr:object rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docentities/doc1#e15" />
   <pgr:actionDoc rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docs/doc1#doc1" />
</pgr:Action>
```

No código anterior, está expresso que a acção ocorre no documento com identificador *doc1* e as entidades envolvidas na acção são as instâncias vistas antes, devidamente identificadas pelos respectivos URIs que são únicos.
Outras referências a *bombeiro* vão ser representadas por novas instâncias da subclasse `bombeiro`.

Para a segunda frase:

O bombeiro da localidade salvou a criança indefesa.

As instâncias obtidas da sua informação semântica, na DRS que está na figura 5.8 (página 61), são as seguintes:

```
<pgre:bombeiro rdf:ID="e22">
     <pgr:entName>bombeiro</pgr:entName>
     <pgr:entDoc rdf:resource=
          "http://abc.di.uevora.pt/~jsaias/owl/docs/doc1#doc1" />
     <pgr:entModifier>
        <pgr:Modifier rdf:ID="mPPe22">
             <pgr:modType>Prepositional</pgr:modType>
             <pgr:modValue>localidade</pgr:modValue>
        </pgr:Modifier>
     </pgr:entModifier>
</pgre:bombeiro>

<pgre:salvar rdf:ID="e23">
     <pgr:entName>salvar</pgr:entName>
     <pgr:entDoc rdf:resource=
          "http://abc.di.uevora.pt/~jsaias/owl/docs/doc1#doc1" />
</pgre:salvar>
```

```
<pgre:criança rdf:ID="e24">
     <pgr:entName>criança</pgr:entName>
     <pgr:entDoc rdf:resource=
          "http://abc.di.uevora.pt/~jsaias/owl/docs/doc1#doc1" />
     <pgr:entModifier>
        <pgr:Modifier rdf:ID="mAe24">
             <pgr:modType>Adjective</pgr:modType>
             <pgr:modValue>indefeso</pgr:modValue>
        </pgr:Modifier>
     </pgr:entModifier>
</pgre:criança>
```

Como se pode ver pelo *ID*, estas são outras instâncias das mesmas classes. Neste caso temos mais informação, existem instâncias da classe `Modifier` que associam o sintagma preposicional ao *bombeiro* e o adjectivo à *criança*.

Tal como está patente na figura 6.1, o processo até aqui apresentado para a construção automática de ontologias está preparado para construir uma hierarquia com apenas dois níveis de classes e as suas instâncias. Na próxima secção descreve-se outra abordagem para estender ontologias mais complexas.

6.3 Ontologia Externa

Para além de gerar instâncias em OWL para a ontologia de classes a dois níveis explicada na secção anterior, o sistema é capaz de ajudar na construção de ontologias mais elaboradas. Aqui é necessária a intervenção externa humana, por parte de especialistas na área temática dos documentos, para a tarefa de construção de uma hierarquia de classes ou simplesmente a reorganização das classes já encontradas na fase anterior numa hierarquia mais refinada e coerente.
A ontologia de classes fornecida ao sistema vai ser estendida com instâncias extraídas das frases dos documentos.

Consideremos a ontologia da figura 6.4, relacionada com assuntos militares, e cujo código é apresentado em seguida.

```
<Class rdf:ID="Cidadão">
      <rdfs:label>Cidadão</rdfs:label>
</Class>
```

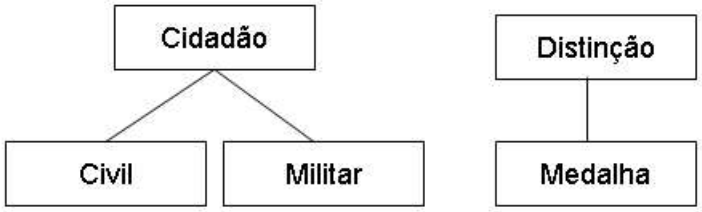

Figura 6.4: Ontologia externa

```
   <DatatypeProperty rdf:ID="nome">
      <rdfs:domain rdf:resource="#Cidadão" />
      <rdfs:range rdf:resource="&xsd;string" />
      <rdf:type rdf:resource="&owl;FunctionalProperty" />
   </DatatypeProperty>

<Class rdf:ID="Civil">
      <rdfs:label>Civil</rdfs:label>
      <owl:subClassOf rdf:resource="#Cidadão" />
</Class>

   <DatatypeProperty rdf:ID="profissão">
      <rdfs:domain rdf:resource="#Civil" />
      <rdfs:range rdf:resource="&xsd;string" />
      <rdf:type rdf:resource="&owl;FunctionalProperty" />
   </DatatypeProperty>

<Class rdf:ID="Militar">
      <rdfs:label>Militar</rdfs:label>
      <owl:subClassOf rdf:resource="#Cidadão" />
</Class>

   <DatatypeProperty rdf:ID="ramo">
      <rdfs:domain rdf:resource="#Militar" />
      <rdfs:range rdf:resource="&xsd;string" />
   </DatatypeProperty>

   <DatatypeProperty rdf:ID="posto">
      <rdfs:domain rdf:resource="#Militar" />
      <rdfs:range>
              <owl:DataRange>
              <owl:oneOf>
```

```
                <rdf:List>
          <rdf:first rdf:datatype="xsd:string">sargento</rdf:first>
          <rdf:rest>
            <rdf:List>
          <rdf:first rdf:datatype="xsd:string">tenente</rdf:first>
          <rdf:rest>
             <rdf:List>
            <rdf:first rdf:datatype="xsd:string">capitão</rdf:first>
            <rdf:rest rdf:resource="&rdf;nil" />
             </rdf:List>
          </rdf:rest>
            </rdf:List>
          </rdf:rest>
                </rdf:List>
              </owl:oneOf>
              </owl:DataRange>
      </rdfs:range>
   </DatatypeProperty>

<Class rdf:ID="Distinção">
      <rdfs:label>Distinção</rdfs:label>
</Class>

   <DatatypeProperty rdf:ID="título">
      <rdfs:domain rdf:resource="#Distinção" />
      <rdfs:range rdf:resource="&xsd;string" />
      <rdf:type rdf:resource="&owl;FunctionalProperty" />
   </DatatypeProperty>

<Class rdf:ID="Medalha">
      <rdfs:label>Medalha</rdfs:label>
      <owl:subClassOf rdf:resource="#Distinção" />
</Class>
```

Existe uma classe `Cidadão` com duas subclasses `Civil` e `Militar`, cada uma com as suas propriedades. A classe `Medalha` vai herdar as propriedades da sua superclasse, tal como `Civil` e `Militar` herdam a propriedade `nome`. A propriedade `posto` é definida com um `rdfs:range` onde se apresenta um conjunto de valores possíveis que poderá tomar. A linguagem OWL permite este e outro tipo de indicações e restrições aos valores das propriedades. Nas próximas secções veremos como utilizar esta informação.

A ideia é obter instâncias OWL para as classes da ontologia externa. Este processo é compreende três passos:

1. tradução da ontologia para Prolog
2. introdução de regras e inferência de instâncias
3. produção de código OWL para as instâncias

6.3.1 Tradução da Ontologia para Prolog

Para lidar com a estrutura das classes e simultaneamente com a informação semântica das frases era necessário:

- uma representação comum
- capacidade de inferência

Assim, optou-se por uma tradução da ontologia para a linguagem de programação em lógica usada pelo sistema e que já fora utilizada na DRS das frases.

A tradução é realizada por uma aplicação desenvolvida em Java, utilizando algumas ferramentas auxiliares para fazer parse a código OWL. O resultado é um ou mais factos Prolog para cada ocorrência de uma classe ou propriedade definida em OWL.

O resultado em Prolog para a definição de uma classe é um facto com o nome e respectivo URL[4], que a identifica inequivocamente:

```
class( cidadão, 'external.owl#Cidadão' ).
class( civil, 'external.owl#Civil' ).
class( militar, 'external.owl#Militar' ).
class( distinção, 'external.owl#Distinção' ).
class( medalha, 'external.owl#Medalha' ).
```

[4]Para facilitar a leitura omite-se o prefixo do URL. Onde se lê `external.owl` entenda-se `http://abc.di.uevora.pt/jsaias/owl/external.owl`

A noção de subclasse é registada com o predicado `subclass`, cujos argumentos são os identificadores URL das classes:

```
subclass( 'external.owl#Civil' , 'external.owl#Cidadão' ).
subclass( 'external.owl#Militar' , 'external.owl#Cidadão' ).
subclass( 'external.owl#Medalha' , 'external.owl#Distinção' ).
```

As propriedades são descritas pelo nome, a classe a que pertencem e o seu tipo, através do predicado `property`:

```
property(nome, 'datatype', 'external.owl#nome',
     'external.owl#Cidadão', 'XMLSchema#string' ).
property(profissão, 'datatype', 'external.owl#profissão',
     'external.owl#Civil', 'XMLSchema#string' ).
property(ramo, 'datatype', 'external.owl#ramo',
     'external.owl#Militar', 'XMLSchema#string' ).
property(título, 'datatype', 'external.owl#título',
     'external.owl#Distinção', 'XMLSchema#string' ).
```

A propriedade `posto` é descrita com mais um predicado. O conjunto de valores possíveis da propriedade é registado para posterior inferência.

```
property(posto, 'datatype' , 'external.owl#posto' ,
     'external.owl#Militar' , 'owl:DataRange' ).

hasPossibleValue( 'external.owl#posto', capitão ).
hasPossibleValue( 'external.owl#posto', tenente ).
hasPossibleValue( 'external.owl#posto', sargento ).
```

6.3.2 Introdução de Regras e Inferência de Instâncias

O segundo passo consiste na adição de regras de programação em lógica. Estas regras visam a inferência de propriedades, instâncias e relações entre instâncias, a partir da representação semântica parcial das frases dos documentos em DRS e também dos termos Prolog da ontologia, criados no passo anterior.
Por exemplo, no caso de uma frase acerca de um militar, o sistema vai procurar caracterizar essa possível instância com valores para as propriedades definidas na ontologia, que no caso são `ramo` e `posto`. Consideremos a seguinte frase:

O capitão salvou a criança.

A sua DRS vai ser:

```
sentence(ex1, [ capitão(X), criança(Y), salvar(X,Y) ],
              [ ref(X), ref(Y) ] ).
```

A partir desta representação, obtida pelo processo descrito no capítulo 5, juntamente com a informação da ontologia e algumas regras Prolog, é possível identificar uma instância e o valor de uma propriedade.

A regra mais geral que se aplica vai percorrer e analisar todas as frases do documento, representadas com o predicado `sentence`, em busca de instâncias para a ontologia externa.

```
instanceFinder:- sentence(Id, A, R),
    checkSentence(Id, A, R),
    fail.
instanceFinder.
```

Para analisar o conjunto das asserções `A` sobre os referentes `R` na DRS da frase, existem regras dedicadas à parte do sujeito, predicado e complemento. O exemplo seguinte apresenta o caso em que surge uma palavra que corresponde ao nome de uma classe. Isto desencadeia a busca de valores para as propriedades dessa classe.

```
checkSubject(SentID, Subj, [SA| _], [ClassURI,InstID,VE1]):-
    Subj=..[E1,VE1], class(E1,ClassURI),
    checkDataPropValue(E1,VE1,ClassURI, [] ,SentID,[SA],InstID), !.

checkDataPropValue(E1,VE1,ClassURI,SDetail,SentID,ADJ,FinalID):-
    ...
    propAbduction(...),
    classHasDatatypeProperty(E1,ClassURI,P,PropURI),
    dataPropertyValue(P,PropURI,VE1,SDetail).
```

O predicado `classHasDatatypeProperty` procura propriedades que podem ter sido definidas na classe identificada por `classURI` ou numa classe acima na hierarquia, daí a utilização de `chainClass/2`.

```
classHasDatatypeProperty(C,ClassURI,P,PropURI):-
    property4Class(P1,'datatype',PropURI,ClassURI,_),
    name(P1,P1L), name(C,CL),
    simpleName(P1L,CL,PL), name(P,PL).
```

```
property4Class(P1,PType,PropURI,ClassURI,Range):-
    chainClass(ClassURI,URI),
    property(P1,PType,PropURI,URI,Range).

chainClass(URI,URI).
chainClass(ClassURI,URI):-
    subclass(ClassURI,URI1), chainClass(URI1,URI).
```

No caso em que existe um sintagma preposicional `Ppp` que se identifica com uma propriedade `P`, pode tomar-se um adjectivo, ou outra expressão que o descreve, como o valor `PVal` inferido para essa propriedade.

