

**Populando ontologias através
de informações em HTML -
o caso do currículo lattes**

André Casado Castaño

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
MESTRE EM CIÊNCIAS

Área de Concentração: Ciência da Computação
Orientadora: Profa. Dra. Renata Wassermann

São Paulo, junho de 2008

**Populando ontologias através
de informações em HTML -
o caso do currículo lattes**

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por (André Casado Castaño)
e aprovada pela Comissão Julgadora.

Banca Examinadora:

- Profa. Dra. Renata Wassermann (orientadora) - IME-USP.
- Prof. Dr. Flávio Soares Correa da Silva - IME-USP.
- Profa. Dra. Anarosa Alves Franco Brandão - EP-USP.

Agradecimentos

Agradeço, antes de tudo, aos meus pais (Joaquim e Mirtes) e a meus irmãos (Fábio e Marcos) pela educação, pela confiança e pelo amor de sempre.

A minha namorada e amiga, Grazielle Yumi, obrigado por me ajudar em tantos momentos e me suportar nos piores. Obrigado pelo companheirismo, carinho e amor.

Aos meus amigos imeanos, principalmente os companheiros dos times de voleibol e handebol, muito obrigado por me proporcionarem sempre grandes momentos e enorme aprendizado através do esporte e da ótima convivência. Agradeço também a amiga Maki Hirai que sempre me ofereceu palavras de incentivo nos momentos mais difíceis. Sem vocês, tenho certeza que a caminhada teria sido muito mais difícil.

A minha orientadora, Renata Wassermann, agradeço muito pela paciência e sabedoria.

Agradeço a todos que, de alguma forma, me ajudaram e me incentivaram em mais esta etapa da minha vida.

Resumo

A Plataforma Lattes é, hoje, a principal base de currículos dos pesquisadores brasileiros. Os currículos da Plataforma Lattes armazenam de forma padronizada dados profissionais, acadêmicos, de produções bibliográficas e outras informações dos pesquisadores.

Através de uma base de Currículos Lattes, podem ser gerados vários tipos de relatórios consolidados. As ferramentas existentes da Plataforma Lattes não são capazes de detectar alguns problemas que aparecem na geração dos relatórios consolidados como duplicidades de citações ou produções bibliográficas classificadas de maneiras distintas por cada autor, gerando um número total de publicações errado. Esse problema faz com que os relatórios gerados necessitem ser revistos pelos pesquisadores e essas falhas deste processo são a principal inspiração deste projeto.

Neste trabalho, utilizamos como fonte de informações currículos da Plataforma Lattes para popular uma ontologia e utilizá-la principalmente como uma base de dados a ser consultada para geração de relatórios.

Analisamos todo o processo de extração de informações a partir de arquivos HTML e seu posterior processamento para inserí-las corretamente dentro da ontologia, de acordo com sua semântica.

Com a ontologia corretamente populada, mostramos também algumas consultas que podem ser realizadas e fazemos uma análise dos métodos e abordagens utilizadas em todo processo, comentando seus pontos fracos e fortes, visando detalhar todas as dificuldades existentes no processo de população (instanciação) automática de uma ontologia.

Palavras-chave: Web Semântica, Ontologia, Currículo Lattes, OWL, SPARQL, Co-referência.

Abstract

Lattes Platform is the main database of Brazilian researchers résumés in use nowadays. It stores in a standardized form professional, academic, bibliographical productions and other data from these researchers.

From these Lattes resumés database, several types of reports can be generated. The tools available for Lattes platform are unable to detect some of the problems that emerge when generating consolidated reports, such as citation duplicity or bibliographical productions misclassified by their authors, generating an incorrect number of publications. This problem demands a revision performed by the researcher on the reports generated, and the flaws of this process are the main inspiration for this project.

In this work we use the Lattes platform resumés database as the source for populating an ontology that is intended to be used to generate reports.

We analyze the whole process of information gathering from HTML files and its post-processing to insert them correctly in the ontology, according to its semantics.

With this ontology correctly populated, we show some new reports that can be generated and we perform also an analysis of the methods and approaches used in the whole process, highlighting their strengths and weaknesses, detailing the difficulties faced in the automated populating process (instantiation) of an ontology.

Keywords: Semantic Web, Ontology, Currículo Lattes, OWL, SPARQL, Coreference

Sumário

Lista de Figuras	ix
1 Introdução	1
2 Linguagens e Ferramentas utilizadas com Ontologias	4
2.1 Ontologia	4
2.2 XML	5
2.3 RDF	5
2.3.1 Notation 3	8
2.3.2 Turtle	8
2.4 OWL	10
2.5 Protégé	13
2.6 DIG	13
2.7 O motor de inferência Pellet	14
2.8 Interação entre Protégé e Pellet	15
2.9 SPARQL	16
3 O Currículo Lattes	20
3.1 Introdução	20

3.2	Relatórios baseados nos Currículos Lattes	21
4	Criação da ontologia base	23
4.1	Introdução	23
4.2	Modelando a ontologia baseado no DTD do Currículo Lattes	24
4.3	Verificação da ontologia	29
5	População da Ontologia do Currículo Lattes	31
5.1	Introdução	31
5.2	Leitura e extração de informações dos arquivos HTML	32
5.3	Processamento das informações extraídas	44
5.4	Casamento de Instâncias	51
5.4.1	Introdução	51
5.4.2	Busca de duplicidades - Algoritmo	53
5.4.3	CrITÉrios de similaridade	54
5.5	Testes com os algoritmos e os critérios de similaridade	56
6	Análise dos Resultados	62
6.1	Introdução	62
6.2	Medidas em Sistemas de Recuperação de Informações	63
6.3	Testes do Sistema	64
7	Consultas na Ontologia	68
7.1	Introdução	68
7.2	Consultas Simples	69
7.3	Consultas Consolidadas	73

<i>SUMÁRIO</i>	viii
7.4 SPARQL vs SQL vs XQuery	76
8 Conclusões e trabalhos futuros	85
Referências Bibliográficas	88

Lista de Figuras

2.1	Exemplo de Grafo RDF.	7
4.1	Arquivo DTD - Endereço Profissional.	25
4.2	Ontologia - Endereço Profissional.	26
5.1	Parte de um Currículo Lattes, mostrando as seções: Dados Pessoais, Formação Acadêmica, Atuação Profissional, Áreas de Atuação e Idiomas.	35
5.2	Instância de um CurriculumVitae.	45
5.3	Idiomas de um currículo da Plataforma Lattes, visualizado em um navegador	47
5.4	Tela do Protégé mostrando uma consulta SPARQL que retorna as produções bibliográficas cujo título contém o texto “multi-robotic”. A coluna <i>Dados</i> mostra as diferentes instâncias.	53
5.5	Tela do Protégé mostrando uma consulta SPARQL que retorna as produções bibliográficas cujo título contém o texto “multi-robotic”. A coluna <i>Dados</i> mostra que temos sempre a mesma instância.	53
6.1	Relações entre os conjuntos de objetos corretos/incorretos e encontrados/não encontrados.	63
7.1	Tela de um Currículo Lattes visto na Web.	70
7.2	Dados Pessoais do pesquisador - original HTML.	71

7.3	Consulta SPARQL buscando Dados Pessoais do pesquisador.	71
7.4	Formação Acadêmica do pesquisador - original HTML.	72
7.5	Consulta SPARQL buscando Formação Acadêmica do pesquisador.	72
7.6	Proficiência em Idiomas do pesquisador - original HTML.	73
7.7	Consulta SPARQL buscando Proficiência em Idiomas do pesquisador.	73
7.8	Proficiência em Idiomas dos pesquisadores - original HTML.	74
7.9	Consulta SPARQL buscando Proficiência em Espanhol dos pesquisadores.	75
7.10	Consulta SPARQL buscando Artigos Publicados entre os anos de 2003 e 2006 que foram escritos por 2 ou mais autores.	76
7.11	Diagrama contendo as classes da ontologia do Currículo Lattes envolvidas na consulta de proficiências de idioma de um pesquisador.	78
7.12	Modelo da estrutura de dados de um banco SQL contendo as mesmas informações e relações existentes na ontologia para o exemplo das proficiências em línguas.	78
7.13	Modelo da estrutura de dados de um banco SQL contendo orientações de Mestrado, Doutorado e Iniciação Científica, baseado na estrutura de classes da ontologia do Currículo Lattes.	81

Capítulo 1

Introdução

A Internet é hoje amplamente utilizada no mundo todo e uma das principais conseqüências dessa popularização é o aumento contínuo da quantidade de dados disponíveis na rede.

Cada vez temos disponíveis ferramentas de buscas mais poderosas e complexas, mas que ainda pecam por não conseguirem associar semântica às buscas que são realizadas. As ferramentas atuais utilizam recursos baseados em variações sintáticas apenas. Essa limitação não permite aos mecanismos de busca classificarem consultas semanticamente, sendo apenas possível dar níveis de relevância de acordo com a presença de palavras-chaves ou outras variações não envolvendo semântica das palavras.

Podemos dizer que o objetivo da Web Semântica [3] é melhorar as potencialidades da Web através da criação de ferramentas e de padrões que permitam atribuir significados claros aos conteúdos das páginas e facilitar a sua publicação e manutenção.

Enquanto a web tradicional foi desenvolvida para ser entendida apenas pelos usuários, a Web Semântica está sendo projetada para ser compreendida pelas máquinas, na forma de agentes computacionais, que são capazes de operar eficientemente sobre as informações, podendo entender seus significados [6]. Desta maneira, elas irão auxiliar os usuários em operações na web.

Neste trabalho, queremos explorar um pouco do poder da Web Semântica e utilizar alguns dos recursos disponíveis através de um estudo de caso: os currículos da Plataforma Lattes.

Vamos criar e popular uma ontologia com currículos da Plataforma Lattes e realizar consultas utilizando as linguagens e ferramentas da Web Semântica.

Mostraremos em detalhes, neste estudo, todo o processo de população automática de uma onto-

logia, descrevendo os problemas encontrados, analisando e propondo soluções para cada caso.

Algumas das soluções propostas apresentarão detalhes específicos para o estudo de caso do Currículo Lattes. Entretanto, as idéias contidas nessas soluções podem ser utilizadas na resolução de problemas semelhantes, desde que sejam feitas as adaptações de acordo com o problema em questão. O foco do trabalho é tratar o caso dos currículos da Plataforma Lattes, mas esperamos que este trabalho apresente temas comuns a outros casos, de modo que as soluções apresentadas possam ser aplicadas em outras situações.

A estrutura dos tópicos do trabalho é descrita a seguir.

No capítulo 2, são apresentadas as linguagens e ferramentas utilizadas neste trabalho, relacionadas a Web Semântica.

Em seguida, temos no capítulo 3 um breve histórico do Currículo Lattes, descrevendo algumas de suas características e também explicando o processo de geração de relatórios a partir de um conjunto de currículos.

No capítulo 4, vemos como foi o processo de modelagem da ontologia base. Começamos falando da história desta ontologia, desde sua criação feita por Ailton Sergio Bonifacio [4] em seu trabalho de mestrado e modificada por Marcos Yoshinori Nakashima [19], passando por todo o processo de transformação, verificação e correção da modelagem, utilizando um motor de inferência.

O quinto capítulo trata de todo o processo de leitura dos arquivos dos currículos, identificação dos tipos de informação contidos neles e inserção dentro da ontologia. Veremos alguns problemas encontrados no processo, em especial o problema de co-referências, bem como soluções propostas e análises dos resultados obtidos.

Após a apresentação do processo de população da ontologia, temos no capítulo 6 os testes realizados com os algoritmos propostos e uma breve explicação das medidas utilizadas.

No sétimo capítulo, temos alguns exemplos de consultas realizadas com a ontologia, desde consultas simples que apenas mostram a forma como algumas informações estão armazenadas, até consultas mais complexas, obtendo dados consolidados dos dados inseridos na ontologia.

Por fim, o capítulo 8 traz as conclusões sobre o trabalho, apresentando e comentando os resultados obtidos. Também são propostos alguns trabalhos futuros a partir de alguns temas importantes que não puderam ser tratados neste projeto.

Os scripts e arquivos deste trabalho estão disponibilizados na web, no endereço <http://www.ime.usp.br/~casado/Mestrado/Arquivos/>.

Capítulo 2

Linguagens e Ferramentas utilizadas com Ontologias

2.1 Ontologia

O termo “ontologia” deriva do grego “onto”, (ser), e “logia”, (discurso escrito ou falado). Este termo tem sido empregado ao longo da história pela Filosofia e estuda as teorias sobre a natureza da existência. Na Inteligência Artificial, o termo ontologia teve seu significado adaptado para um outro sentido: “Ontologia é uma definição de relações entre termos e conceitos de um domínio” [3].