```
dataPropertyValue(P,PropURI,VE1,  [PP, rel(VE1,VPP1), PPA |_]  ):-
    PP=..[Ppp,VPP1], PPA=..[PVal,VPP1],
    synonym(P,Ppp),
    write('DATATYPE PROPERTY'),nl,
    write(P),nl,
    write(PropURI),nl,
    write(PVal) ,nl,nl.
```

O predicado `propAbduction` é usado para processos de **abdução** sobre as propriedades de uma instância.
Voltando ao exemplo, sabemos que *capitão* é um valor possível para a propriedade `posto`, na classe `Militar`. A DRS pode ser reescrita de modo a ser tratável pelas regras que procuram instâncias OWL:

```
sentence(ex1, [ militar(X), posto(X1), rel(X,X1),
                            capitão(X1), criança(Y), salvar(X,Y) ],
              [ ref(X), ref(Y), ref(X1) ] ).
```

Na área de Processamento de Língua Natural, este processo é usualmente designado por interpretação pragmática de frases e pode ser visto como um processo de abdução no qual as propriedades (antecedentes) são inferidas dos seus valores (consequentes) [HSAM90].

Por vezes, a palavra que surge na frase tem igual significado a outra que podia mapear com o nome de uma propriedade, mas por ser diferente essa propriedade não seria identificada. Para melhorar o desempenho a este nível, o sistema recorre a um dicionário auxiliar ao processo de inferência:

```
synonym(X,X).
synonym(A,B):- synonymPair(A,B).
synonym(A,B):- synonymPair(B,A).

synonymPair(posto,patente).
synonymPair(capitao,capitão).
```

Adicionando estes predicados Prolog, permitimos que a propriedade `posto` tenha um valor encontrado para *patente*. Por outro lado, no caso da existência de pequenos erros na frase em língua natural, como a falta de acentuação, o sistema capta os dados como se estivessem correctamente escritos.

Com abordagens do mesmo género é possível obter características diferentes das frases em linguagem natural. Uma das possibilidades é relacionar adjectivos ou sintagmas preposicionais com o substantivo no sujeito. Vejamos mais um caso com a DRS da frase:

O sargento do exército salvou a criança.

```
sentence(ex2, [ sargento(X), exército(X1), rel(X,X1),
                                        criança(Y), salvar(X,Y) ],
              [ ref(X), ref(Y), ref(X1) ] ).
```

Relacionando as classes e propriedades da ontologia com a DRS, as regras e técnicas de abdução permitem inferir uma nova representação:

```
sentence(ex2, [ militar(X), posto(X, sargento), ramo(X, exército),
                                        criança(Y), salvar(X,Y) ],
              [ ref(X), ref(Y) ] ).
```

Esta última versão é mais apropriada para a resposta a interrogações sobre o conteúdo dos documentos, no âmbito de um sistema de recuperação de informação que pode ser baseado na ontologia.

6.3.3 Produção de OWL para as Instâncias

A última fase do processo consiste em transformar os resultados obtidos pela interpretação pragmática de cada frase em instâncias da ontologia propriamente ditas, no formato OWL.
Os resultados do passo anterior são considerados sugestões de instâncias, relações

ou propriedades e vão ser consumidos por uma aplicação em Java. Sempre que essas sugestões se possam materializar em OWL é gerado o respectivo código.

Voltando o último exemplo, as sugestões para o sujeito da frase seriam as seguintes:

```
INSTANCE
idmilitar1ex2
http://abc.di.uevora.pt/~jsaias/owl/external.owl#Militar

DATATYPE PROPERTY
http://abc.di.uevora.pt/~jsaias/owl/external.owl#posto
sargento

DATATYPE PROPERTY
http://abc.di.uevora.pt/~jsaias/owl/external.owl#ramo
exército
```

Isto representa uma instância de `Militar` com um identificador *idmilitar1ex2* e para a qual foram inferidas duas propriedades. Estas indicações são transformadas no código OWL:

```
<extont:Militar rdf:ID="idmilitar1ex2">
    <extont:posto>sargento</extont:posto>
    <extont:ramo>exército</extont:ramo>
</extont:Militar>
```

O novo código, automaticamente gerado, estende a ontologia fornecida ao sistema, com as instâncias para aquelas classes. O prefixo `extont` é um XML namespace que indica onde estão definidas a classe `Militar` e as propriedades `posto` e `ramo`.
Se na ontologia existissem uma classe `Ramo` com uma subclasse `Exército` e a propriedade `ramo` fosse uma ObjectProperty do tipo `Ramo`, o código anterior podia ainda ser reescrito em:

```
<extont:Militar rdf:ID="idmilitar1ex2">
    <extont:posto>sargento</extont:posto>
    <extont:ramo rdf:resource="&extont;Exército" />
</extont:Militar>
```

Esta ontologia pode coexistir com a anterior. É possível reorganizar manualmente a ontologia base, combinando-a com a ontologia externa, e associar instâncias

de ambos os lados. Por exemplo, se a classe `Militar` descender de `Entity`[5], então será possível utilizar a instância anterior no campo `subject` de uma acção da ontologia base:

```
<pgre:salvar rdf:ID="e32">
    <pgr:entName>salvar</pgr:entName>
    <pgr:entDoc rdf:resource=
             "http://abc.di.uevora.pt/~jsaias/owl/docs/ex2#ex2" />
</pgre:salvar>

<pgre:criança rdf:ID="e33">
    <pgr:entName>criança</pgr:entName>
    <pgr:entDoc rdf:resource=
             "http://abc.di.uevora.pt/~jsaias/owl/docs/ex2#ex2" />
</pgre:criança>

<pgr:Action rdf:ID="action-ex2_1">
   <pgr:subject rdf:resource="#idmilitar1ex2" />
   <pgr:verb rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docentities/ex2#e32" />
   <pgr:object rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docentities/ex2#e33" />
   <pgr:actionDoc rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docs/ex2#ex2" />
</pgr:Action>
```

Vejamos outros exemplos de instâncias OWL para a ontologia externa, retiradas de algumas frases dos documentos:

1. Duas instâncias, uma relativa ao sujeito e outra ao complemento onde se encontra uma propriedade herdada para a classe `Medalha`.

   O militar de posto tenente recebe uma medalha a título de honra.

   ```
   <extont:Militar rdf:ID="idmilitar33P001">
       <extont:posto>tenente</extont:posto>
   </extont:Militar>

   <extont:Medalha rdf:ID="idmedalha33P001">
       <extont:título>honra</extont:título>
   </extont:Medalha>
   ```

[5]Porque a propriedade `subject` definida na classe `Action` toma valores do tipo `Entity` (secção 6.2.1).

Como já foi dito, estas instâncias podem ser relacionadas por uma acção, em que o verbo, neste caso, é *receber*:

```
<pgre:receber rdf:ID="e400">
    <pgr:entName>receber</pgr:entName>
    <pgr:entDoc rdf:resource=
        "http://abc.di.uevora.pt/~jsaias/owl/docs/P001#P001"/>
</pgre:receber>

<pgr:Action rdf:ID="action-P001_1">
   <pgr:subject rdf:resource="#idmilitar33P001" />
   <pgr:verb rdf:resource=
   "http://abc.di.uevora.pt/~jsaias/owl/docentities/P001#e400"/>
   <pgr:object rdf:resource="#idmedalha33P001" />
   <pgr:actionDoc rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docs/P001#P001" />
</pgr:Action>
```

2. Tratamento especial do verbo *ter* por acrescentar informação sobre o sujeito. Uso de dicionário para auxiliar a identificação do nome da propriedade `posto` pelo sinónimo *patente*.

   O militar tem a patente de general.

   ```
   <extont:Militar rdf:ID="idmilitar34P001">
       <extont:posto>general</extont:posto>
   </extont:Militar>
   ```

3. Caso em que a classe e propriedade são conseguidas por abdução, já que não há palavras na frase para mapear directamente com os nomes da classe ou propriedades.

   A Maria viu o capitão.

   ```
   <extont:Militar rdf:ID="idmilitar35P001">
       <extont:posto>capitão</extont:posto>
   </extont:Militar>
   ```

   A análise sintáctica identifica *Maria* como um nome próprio. Com uma indicação especial no dicionário, é possível deduzir que este nome corresponde ao nome de uma pessoa ou cidadão:

```
<extont:Cidadão rdf:ID="idcidadão35P001">
    <extont:nome>Maria</extont:nome>
</extont:Cidadão>
```

Tal como antes, estas instâncias podem ser relacionadas por uma acção.

4. Uso do dicionário para ajudar a encontrar a propriedade a partir da respectiva lista de valores possíveis. A palavra contida na frase tem um pequeno erro de acentuação, o que é ultrapassado pela aplicação correctiva do dicionário.

   A Maria viu o capitao.

```
<extont:Cidadão rdf:ID="idcidadão35P001">
    <extont:nome>Maria</extont:nome>
</extont:Cidadão>

<extont:Militar rdf:ID="idmilitar35P001">
    <extont:posto>capitão</extont:posto>
</extont:Militar>
```

5. Exemplo que mistura várias técnicas apresentadas. A análise à frase resulta na identificação de três propriedades para `Militar` e uma para `Medalha`.

   O sargento do exército teve actividade heróica e recebeu uma medalha a título de honra por isso.

```
<extont:Militar rdf:ID="idmilitar36P001">
    <extont:posto>sargento</extont:posto>
    <extont:ramo>exército</extont:ramo>
    <extont:actividade>heróica</extont:actividade>
</extont:Militar>

<extont:Medalha rdf:ID="idmedalha36P001">
    <extont:título>honra</extont:título>
</extont:Medalha>

<pgre:receber rdf:ID="e491">
    <pgr:entName>receber</pgr:entName>
    <pgr:entDoc rdf:resource=
        "http://abc.di.uevora.pt/~jsaias/owl/docs/P001#P001"/>
</pgre:receber>
```

```
<pgr:Action rdf:ID="action-P001_28">
   <pgr:subject rdf:resource="#idmilitar36P001" />
   <pgr:verb rdf:resource=
   "http://abc.di.uevora.pt/~jsaias/owl/docentities/P001#e491"/>
   <pgr:object rdf:resource="#idmedalha36P001" />
   <pgr:actionDoc rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docs/P001#P001" />
</pgr:Action>
```

Depois de aplicar o processo aqui descrito ao conjunto completo dos textos, os documentos já representados em OWL são enriquecidos com estas instâncias da ontologia, obtidas pela interpretação pragmática do seu conteúdo.

Capítulo 7

Motor de Inferência

Este capítulo descreve o processo de inferência utilizado para responder a interrogações sobre os documentos.

As secções 7.1 e 7.2 descrevem as ferramentas utilizadas, num contexto de programação em lógica. A secção 7.3 é dedicada à análise da interrogação e a última secção, 7.4, inclui exemplos de interrogações ao sistema.

7.1 GNU Prolog e ISCO

Com a informação semântica recolhida do texto dos documentos vão fazer-se operações como:

- manipulação de informação
- validação ou prova de asserções
- inferência
- interrogações

Para realizar este género de operações de um modo eficaz, optou-se pela utilização de uma linguagem declarativa de programação em lógica que permite a inferência dedutiva sobre cláusulas, Prolog. Mais especificamente, a ferramenta utilizada é **GNU Prolog**.

Enquanto outras ferramentas são apenas interpretadores Prolog, o GNU Prolog é efectivamente um compilador de código Prolog, baseado na Warren Abstract Machine (WAM)[War83]. O código Prolog é compilado para instruções WAM, que

por sua vez são traduzidas para uma linguagem máquina independente de baixo nível. Por último, essa linguagem é compilada para código nativo da máquina.
A vantagem de produzir ficheiros executáveis *stand alone* é conseguir melhor performance, nomeadamente ao nível da velocidade, relativamente a um interpretador.

Outra característica do GNU Prolog é um eficiente *constraint solver* sobre domínios finitos, que permite ao utilizador tirar partido da junção entre programação por restrições e a declaratividade da programação em lógica.

ISCO [Abr01] é uma linguagem implementada sobre GNU Prolog, oferecendo as seguintes possibilidades:

- Funcionalidades *Object Oriented*: classes, hierarquia, herança.
- Programação Lógica por Restrições: queries com restrições de domínio finito.
- Armazenamento em bases de dados relacionais via ODBC. Tem um *backend* para PostgreSQL e Oracle.
- Acesso a bases de dados relacionais externas, tratando-as como bases de dados declarativas/dedutivas e *object oriented*. Entre outras possibilidades, mapeia tabelas da BD em classes. As classes vão corresponder a predicados Prolog.
- Usa uma linguagem simples para descrever a estrutura da base de dados. Existem ferramentas para obter a descrição ISCO da BD a partir do schema da BD relacional e também para a operação contrária.

Um predicado em ISCO é similar a um predicado em Prolog com uma pequena *nuance* ao nível da implementação. Em ISCO o predicado pode ser proveniente de um repositório exterior ao Prolog, como uma base de dados, o que permite ter persistência nos factos e eficiência no tratamento de grandes quantidades de dados.

O GNU Prolog foi adoptado desde o início do trabalho como a linguagem a utilizar para as operações do sistema que envolvem o conhecimento. Numa segunda fase, realizaram-se algumas experiências envolvendo também ISCO, aproveitando a possibilidade de ligar o Prolog com uma base de dados. Deste modo, as diversas instâncias associadas aos documentos podem estar em bases de dados relacionais, ultrapassando possíveis problemas de escalabilidade.
Eis o código SQL para uma tabela:

```
CREATE TABLE document (
  numero int4 NOT NULL,
  titulo text,
  relator text,
  PRIMARY KEY (number)
);
```

Esta tabela em PostgreSQL vai mapear numa classe em ISCO definida por:

```
external(pgr) class document.
  numero: int. key.
  titulo: text.
  relator: text.
```

Foi elaborada uma ferramenta para construir a definição das classes ISCO a partir das classes de uma ontologia em OWL. Aplicando este processo à ontologia de classes gerada a partir do conteúdo dos documentos obtém-se o código SQL para as tabelas e o código ISCO para as respectivas classes. Por outro lado, cada instância de uma classe na ontologia é transformada num registo da tabela PostgreSQL relativa aquela classe e ainda no facto de programação em lógica ISCO correspondente.

Para o caso concreto de uma instância de `Action`, como:

```
<pgr:Action rdf:ID="a1">
   <pgr:subject rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docentities/doc1#e13" />
   <pgr:verb rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docentities/doc1#e14" />
   <pgr:object rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docentities/doc1#e15" />
   <pgr:actionDoc rdf:resource=
       "http://abc.di.uevora.pt/~jsaias/owl/docs/d1#d1" />
</pgr:Action>
```

O facto ISCO que a vai representar é:

```
action(ID=a1, subject='#e13', object='#e15',
   verb='#e14', actionDoc='#d1').
```

As declarações em ISCO podem envolver herança, restrições simples de integridade sobre o domínio e restrições globais. Existem predicados Prolog que permitem operações básicas query, insert, update e delete, e que se podem aplicar a cada

classe.
As variáveis de uma interrogação ISCO podem ter restrições associadas (programação lógica por restrições sobre domínio finito). Se está a ser usada uma base de dados, então essas restrições são traduzidas para a query SQL, sempre que possível.
Por exemplo, consideremos a variável X de domínio finito (1..1000), na interrogação:

```
document(number=X, title=Y).
```

O resultado será um conjunto de pares (X,Y) em que X é o número do documento e Y o respectivo título. Todos os resultados observam a restrição do domínio de X, isto é, só são vistos os documentos com número entre 1 e 1000.
Para um desempenho óptimo, o código ISCO pode ser compilado, resultando num executável GNU Prolog.

7.2 EVOLP

Até aqui vimos como usar a linguagem ISCO para obter uma representação declarativa das classes e instâncias da ontologia. Contudo, há também a necessidade de representar acções e modelar a evolução do conhecimento.

Em [APP+99] foi apresentada uma linguagem declarativa de alto nível para a actualização do conhecimento (knowledge update) designada LUPS (Language of UPdateS) que descreve a transição entre estados de conhecimento consecutivos.
Posteriormente, foi proposta uma nova linguagem em que a formulação da actualização lógica resulta mais simples. Essa linguagem designa-se **EVOLP** [ABLP02] e é resumidamente descrita nesta secção para se compreender a sua aplicação no sistema. Para uma descrição mais detalhada da linguagem recomenda-se o artigo citado.

EVOLP permite especificar a evolução de um programa através de regras que introduzem asserções ao programa. Os programas EVOLP consistem em conjuntos de regras lógicas definidas sobre uma linguagem proposicional estendida L_{assert}, que por sua vez pode ser definida sobre qualquer linguagem proposicional L onde não existe a noção de *assert*, do seguinte modo[1]:

- Todos os átomos proposicionais L são átomos proposicionais em L_{assert}

[1]Para uma explicação mais detalhada recomenda-se a leitura do artigo citado, [ABLP02].

- Se cada $L_0, \ldots, L_n$ é um literal em L_{assert}, então $L_0 \leftarrow L_1, \ldots, L_n$ é uma regra lógica do programa em L_{assert}.
- Se R é uma regra em L_{assert} então $assert(R)$ é um átomo proposicional em L_{assert}
- Nada mais é um átomo proposicional em L_{assert}

A definição formal da semântica de EVOLP pode consultar-se no artigo referido, mas a ideia é a seguinte: sempre que o átomo $assert(R)$ pertence à interpretação, isto é, pertence a um modelo de acordo com a semântica do modelo estável do programa actual, então R deve pertencer ao programa no estado seguinte.
Por exemplo, a regra seguinte:

$$assert(b \leftarrow a) \leftarrow c \tag{7.1}$$

significa que se c é verdadeiro num estado, então o próximo estado deve ter a regra $b \leftarrow a$.

Em EVOLP também existe a noção de eventos externos, ou por outras palavras, asserções que deixam de persistir por inércia ou o contrário, introdução de novas asserções. Esta noção é fundamental para um modelo de interacção entre agentes e para representar acções. É importante que através de uma regra se consigam representar os efeitos de uma acção, tendo em conta determinadas condições prévias. Exemplo:

$$assert(Effect) \leftarrow Action, PreConditions \tag{7.2}$$

Para um determinado estado, se existir um evento *Action* e ao mesmo tempo se verificarem as condições *PreConditions*, então o estado seguinte vai ter *Effect*.

A próxima secção mostra como esta linguagem e as ferramentas anteriores se aplicam ao sistema.

7.3 Tratamento de Interrogações

Primeiro construiu-se uma representação dos documentos através de uma ontologia de classes relativas a conceitos, bem como as instâncias dessas classes referidas no texto. Este é o formato apropriado para publicar um documento na Web, aliando o texto em língua natural para leitura por humanos às anotações semânticas numa linguagem Semantic Web.

Para o sistema conseguir fazer deduções lógicas sobre a informação no formato OWL, foi efectuada uma replicação dessa informação para um modelo **ISCO / GNUProlog + EVOLP**, que é o ambiente adoptado no sistema para a evolução do conhecimento e inferência.
Um objectivo é conseguir responder a interrogações do género:

1. Quais os documentos em que ocorre a acção A?
2. Quais os documentos em que S é agente de uma acção A?
3. Quem foi o agente da acção A sobre C?
4. Quais os agentes da acção A que também sejam agentes da acção B?
5. Quais os documentos em que S é agente de uma acção?
6. Quais as acções referidas no documento D?

Note-se que o motor de inferência vai necessariamente percorrer as relações da ontologia. Pegando no caso em que pretendemos "os documentos onde S é agente da acção A", isso significa procurar os documentos em que uma acção A ou desse tipo (instância de A ou de uma subclasse de A) e contenha um sujeito S ou do tipo S (instância de S ou, eventualmente, de uma subclasse de S).

O processamento de uma interrogação envolve duas tarefas:

- análise da interrogação
- interacção com a base de conhecimento

7.3.1 Análise da Interrogação

A interrogação ao sistema pode incidir directamente sobre a ontologia ou sobre uma base de dados associada ao ISCO, pode ser uma interrogação em prolog, ou pode ser uma interrogação escrita em língua natural pelo utilizador. O último caso é o mais relevante e esta secção é dedicada a esse caso.
A interrogação em língua natural é sujeita a análises sintáctica e semântica. Em seguida faz-se uma interpretação pragmática.

Análise Sintáctica

O texto da interrogação é analisado com o analisador sintáctico desenvolvido por E. Bick, já antes referido [Bic00]. O output em formato VISL é traduzido para factos Prolog, aplicando a mesma ferramenta utilizada na secção 5.2.2, descrita em [GVGQ03]. Vejamos um exemplo com uma possível interrogação do utilizador do sistema:

Quais os documentos em que bombeiros salvaram crianças?

Esta interrogação é traduzida para uma estrutura sintáctica em Prolog:

```
sentence(
  sc(pron_det('qual','M/F','P','<interr>'),'Quais'),
  subj(np),
  n(art('o','M','P'),'os'), h(n('documento','M','P'),'documentos'),
  advl(prp('em'),'em'),
  acc(fcl,
    sub(conj_s('que'),'que'),
    subj(n('bombeiro','M','P'),'bombeiros'),
    p(v_fin('salvar','PS/MQP','3P','IND'),'salvaram'),
    acc(n('criança','F','P'),'crianças','?'))).
```

Analisando o código, destaca-se a etiqueta fcl, que marca uma oração finita (Finite CLause) dentro da questão (*"que bombeiros salvaram crianças"*). O pronome determinativo *qual* surge associado ao nome *documento*, identificando o que se procura com a interrogação.

Análise Semântica

À semelhança do que já antes se fez com as frases dos documentos, na secção 5.3.2, a estrutura sintáctica da interrogação é traduzida para uma expressão em Lógica de Primeira Ordem. Essa expressão vai ter a representação semântica da interrogação efectuada, através de uma DRS incluindo duas listas, uma com a frase reescrita e a outra com os referentes do discurso (ver secção 5.3).
A representação semântica para a frase da interrogação mostrada como exemplo será uma expressão com os elementos:

```
documento(A),
bombeiro(B), salvar(C), criança(D)
action(E),
rel(E, B, C, D)
```

```
rel(A, E)
```

e uma lista com os referentes do discurso:

```
[ ref(A), ref(B), ref(C), ref(D), ref(E) ]
```

Esta estrutura envolverá os conceitos que na ontologia correspondem a instâncias de bombeiro B, instâncias de criança D e instâncias de salvar C, que são relacionadas com instâncias de action E em documentos D.

Importa salientar que, de momento, o sistema não está preparado para lidar com todo o tipo de interrogações e traduzi-las do formato sintáctico para a estrutura semântica. De facto, este é um complexo problema na área de Processamento de Língua Natural e foi decidido que este trabalho devia focar-se apenas em alguns aspectos da Língua Portuguesa, nomeadamente as interrogações sobre domínios específicos.

Interpretação Pragmática

O sistema tem um módulo de carácter pragmático, que recebe a representação semântica da interrogação e procura interpretar isso no contexto global de toda a informação disponível, que resultou da tradução das instâncias da ontologia em OWL para factos ISCO/GNUProlog, de acordo com o procedimento antes descrito.

Para conseguir uma interpretação, o sistema procura encontrar a melhor explicação para a expressão lógica DRS da interrogação ser verdadeira. O sistema vai tentar fazer prova da frase na interrogação usando a base de conhecimento. Como já foi referido, esta estratégia para a interpretação é conhecida como *interpretation as abuction*, apresentada em [HSAM90]. Este processo foi também utilizado num trabalho mais recente apresentado em [QRA01].

Utilizando ISCO e a partir da descrição da ontologia, é possível representar a interrogação de um modo que reflecte o conhecimento dos campos das classes:

```
document(id=A),
bombeiro(id=B),
salvar(id=C),
criança(id=D),
action(id=I, subject=B, verb=C, object=D, actionDoc=A).
```

Esta versão da interrogação é conseguida com regras de programação em lógica adicionais, introduzidas para o processo de inferência. Algumas dessas regras têm a ver com os referentes do discurso, outras com o equivalente às classes e propriedades de classes OWL. Um exemplo:

```
rel(A, B, C, D)  <-  action(A),
                     entity(B),
                     verb(C),
                     entity(D),
                     abduct( action(A,B,C,D,_) ).
```

Note-se que, mais uma vez, a hierarquia da ontologia influencia o modo como se procura uma solução. Procuram-se instâncias de `Action` envolvendo instâncias de `Entity` (que no caso seriam instâncias das subclasses `Bombeiro` e `Criança`).

A interpretação dos predicados ISCO é levada a cabo com o acesso à base de dados, coleccionando instâncias e eventualmente restringindo identificadores. Algumas restrições possíveis para a interrogação ISCO acima:

```
- $I=_\#(104..109:156..157)$
- $A=_\#(123:145)$
```

Ao limitar a variável I a um conjunto de valores, vamos considerar apenas as acções em que o `id` pertence a esse conjunto. Os documentos onde que se procuram as acções podem ser limitados a um grupo mais reduzido, bastando restringir a variável A. Estas restrições limitam a pesquisa à base de dados no processo de interpretação da interrogação, o que poderá ser útil para acelerar o processo.

7.3.2 Interacção com a Base de Conhecimento

Para aceder à base de conhecimento, por exemplo no processamento de uma interrogação, é necessário representar efectivamente as acções associadas com a interacção ($inform$ ou $request$) e construir um modelo de intenções e crenças do utilizador ($intentions$ and $beliefs$).

Esta tarefa é alcançada mediante a utilização da linguagem EVOLP. Para uma descrição detalhada do procedimento consultar [QR01a] e [QL95]. Na prática, as regras que definem o efeito de uma acção $inform$ ou uma acção $request$ são:

$$assert(bel(A, bel(B, P))) \leftarrow inform(B, A, P). \tag{7.3}$$

$$assert(bel(A, int(B, Action))) \leftarrow request(B, A, Action). \tag{7.4}$$

Estas regras significam que, se um agente A é informado de uma propriedade P, então irá começar a acreditar que outro agente também acredita em P, e por outro lado, se B solicita a A que efectue a acção $Action$, então A passa a acreditar que B pretende que a acção seja efectuada.

Para expressar um comportamento colaborativo é ainda necessário modelar a transferência de informação entre agentes:

$$assert(bel(A, P)) \leftarrow bel(A, bel(B, P)). \tag{7.5}$$
$$assert(int(A, Action)) \leftarrow bel(A, int(B, Action)). \tag{7.6}$$

Isto significa que, se um agente A acredita que outro agente acredita em P, então A passa a acreditar em P também (assume verdadeira a crença dos interlocutores); para além disso o agente A vai adoptar para si as intenções que crê que outros agente têm.

Existe ainda a necessidade de regulamentar, definir regras para, a relação entre as intenções do sistema e os acessos à base de conhecimento:

$$assert(inf(A, B, P)) \leftarrow int(A, inf(A, B, P)), isco(P). \tag{7.7}$$
$$assert(not\ int(A, B, inf(A, B, P))) \leftarrow inf(A, B, P). \tag{7.8}$$

Com a primeira regra define-se que, se o sistema tem a intenção de informar o utilizador acerca de uma propriedade, então o sistema usa a base de dados em ISCO e realiza uma operação $inform$. Na segunda regra diz-se que a execução da acção $inform$ termina a intenção de efectuar o $inform$.

7.4 Exemplo de Interrogações ao Sistema

Recuperando o exemplo da interrogação:

Quais os documentos em que bombeiros salvaram crianças?

O sistema efectua análises sintáctica e semântica ao texto da interrogação, que após a interpretação pragmática é representada por:

```
Q = [ document(id=A),
      bombeiro(id=B), salvar(id=C), criança(id=D),
      action(id=I, subject=B, verb=C, object=D, actionDoc=A) ].
```

Formulada a representação lógica da interrogação, a interacção com a base de conhecimento inicia-se com uma acção *request*:

```
request(user, system, inform(user, system, Q))
```

De acordo com as regras da linguagem EVOLP antes apresentadas, designadamente sobre a transferência de intenções, este pedido de informação por parte do utilizador será transformado numa intenção, do sistema, de informar o utilizador sobre Q:

```
int(system, inform(system, user, Q))
```

Em seguida, e ainda de acordo com as regras, para o sistema realizar o $inform$ é necessário consultar a base de dados em ISCO, de modo a alcançar na resposta um conjunto de restrições sobre as variáveis de Q (relativas aos referentes do discurso), mediante as quais Q se pode provar verdadeiro. O resultado pode ser:

```
- $I=_\#(107..109:151..153)$
- $A=_\#(133:140)$
```

Pelas restrições inferidas da base de conhecimento é possível identificar o conjunto de soluções para a pergunta. O sistema pode agora responder:

> *São os documentos com id 133 e 140.*

Se a pergunta for elaborada de uma forma mais geral, como:

> *Quais os documentos em que crianças foram salvas?*

A única diferença vai ser a interpretação pragmática da interrogação (e provavelmente os resultados, com polícias, médicos ou outras entidades a salvarem crianças, de acordo com os documentos):

```
Q = [ document(id=A),
      salvar(id=B), crianças(id=C),
      action(id=I, verb=B, object=C, actionDoc=A) ].
```

Capítulo 8

Utilização do Sistema

Este capítulo é dedicado à utilização do sistema, ao modo como o utilizador interage com o mesmo e às experiências efectuadas.
A secção 8.1 descreve a interface Web desenvolvida para o sistema. Na secção 8.2 são apresentados alguns casos onde o sistema foi aplicado.

8.1 Interface Web

Antes de aplicar o sistema ao texto dos documentos jurídicos, foram feitos alguns testes com frases e textos mais curtos. Como a aplicação do sistema a um texto, com o objectivo de obter elementos para uma ontologia, é uma operação recorrente, foi elaborada uma interface Web, sobre Java[1], para facilitar todo o processo.
Para além de ser útil na fase de desenvolvimento torna possível o processamento de outros textos, introduzidos pelo utilizador, para além dos extensos documentos jurídicos.

Para processar um texto, o utilizador só tem de introduzir as frases numa *textbox* e submeter o formulário.
A figura 8.1 contém o formulário de entrada, onde o utilizador inseriu a frase:

O bombeiro da localidade salvou a criança indefesa.

[1] A interface é baseada em Java Servlets, juntamente com um Apache Web Server.

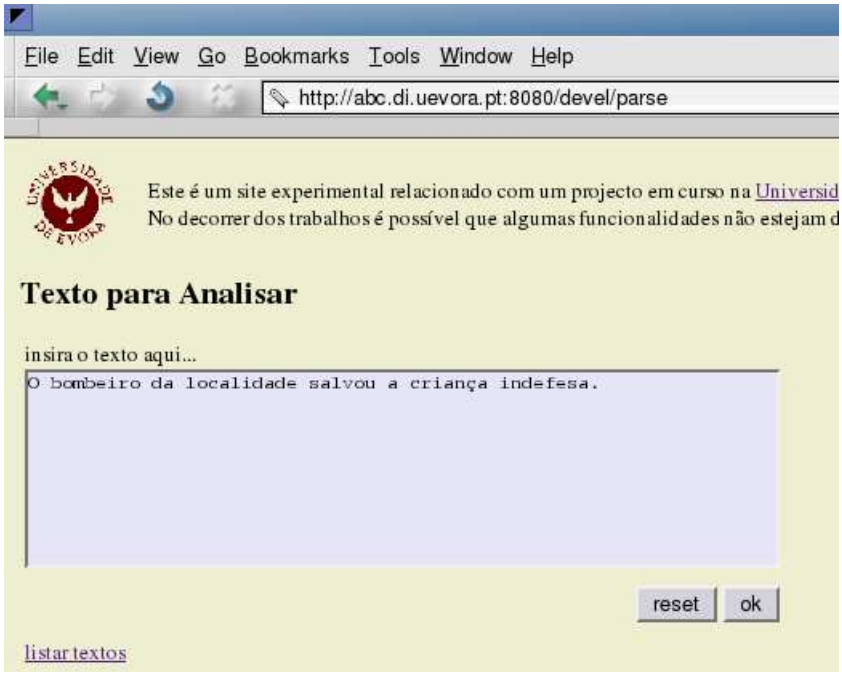

Figura 8.1: Introdução de texto pelo utilizador

Existe uma opção para listar o conjunto de textos já processados (que veremos mais adiante) e os botões do formulário. A submeter o texto (botão *Ok*), os dados são enviados via *HTTP POST* para uma *Java Servlet*, que por sua vez faz o processamento inicial junto do sistema.

Após o envio da frase anterior, o utilizador é informado do resultado do processo, conforme indicado na figura 8.2. Nessa altura, o utilizador pode optar entre:

- voltar atrás e inserir outros textos, seguindo o mesmo procedimento
- efectuar a análise sintáctica ao texto

A análise sintáctica correspondente à frase do exemplo é apresentada na figura 8.3. Estes resultados são obtidos com o analisador VISL. Podem identificar-se os substantivos e o verbo, devidamente associados ao sujeito ($SUBJ$), ao predicado (P) e ao complemento (ACC) da frase.

Figura 8.2: Resultado do envio de um texto ao sistema

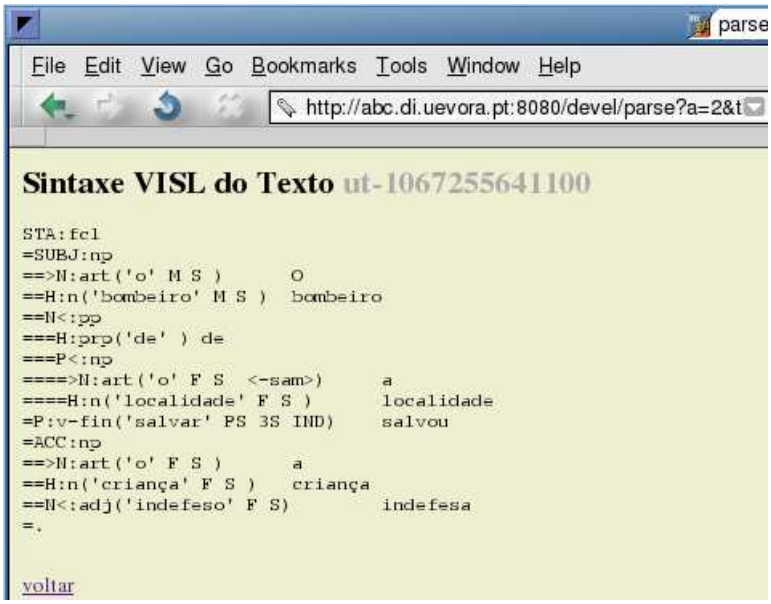

Figura 8.3: Análise sintáctica ao texto do utilizador

Na fase seguinte, o utilizador pode ver uma lista com todos os textos inseridos, como mostra a figura 8.4.

Em cada coluna existe um *link*, a que corresponde determinada operação relacionada com o texto. A primeira opção, da esquerda para a direita, serve para voltar a consultar o texto introduzido. A figura 8.5 ilustra essa opção.

Figura 8.4: Lista de opções sobre os textos inseridos

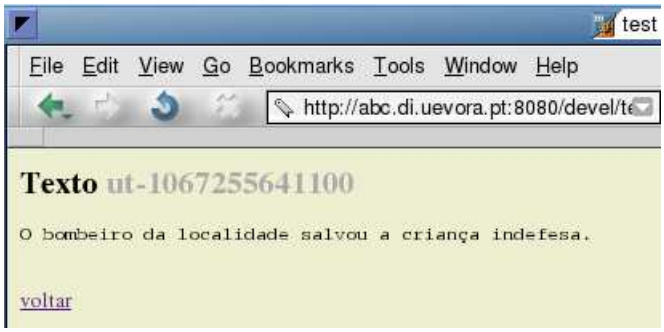

Figura 8.5: Opção de consulta de um texto inserido

As restantes opções permitem ao utilizador acompanhar cada passo intermédio do processo. A tradução da representação sintáctica VISL para factos Prolog, conforme descrito no capítulo 5, pode ser vista na figura 8.6.

Pela estrutura sintáctica da frase, escolhem-se as entidades a usar na representação semântica parcial. As entidades seleccionadas no exemplo estão na figura 8.7.

Figura 8.6: Sintaxe em Prolog

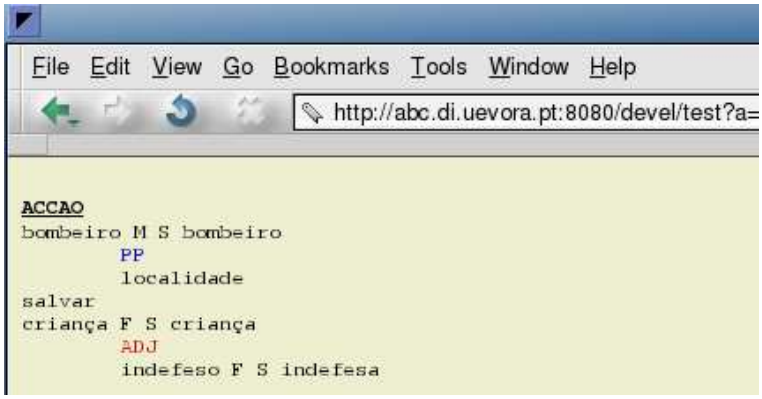

Figura 8.7: Entidades seleccionadas do texto

Na opção *acções em Prolog* é possível ver uma versão simplificada das estruturas DRS (secção 5.3.2) que irão guardar a informação semântica parcial das frases do texto, como se pode ver na figura 8.8. Cada DRS aparece representada por um facto Prolog (e pode ser vista como uma acção efectuada pelo sujeito relativamente ao complemento).

Figura 8.8: Versão simplificada das DRS que representam o texto

As duas últimas opções do quadro da figura 8.4 permitem ver o código OWL com instâncias, de classes da ontologia, obtidas no respectivo texto. As instâncias de subclasses de `Entity` (de acordo com a secção 6.2), que representam entidades encontradas no texto do exemplo são apresentadas em seguida.

```
<!DOCTYPE owl [
  <!ENTITY owl ‘‘http://www.w3.org/2002/07/owl#’’ >
  <!ENTITY pgr ‘‘http://abc.di.uevora.pt/~jsaias/owl/pgr.owl#’’ >
  <!ENTITY pgre ‘‘http://abc.di.uevora.pt/~jsaias/owl/entity.owl#’’>
  <!ENTITY xsd ‘‘http://www.w3.org/2000/10/XMLSchema#’’ >   ] >

<rdf:RDF
  xmlns:owl =’’http://www.w3.org/2002/07/owl#’’
  xmlns:rdf =’’http://www.w3.org/1999/02/22-rdf-syntax-ns#’’
  xmlns:rdfs=’’http://www.w3.org/2000/01/rdf-schema#’’
  xmlns:xsd =’’http://www.w3.org/2000/10/XMLSchema#’’
  xmlns:pgr =’’http://abc.di.uevora.pt/~jsaias/owl/pgr.owl#’’
  xmlns:pgre =’’http://abc.di.uevora.pt/~jsaias/owl/entity.owl#’’
>

<pgre:bombeiro rdf:ID=’’e1’’>
     <pgr:entName>bombeiro</pgr:entName>
```

```
     <pgr:entDoc rdf:resource=''http://abc.di.uevora.pt/~jsaias/
owl/docs/ut-1067255641100#ut-1067255641100'' />
     <pgr:entModifier>
        <pgr:Modifier rdf:ID=''mPPe1''>
             <pgr:modType>Prepositional</pgr:modType>
             <pgr:modValue>localidade</pgr:modValue>
        </pgr:Modifier>
     </pgr:entModifier>
</pgre:bombeiro>

<pgre:salvar rdf:ID=''e2''>
     <pgr:entName>salvar</pgr:entName>
     <pgr:entDoc rdf:resource=''http://abc.di.uevora.pt/~jsaias/
owl/docs/ut-1067255641100#ut-1067255641100'' />
</pgre:salvar>

<pgre:criança rdf:ID=''e3''>
     <pgr:entName>criança</pgr:entName>
     <pgr:entDoc rdf:resource=''http://abc.di.uevora.pt/~jsaias/
owl/docs/ut-1067255641100#ut-1067255641100'' />
     <pgr:entModifier>
        <pgr:Modifier rdf:ID=''mAe3''>
             <pgr:modType>Adjective</pgr:modType>
             <pgr:modValue>indefeso</pgr:modValue>
        </pgr:Modifier>
     </pgr:entModifier>
</pgre:criança>

</rdf:RDF>
```

A acção no mesmo exemplo e envolvendo as referidas entidades é uma instância de `Action`, representada em OWL por:

```
<!DOCTYPE owl [
  <!ENTITY owl ‘‘http://www.w3.org/2002/07/owl#'' >
  <!ENTITY pgr ‘‘http://abc.di.uevora.pt/~jsaias/owl/pgr.owl#'' >
  <!ENTITY pgre ‘‘http://abc.di.uevora.pt/~jsaias/owl/entity.owl#''>
  <!ENTITY xsd ‘‘http://www.w3.org/2000/10/XMLSchema#'' >   ] >

<rdf:RDF
  xmlns:owl =''http://www.w3.org/2002/07/owl#''
```

```
  xmlns:rdf =''http://www.w3.org/1999/02/22-rdf-syntax-ns#''
  xmlns:rdfs=''http://www.w3.org/2000/01/rdf-schema#''
  xmlns:xsd =''http://www.w3.org/2000/10/XMLSchema#''
  xmlns:pgr =''http://abc.di.uevora.pt/~jsaias/owl/pgr.owl#''
  xmlns:pgre =''http://abc.di.uevora.pt/~jsaias/owl/entity.owl#''
>

<pgr:Action rdf:ID=''aut-1067255641100_1''>
  <pgr:subject rdf:resource=''http://abc.di.uevora.pt/~jsaias/
owl/docentities/ut-1067255641100#e1'' />
  <pgr:verb rdf:resource=''http://abc.di.uevora.pt/~jsaias/
owl/docentities/ut-1067255641100#e2'' />
  <pgr:object rdf:resource=''http://abc.di.uevora.pt/~jsaias/
owl/docentities/ut-1067255641100#e3'' />
  <pgr:actionDoc rdf:resource=''http://abc.di.uevora.pt/~jsaias/
owl/docs/ut-1067255641100#ut-1067255641100'' />
</pgr:Action>

</rdf:RDF>
```

8.2 Exemplos

Os documentos jurídicos da PGR são tratados do mesmo modo que os textos do utilizador, com a diferença que estes são mais vastos e já se encontram junto ao sistema.
Para verificar cada fase intermédia do processamento dos documentos, o utilizador dispõe da interface na figura 8.9. Existem duas possibilidades de consulta novas, relativamente aos textos do utilizador. Isto tem a ver com a estrutura específica dos documentos jurídicos analisados.
No formato original dos documentos existe um conjunto de descritores ou palavras-chave sobre o respectivo texto. O utilizador pode ter interesse em comparar esses descritores com a lista de entidades e acções em OWL geradas automaticamente.

Tomando como exemplo o documento com o identificador *P000061997*, que tem a ver com a atribuição do estatuto de trabalhador-estudante a militares da GNR, uma parte do texto é exibida na figura 8.10.

Figura 8.9: Opções para consultar fases intermédias do processamento dos documentos jurídicos

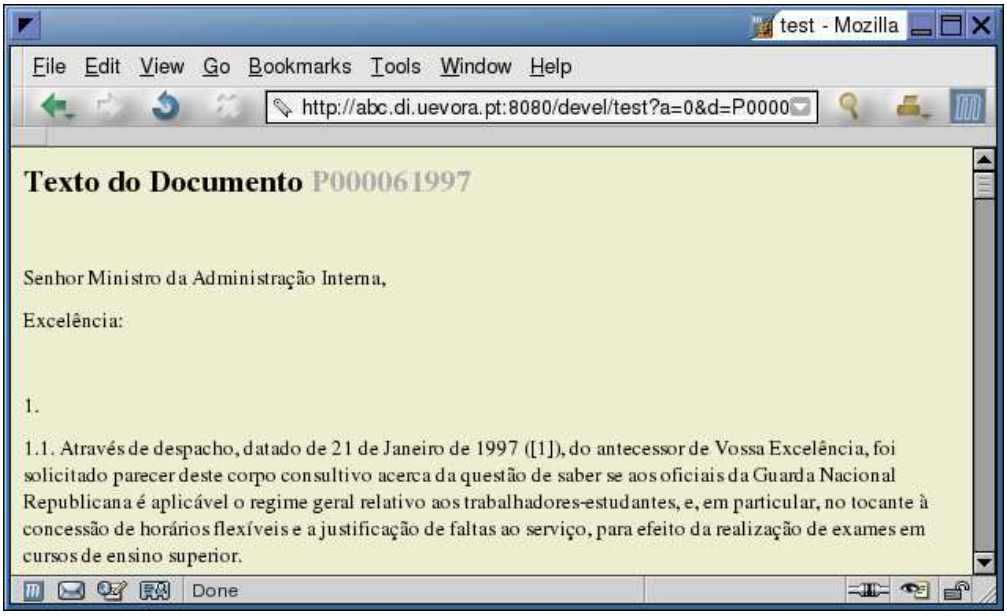

Figura 8.10: Consulta ao texto do documento jurídico

Figura 8.11: Descritores previamente atribuídos ao documento

A figura 8.11 contém uma lista com os descritores do documento anterior. Estes descritores estão também representados na ontologia, conforme já foi descrito na secção 6.1.2, onde se pode ver a classe OWL e as propriedades que relacionam estes conceitos.
Antes da extracção de entidades, do texto para a ontologia, é construída uma representação dos documentos em OWL, através de instâncias de `Document`. A instância tem todo o texto do documento e inclui ainda outros campos, nomeadamente referências para as instâncias de `Concept` com os respectivos descritores e para instâncias de outros documentos relacionados, como se pode ver no exemplo seguinte.

```
<!DOCTYPE owl [
  <!ENTITY owl ‘‘http://www.w3.org/2002/07/owl#’’ >
  <!ENTITY pgr ‘‘http://abc.di.uevora.pt/~jsaias/owl/pgr.owl#’’ >
  <!ENTITY pgrc ‘‘http://abc.di.uevora.pt/~jsaias/owl/concept.owl#’’>
  <!ENTITY xsd ‘‘http://www.w3.org/2000/10/XMLSchema#’’ >   ] >

<rdf:RDF
   xmlns:owl =’’http://www.w3.org/2002/07/owl#’’
   xmlns:rdf =’’http://www.w3.org/1999/02/22-rdf-syntax-ns#’’
   xmlns:rdfs=’’http://www.w3.org/2000/01/rdf-schema#’’
```

```
   xmlns:xsd =''http://www.w3.org/2000/10/XMLSchema#''
   xmlns:pgr =''http://abc.di.uevora.pt/~jsaias/owl/pgr.owl#''
   xmlns:pgre =''http://abc.di.uevora.pt/~jsaias/owl/entity.owl#''
   xmlns:pgrc =''http://abc.di.uevora.pt/~jsaias/owl/concept.owl#''
>

<pgr:Document rdf:ID=''P000061997''>
<pgr:numero>PPA19990609000600</pgr:numero>
<pgr:parecer>6/1997</pgr:parecer>
<pgr:descritor rdf:resource=''...owl/concept.owl#c7407''/>
<pgr:descritor rdf:resource=''...owl/concept.owl#c4542''/>
<pgr:descritor rdf:resource=''...owl/concept.owl#c8881''/>
<pgr:descritor rdf:resource=''...owl/concept.owl#c9080''/>
<pgr:descritor rdf:resource=''...owl/concept.owl#c8882''/>
<pgr:descritor rdf:resource=''...owl/concept.owl#c3808''/>
<pgr:descritor rdf:resource=''...owl/concept.owl#c3509''/>
<pgr:descritor rdf:resource=''...owl/concept.owl#c1596''/>
<pgr:descritor rdf:resource=''...owl/concept.owl#c3234''/>
<pgr:descritor rdf:resource=''...owl/concept.owl#c1483''/>
<pgr:livro>00</pgr:livro>
<pgr:pedido>28-01-1997</pgr:pedido>
<pgr:distrib>06-02-1997</pgr:distrib>
<pgr:relator>ANA EXEMPLO</pgr:relator>
<pgr:sessoes>01</pgr:sessoes>
<pgr:votacao>09-06-1999</pgr:votacao>
<pgr:tipVot>MAIORIA COM 1 VOT VENC</pgr:tipVot>
<pgr:orgCons1>MAI</pgr:orgCons1>
<pgr:entCons1>MIN DA ADMINISTRAÇÃO INTERNA</pgr:entCons1>
<pgr:indev2>ASSESSOR: JOSÉ EXEMPLO</pgr:indev2>
<pgr:areaTematica>DIR ADM * ADM PUBL * DIR FUND /
          DISC MIL /  DIR TRAB.</pgr:areaTematica>
<pgr:reference rdf:resource=''...owl/docs/P000641992#P000641992''/>
<pgr:reference rdf:resource=''...owl/docs/P000171998#P000171998''/>
<pgr:reference rdf:resource=''...owl/docs/P000271982#P000271982''/>
<pgr:reference rdf:resource=''...owl/docs/P000231986#P000231986''/>
<pgr:legislacao>
CONST76 ART73 ART74.
ART18 ART58 ART59 ART269 ART270 ART271.
L 26/81 DE 1981/08/21.
L 271/86 DE 1986/09/04.
</pgr:legislacao>
<pgr:conclusoes>
1- Aos militares da Guarda Nacional Republicana...
```