Uma ontologia [12] [27] descreve os conceitos e relações que são importantes em um domínio particular, fornecendo um vocabulário para esse domínio assim como especificações computacionais do significado dos termos usados no vocabulário. Ontologias podem ser taxonomias e classificações, esquemas de banco de dados, até teorias completamente axiomatizadas. Recentemente, ontologias têm sido adotadas em muitos negócios e comunidades científicas como uma forma de compartilhar, reusar e processar domínios de conhecimento.

Podemos, resumidamente, dizer que uma ontologia é uma representação computacional de um determinado domínio de conhecimento.

Ontologias são agora fundamentais para muitas aplicações como portais de conhecimento científico, gerenciamento de informações e integração de sistemas, comércio eletrônico, e principalmente serviços relacionados com web semântica.

2.2 XML

XML¹ –*Extensible Markup Language*– é um formato de texto de uso geral, derivado do SGML (*Standard Generalized Markup Language*)², que tem como objetivo principal facilitar o compartilhamento de dados estruturados através de sistemas de informações diferentes, principalmente via Internet [6]. O XML é classificado como uma linguagem extensível porque permite que seus usuários definam seus próprios elementos, diferente de linguagens como o HTML, por exemplo, que possuem elementos pré-definidos como as marcas <p> (parágrafo) ou <table> (tabela).

A linguagem XML tem o propósito de armazenar e transportar informações, diferente do HTML que é uma linguagem para mostrar informações.

O uso do XML tem crescido desde sua criação e hoje em dia temos exemplos de arquivos XML sendo usados em diversas situações, até mesmo para fins bastante diferentes do seu propósito original. Vemos arquivos XML sendo usados na comunicação entre muitos sistemas, por conta de sua praticidade de criação e também por serem amplamente conhecidos entre os desenvolvedores de sistemas.

2.3 RDF

RDF³ –*Resource Description Framework*– pode ser descrito inicialmente como um arcabouço que fornece uma sintaxe padronizada para que aplicações e ferramentas de desenvolvimento possam descrever recursos da Web.

O modelo de dados RDF representa as propriedades de um recurso e os valores dessa propriedades através de triplas. A estrutura de uma tripla que representa uma expressão RDF é:

- Recurso (sujeito): é o que será descrito
- Objeto: um valor ou uma expressão RDF
- Propriedade (predicado): é uma característica ou atributo do recurso. Representa um relacionamento entre o recurso e o objeto.

Essa estrutura permite que o RDF seja definido como um grafo, onde os nós representam recursos

¹<http://w3.org/XML/>

²<http://www.w3.org/MarkUp/SGML/>

³<http://www.w3.org/RDF/>

e objetos e os arcos representam propriedades. Podemos dizer que um grafo RDF é um conjunto de triplas RDF.

O RDF possui uma sintaxe baseada em XML chamada RDF/XML que é a principal forma de gravar e compartilhar os grafos RDFs.

A sintaxe RDF/XML é a mais utilizada para trabalhar com RDF. Ela permite que agentes computacionais a utilizem facilmente por utilizar a sintaxe da linguagem XML, mas é mais difícil de ser lida e entendida por humanos. Veremos nas próximas seções outros formatos de representação de arquivos RDF que são sintaxes alternativas ao RDF/XML.

A seguir, temos um exemplo de um trecho de um arquivo RDF, em RDF/XML. O trecho descreve uma Pessoa com duas propriedades: nomeCompleto e sexo. Segue também uma tabela com as propriedades e seus valores, para facilitar o entendimento do formato RDF:

Tabela de exemplo com propriedades e valores:

Recurso	Propriedade	Valor
Individuo01	rdf:type	Pessoa
Individuo01	sexo	Masculino
Individuo01	nomeCompleto	André Casado Castaño

Código RDF representando os dados do exemplo:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns="http://example.com/ontology#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <Pessoa rdf:about="Individuo01">
    <sexo>Masculino</sexo>
    <nomeCompleto>André Casado Castaño</nomeCompleto>
  </Pessoa>
</rdf:RDF>
```

Para entendermos melhor a representação do RDF como um grafo, vamos ver um exemplo baseado nas informações da tabela anterior:

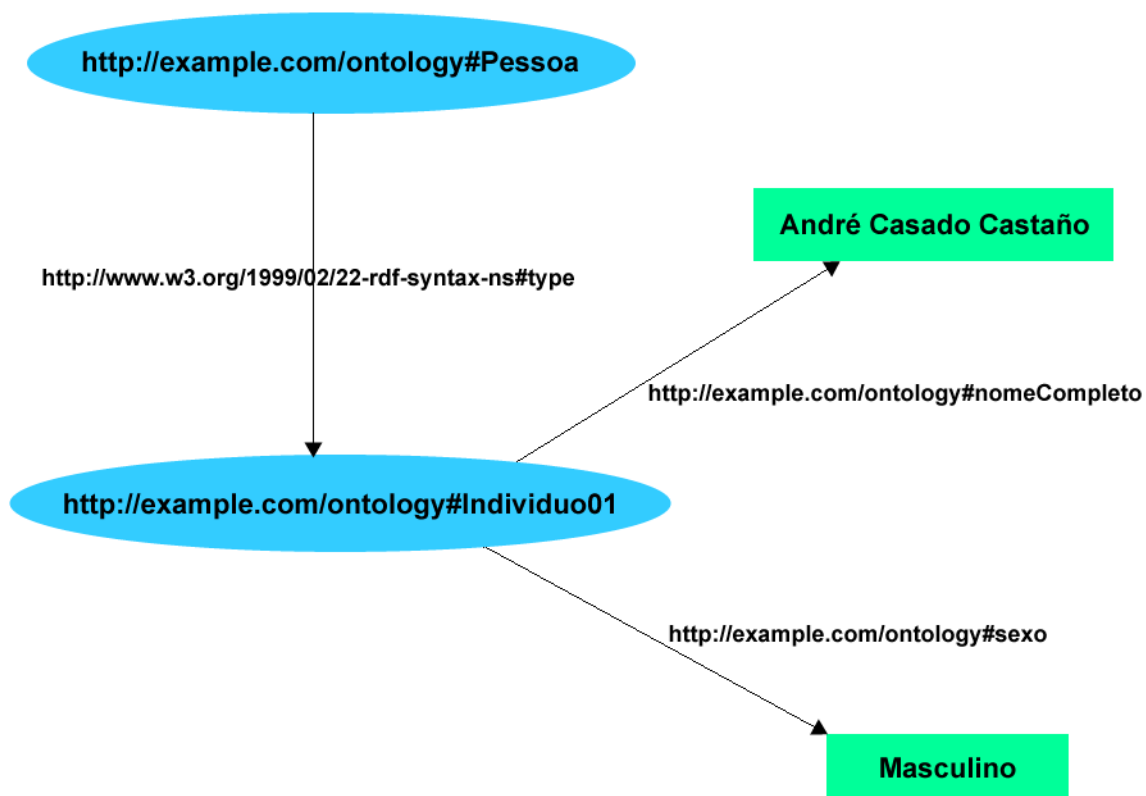


Figura 2.1: Exemplo de Grafo RDF.

O grafo da figura 2.1 representa uma “Pessoa do sexo masculino e de nome completo André Casado Castaño”.

2.3.1 Notation 3

N3⁴ –*Notation 3*– é uma sintaxe alternativa para a sintaxe RDF/XML, tentando possibilitar uma melhor leitura do seu conteúdo, mas também permitindo maior expressividade.

Os principais objetivos da N3 são:

- otimizar a expressão de lógica e dados em uma mesma linguagem
- permitir a expressão de RDF
- permitir a integração de regras com RDF
- permitir citações para que possam ser feitas declarações sobre declarações
- ser o mais natural, legível e simétrica possível, facilitando a leitura e entedimento pelos usuários

Podemos citar alguns itens da sintaxe da N3 que não existem em RDF/XML como:

- Variáveis quantificadoras `@forAll` e `@forSome` (fórmula válida para todos os valores e fórmula válida para pelo menos um valor)
- `=>` implica

2.3.2 Turtle

Turtle⁵ –*Terse RDF Triple Language*– é uma sintaxe de texto para RDF que permite que grafos RDF sejam escritos de maneira que possam ser mais facilmente lidos e entendidos por seus usuários, semelhante a sintaxe N3.

A sintaxe Turtle possui alguns elementos de sintaxe a menos do que os existentes em N3, mas ela não pode ser considerada totalmente um subconjunto da N3 porque possui 2 elementos de sintaxe que não existem na N3 (literais booleanos e literais com decimais de qualquer tamanho).

Para mostrar a diferença entre Turtle e N3, exibimos a seguir algumas sintaxes que estão presentes em N3 mas não em Turtle:

⁴<http://www.w3.org/DesignIssues/Notation3>

⁵<http://www.dajobe.org/2004/01/turtle/>

- { ... }
- # is of
- caminhos como :a.:b.:c e :a^:b^:c
- @keywords
- => implica
- = equivalente
- @forAll
- @forSome
- <=

Todo RDF escrito no formato Turtle pode ser usado dentro de uma consulta SPARQL (seção 2.9). Isso facilita o desenvolvimento de consultas, uma vez que podemos fazer consultas baseadas em triplas retiradas diretamente do arquivo RDF em questão, posteriormente modificando as consultas para chegar exatamente nos resultados desejados.

Este formato para representação de ontologias é um dos que pode ser mais facilmente entendido e portanto será usado algumas vezes, neste trabalho, para mostrar exemplos de trechos de códigos de ontologias.

Exemplo:

```
@prefix default: <http://example.com/ontology#> .
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

default:Individuo01
  rdf:type default:Pessoa ;
  default:nomeCompleto
    "Andre Casado Castano"^^<http://www.w3.org/2001/XMLSchema#string> ;
  default:sexo "Masculino"^^<http://www.w3.org/2001/XMLSchema#string> .
```

2.4 OWL

OWL⁶ (*Web Ontology Language*) é a linguagem padrão para descrição de ontologias na Web. OWL é baseada em lógicas de descrição de grande expressividade [2].

O padrão OWL foi anunciado oficialmente como uma recomendação da W3C⁷ no dia 10 de fevereiro de 2004. O OWL é derivado da linguagem DAML+OIL [23], a qual foi utilizada no desenvolvimento inicial da ontologia do Currículo Lattes, criada por Ailton Sergio Bonifacio [4], no seu trabalho de mestrado. No trabalho de iniciação científica de Marcos Yoshinori Nakashima [19], foi feita a transformação da ontologia da linguagem DAML+OIL para OWL, e é esta ontologia transformada em OWL que está sendo usada neste projeto.

A linguagem OWL possibilita que ontologias sejam representadas e processadas por sistemas através da Web, utilizando como base o arcabouço RDF.

OWL adiciona mais vocabulário para descrever propriedades e classes em relação ao RDF como por exemplo: relações entre classes, cardinalidade, igualdade, propriedades com tipos mais complexos, características de propriedades, e classes enumeradas.

A linguagem OWL tem 3 sublinguagens. São elas:

- OWL Full - Utiliza todos os recursos da linguagem OWL, sem restrições. Possui a máxima expressividade da linguagem OWL, mas é indecidível. Nenhum motor de inferência é capaz de suportar todos os recursos da linguagem OWL Full.

⁶<http://www.w3.org/2004/OWL/>

⁷<http://www.w3.org/>

- OWL DL (*Description Logic*) - A linguagem OWL DL possui o mesmo vocabulário da linguagem OWL Full. Entretanto, a OWL DL possui restrições. O OWL DL requer separação de tipos (uma classe não pode ser também um indivíduo ou propriedade, uma propriedade não pode ser um indivíduo ou classe). As propriedades só podem ser *ObjectProperties* ou *DatatypeProperties*. *DatatypeProperties* são relações entre instâncias de classes e literais RDF e tipos de dados XML, enquanto que *ObjectProperties* são relações entre instâncias de duas classes.

O OWL DL tem expressividade menor que o OWL Full, mas é mais eficiente computacionalmente, mantendo a computabilidade (garante-se que todas as conclusões sejam computáveis) e decidibilidade (todas as computações terminarão em tempo finito). Já existem motores de inferência que aceitam totalmente arquivos na linguagem OWL-DL, como o motor de inferência utilizado neste trabalho que será descrito em detalhes mais adiante.

- OWL Lite - Possui um subconjunto dos construtores usados na linguagem OWL Full e possui algumas limitações. No OWL Lite, somente certos tipos de restrições podem ser usadas. Equivalência entre classes e subclasses entre classes são permitidas somente com classes nomeadas (as classes não podem ser expressões quaisquer). Da mesma forma, restrições de propriedades também usam sempre classes nomeadas. O OWL Lite também tem uma limitação com o conceito de cardinalidade. Somente são aceitos os valores 0 e 1 para cardinalidades.

O OWL Lite também é totalmente computável e decidível e a linguagem é mais eficiente computacionalmente que o OWL DL.

A seguir, veremos um trecho de código OWL. Basicamente, temos a definição da Classe “AtividadesDeDirecaoEAdministracao” e da classe “AtividadesDeEnsino”.

Para a classe “AtividadesDeDirecaoEAdministracao”, vemos na segunda linha a definição que diz que esta é uma subclasse da classe “Atividades”. Em seguida, são definidas duas propriedades (“cargoOuFuncao” e “formatoCargoOuFuncao”). A propriedade “cargoOuFuncao” tem cardinalidade máxima 1 e a propriedade “formatoCargoOuFuncao” tem cardinalidade 1.

Dentro do contexto da Ontologia, a propriedade cargoOuFuncao armazena o nome do cargo do pesquisador (Ex: Professor Titular do Departamento de Ciência da Computação do IME-USP) e a propriedade formatoCargoOuFuncao armazena um valor de cargo ou funcao dentro de valores

pré-definidos, de acordo com o que está estabelecido no arquivo DTD do Currículo Lattes⁸. Alguns possíveis valores são: CHEFE_DE_DEPARTAMENTO, COORDENADOR_DE_CURSO, REITOR, OUTRO, LIVRE.

A classe “AtividadesDeEnsino” também é subclasse de Atividades e possui a propriedade “disciplina”, cujo valor tem que ser uma instância da classe Disciplina, e temos a propriedade tipo-Ensino, cujo valor (também definido no DTD do Lattes) pertence ao conjunto (GRADUACAO | POS-GRADUACAO | ESPECIALIZACAO | APERFEICOAMENTO | ENSINO-FUNDAMENTAL | ENSINO-MEDIO | OUTRO), com cardinalidade máxima 1.