```
</pgr:conclusoes>
<pgr:docTIntegral>
Senhor Ministro da Administração Interna,
Excelência:
1.
1.1. Através de despacho, datado de 21 de Janeiro de 1997 ([1]), do
antecessor de Vossa Excelência, foi solicitado parecer deste corpo
consultivo acerca da questão de saber se aos oficiais da Guarda
Nacional Republicana é aplicável o regime geral relativo aos
trabalhadores-estudantes, e, em particular, no tocante à concessão
de horários flexíveis e a justificação de faltas ao serviço, para
efeito da realização de exames em cursos de ensino superior.
...
</pgr:docTIntegral>
</pgr:Document>
</rdf:RDF>
```

O resultado da análise sintáctica pode ser consultado, tal como no caso dos textos inseridos pelo utilizador. A figura 8.12 mostra a sintaxe do documento em análise, contendo todas as frases do texto.

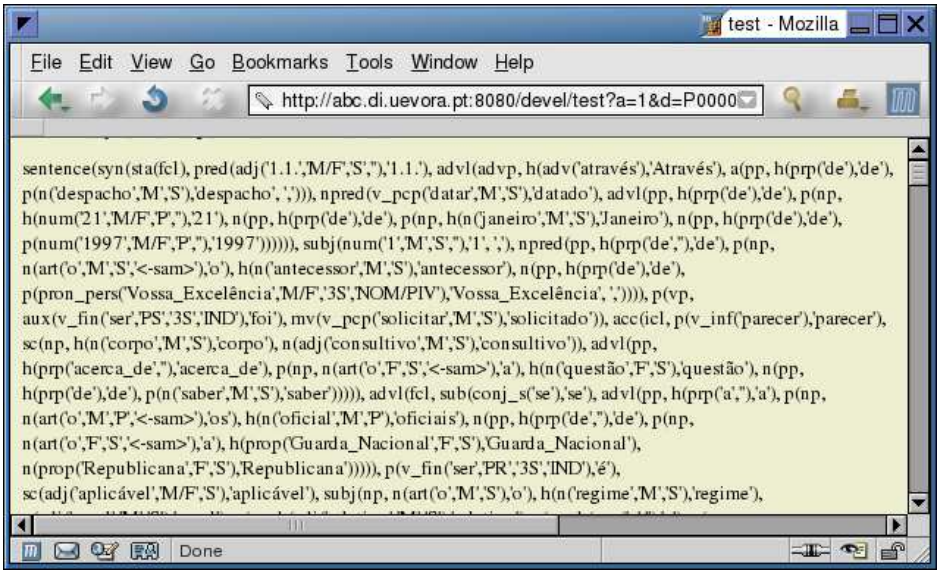

Figura 8.12: Sintaxe em Prolog de todas as frases do documento

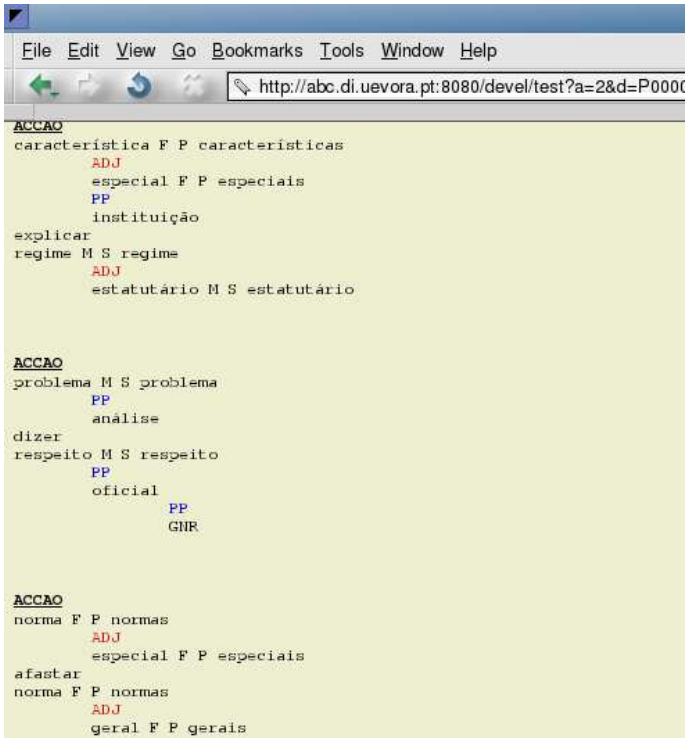

Figura 8.13: Algumas entidades a representar na ontologia

Algumas das entidades obtidas da estrutura sintáctica aparecem na figura 8.13. De acordo com o capítulo 5, apenas as frases com um triplo (sujeito, verbo, complemento) são consideradas. Esse triplo será representado na ontologia como uma acção.
As expressões alinhadas para a direita são modificadores que dão mais informação sobre o elemento a que estão associados. Normalmente esse modificador é um adjectivo, um sintagma preposicional ou um nome.

Sempre que uma entidade corresponde a um termo cuja expressão geral ainda não tinha sido encontrada, a ontologia da secção 6.2 é estendida com uma nova classe (com essa expressão geral para o termo), como as que se seguem.

```
<owl:Class rdf:ID=''estatuto''>
  <rdfs:subClassOf rdf:resource=''&pgr;Entity'' />
</owl:Class>

<owl:Class rdf:ID=''trabalhador-estudante''>
  <rdfs:subClassOf rdf:resource=''&pgr;Entity'' />
```