```
<owl:Class rdf:ID="AtividadesDeDirecaoEAdministracao">
  <rdfs:subClassOf rdf:resource="#Atividades"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#cargoOuFuncao"/>
      <owl:maxCardinality rdf:datatype="&xsd:int">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#formatoCargoOuFuncao"/>
      <owl:cardinality rdf:datatype="&xsd:int">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="AtividadesDeEnsino">
  <rdfs:subClassOf rdf:resource="#Atividades"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#disciplina"/>
      <owl:allValuesFrom rdf:resource="#Disciplina"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
```

⁸<http://lmpl.cnpq.br/lmpl/Gramaticas/Curriculo/DTD/Documentacao/DTDCurriculo.pdf>

```
<owl:onProperty rdf:resource="#tipoEnsino"/>
  <owl:maxCardinality rdf:datatype="&xsd:int">1</owl:maxCardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

2.5 Protégé

Protégé⁹ é uma plataforma de código aberto que fornece ferramentas para construir e manipular ontologias.

Esta ferramenta possui um editor OWL que facilita a manipulação de ontologias, além de apresentar a possibilidade de uso de uma série de *plug-ins*, como por exemplo um *plug-in* para visualização da ontologia que está sendo editada.

A ferramenta oferece a possibilidade de se integrar um motor de inferência externo, o que deveria facilitar a classificação, inferência e checagem de consistência dentro do ambiente do Protégé. Infelizmente a interface entre o Protégé e o motor de inferência tem limitações que prejudicam seu funcionamento. Veremos esse problema detalhadamente mais adiante.

Durante o desenvolvimento deste trabalho, foram lançadas algumas versões do Protégé, desde a versão 3.1 até a última que foi a 3.3.1.

O Protégé ainda possui alguns detalhes a serem melhorados mas é uma ferramenta bastante poderosa para interagir com ontologias e foi uma ferramenta de grande importância neste projeto.

2.6 DIG

A interface DIG¹⁰ (*DL Implementation Group*) fornece acesso uniforme aos motores de inferências de Lógica de Descrição. A interface define um protocolo simples (baseado em HTTP PUT/GET) junto com esquemas XML que descrevem uma linguagem conceitual e operações relacionadas.

Podemos dizer que a interface DIG é um tradutor que permite por exemplo que arquivos RDF sejam transformados e enviados a um serviço web de raciocínio, que entenderá o arquivo de entrada já traduzido pelo DIG e retornará uma resposta.

⁹<http://protege.stanford.edu/>

¹⁰<http://dl.kr.org/dig/>

Através da interface DIG, é possível acoplar motores de inferência ao Protégé e testar consistência, por exemplo, de uma ontologia conforme ela vai sendo construída.

O DIG possui algumas limitações como por exemplo o fato de não comportar restrições de cardinalidade. Isso faz com que arquivos da linguagem OWL (incluindo arquivos OWL-DL) que possuam restrições de cardinalidade não possam ser perfeitamente convertidos pelo DIG e portanto, a integração entre Protégé e um motor de inferência através do DIG não funciona. Por essa razão, torna-se necessário que o motor de inferência seja utilizado diretamente no arquivo OWL de origem da ontologia em questão.

2.7 O motor de inferência Pellet

Pellet¹¹ é um motor de inferência (*reasoner*) OWL-DL de código aberto escrito em Java, originalmente desenvolvido no Laboratório Mindswap da Universidade de Maryland. Pellet é baseado em algoritmos de tableaux desenvolvidos para lógicas de descrição (DL) expressivas. Ele suporta a expressividade OWL-DL completa, incluindo raciocínio sobre nominais (classes enumeradas).

O motor de inferência Pellet fornece muitos serviços diferentes de raciocínio. Ele também incorpora várias técnicas de otimização descritas na literatura de DL e contém várias otimizações para nominais, resultados de consultas conjuntivas, e raciocínio incremental.

O pacote Pellet já possui uma versão do motor de inferência que funciona como um pequeno servidor web utilizando o protocolo DIG para interagir com outras aplicações. Através dessa interface web, é possível fazer com que o Protégé utilize diretamente motor de inferência Pellet, bastando apenas que a ontologia utilize somente construtores suportados pelo DIG.

Várias versões do Pellet foram utilizadas durante o período deste projeto, desde a versão 1.3 até a versão 1.5, incluindo versões completas e versões beta.

O Pellet ainda apresenta algumas limitações dentro do seu projeto mas notou-se grande evolução no avanço de suas versões.

Em suas primeiras versões, quando o Pellet era executado para verificar a consistência da ontologia com uma versão dela que possuía erros, o motor de inferência simplesmente encontrava o erro e não o relatava de forma muito clara, ou seja, relatava que existia um problema mas sem informar exatamente qual era o problema, dificultando o processo de depuração da ontologia.

¹¹<http://pellet.owldl.com/>

Nas versões seguintes, o Pellet mostrou melhorias e os erros que anteriormente apareciam sem muitos detalhes, passaram a ser bem descritos, facilitando bastante a depuração da ontologia.

O Pellet está evoluindo e podemos notar pela crescente quantidade de informações disponíveis na Internet relacionadas a esse motor de inferência que seu uso vem aumentando consideravelmente.

2.8 Interação entre Protégé e Pellet

O Protégé permite que o usuário configure qual motor de inferência prefere utilizar para realizar as verificações e classificações da ontologia. Neste projeto, estamos utilizando o motor de inferência Pellet, que foi escolhido após termos verificado também a possibilidade de usar os motores de inferências Racer¹² e FACT++¹³. O Racer é um motor de inferência que era usado por alguns alunos em seus trabalhos mas que agora tornou-se um software comercial e por isso perdeu espaço no meio acadêmico. O FACT++ ainda é um software que ainda não apresenta um bom funcionamento. Ainda não foi feita uma versão suficientemente boa para ser usada como um motor de inferência confiável. Além disso, o Pellet é o único que é específico para OWL, ou seja, não dependendo de traduções para lógica de descrição internamente no seu algoritmo.

A integração direta entre o Protégé e o Pellet é feita através da interface DIG. Assim que o Pellet é instalado no sistema, ele cria um executável que realiza o raciocínio em arquivos RDF e outro executável que funciona como um pequeno servidor web que utiliza a interface DIG para se comunicar com outros programas, como o Protégé. A integração do Protégé com o Pellet através da interface DIG dá um maior dinamismo ao desenvolvimento dentro do Protégé e facilita o desenvolvimento e a verificação de consistências na ontologia.

Entretanto, a interface DIG possui certas limitações, como por exemplo não ser compatível com restrições de cardinalidade e também com o uso de tipos de dados, que infelizmente a tornam um pouco fraca. Para que seja possível utilizar a facilidade de usar o Pellet de dentro do Protégé, é necessário abrir mão desses conceitos conflitantes para que a interface DIG funcione sem problemas. Em alguns casos, remover essas restrições da ontologia não é uma tarefa fácil, pois isso diminui a compreensibilidade ao permitir que informações irrelevantes ou sem uma semântica adequada possam ser inseridas nas instâncias da ontologia.

Uma alternativa para este problema é utilizar o Pellet diretamente com o arquivo OWL gerado

¹²<http://www.sts.tu-harburg.de/~r.f.moeller/racer/>

¹³<http://owl.man.ac.uk/factplusplus/>

pelo Protégé. O principal problema dessa solução é a perda da interação das funções de classificação, inferência e verificação de consistência internas do Protégé, o que acarreta uma perda de agilidade no desenvolvimento da ontologia.

Diante da complexidade da ontologia tratada neste projeto, optamos por não utilizar a interface DIG e portanto todos os testes feitos com a ontologia eram feitos diretamente no Pellet, acessando o arquivo OWL original da ontologia, conforme já foi explicado anteriormente.

2.9 SPARQL

SPARQL¹⁴ é uma linguagem de consulta que está sendo desenvolvida sob supervisão do W3C e que foi definida como a linguagem padrão para consultas da Web Semântica em arquivos RDF em Janeiro de 2008.

A sintaxe e a forma como são feitas as consultas na linguagem SPARQL é baseada no esquema de triplas que é a base de representação dos arquivos RDF.

Os resultados de uma consulta SPARQL podem ser ordenados, limitados e transladados por um *offset* (Ex: Se uma consulta normalmente devolve 20 resultados e o parâmetro *offset* for definido como 7, então essa consulta agora irá retornar apenas 13 resultados, sendo que os resultados retornados se iniciarão a partir do que seria o oitavo resultado sem o *offset*), assim como ocorre com a linguagem SQL para banco de dados.

Vejam os alguns exemplos simples de consultas da linguagem SPARQL que podem ser feitas na nossa ontologia:

1. Consulta que relaciona os pares (classes, superclasses) através da propriedade subClassOf:

```
SELECT ?classe ?superclasse
WHERE {
    ?classe rdfs:subClassOf ?superclasse .
    ?superclasse a owl:Class .
}
ORDER BY ?superclasse
```

Primeiros 5 resultados da resposta da Consulta:

¹⁴<http://www.w3.org/TR/rdf-sparql-query/>

classe	superclasse
ArtigosAceitosParaPublicacao	Artigos
ArtigosPublicados	Artigos
AtividadesDeConselhoComissaoEConsultoria	Atividades
AtividadesDeDirecaoEAdministracao	Atividades
AtividadesDeEnsino	Atividades

Trechos da ontologia (no formato Turtle) que geram os resultados da consulta:

```

default:ArtigosAceitosParaPublicacao
    rdf:type owl:Class ;
    rdfs:subClassOf default:Artigos .
.
.
.
default:ArtigosPublicados
    rdf:type owl:Class ;
    rdfs:subClassOf default:Artigos .
.
.
.
default:AtividadesDeConselhoComissaoEConsultoria
    rdf:type owl:Class ;
    rdfs:subClassOf default:Atividades ;
.
.
.
default:AtividadesDeDirecaoEAdministracao
    rdf:type owl:Class ;
    rdfs:subClassOf default:Atividades ;

```

```

.
.
.
default:AtividadesDeEnsino
    rdf:type owl:Class ;
    rdfs:subClassOf default:Atividades ;

```

2. Consulta que apresenta as instâncias que possuem a propriedade nomeDaEspecialidade e que tenham a *substring* “soft” como parte do valor de ?nome:

```

SELECT ?instancia ?nome
WHERE {
    ?instancia :nomeDaEspecialidade ?nome .
    FILTER regex(?nome, "soft", "i") .
}
ORDER BY DESC(?nome)

```

Resultados da resposta da Consulta:

instancia	nome
Especialidade_17	Software Básico
Especialidade_15	Métodos Ágeis de Desenvolvimento de Software
Especialidade_4	Engenharia de Software

Trecho da ontologia (no formato Turtle) que gera os resultados da consulta:

```

default:Especialidade_17
    rdf:type default:Especialidade ;
    default:nomeDaEspecialidade

```

```
        "Software Básico"^^xsd:string .  
.  
.  
.  
default:Especialidade_15  
    rdf:type default:Especialidade ;  
    default:nomeDaEspecialidade  
        "Métodos Ágeis de Desenvolvimento de Software"^^xsd:string .  
.  
.  
.  
default:Especialidade_4  
    rdf:type default:Especialidade ;  
    default:nomeDaEspecialidade  
        "Engenharia de Software"^^xsd:string .
```

Capítulo 3

O Currículo Lattes

O Currículo Lattes é hoje a principal ferramenta que armazena informações dos pesquisadores brasileiros. Os currículos da Plataforma Lattes armazenam de forma padronizada informações profissionais e acadêmicas dos pesquisadores, bem como informações sobre suas produções.

Vamos, neste capítulo, contar um pouco da história do Currículo Lattes, das suas principais características e dos problemas atualmente enfrentados pelos pesquisadores.

3.1 Introdução

O Currículo Lattes é um documento criado pelo CNPq com o objetivo de centralizar em um único lugar as informações acadêmicas, profissionais e até pessoais da comunidade científica do país.

Através do Currículo Lattes, os pesquisadores mantêm suas informações acadêmicas atualizadas, permitindo que outras pessoas tenham acesso a esses dados através da Web. Dessa forma, pode-se verificar as publicações e projetos já realizados e que ainda estão em desenvolvimento para cada pesquisador, além de todas as outras informações relevantes que podem ser encontradas como por exemplo informações da Instituição de Ensino que o pesquisador faz parte, seus contatos, orientações de Iniciação Científica, Mestrado e Doutorado realizados e em andamento.

A implantação do Currículo Lattes ocorreu em 1999 e trata-se de um repositório de dados sobre as atividades científicas dos pesquisadores que tem sido bastante utilizado, sendo até mesmo obrigatório para algumas agências de fomento. Entretanto, existem algumas limitações sobre a disponibilização dos dados na Web. Atualmente, para realizar a busca através da Web¹ por algum currículo cadas-

¹<http://buscatextual.cnpq.br/buscatextual/index.jsp>

trado, é necessário inserir um código numérico somente visível na tela do monitor, impedindo que se faça buscas de currículos por nomes de pesquisadores através de scripts.

Vale ressaltar que cada currículo publicado na plataforma Lattes possuem uma URL fixa de acesso direto e aberto, ou seja, conhecendo a URL referente a um determinado currículo, é possível acessá-lo automaticamente quantas vezes for necessário pois o acesso direto não exige a autenticação por imagem que existe na busca.

Apesar de o CNPq armazenar os currículos originalmente como documentos XML, o que é disponibilizado na Internet é apenas a versão HTML, sem nenhuma marcação específica para as diferentes informações, como é o caso do arquivo original XML.

Optamos por utilizar neste trabalho os arquivos HTML pois são os arquivos que estão disponíveis publicamente a qualquer pessoa, diferentemente dos arquivos originais XML que seriam mais simples de serem trabalhados mas que não estão disponíveis.

3.2 Relatórios baseados nos Currículos Lattes

Atualmente, existe um problema que grande parte dos pesquisadores enfrentam, que é a dificuldade na geração dos diversos relatórios a serem entregues para as instituições de ensino e agências de fomento. Como um exemplo, temos os relatórios dos programas de pós-graduação para a CAPES.

O processo consiste inicialmente na inserção de dados dos currículos dos pesquisadores. Depois, esses dados são exportados em arquivos no formato XML contendo suas informações, incluindo dados pessoais, publicações e projetos. Os arquivos gerados por vários professores de uma determinada instituição são consolidados através de um outro programa que gera um relatório global.

Após a geração desse relatório, os pesquisadores são obrigados a revisar o que foi gerado e acabam tendo que corrigir um grande número de informações. Existem alguns pequenos detalhes no preenchimento dos currículos que acabam gerando problemas no momento em que se gera um relatório global, como por exemplo, o aparecimento de uma mesma publicação mais de uma vez, pelo fato de ter sido escrita por mais de uma pessoa e o sistema não ser capaz de detectar e retirar essa duplicidade.

Esse problema ilustra bem a principal motivação deste projeto, ou seja, neste mestrado queremos, através do uso de uma ontologia, oferecer a possibilidade de que informações possam ser disponibilizadas e consolidadas automaticamente e com maior consistência semântica, ou seja, evitando que uma mesma informação apareça em duplicidade ou em um contexto errado.

Isso será feito utilizando o exemplo prático do caso de geração de relatórios a partir do Currículo Lattes. A idéia inicial do projeto é popular uma ontologia com as informações do Currículo Lattes e extrair relatórios dessa base de dados, baseados em consultas na linguagem SPARQL usando o poder da Web Semântica, esperando que os resultados sejam melhores do que os obtidos atualmente.

Capítulo 4

Criação da ontologia base

A ontologia que será utilizada neste trabalho é baseada na que foi criada por Ailton Sergio Bonifacio [4] no seu trabalho de mestrado. Esta primeira ontologia foi criada na linguagem DAML+OIL e depois foi melhorada por Marcos Yoshinori Nakashima [19] na sua Iniciação Científica, que teve como um dos objetivos transformar a ontologia que estava em DAML+OIL para a linguagem OWL.

Essa ontologia gerada por Marcos foi o princípio da ontologia que efetivamente está sendo usada neste trabalho, ou seja, a partir da ontologia final OWL do trabalho do Marcos, realizamos todo o trabalho de verificação e análise para adequar a ontologia para o nosso uso, modificando classes e propriedades de acordo com o que era necessário, tentando sempre seguir as boas práticas de modelagem indicadas pela literatura [21] [11].

Os dois trabalhos citados [4] [19] estão relacionados com o uso da Web Semântica com o Currículo Lattes e foram o princípio do desenvolvimento deste projeto.

Veremos, a seguir, como a ontologia base foi modificada, como foi feita sua modelagem e também outras informações relevantes desta que é a base para todo o restante do trabalho.

O arquivo com a ontologia está disponível em <http://www.ime.usp.br/~casado/Mestrado/Arquivos/>.

4.1 Introdução

Uma ontologia descreve os conceitos e relações que são importantes em um domínio particular, como já foi explicado anteriormente na seção 2.1.

Neste trabalho, estamos interessados no domínio do Currículo Lattes. Trata-se de um domínio

complexo, com muitas relações entre seus elementos. Isso torna o trabalho de modelagem da ontologia difícil, sendo necessário que todo o trabalho seja feito com bastante atenção.

Entretanto, apesar da dificuldade de se trabalhar com uma ontologia grande, utilizamos como base da modelagem o arquivo DTD do Currículo Lattes, arquivo que basicamente descreve todas as informações existentes no Currículo Lattes e a forma como essas informações estão relacionadas. Vejamos a seguir um exemplo de como uma descrição existente no arquivo DTD se transforma em uma modelagem dentro da Ontologia.

4.2 Modelando a ontologia baseado no DTD do Currículo Lattes

A ontologia inicialmente utilizada foi produzida por Marcos [19] no seu trabalho de Iniciação Científica. Seu trabalho consistiu em transformar uma ontologia do Currículo Lattes que antes estava em DAML+OIL para a linguagem OWL.

Um dos trabalhos realizados neste projeto foi analisar essa ontologia criada anteriormente e corrigir defeitos na sua modelagem. Muitos dos problemas na modelagem somente puderam ser percebidos durante o processo de instanciação. Por essa razão, podemos dizer que a modelagem da ontologia foi sendo feita no decorrer de todo o desenvolvimento do trabalho, sempre procurando adaptá-la para solucionar os problemas gerados nas instanciações.

Para auxiliar a modelagem, utilizamos como base o DTD do Currículo Lattes.

A modelagem é realizada basicamente seguindo sempre o mesmo procedimento. Para cada elemento existente no DTD, fazemos uma espécie de transformação desse elemento para uma classe da ontologia, incluindo também as suas propriedades.

Vamos usar um exemplo para detalhar o processo.

Na figura 4.1, temos um trecho do arquivo DTD que contém o elemento Endereço Profissional e suas propriedades.

```
<!ELEMENT ENDERECO-PROFISSIONAL EMPTY>
<!ATTLIST ENDERECO-PROFISSIONAL
  CODIGO-INSTITUICAO-EMPRESA CDATA #IMPLIED
  NOME-INSTITUICAO-EMPRESA CDATA #IMPLIED
  CODIGO-ORGAO CDATA #IMPLIED
  NOME-ORGAO CDATA #IMPLIED
  CODIGO-UNIDADE CDATA #IMPLIED
  NOME-UNIDADE CDATA #IMPLIED
  LOGRADOURO-COMPLEMENTO CDATA #IMPLIED
  PAIS CDATA #IMPLIED
  UF CDATA #IMPLIED
  CEP CDATA #IMPLIED
  CIDADE CDATA #IMPLIED
  BAIRRO CDATA #IMPLIED
  DDD CDATA #IMPLIED
  TELEFONE CDATA #IMPLIED
  RAMAL CDATA #IMPLIED
  FAX CDATA #IMPLIED
  CAIXA-POSTAL CDATA #IMPLIED
  E-MAIL CDATA #IMPLIED
  HOME-PAGE CDATA #IMPLIED
>
```

Figura 4.1: Arquivo DTD - Endereço Profissional.

Agora, vamos mostrar na figura 4.2 uma visualização no Protégé da classe criada dentro da ontologia a partir desse elemento.

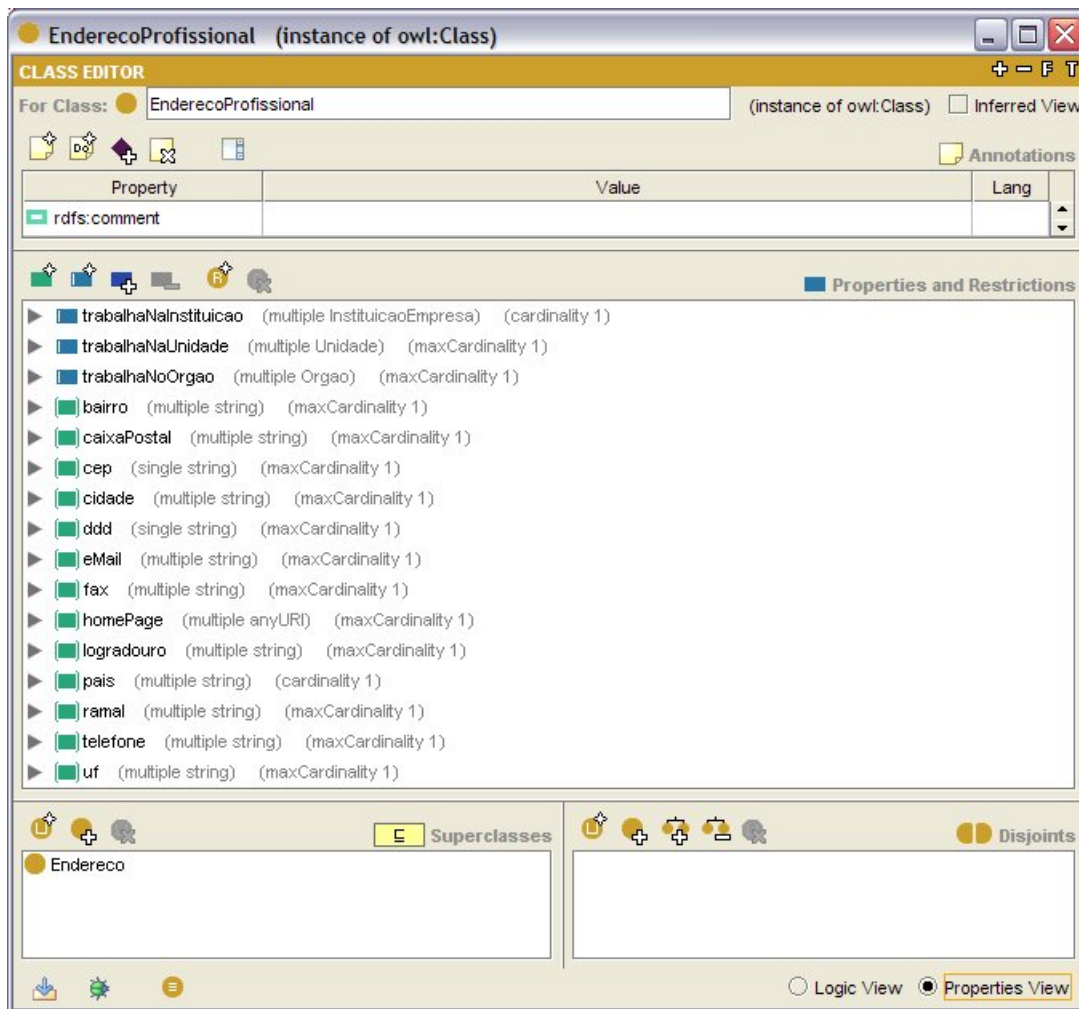


Figura 4.2: Ontologia - Endereço Profissional.