```
</owl:Class>

<owl:Class rdf:ID=''diploma''>
  <rdfs:subClassOf rdf:resource=''&pgr;Entity'' />
</owl:Class>
```

As instâncias destas classes, geradas automaticamente a partir do texto, estão disponíveis para consulta, como se mostra na figura 8.14.

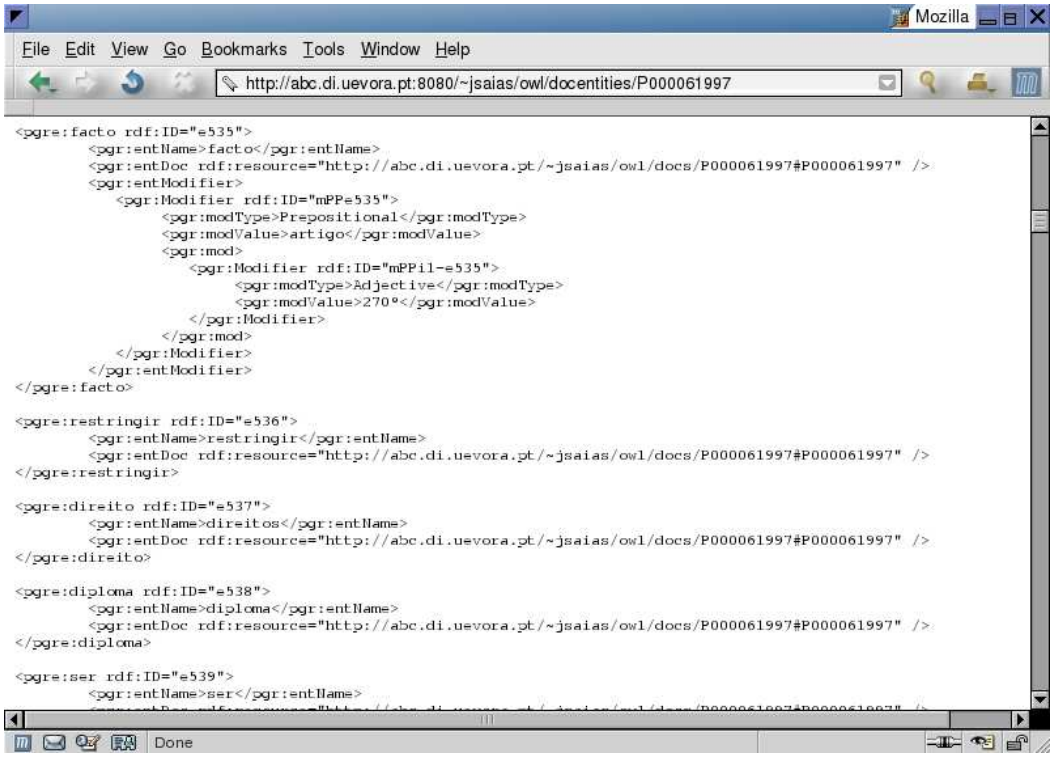

Figura 8.14: Instâncias da ontologia: entidades obtidas do texto

Também o código OWL que representa as instâncias de `Action`, com as acções que envolvem as entidades já vistas, pode ser visto para cada documento, como está patente na figura 8.15.

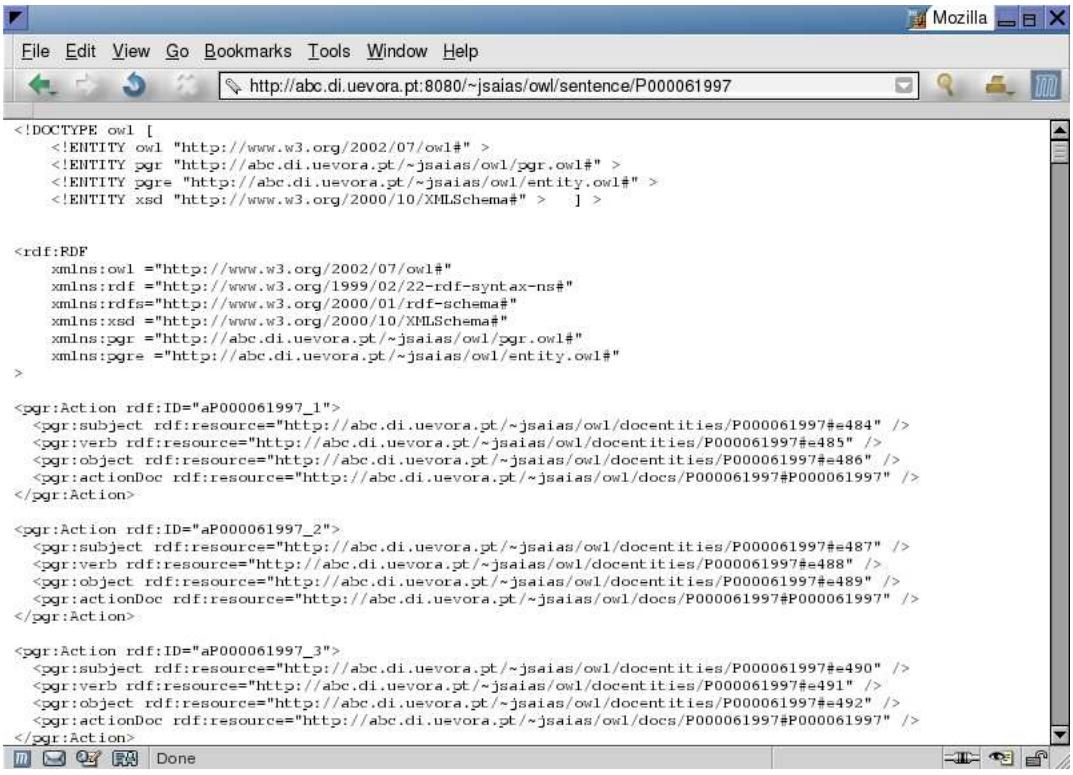

Figura 8.15: Instâncias da ontologia: acções relativas às frases do texto

Para exemplificar uma vez mais a extracção de entidades do texto, consideremos a seguinte frase:

> *A declaração de afectação permite a constituição de direitos reais de habitação periódica sobre o imóvel.*

A figura 8.16 ilustra a acção e entidades retiradas da frase e que vão ser representadas numa estrutura DRS e posteriormente na ontologia.

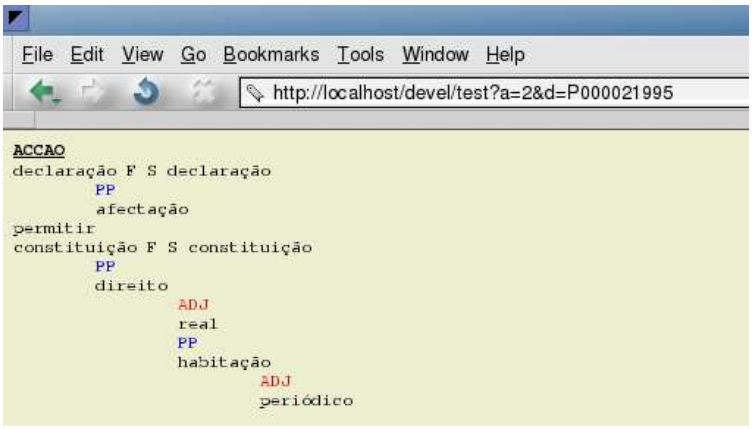

Figura 8.16: Exemplo com as entidades de uma frase

Relativamente ao processo de construção de instâncias para ontologias externas, considera-se que a secção 6.3.3 (página 82) inclui exemplos elucidativos em número suficiente.

O código OWL gerado pelo sistema pode ser utilizado de dois modos. O código pode ser guardado, tal como aqui é apresentado, em documentos que só contém *markup* OWL, formando um repositório de informação em Semantic Web. Por outro lado, o código com as entidades e acções pode ser adicionado aos respectivos documentos HTML onde se publica o texto. Isto permite associar as anotações semânticas em OWL ao texto em língua natural.

Capítulo 9

Conclusões

Após a apresentação do trabalho realizado, com a descrição da metodologia seguida em cada fase e com exemplos elucidativos da sua aplicação em casos concretos, é altura de efectuar um balanço. Este capítulo contém uma análise ao trabalho, recapitulando os objectivos alcançados, na secção 9.1, apontando as limitações encontradas, na secção 9.3, e enumerando os aspectos a melhorar em trabalho posterior, na secção 9.4. A secção 9.2 faz uma comparação com os trabalhos relacionados.

9.1 Objectivos Alcançados

O trabalho apresentado tem a ver com as áreas de Semantic Web e Recuperação de Informação. Foi descrita a metodologia a seguida no tratamento de um conjunto de documentos, de acordo com os objectivos traçados.

Esta metodologia começa com técnicas de Processamento de Língua Natural. Primeiro aplica-se um analisador sintáctico para obter uma arvore sintáctica representativa das frases de cada texto. Em seguida faz-se uma análise semântica parcial, de onde se extraem triplos (sujeito, verbo, complemento), que vão dar origem a instâncias de entidades, ou conceitos, e acções envolvendo esses conceitos.

A inferência das instâncias OWL associadas a cada frase é efectuada através de um processo de abdução.
A ontologia de conceitos gerada, envolvendo classes e instâncias, é representada em OWL e associada aos documentos, também já em versão OWL (é feito um enriquecimento semântico).
Para possibilitar a resposta a interrogações sobre o conteúdo dos documentos, foi

implementado um motor de inferência, baseado em regras de programação em lógica e *interpretation as abduction*.

No decorrer dos trabalhos, foram elaborados alguns módulos ou simplesmente tarefas intermédias que, pela importância que se julga terem ou a eventual utilidade para outros sistemas, vale a pena enumerar:

1. mecanismo para identificação automática de conceitos mencionados nas frases do texto
2. construção automática de uma ontologia, com uma hierarquia de classes
3. construção automática das instâncias para a ontologia OWL antes construída ou outra externa ao sistema
4. representação dos documentos num formato Semantic Web, ideal para publicar e que permite a terceiros usarem as suas ferramentas de pesquisa
5. ferramenta de conversão OWL-Prolog
6. ferramenta para conversão dos documentos em OWL para ISCO/GNU Prolog (e simultaneamente PostgreSQL)
7. motor de inferência para responder a interrogações do utilizador

Os objectivos delineados no começo do trabalho foram, na sua essência, atingidos, ainda que existam algumas limitações (conforme apontado na próxima secção).

Por um lado, este trabalho constitui um importante contributo na forma de construção de uma ontologia, introduzindo uma metodologia para a construção automática de ontologias (classes e respectivas instâncias) directamente a partir do texto dos documentos em língua natural.
Por outro lado, demonstra-se que para além de ser um meio ideal para a publicação dos documentos, unindo a versão para humanos à representação semântica para as máquinas, a Semantic Web, através de uma ontologia, oferece uma plataforma que pode ser adaptada e servir como base de conhecimento em que é possível:

- inferir
- responder a interrogações em língua natural, introduzidas por utilizadores
- provar asserções

9.2 Discussão e Comparação com Trabalhos Relacionados

Neste trabalho, a construção de classes (numa hierarquia com dois níveis de profundidade) e instâncias segue uma abordagem *bottom-up*. Primeiro são identificadas as instâncias e em seguida procura-se a classe a que devem pertencer. Se ainda não há uma classe apropriada, então a nova classe é adicionada como uma "folha" na árvore de classes (no nível inferior).
Embora os documentos do sistema pertençam a um domínio específico, a metodologia seguida permite trabalhar não só em ontologias verticais como também em ontologias horizontais.

O sistema também permite gerar instâncias para uma ontologia externa, o que usualmente se designa por *automatic ontology population*. Este processo não altera a hierarquia de classes, consiste apenas na identificação (e caracterização) de instâncias para as respectivas classes.
O facto da ontologia externa ser vertical ou horizontal não influencia o processo descrito. É mais importante a adequação da semântica da ontologia à semântica dos documentos analisados.

A área de Semantic Web é relativamente recente, pelo que não existem demasiados trabalhos sobre construção de ontologias neste contexto. Há muito que se desenvolve trabalho sobre classificação e definição de taxonomias, mas numa abordagem mais teórica. As linguagens Semantic Web constituem mecanismos apropriados para representar as ontologias desses estudos, pelas suas aptidões em representação semântica.

Os exemplos apresentados na secção 3.2 são representativos dos métodos de construção de ontologias existentes. No trabalho apresentado nesta dissertação não há construção manual de classes para a ontologia, excepto as cinco classes da ontologia base, vistas na secção 6.1.2. Destas, três classes são de topo e genéricas, as restantes são auxiliares. Na definição destas classes iniciais não foi utilizada nenhuma ferramenta das indicadas na secção 3.2.1. A sintaxe OWL da ontologia base foi escrita directamente com um editor de texto.

O modo automático como são geradas as classes e instâncias tem semelhanças com alguns dos exemplos na secção 3.2.2.
A parte deste trabalho que procura construir instâncias para as classes de uma ontologia externa envolve um processo semelhante ao descrito no trabalho de Alani

et al. [AKM+02]. Ambos os processos procuram informação em documentos, os quais são analisados sintacticamente com vista à identificação de termos relacionados com uma ontologia já existente. No exemplo citado, as instâncias geradas serão sempre para a mesma ontologia. Já neste trabalho, o domínio da ontologia externa é desconhecido, sendo a ontologia automaticamente analisada apenas no momento em que se procuram as instâncias nos documentos.

O trabalho descrito em [Lam03] é também aplicado a documentos da área jurídica. O processo começa da mesma forma, com uma análise sintáctica. No trabalho descrito nesta dissertação, os termos seleccionados para a ontologia dão lugar a uma hierarquia de classes simples mas precisa. Posteriormente, para melhorar a ontologia em termos de profundidade seria necessária uma reorganização manual da hierarquia de classes.
No trabalho de Lame procura-se uma ontologia mais elaborada, através da análise dos contextos em que surgem os termos a incluir na ontologia. Apesar de poder alcançar mais relações semânticas, esta ontologia requer um supervisionamento humano para validar as relações semânticas (e hierárquicas) encontradas pelo contexto dos termos.
Os métodos apresentam diferenças que lhes conferem um potencial ganho num dado aspecto, em detrimento de outro aspecto relegado para segundo plano. Relativamente a estes métodos de construção automática, esses aspectos seriam a complexidade/profundidade da ontologia por um lado, e por outro a precisão das relações hierárquicas definidas.

9.3 Limitações

O trabalho apresentado envolve Semantic Web, Processamento de Língua Natural e Programação em Lógica por Restrições e os resultados nem sempre são rigorosamente correctos.

Tanto na construção da ontologia de dois níveis como das instâncias para uma ontologia externa, um dos factores para os resultados não serem mais satisfatórios é o facto de se perder alguma informação na fase anterior, ao retirar a representação semântica parcial da frase.

À partida, o facto de lidarmos apenas com triplos[1] representativos de uma acção já deixa algum conteúdo de fora, nomeadamente frases sem complemento.

[1] triplos com (sujeito, verbo, complemento) extraídos de uma frase

Para além disso, a análise semântica está limitada ao interior da frase. Seria desejável também uma análise inter-frase, para além da intra-frase, de modo a juntar mais informações sobre as entidades referidas e desse modo identificar outras propriedades da ontologia.

A produção de instâncias em OWL para uma ontologia externa tem mais sucesso se a ontologia fornecida ao sistema for de facto relacionada com o conteúdo dos textos.

A perda de informação durante a análise semântica influencia a interpretação pragmática, e não afecta apenas o processo de construção da ontologia como também, e consequentemente, o processo de inferência sobre a base de conhecimento, usada para responder a interrogações.

Há um aspecto alheio a este trabalho que acaba por contribuir para resultados menos bons. O facto de os documentos utilizados serem pareceres jurídicos, e como tal, repletos de expressões retóricas e um discurso com demasiada eloquência, dificulta a fase de extracção de informação.
Inclusivamente, foi preparada uma funcionalidade para analisar textos escritos pelo utilizador, os quais seriam representados na ontologia e usados em processos de inferência. Isto possibilitou a aplicação do sistema a textos arbitrários.

Com a metodologia seguida, não se obtém uma hierarquia de classes com mais de dois níveis de profundidade. Actualmente, para obter uma ontologia mais complexa é preciso alterar manualmente as relações hierárquicas das classes automaticamente geradas. Em particular no caso de ontologias verticais, seria importante atingir uma profundidade maior de modo automático.

Finalmente uma nota sobre a robustez e eficiência do trabalho. Existe um elevado número de tarefas intermédias, desempenhadas por ferramentas diferentes, designadamente em Java ou Prolog (GNU Prolog, ISCO), o que obriga a ter vários processos, onde o segundo recebe como input o resultado do primeiro. Isto introduz alguma morosidade ao nível da "comunicação", por um lado, e se um módulo deixar de funcionar então o seguinte fica necessariamente parado, por outro.
Haveria soluções mais eficientes em que tudo seria desenhado de raíz. Neste caso houve antes uma preocupação de escolher a ferramenta ideal para cada tarefa e conjugar os resultados parciais. É preciso ter em conta que este trabalho é uma experiência materializada num protótipo, que funciona, mas em que se deu mais importância à eficácia, em detrimento da eficiência.

9.4 Trabalho Futuro

Foram identificadas algumas medidas a tomar para aperfeiçoar o sistema apresentado. Para superar as limitações apontadas na secção anterior e melhorar o desempenho global do sistema, é necessário desenvolver algum trabalho sobre os seguintes aspectos:

- extracção de informação: Captar mais detalhes para a representação semântica da frase, para que a interpretação pragmática tenha melhores resultados. A análise semântica deve abarcar simultaneamente várias frases permitindo um histórico sobre os referentes do discurso. Deve efectuar-se a resolução de anáforas e elipses.

- criação da ontologia: Apesar de se tratar de um processo automático que produz resultados úteis, ainda não é possível estabelecer relações hierárquicas com vários níveis de profundidade entre as classes, pelo que será necessário:

  - consultar hierarquias em ontologias já existentes
  - definir manualmente hierarquias para domínios específicos

- normalização de conceitos: O processo de extracção de entidades não elimina todos os duplicados, sobretudo por causa de incorrecções no texto em língua natural.

- tradução OWL para Prolog: Era bastante útil dispor de uma ferramenta de tradução, de toda ou grande parte, da linguagem OWL, já que actualmente apenas se consegue trabalhar com uma parte elementar da linguagem.

- diversificar a base de textos: Devem ser feitas experiências com textos de natureza diferente e comparar o comportamento do sistema relativamente ao observado com os documentos actuais.

- avaliação: O sistema necessita ser testado e avaliado por utilizadores e o *feed-back* dessa experiência usado para tornar o sistema mais robusto.

Apêndice A

Código da Ontologia Base

Este código define as classes na linguagem OWL da Ontologia Base antes de se adicionarem outras classes e instâncias inferidas.