Apresentamos também a seguir o código (no formato Turtle) desse trecho da ontologia.

```
default:Endereco
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
  rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
```

```
        owl:onProperty default:uf
    ] ;
rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:cep
    ] ;
rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:caixaPostal
    ] ;
rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:ddd
    ] ;
rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:cardinality "1"^^xsd:int ;
      owl:onProperty default:pais
    ] ;
rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:email
    ] ;
rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:bairro
    ] ;
rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:telefone
    ] ;
rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
```

```

        owl:onProperty default:logradouro
    ] ;
    rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:ramal
    ] ;
    rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:cidade
    ] ;
    rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:fax
    ] ;
    rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:homePage
    ] .
.
.
.
default:EnderecoProfissional
    rdf:type owl:Class ;
    rdfs:subClassOf default:Endereco ;
    rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:trabalhaNoOrgao
    ] ;
    rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:maxCardinality "1"^^xsd:int ;
      owl:onProperty default:trabalhaNaUnidade
    ] ;
    rdfs:subClassOf
    [ rdf:type owl:Restriction ;

```

```
owl:cardinality "1"^^xsd:int ;  
owl:onProperty default:trabalhaNaInstituicao  
] .
```

Basicamente, a idéia do processo é fazer uma relação direta entre os elementos do arquivo DTD e classes na ontologia. Cada elemento representa uma classe e as propriedades dos elementos transformam-se diretamente em propriedades da classe, tomando cuidado para que os tipos de dados e as cardinalidades fiquem corretas. Esse foi o procedimento adotado para criar a modelagem principal da ontologia. Como dito anteriormente, alguns ajustes foram feitos para que a modelagem ficasse melhor. Podemos citar como exemplo as classes Idioma e proficienciaEmIdioma. No arquivo DTD, não existem esses elementos definidos da forma como foi definido na modelagem. Entretanto, sabendo que o universo de idiomas é limitado e buscando tornar a modelagem mais correta, foi criada a classe Idioma que tem uma instância para cada idioma e também criada a classe proficienciaEmIdioma, onde é feita a relação da proficiência de cada pesquisador em cada idioma.

Existem algumas propriedades que foram inseridas na ontologia mas que não são utilizadas dentro deste trabalho. Isso ocorre porque existem propriedades que são informações não disponibilizadas na Web pela Plataforma Lattes, ou seja, informações que não podem ser preenchidas a partir dos arquivos HTML dos currículos. De qualquer maneira, para que a modelagem fique completa e possa ser utilizada para outros fins, essas propriedades foram mantidas dentro da ontologia.

A modelagem da ontologia base do Currículo Lattes foi feita sempre tentando seguir as melhores práticas de criação de ontologias [21]. Dificilmente teremos uma modelagem sem nenhum tipo de problema, mesmo porque alguns problemas só aparecem quando utiliza-se a ontologia de alguma forma diferente do que se esperava no princípio. Entretanto, pode-se dizer que a modelagem criada está atendendo sem problemas as necessidades deste trabalho.

4.3 Verificação da ontologia

O motor de inferência Pellet permite que seja verificada uma determinada ontologia, dizendo se ela é OWL-Full, OWL-DL ou OWL-Lite. A ontologia elaborada neste trabalho é OWL-DL e o objetivo no decorrer de todo o desenvolvimento do trabalho sempre foi de manter a ontologia compatível com OWL-DL.

Durante todo o processo de desenvolvimento, vários pequenos problemas de inconsistências foram

encontrados dentro da ontologia. O processo de depuração e correção da ontologia era:

1. Salvar a ontologia populada em um arquivo OWL.
2. Executar o Pellet, para que ele verifique a consistência e a classificação da ontologia populada, usando como entrada o arquivo OWL.
3. Se a ontologia não apresentar problemas de consistência, o Pellet retornará a classificação da ontologia. Se a ontologia apresentar problemas, então iniciamos a depuração da ontologia, olhando os erros apresentados pelo motor de inferência para corrigí-los.

Muitos dos problemas somente apareciam quando a ontologia era populada, ou seja, quando instâncias eram criadas. Por essa razão, o processo de depuração da ontologia foi realizado durante todo o trabalho, pois era necessário verificar a consistência da ontologia para cada novo tipo de instância inserida.

Os principais problemas detectados pelo motor de inferência eram:

- Uma propriedade ser simultaneamente *Object Property* e *Datatype Property*
- Uma instância conter uma propriedade com o tipo de dados diferente do que foi declarado na classe, ou seja, se temos uma classe que contém uma propriedade com o tipo de dados *string*, as instâncias obrigatoriamente tem que ser declaradas com a mesma propriedade usando o mesmo tipo de dados *string*. Esse problema parece ser óbvio mas, no processo de extração de informações desde HTML e posterior mapeamento para a ontologia, as falhas em relação ao tipo de dados aconteciam com frequência e portanto vale o destaque para esse problema.

A evolução das novas versões do Pellet também auxiliou no processo de depuração da ontologia pois as informações retornadas por ele referentes aos erros encontrados na ontologia apresentavam-se mais detalhadas e completas, ou seja, mostrando com maior precisão onde foi encontrado o erro e dizendo exatamente o tipo de problema.

Capítulo 5

População da Ontologia do Currículo Lattes

Nos capítulos anteriores, falamos do Currículo Lattes e da ontologia que foi criada para o Currículo Lattes, através do arquivo DTD. Essa ontologia com toda a modelagem pronta será agora utilizada e iremos popular a ontologia através de um processo que começa com a leitura das informações de Currículos Lattes disponíveis na Web e posterior inserção dessas informações dentro da Ontologia. Vejamos, a seguir, os detalhes de todo esse processo.

Os scripts que serão descritos neste capítulo podem ser encontrados em <http://www.ime.usp.br/~casado/Mestrado/Arquivos/>.

5.1 Introdução

Foi construído um script em Perl para ler os arquivos HTML dos currículos, extrair as informações desses arquivos e popular uma ontologia OWL.

A linguagem Perl foi escolhida por dois principais motivos: por ser uma linguagem poderosa e fácil para trabalhar com textos e pela experiência anterior adquirida por mim, tendo já realizado diversos trabalhos nessa linguagem, incluindo um projeto de iniciação científica e muitos outros scripts desenvolvidos em ambiente profissional.

O script recebe como entrada um diretório contendo arquivos HTML de currículos da Plataforma Lattes.

Todos os arquivos de currículos contidos nesse diretório serão inseridos na base de conhecimento final¹. A entrada do script poderia, até mesmo, ser apenas uma lista de URLs, representando os

¹Chamamos de base de conhecimento a ontologia populada com informações.

currículos, que poderiam ser dinamicamente baixados da Web em tempo real. No entanto, para o trabalho que está sendo feito, acreditamos que não haja necessidade de implementar essa funcionalidade e portanto estamos trabalhando com uma base de currículos salva localmente.

O processo de extração das informações desde o arquivo HTML e a inserção na ontologia são realizados separadamente para cada currículo, ou seja, cada passo de execução do script consiste em ler um arquivo HTML, procurar marcas dentro desse arquivo para extração das informações, classificar as informações e inserí-las na ontologia final. Os passos se repetem para cada um dos currículos até que no final tenhamos a ontologia populada com dados de todos os arquivos.

Vamos inicialmente apresentar um pseudo-código do algoritmo para a extração das informações e criação da base de conhecimento, dando uma visão geral do processo:

Algoritmo 1: Algoritmo de criação da Base de conhecimento a partir de arquivos HTML

```

foreach ( currículo: cada arquivo de currículo em HTML ) do
  currículo_dividido ← ler_curriculo_e_dividir_em_secoes() ;
  foreach ( parte: cada parte do currículo_dividido ) do
    seção ← identificar_secao(parte) ;
    foreach informação: cada informação diferente da seção seção ) do
      if ( informação ainda não existe na ontologia populada ) then
        inserir_informacao_no_arquivo_owl(informação) ;
      end
    end
  end
end

```

A seguir, vamos descrever em mais detalhes os passos executados pelo script para cada currículo.

5.2 Leitura e extração de informações dos arquivos HTML

Como dito anteriormente, o arquivo original que estamos trabalhando é um arquivo HTML. O arquivo HTML tem como finalidade permitir uma boa visualização das informações pelos usuários que acessam a Plataforma Lattes na Web. Isso faz com que os dados sejam apresentados de forma clara e com boa aparência para os usuários que estão acessando as informações através de um navegador. Entretanto, as marcas que facilitam a visualização através de um navegador acabam dificultando a extração do conteúdo uma vez que não existem marcas específicas definindo e separando cada uma

das informações. Essa falta de marcas específicas separando as informações dentro de cada arquivo é a primeira dificuldade encontrada no processo de transformação de HTML para OWL. Além da falta de marcas separadoras específicas, o arquivo HTML naturalmente possui marcas que auxiliam somente na parte visual mas que as vezes podem ser uma dificuldade a mais quando estamos retirando informações do seu código.

No trabalho de mestrado de Silva [24], existe uma classificação relacionada com a maneira que se realiza a extração de informação (EI) a partir de um texto. Existem duas principais subdivisões de aborgadens de como realizar a EI. São elas: EI baseado em conhecimento e EI baseado em *wrappers*. Os sistemas baseados em conhecimento, em geral, utilizam técnicas de Processamento de Linguagem Natural, que usam aprendizado para aquisição de conhecimento aplicando técnicas entre termos e sentenças. Os sistemas baseados em *wrappers* atuam em textos semi-estruturados, identificando dados de interesse e mapeando-os para um formato estruturado.

No caso deste trabalho, estamos lidando com códigos HTML bem definidos, ou seja, que apresentem padrões bem definidos na forma como as marcas aparecem dentro do arquivo. Por essa razão, utilizaremos a idéia de *wrappers* como base para fazer a EI nos arquivos HTML. Vale ressaltar que em algumas partes do processo serão usados padrões de textos que não necessariamente são marcas HTML. Veremos mais adiante que para identificar algumas informações mais detalhadas, utilizaremos caracteres de texto como por exemplo “vírgulas” e “dois pontos”. Além disso, para buscarmos as marcas HTML, utilizaremos expressões regulares. Isso não impede que nosso sistema de EI possa ser classificado como um sistema baseado em *wrappers*.

Para que fosse possível então extrair corretamente a informação dos arquivos originais HTML, foi feita uma análise do código fonte dos arquivos HTML na busca de *strings* que pudessem servir de marcas para a leitura das informações.

No trabalho de Nanno et al. [26] é proposta uma forma de se extrair informação estruturada a partir da repetição de elementos de um código HTML. Podemos dizer inicialmente que essa é a idéia que vamos utilizar para extrair as informações dos currículos. No entanto, é importante notar que no nosso caso, teremos uma busca mais específica por elementos repetitivos e necessitamos de uma extração de informações bastante precisa, um pouco diferente da abordagem proposta que é mais generalista e pode não conseguir encontrar algumas informações dentro de um arquivo HTML. De qualquer maneira, esse método proposto de estruturação de informações é bastante útil para trabalhos que utilizem informações de códigos HTML.

A maneira como estamos fazendo a extração de informações a partir de arquivos HTML neste trabalho utiliza um pouco da idéia de alguns trabalhos na área [26] [8] [15] [5]. A primeira parte do processo de extração de informações dos arquivos HTML é baseada na divisão do arquivo em seções separadas por uma marca que se repete sempre entre as seções e é justamente essa a idéia base dos trabalhos citados. Nanno et al. [26] propõe uma forma de buscar marcas HTML repetitivas para encontrar informações relevantes. No caso deste trabalho, essa busca por termos repetitivos foi feita manualmente, ou seja, os arquivos de origem foram lidos e visualmente foram procuradas marcas HTML repetitivas entre as seções que pudessem ser usadas para dividi-las dentro de um currículo, pois não podemos ter erros nessa divisão e qualquer resultado que não seja 100% eficiente nesse passo não atende as necessidades do trabalho. Veremos detalhadamente, mais adiante, como funciona esse processo.

A dificuldade, no caso deste trabalho, será a busca das informações específicas dentro dessas seções previamente separadas. A falta de marcações HTML faz com que essa busca de informações específicas exija que sejam utilizadas expressões regulares um pouco complexas para evitar que informações sejam classificadas de forma errada.

Vamos, então, detalhar os passos do processo de extração das informações.

O primeiro passo, portanto, foi procurar separar o conteúdo todo em seções, de acordo com divisões feitas nas informações dos currículos (seguindo as divisões da estrutura de um currículo, como pode ser visto no arquivo DTD do Currículo Lattes).

Mostraremos a seguir a figura [5.1](#) contendo uma parte de um Currículo Lattes.

The image shows a screenshot of a Lattes Curriculum with five sections, each in a blue-bordered box with a left-pointing arrow icon in the bottom right corner.

- Dados pessoais**:

Nome	André Casado Castano
Nome em citações bibliográficas	CASTANO, A. C.
Sexo	Masculino
Endereço profissional	Instituto de Matemática e Estatística. Cidade Universitária 05508-090 - Sao Paulo, SP - Brasil
Endereço eletrônico	casado@ime.usp.br
- Formação acadêmica/Titulação**:

1999 - 2003	Graduação em Bacharelado em Ciência da Computação. Instituto de Matemática e Estatística, IME USP, Brasil.
--------------------	------------------------------------------------------------------------------------------------------------
- Atuação profissional**: (Empty section)
- Áreas de atuação**: (Empty section)
- Idiomas**:

Compreende	Inglês (Razoavelmente).
Fala	Inglês (Razoavelmente).
Lê	Inglês (Bem).
Escreve	Inglês (Razoavelmente).

Figura 5.1: Parte de um Currículo Lattes, mostrando as seções: Dados Pessoais, Formação Acadêmica, Atuação Profissional, Áreas de Atuação e Idiomas.

O código HTML referente ao trecho representado na figura 5.1 é apresentado a seguir:

```
<div>
<a name="DadosPessoais"></a>
<a style="width:180px; position: relative; top: 1px; float: left;" class="aba">
```

```

<b class="b1"></b>
<b class="b2"></b><span align="left" class="conteudo">Dados pessoais</span></a>
<div id="caixa">
<b class="top"><b class="b-um"></b><b class="b-dois"></b><b class="b-tres">
</b><b class="b-quatro"></b></b>

<table width="746px" class="IndicProdTabela">
<tr class="IndicProdTabelaLinha">
<td width="148px" class="DadGerTabelaCelula26">Nome</td>
<td width="624px" class="DadGerTabelaCelula74">Andr&eacute; Casado Castano</td>
</tr>
<tr class="IndicProdTabelaLinha">
<td width="148px" class="DadGerTabelaCelula26">
Nome em cita&ccedil;&otilde;es bibliogr&aacute;ficas</td>
<td width="624px" class="DadGerTabelaCelula74">CASTANO, A. C.</td>
</tr>
.
.
.
</div>
<p align="left"></p>
<a name="Atuacaoprofissional"></a>
<a style="width:180px; position: relative; top: 1px; float: left;" class="aba">
<b class="b1"></b><b class="b2"></b><span align="left" class="conteudo">
Atua&ccedil;&atilde;o profissional</span></a>
<br>
<div id="caixa">
<b class="top"><b class="b-um"></b><b class="b-dois"></b><b class="b-tres"></b>
.
.
.
<p></p>
<a name="Areasdeatuacao"></a>
<a style="width:180px; position: relative; top: 1px; float: left;" class="aba">
<b class="b1"></b><b class="b2"></b><span align="left" class="conteudo">
&Aacute;reas de atua&ccedil;&atilde;o</span></a>
<div id="caixa">
<b class="top"><b class="b-um"></b><b class="b-dois"></b>
<b class="b-tres"></b>
<b class="b-quatro"></b></b>

```

```

<table width="746px" class="IndicProdTabela"></table>
.
.
.
</div>
<p></p>
<a name="Idiomas"></a>
<a style="width:180px; position: relative; top: 1px; float: left;" class="aba">
<b class="b1"></b><b class="b2"></b><span align="left" class="conteudo">
Idiomas</span></a>
<div id="caixa">
<b class="top"><b class="b-um"></b><b class="b-dois"></b><b class="b-tres"></b>
<b class="b-quatro"></b></b>
<table width="746px" class="IndicProdTabela">
<tr valign="top" class="IndicProdTabelaLinha">
<td width="148px" class="DadGerTabelaCelula10Gray"> Compreende </td>
<td width="624px" class="DadGerTabelaCelula90">Ingl&ecirc;s (Razoavelmente).</td>
</tr>
.
.
.
<div class="linha">
<a href="#Indice">
</a>
</div>
</div>
</div>

```

Analisando o código HTML dos currículos em busca de textos repetitivos entre as seções, encontramos um padrão no código na parte inicial de cada seção e portanto esse padrão foi utilizado como separador das seções. Para encontrar as *strings* que seguissem esse padrão, foi criada uma expressão regular que asseguraria que nenhum separador deixaria de ser corretamente encontrado.

A expressão regular que encontrou todas as *strings* de separações, seguindo a sintaxe da linguagem Perl, foi:

```

(<a name="\[^"]*\")?.{1,250}?<div id="\caixa">|
(?:class="\AtuaProfTabelaCelula95">\s*(<a name="\[^"]*\")

```

Podemos notar que a expressão regular possui um OU lógico porque temos, na verdade, duas variações de *strings* que são usadas como separadoras. A utilização somente da primeira parte da expressão regular não permitia que a separação das seções fosse feita corretamente por completo com algumas partes presentes em alguns currículos.

Através dessa expressão regular, portanto, foi concretizado o primeiro passo de dividir o arquivo inteiro em seções, onde cada seção representa uma parte do currículo como por exemplo “Dados Pessoais” ou “Atuação Profissional”.

Outra maneira de fazer a divisão do arquivo em seções seria buscando no texto as palavras chaves dos títulos das seções, ou seja, ao invés de procurar por uma única expressão regular para dividir o texto, poderíamos buscar diretamente no arquivo as seções. Por exemplo, na seção de “Dados Pessoais”, buscaríamos por uma expressão ou *string* que pudesse identificar somente essa seção. Entretanto, utilizando essas buscas diretas sem uma expressão regular mais geral, o script ficaria bem menos flexível no caso de uma mudança no layout do Currículo Lattes e procuramos no decorrer de todo desenvolvimento do script torná-lo, na medida do possível, adaptável a mudanças de layout. No entanto, não podemos deixar de dizer que essa abordagem de busca direta no texto de entrada para cada seção seria uma opção que provavelmente traria bons resultados também.

Apesar de todo o cuidado para tornar o script flexível, vale ressaltar que quando mudanças de layout ocorrerem, serão necessárias adaptações no script para que ele torne a gerar os resultados desejados. É difícil quantificar quantas exatamente seriam as mudanças necessárias, pois isso depende diretamente das mudanças efetuadas no layout. Entretanto, no caso de mudanças simples de visual, é bem provável que com poucas modificações seja possível tornar o script efetivo novamente.

Seguindo com a extração das informações, continuamos com o processo seguinte que consiste em ler cada uma das seções e extrair a informação detalhada dentro dessas seções.

Entende-se por informação detalhada a informação já da maneira que será utilizada na classificação existente na ontologia. Exemplificando, podemos dizer que “Sexo” e “Nome” são informações detalhadas retiradas da seção “Dados Pessoais”.

Mais uma vez buscamos padrões de *strings* que pudessem ajudar na localização das informações úteis. Para encontrar as informações detalhadas, a busca por padrões vai se tornando um pouco mais complexa pois diferentes tipos de informações são mostrados de maneiras distintas. Isso significa que o código HTML para determinadas informações é diferente de outros e portanto é necessário buscar

padrões de textos específicos para alguns tipos de informações.

Para exemplificar a diferença dos códigos HTML e do tratamento necessário, vamos ver dois trechos retirados de um currículo e comentar um pouco do processo de extração das informações a partir deles e mostrar também o código OWL gerado ao final do processo.

Exemplo 1:

```
<tr valign="top" class="IndicProdTabelaLinha">
<td width="148px" class="DadGerTabelaCelula10Gray"> Compreende </td>
<td width="624px" class="DadGerTabelaCelula90">
Ingl&ecirc;s (Bem), Alem&atilde;o (Bem), Franc&ecirc;s (Bem),
Espanhol (Bem), Italiano (Bem), Holand&ecirc;s (Bem).</td>
</tr>
```

O Exemplo 1 contém informações de idiomas (Proficiência de Compreensão dos idiomas) de um pesquisador. Vamos ver agora um outro trecho:

Exemplo 2:

```
<tr valign="top" class="IndicProdTabelaLinha">
<td width="148px" class="AreadeAtuacaoCelula5">2. </td>
<td width="624px" class="textoProducao"><i>Grande &aacute;rea:
</i><i>Ci&ecirc;ncias Exatas e da Terra /
<i>&Aacute;rea: </i><i>Ci&ecirc;ncia da Computa&ccedil;&atilde;o /
<i>Sub&aacute;rea:
</i>Metodologia e T&eacute;cnicas da Computa&ccedil;&atilde;o /
<i>Especialidade: </i>Sistemas de Informa&ccedil;&atilde;o. <br></td>
</tr>
```

No Exemplo 2, temos informações sobre uma Área de Atuação de um pesquisador. Notamos que temos a primeira linha igual para ambos os trechos e a partir da segunda linha começam as diferenças de código entre os diferentes tipos de informações.

Agora vejamos o código OWL gerado pelo script, baseado nesses trechos de entrada:

Código OWL relativo ao Exemplo 1:

```
<temIdiomas>
  <Idiomas rdf:ID="Idiomas_0004_Ingles">
    <idioma rdf:resource="#ingles"/>
    <proficienciaDeCompreensao rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Bem</proficienciaDeCompreensao>
    </Idiomas>
  </temIdiomas>
<temIdiomas>
  <Idiomas rdf:ID="Idiomas_0004_Italiano">
    <idioma rdf:resource="#italiano"/>
    <proficienciaDeCompreensao rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Bem</proficienciaDeCompreensao>
    </Idiomas>
  </temIdiomas>
<temIdiomas>
  <Idiomas rdf:ID="Idiomas_0004_Alemao">
    <idioma rdf:resource="#alemao"/>
    <proficienciaDeCompreensao rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Bem</proficienciaDeCompreensao>
    </Idiomas>
  </temIdiomas>
<temIdiomas>
  <Idiomas rdf:ID="Idiomas_0004_Holandes">
    <idioma rdf:resource="#holandes"/>
    <proficienciaDeCompreensao rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Bem</proficienciaDeCompreensao>
    </Idiomas>
  </temIdiomas>
<temIdiomas>
  <Idiomas rdf:ID="Idiomas_0004_Espanhol">
    <idioma rdf:resource="#espanhol"/>
    <proficienciaDeCompreensao rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Bem</proficienciaDeCompreensao>
    </Idiomas>
  </temIdiomas>
<temIdiomas>
  <Idiomas rdf:ID="Idiomas_0004_Frances">
    <idioma rdf:resource="#frances"/>
    <proficienciaDeCompreensao rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

```

    Bem</proficienciaDeCompreensao>
  </Idiomas>
</temIdiomas>

```

Agora vamos ver o código OWL para o segundo exemplo, apresentando uma estrutura bastante diferente e mais complexa do que a do Exemplo 1.

Código OWL relativo ao Exemplo 2:

```

<temAreadeAtuacao>
  <AreasDeAtuacao rdf:ID="AreasDeAtuacao_Sistemas de Informacao">
    <especialidade rdf:resource="#Especialidade_9"/>
  </AreasDeAtuacao>
</temAreadeAtuacao>
.
.
.
<GrandeAreaDeConhecimento rdf:ID="GrandeAreaDeConhecimento_5">
  <nomeDaGrandeArea rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Ciencias Exatas e da Terra</nomeDaGrandeArea>
  <temArea>
    <Area rdf:ID="Area_6">
      <nomeDaArea rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Ciencia da Computacao</nomeDaArea>
      <temSubArea>
        <SubArea rdf:ID="SubArea_7">
          <nomeDaSubArea rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
            Metodologia e Tecnicas da Computacao</nomeDaSubArea>
          <temEspecialidade>
            <Especialidade rdf:ID="Especialidade_9">
              <nomeDaEspecialidade rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                Sistemas de Informacao</nomeDaEspecialidade>
            </Especialidade>
          </temEspecialidade>
        </SubArea>
      </temSubArea>
    </Area>
  </temArea>
</GrandeAreaDeConhecimento>

```

```
</temArea>  
</GrandeAreaDeConhecimento>
```

Notamos a partir dos fragmentos anteriores como dois pequenos trechos do código HTML original de um currículo necessitam transformar-se em códigos OWL bastante diferentes por representarem conhecimento distinto. Isso ilustra bem o fato de que é bastante difícil generalizar a conversão de HTML para OWL neste caso do Currículo Lattes, pois informações que possuem pequenas diferenças no código HTML podem ter uma modelagem completamente diferente dentro da ontologia, por representarem informações distintas, e portanto necessitam de um tratamento totalmente diferenciado.

O fato de termos códigos HTML diferentes em seções diferentes é apenas o primeiro problema com relação a extração da informação detalhada. Boa parte das informações dos currículo, principalmente da parte da Produção Bibliográfica, possui muitas informações detalhadas que sequer possuem marcas HTML separadoras. Diante desse problema, passamos agora a realizar um tratamento diferente nos textos. Ao invés de buscarmos por padrões em marcas HTML, passamos agora a buscar por quaisquer tipo de padrões de textos que possam nos ajudar a separar e identificar diferentes tipos de informações.

Vejamos um exemplo da citação de um Artigo:

```
<td width="702px" class="textoProducao">WASSERMANN, R. . Generalized  
Change and the Meaning of Rationality Postulates.  
Studia logica, v. 73, n. 2, p. 299-319, 2003. </td>
```

Neste trecho de código apresentado, identificamos as seguintes informações detalhadas:

Tipo de Informação	Valor
Autor	WASSERMANN, R
Título	Generalized Change and the Meaning of Rationality Postulates
Título do Periódico ou Revista	Studia logica
Volume	73
Fascículo	2
Página Inicial	299
Página Final	319
Ano	2003

No exemplo apresentado, vemos que não existem marcas HTML separando as diferentes informações e portanto fez-se necessário o uso de expressões regulares para identificar e separar as informações contidas em textos sem marcas.

Cada tipo de Produção Bibliográfica, por exemplo, possui diferentes tipos de informações. Quando tratamos de um Artigo, temos informações detalhadas como: página inicial, página final, título, título do periódico ou revista, fascículo, volume, etc. Se olharmos para um Livro Publicado teremos as seguintes informações detalhadas: título, número de volumes, nome da editora, cidade da editora, número de páginas, etc.

Esses dois exemplos apresentados demonstram claramente que as informações existentes nos currículos apresentam-se sempre de maneiras variadas e portanto cada diferente tipo de informação precisa de tratamento específico. No caso das Produções Bibliográficas, podemos dizer que cada tipo de Produção requer expressões regulares diferentes para extrair corretamente suas informações detalhadas.

Algumas vezes, mesmo dentro de um mesmo tipo de Produção, foi necessário usar variações nas expressões regulares, ou seja, somente uma expressão regular não foi capaz de retirar corretamente as informações. Isso ocorre pela heterogeneidade na forma como os pesquisadores inserem os dados. Algumas citações possuem informações completas, outras não tem alguns detalhes como por exemplo *número de páginas*, ou então sem o *fascículo*. Essas diferenças podem não aparentar ter muita importância mas conseguir encontrar expressões regulares que tratem de informações que são opcionais (ou seja, que podem não estar presentes se o usuário não as inserir quando cria seu currículo) foi um

dos desafios nessa parte do processo de extração das informações e sem dúvida algo que demandou um esforço muito grande.

Após essa análise do tratamento diferenciado que cada seção do currículo recebe, seguimos com o processo de leitura do arquivo HTML e extração das informações.

5.3 Processamento das informações extraídas

A partir das informações captadas no arquivo HTML, é criada uma estrutura de dados que, para cada currículo, divide as informações em seções da forma como foi explicado anteriormente e dentro de cada seção pode-se encontrar um segundo nível de divisão de informações. Dentro desse segundo nível encontram-se as informações que estavam contidas originalmente, ou seja, as informações detalhadas que serão efetivamente inseridas na base de conhecimento.

Com essa estrutura de dados criada após a leitura do arquivo HTML, seguimos então para a segunda parte do processo que consiste em colocar essas informações lidas corretamente dentro da base de conhecimento que estamos gerando. Para isso, necessitamos saber exatamente como classificar cada parte de informação para que possamos inserí-la corretamente na base.

Esta parte do processo começa com o uso de uma ontologia que serve de base para a inserção das informações, ou seja, antes do processo de leitura dos currículos, já temos um arquivo OWL com uma ontologia montada do Currículo Lattes (ver seção 4). Essa ontologia somente possui as definições de classes, propriedades e relações, sem ter nenhum tipo de instâncias. A idéia é que esse arquivo base carregue o modelo de classes da ontologia do Currículo Lattes e que o script, a partir desse arquivo, popule a ontologia de modo que ao final do processo, temos uma base de conhecimento com o modelo de classes completo juntamente de instâncias criadas a partir dos arquivos HTML dos Currículos Lattes utilizados.

Usando então o arquivo base, fazemos a leitura das informações de cada currículo e de acordo com cada tipo de informação, populamos a ontologia anexando à ontologia modelo as informações das instâncias. Nessa parte do processo, vale ressaltar a importância de fazer corretamente o vínculo entre informações que pertencem a classes diferentes para que elas fiquem corretamente classificadas na base de conhecimento de maneira que informações de um determinado currículo não fiquem perdidas ou até mesmo sejam relacionadas erroneamente com informações de outros currículos.

Para verificar a importância do correto vínculo entre classes, vamos examinar um exemplo.

Vamos apresentar uma instância da Classe *CurriculoVitae* na figura 5.2.

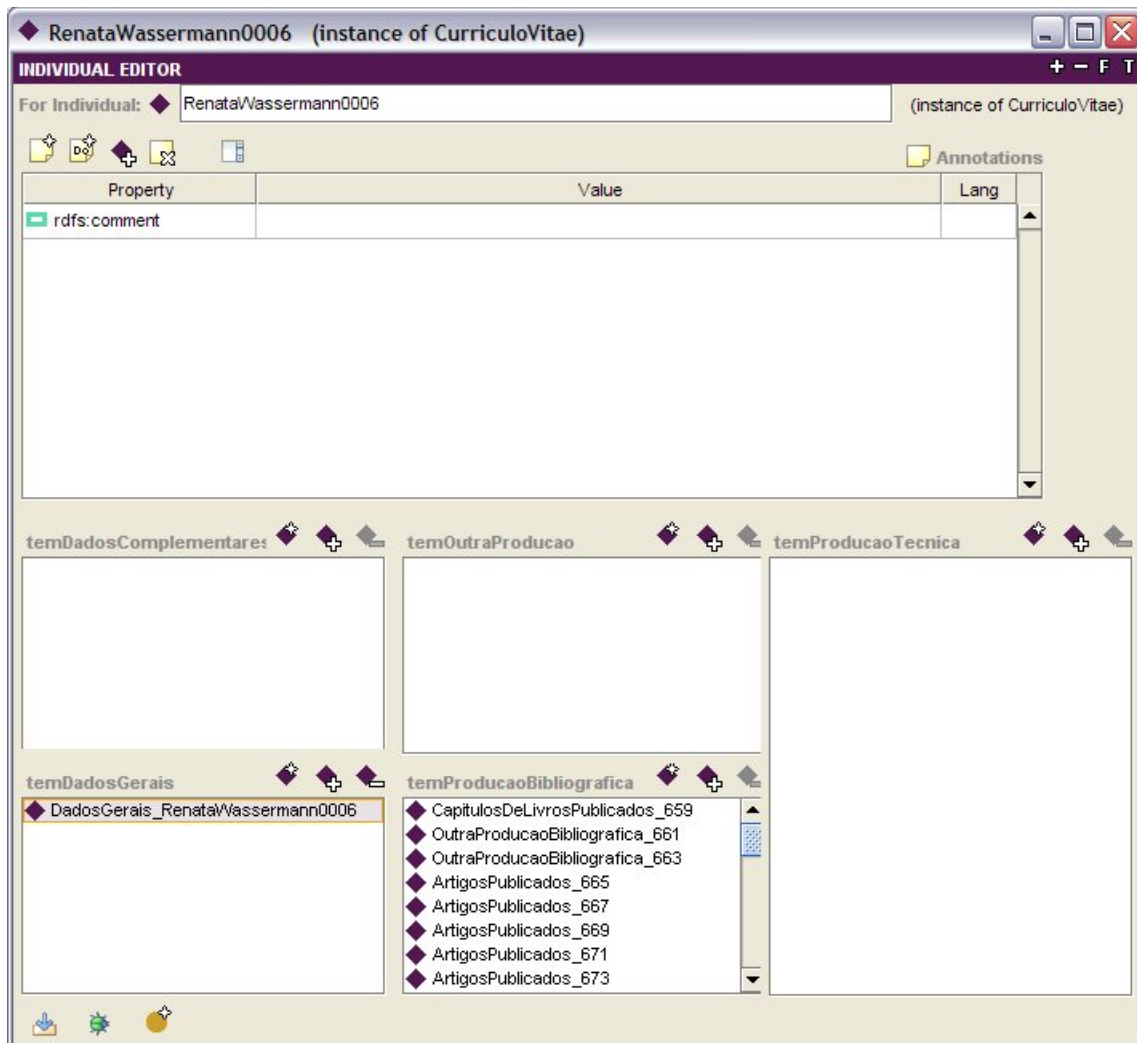


Figura 5.2: Instância de um *CurriculoVitae*.

A propriedade *temDadosGerais* da instância de *CurriculoVitae* serve para relacionar uma instância de *CurriculoVitae* com uma instância de *DadosGerais*. Podemos notar na figura 5.2 que a propriedade *temDadosGerais* possui um objeto e ele assegura a ligação entre as classes.

Se essa propriedade não estivesse relacionada com nenhum valor, então essa instância não estaria

relacionada com nenhuma instância da classe *DadosGerais*. Esse tipo de problema requer sempre atenção quando se criam instâncias dentro de uma base de conhecimento e, no caso deste trabalho, foi um ponto tratado com muita importância para que informações inseridas na base não ficassem “perdidas”, ou seja, sem conexão com outras informações, o que impediria que algumas informações fossem corretamente acessadas através de consultas na base de conhecimento.

Apresentaremos a seguir o código Turtle da instância da classe *CurriculoVitae* e o código Turtle da instância da classe *DadosGerais*. Podemos notar como a propriedade *temDadosGerais* faz a ligação de uma instância com a outra através do código:

```
default:temDadosGerais
```

```
default:DadosGerais_RenataWassermann0006 ;
```

Exemplo de Código de uma instância de *CurriculoVitae*

```
default:RenataWassermann0006
  rdf:type default:CurriculoVitae ;
  default:temDadosGerais
    default:DadosGerais_RenataWassermann0006 ;
.
.
.
```

Exemplo de Código de uma instância de *DadosGerais*

```
default:DadosGerais_RenataWassermann0006
  rdf:type default:DadosGerais ;
  default:nomeCompleto
    "Renata Wassermann"^^xsd:string ;
  default:nomeEmCitacoesBibliograficas
    "WASSERMANN, R."^^xsd:string ;
  default:sexo "Feminino"^^xsd:string ;
.
```

Vamos agora apresentar o funcionamento do script através de um exemplo, mostrando passo-a-passo as ações executadas, desde a leitura dos arquivos de origem até a inserção das informações na ontologia, para um pequeno trecho de currículo.

Primeiro, apresentamos uma imagem (Figura 5.3) representando a visualização de uma parte de um Currículo Lattes a partir de um navegador.



Figura 5.3: Idiomas de um currículo da Plataforma Lattes, visualizado em um navegador

Agora veremos a estrutura de dados gerada pelo script relativa a esta parte do currículo apresentada:

PARTE: 'Idiomas'

```
{{Compreende}}->[[Inglês (Bem), Alemão (Bem), Francês (Bem), Espanhol (Bem), Italiano (Bem), Holandês (Bem).]]
```