```
<!DOCTYPE owl [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2000/10/XMLSchema#" >   ] >

<rdf:RDF
    xmlns:owl ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#"
>

<owl:Ontology rdf:about="">
  <rdfs:comment>
     representation of a PGR document
     at Universidade de Evora
     last revision: June 6, 2003
  </rdfs:comment>
  <rdfs:label>ontopgr</rdfs:label>
</owl:Ontology>


<!--  ***************  -->
<!-- a PGR document  -->

<owl:Class rdf:ID="Document">
```

```
  <rdfs:comment>a PGR document</rdfs:comment>
</owl:Class>

        <owl:DatatypeProperty rdf:ID="numero">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:DatatypeProperty rdf:ID="parecer">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:ObjectProperty rdf:ID="descritor">
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="#Concept" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="referencia">
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="#Document" />
        </owl:ObjectProperty>


        <owl:DatatypeProperty rdf:ID="livro">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:DatatypeProperty rdf:ID="pedido">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:DatatypeProperty rdf:ID="distrib">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Document" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="relator">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="sessoes">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="votacao">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="tipVot">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="orgCons1">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="entCons1">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="homolog1">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="dataDes1">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="areaTematica">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="legislacao">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="conclusoes">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="textoIntegral">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="jurisprudencia">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="refPub">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="servCon1">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="dirInternacional">
```

```
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="dirEstrangeiro">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="refComplementar">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="indev1">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="rectif">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="dataInf">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="despPGR">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="docInternacional">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="dirComunitario">
  <rdfs:domain rdf:resource="#Document" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
```

```
        <!-- ***  e outros campos não relevantes ***  -->


<owl:Class rdf:ID="Concept">
  <rdfs:comment>a concept in field descritor
  </rdfs:comment>
</owl:Class>

        <owl:DatatypeProperty rdf:ID="conceptCode">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Concept" />
          <rdfs:range rdf:resource="&xsd;integer" />
        </owl:DatatypeProperty>

        <owl:DatatypeProperty rdf:ID="conceptName">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Concept" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:ObjectProperty rdf:ID="moreAbstractThan">
          <rdfs:domain rdf:resource="#Concept" />
          <rdfs:range rdf:resource="#Concept" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="moreSpecificThan">
          <rdfs:domain rdf:resource="#Concept" />
          <rdfs:range rdf:resource="#Concept" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="relatedWith">
          <rdfs:domain rdf:resource="#Concept" />
          <rdfs:range rdf:resource="#Concept" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="equivalentWith">
          <rdfs:domain rdf:resource="#Concept" />
          <rdfs:range rdf:resource="#Concept" />
        </owl:ObjectProperty>
```

```
<!--  ****************************************  -->
<!-- action information retrieved from text sentence -->


<owl:Class rdf:ID="Entity">
  <rdfs:comment>to represent an entity found in the document</rdfs:comment>
</owl:Class>

        <owl:DatatypeProperty rdf:ID="entName">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Entity" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:ObjectProperty rdf:ID="entDoc">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Entity" />
          <rdfs:range rdf:resource="#Document" />
        </owl:ObjectProperty>

        <!-- zero or more modifiers with part-of-speech info -->
        <owl:ObjectProperty rdf:ID="entModifier">
          <rdfs:domain rdf:resource="#Entity" />
          <rdfs:range rdf:resource="#Modifier" />
        </owl:ObjectProperty>



<owl:Class rdf:ID="Modifier">
  <rdfs:comment>to add info to the collected entities</rdfs:comment>
</owl:Class>

        <owl:DatatypeProperty rdf:ID="modType">
          <rdfs:domain rdf:resource="#Modifier" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>

        <owl:DatatypeProperty rdf:ID="modValue">
          <rdfs:domain rdf:resource="#Modifier" />
          <rdfs:range rdf:resource="&xsd;string" />
        </owl:DatatypeProperty>
```

```
        <owl:ObjectProperty rdf:ID="mod">
          <rdfs:domain rdf:resource="#Modifier" />
          <rdfs:range rdf:resource="#Modifier" />
        </owl:ObjectProperty>



<owl:Class rdf:ID="Action">
  <rdfs:comment>action retrieved from a sentence in the document
  </rdfs:comment>
</owl:Class>

        <owl:ObjectProperty rdf:ID="subject">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action" />
          <rdfs:range rdf:resource="#Entity" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="verb">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action" />
          <rdfs:range rdf:resource="#Entity" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="object">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action" />
          <rdfs:range rdf:resource="#Entity" />
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="actionDoc">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action" />
          <rdfs:range rdf:resource="#Document" />
        </owl:ObjectProperty>


</rdf:RDF>
```

Bibliografia

[ABC+99] R. Altman, M. Bada, X.J. Chai, M. Whirl Carillo, R.O. Chen, and N.F. Abernethy. Riboweb: An ontology-based system for collaborative molecular biology. *IEEE Intelligent Systems*, 14(5):68–76, 1999.

[ABLP02] J. Alferes, A. Brogi, J. Leite, and L. Pereira. Evolving logic programs. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *JELIA'02 – Proceedings of the 8th European Conference on Logics and Artificial Intelligence*, pages 50–61. Springer-Verlag LNCS 2424, 2002.

[Abr01] Salvador Abreu. Isco: A practical language for heterogeneous information system construction. In *Proceedings of INAP'01*, Tokyo, Japan, October 2001. INAP.

[AKM+02] Harith Alani, Sanghee Kim, David E. Millard, Mark J. Weal, Wendy Hall, Paul H. Lewis, and Nigel R. Shadbolt. Automatic ontology-based knowledge extraction and tailored biography generation from the web. Technical Report 02-049, Equator, 2002.

[APP+99] J. J. Alferes, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, and P. Quaresma. Preliminary exploration on actions as updates. In M. C. Meo and M. Vilares-Ferro, editors, *Procs. of the 1999 Joint Conference on Declarative Programming (AGP'99)*, pages 259–271, L'Aquila, Italy, September 1999.

[APP+00] J. J. Alferes, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, and P. Quaresma. Dynamic knowledge representation and its applications. In S. Cerri and D. Dochev, editors, *Proceedings of the 9th International Conference on Artificial Intelligence - Methodology, Systems, Applications (AIMSA'2000)*, number 1904 in Lecture Notes in Artificial Intelligence, pages 1–10, Varna, Bulgary, September 2000. Springer Verlag.

[BCC+03] V.R. Benjamins, J. Contreras, P. Casanovas, M. Ayuso, M. Becue, L. Lemus, and C. Urios. Ontologies of professional legal knowledge as

the basis for intelligent it support for judges. In *Workshop on Legal Ontologies & Web Based Legal Information Management, ICAIL'03*, March 2003.

[BGB+99] P.G. Baker, C. Goble, S. Bechhofer, N. Paton, R. Stevens, and A. Brass. An ontology for bioinformatics applications. *Bioinformatics*, 15(6):510–520, September 1999.

[Bic00] Eckhard Bick. *The Parsing System "Palavras". Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. Aarhus University Press, 2000.

[BLF99] Tim Berners-Lee and M. Fischetti. Weaving the web. *Harper, San Francisco*, 1999.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[BvEW03] A. Boer, T. van Engers, and R. Winkels. Using ontologies for comparing and harmonizing legislation. In ACM Press USA, editor, *Proceedings of ICAIL2003*, 2003.

[GFH+03] Jennifer Golbeck, Gilberto Fragoso, Frank Hartel, Jim Hendler, Jim Oberthaler, and Bijan Parsia. The national cancer institute's thesaurus and ontology. *Web Semantics*, 1(1), 2003.

[GVGQ03] Caroline Gasperin, Renata Vieira, Rodrigo Goulart, and Paulo Quaresma. Extracting xml syntactic chunks from portuguese corpora. In *TALN'2003 - Workshop on Natural Language Processing of Minority Languages and Small Languages of the Conference on "Traitement Automatique des Langues Naturelles'*, Batz-sur-Mer, France, June 2003.

[HH00] Jeff Heflin and James Hendler. Searching the web with shoe. In *AAAI-2000 Workshop on AI for Web Search*, 2000.

[HKQ02] D. Huynh, D. Karger, and D. Quan. Haystack: A platform for creating, organizing and visualizing information using rdf. In *Semantic Web Workshop, The Eleventh World Wide Web Conference 2002*, 2002.

[HSAM90] Jerry Hobbs, Mark Stickel, Douglas Appelt, and Paul Martin. Interpretation as abduction. Technical Report 499, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, 1990.

[KMV00] Joerg-Uwe Kietz, Alexander Maedche, and Raphael Volz. A method for semi-automatic ontology acquisition from a corporate intranet. In *Proceedings of EKAW-2000 Workshop Ontologies and Text*, number 1937 in Springer Lecture Notes in Artificial Intelligence (LNAI), France, October 2000.

[KPHG03] Aditya Kalyanpur, Bijan Parsia, James Hendler, and Jennifer Golbeck. Smore - semantic markup, ontology, and rdf editor. Technical report, http://www.mindswap.org/, University of Maryland, 2003.

[KR93] Hans Kamp and Uwe Reyle. *From Discourse to Logic: An Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Dordrecht: D. Reidel, 1993.

[KW02] Latifur Khan and Lei Wang. Automatic ontology derivation using clustering for image classification. In *Proceedings of Eighth International Workshop on Multimedia Information Systems*, pages 56–65, October 2002.

[Lam01] Guiraude Lame. Constructing an ir-oriented legal ontology. In *Second International Workshop on Legal Ontologies*, Netherlands, December 2001.

[Lam03] Guiraude Lame. Using text analysis techniques to identify legal ontologies' components. In *Workshop on Legal Ontologies of the International Conference on Artificial Intelligence and Law*, 2003.

[MK60] M.E. Maron and J.L. Kuhns. On relevance, probabilistic indexing and information retrieval. Technical report, Journal of the ACM, July 1960.

[Ome01] Borys Omelayenko. Learning of ontologies for the web: the analysis of existent approaches. In *Proceedings of the International Workshop on Web Dynamics, ICDT'01*, London, UK, January 2001.

[PQ98] Luis Moniz Pereira and Paulo Quaresma. Modelling agent interaction in logic programming. In Osamu Yoshie, editor, *INAP'98 - The 11th International Conference on Applications of Prolog*, pages 150–156, Tokyo, Japan, September 1998. Science University of Tokyo.

[QL95] Paulo Quaresma and José Gabriel Lopes. Unified logic programming approach to the abduction of plans and intentions in information-seeking dialogues. *Journal of Logic Programming*, 24(54):103,121, 1995.

[QR01a] Paulo Quaresma and Irene Rodrigues. Using logic programming to model multi-agent web legal systems - an application report. In *Proceedings of the ICAIL'01 - International Conference on Artificial Intelligence and Law*, St. Louis, USA, May 2001. ACM.

[QR01b] Paulo Quaresma and Irene Pimenta Rodrigues. PGR: Portuguese atorney general's office decisions on the web. In Osamu Yoshie, editor, *Proceedings of the 14th International Conference on Applications of Prolog*, University of Tokyo, Tokyo, Japan, October 2001. REN Associates, Inc. ISSN 1345-0980. To be published by Springer Verlag's LNAI.

[QR03] Paulo Quaresma and Irene Pimenta Rodrigues. A natural language interface for information retrieval on semantic web documents. In E. Menasalvas, J. Segovia, and P. Szczepaniak, editors, *AWIC'2003 - Atlantic Web Intelligence Conference*, Lecture Notes in Artificial Intelligence LNCS/LNAI 2663, pages 142–154, Madrid, Spain, May 2003. Springer-Verlag.

[QRA01] Luis Quintano, Irene Rodrigues, and Salvador Abreu. Relational information retrieval through natural lanaguage analysis. In *Proceedings of INAP'01*, Tokyo, Japan, October 2001. INAP.

[vEV02] Tom M. van Engers and Radboud A.W. Vanlerberghe. The power light-version: Improving legal quality under time pressure. In R. Traunmüller and K. Lenk, editors, *Proceedings of the E-Gov 2002 Conference (DEXA 2002)*, pages 75–83, Berlin, 2002.

[War83] David HD Warren. An abstract prolog instruction set. Technical Report 309, SRI International, Menlo Park, California, USA, October 1983.