```
{{Fala}}->[[Inglês (Bem), Alemão (Razoavelmente), Francês (Razoavelmente), Espanhol (Bem), Italiano (Razoavelmente), Holandês (Bem).]]
```

```
{{Lê}}->[[Inglês (Bem), Alemão (Bem), Francês (Bem), Espanhol (Bem),
```

```
Italiano (Bem), Holandês (Bem).]]
```

```
{{Escreve}}->[[Inglês (Bem), Alemão (Razoavelmente), Francês (Pouco),  
Espanhol (Pouco), Italiano (Pouco), Holandês (Bem).]]
```

A partir dessa estrutura de dados e da modelagem pré-existente da ontologia, o script gera o código necessário para inserir essas informações dentro da base de conhecimento. O código gerado depende do tipo de informações que estamos tratando. Dentro do script, identificamos que tipo de informação está sendo tratada e de acordo com o tipo de informação, o script utiliza adequadamente padrões de texto em busca dos dados mais detalhados contidos dentro dos currículos.

O código final (formato Turtle) fica desta forma:

```
default:DadosGerais_RenataWassermann0006
  rdf:type default:DadosGerais ;
.
.
.
  default:temIdiomas default:Idiomas_RenataWassermann0006_alemao ,
default:Idiomas_RenataWassermann0006_holandes , default:Idiomas_RenataWassermann0006_italiano ,
default:Idiomas_RenataWassermann0006_ingles , default:Idiomas_RenataWassermann0006_espanhol ,
default:Idiomas_RenataWassermann0006_frances .
.
.
.
default:Idiomas_RenataWassermann0006_espanhol
  rdf:type default:Idiomas ;
  default:idioma default:espanhol ;
  default:proficienciaDeCompreensao
    "Bem"^^xsd:string ;
  default:proficienciaDeEscrita
    "Pouco"^^xsd:string ;
  default:proficienciaDeFala
    "Bem"^^xsd:string ;
  default:proficienciaDeLeitura
    "Bem"^^xsd:string .
.
.
```

```
.
default:Idiomas_RenataWassermann0006_ingles
  rdf:type default:Idiomas ;
  default:idioma default:ingles ;
  default:proficienciaDeCompreensao
    "Bem"^^xsd:string ;
  default:proficienciaDeEscrita
    "Bem"^^xsd:string ;
  default:proficienciaDeFala
    "Bem"^^xsd:string ;
  default:proficienciaDeLeitura
    "Bem"^^xsd:string .

.
.
.
default:Idiomas_RenataWassermann0006_holandes
  rdf:type default:Idiomas ;
  default:idioma default:holandes ;
  default:proficienciaDeCompreensao
    "Bem"^^xsd:string ;
  default:proficienciaDeEscrita
    "Bem"^^xsd:string ;
  default:proficienciaDeFala
    "Bem"^^xsd:string ;
  default:proficienciaDeLeitura
    "Bem"^^xsd:string .

.
.
.
default:Idiomas_RenataWassermann0006_italiano
  rdf:type default:Idiomas ;
  default:idioma default:italiano ;
  default:proficienciaDeCompreensao
    "Bem"^^xsd:string ;
  default:proficienciaDeEscrita
    "Pouco"^^xsd:string ;
  default:proficienciaDeFala
    "Razoavelmente"^^xsd:string ;
  default:proficienciaDeLeitura
    "Bem"^^xsd:string .
```

```

.
.
.
default:Idiomas_RenataWassermann0006_frances
  rdf:type default:Idiomas ;
  default:idioma default:frances ;
  default:proficienciaDeCompreensao
    "Bem"^^xsd:string ;
  default:proficienciaDeEscrita
    "Pouco"^^xsd:string ;
  default:proficienciaDeFala
    "Razoavelmente"^^xsd:string ;
  default:proficienciaDeLeitura
    "Bem"^^xsd:string .
.
.
.
default:Idiomas_RenataWassermann0006_alemao
  rdf:type default:Idiomas ;
  default:idioma default:alemao ;
  default:proficienciaDeCompreensao
    "Bem"^^xsd:string ;
  default:proficienciaDeEscrita
    "Razoavelmente"^^xsd:string ;
  default:proficienciaDeFala
    "Razoavelmente"^^xsd:string ;
  default:proficienciaDeLeitura
    "Bem"^^xsd:string .

```

Repetindo-se este processo descrito anteriormente, fazemos o processamento de todos os arquivos, gerando então a base de conhecimento. Contudo, até aqui estivemos trabalhando com os arquivos de origem sem verificar que algumas informações podem já terem sido inseridas na base de conhecimento. Isso naturalmente evidenciou o principal problema que existe nesse processo que são as referências duplicadas que tratam de um mesmo objeto. Este problema é o grande desafio de todo o processo e as soluções propostas serão detalhadas nas próximas seções.

Após todo o processo, o arquivo gerado no final do processo é uma base de conhecimento válida e consistente, em formato OWL, que pode ser aberta no Protégé para ser visualizada e manipulada.

5.4 Casamento de Instâncias

5.4.1 Introdução

O processo de leitura dos arquivos HTML e geração de instâncias a partir das informações apresenta alguns problemas relacionados com a base de conhecimento recém criada. Quando transformamos automaticamente as informações em instâncias, perdemos o completo controle de qualidade que teríamos ao popular a ontologia manualmente, o que permitiria que todas as informações inseridas fossem verificadas. Entretanto, mesmo manualmente é difícil ter um bom controle de qualidade quando o número de instâncias vai crescendo dentro da base de conhecimento e por essa razão a idéia de automatizar algumas verificações é bastante interessante.

O principal problema que trataremos agora é da duplicidade de instâncias que se referem a um mesmo objeto, ou seja, instâncias que representam um mesmo Artigo ou um mesmo Capítulo de Livro, por exemplo, dentro da base de conhecimento do Currículo Lattes.

A duplicidade de instâncias que referenciam um mesmo objeto é uma falha na modelagem que pode causar problemas nas buscas realizadas pois estaríamos tratando mais de uma vez um mesmo objeto.

Para evitar que essas duplicidades aconteçam dentro da ontologia gerada, vamos melhorar o script para que ele passe a utilizar alguns critérios de modo que seja possível identificar as duplicidades e evitá-las.

Vamos mostrar melhor o problema das duplicidades através de um exemplo.

Primeiro, mostraremos três pequenos textos retirados de arquivos HTML de currículos dos pesquisadores:

Currículo 1:

SILVA, F. S. C. ; WASSERMANN, R. ; MELO, A. C. V. ; BARROS, L. N. ;
Marcelo Finger . Intelligent mobile multi-robotic systems: some challenges and
possible solutions. In: International Conference on Informatics in Control,
Automation and Robotics (ICINCO), 2005, Barcelona. Proceedings of the Second
International Conference on Informatics in Control, Automation and Robotics,

2005. p. 479-485.

Currículo 2:

SILVA, F. S. C. ; Wasserman, R. ; MELO, A. C. V. ; BARROS, L. N. ; FINGER, M. . Intelligent Mobile Multi-robotic Systems: some Challenges and Possible Solutions. In: International Conference on Informatics in Control, Automation and Robotics, 2005, Barcelona. Proceedings of ICINCO - 2nd International Conference on Informatics in Control, Automation and Robotics, 2005.

Currículo 3:

CORREA DA SILVA, F. S. ; WASSERMAN, Renata ; MELO, Ana Cristina Vieira de ; BARROS, Leliane Nunes de ; FINGER, Marcelo . Intelligent Mobile Multi-robotic Systems: some Challenges and Possible Solutions. In: 2nd International Conference on Informatics in Control, Automation and Robotics, 2005, Barcelona. Proceedings of ICINCO - 2nd International Conference on Informatics in Control, Automation and Robotics, 2005. v. 0. p. 0-0.

Podemos notar que todos os trechos referem-se a uma mesma produção bibliográfica. Porém, sem usarmos nenhum método para retirar duplicidades, a base de conhecimento gerada a partir desses currículos geraria três instâncias diferentes, ou seja, uma para cada uma das citações.

Na figura 5.4, podemos ver uma consulta feita em SPARQL que retorna as instâncias para o exemplo em questão. A coluna Dados possui o nome das instâncias e através dela podemos notar que temos três instâncias diferentes para as três linhas retornadas.

The screenshot shows the Protégé SPARQL query editor and results window. The query is as follows:

```

SELECT ?Nome ?Dados ?titulo ?tipo
WHERE {
  ?prod :temDadosBasicos ?Dados .
  ?prod rdf:type ?tipo .
  ?pess :temProducaoBibliografica ?prod .
  ?pess :temDadosGerais ?dg .
  ?dg :nomeCompleto ?Nome .
  ?Dados :titulo ?titulo .
  FILTER regex(?titulo, "multi-robotic", "i") .
}
ORDER BY ?nome

```

The results table is:

Nome	Dados	titulo	tipo
Flavio Soares Correa da Silva	DadosBasicos_72	Intelligent Mobile Multi-robotic Systems: some Challenges and Possible Solution...	TrabalhosEmEventos
Leliane Nunes de Barros	DadosBasicos_390	Intelligent Mobile Multi-robotic Systems: some Challenges and Possible Solution...	TrabalhosEmEventos
Renata Wassermann	DadosBasicos_454	Intelligent mobile multi-robotic systems: some challenges and possible solution...	TrabalhosEmEventos

Figura 5.4: Tela do Protégé mostrando uma consulta SPARQL que retorna as produções bibliográficas cujo título contém o texto “multi-robotic”. A coluna *Dados* mostra as diferentes instâncias.

Queremos, através de uma eficiente busca de duplicidades, fazer com que um mesmo item seja representado por uma única instância. Exemplificando, podemos dizer que queremos que o resultado apresentado na figura 5.4 se transforme no resultado apresentado na figura 5.5

The screenshot shows the Protégé SPARQL query editor and results window. The query is identical to the one in Figure 5.4:

```

SELECT ?Nome ?Dados ?titulo ?tipo
WHERE {
  ?prod :temDadosBasicos ?Dados .
  ?prod rdf:type ?tipo .
  ?pess :temProducaoBibliografica ?prod .
  ?pess :temDadosGerais ?dg .
  ?dg :nomeCompleto ?Nome .
  ?Dados :titulo ?titulo .
  FILTER regex(?titulo, "multi-robotic", "i") .
}
ORDER BY ?nome

```

The results table is:

Nome	Dados	titulo	tipo
Flavio Soares Correa da Silva	DadosBasicos_72_IntelligentMobileMultiroboticS...	Intelligent Mobile Multi-robotic Systems: some Challe...	TrabalhosEmEventos
Leliane Nunes de Barros	DadosBasicos_72_IntelligentMobileMultiroboticS...	Intelligent Mobile Multi-robotic Systems: some Challe...	TrabalhosEmEventos
Renata Wassermann	DadosBasicos_72_IntelligentMobileMultiroboticS...	Intelligent Mobile Multi-robotic Systems: some Challe...	TrabalhosEmEventos

Figura 5.5: Tela do Protégé mostrando uma consulta SPARQL que retorna as produções bibliográficas cujo título contém o texto “multi-robotic”. A coluna *Dados* mostra que temos sempre a mesma instância.

5.4.2 Busca de duplicidades - Algoritmo

Damos o nome de co-referência para o problema que aparece quando temos dois nomes referenciando a mesma coisa [1]. Dentro do nosso contexto, estamos preocupados com instâncias dentro da mesma base de conhecimento que referenciem um mesmo objeto, como por exemplo uma mesma

produção bibliográfica.

O artigo de Alani et al. [1] explica um método para encontrar duplicidades em grandes bases de conhecimento. Os critérios utilizados no artigo são baseados principalmente na distância de Levenshtein entre textos similares. O objetivo do trabalho realizado no artigo é retirar duplicidades de ontologias já populadas, ao passo que no nosso trabalho queremos evitar que as duplicidades sejam inseridas na base de conhecimento, ou seja, utilizaremos o algoritmo de Levenshtein para encontrar similaridades entre *strings* e usaremos esses resultados para não inserir duplicidades, na medida que o script for detectando essas duplicidades.

5.4.3 Critérios de similaridade

Encontrar co-referências no processo de instanciação da ontologia não é simples. As referências para um mesmo objeto podem ter várias diferenças e o que procuramos buscar são critérios adequados que maximizem o número de duplicidades encontradas, sem que sejam encontradas falsas duplicidades.

É praticamente impossível conseguir critérios que consigam obter 100% de êxito nos seus resultados. Entretanto, vamos aqui listar os critérios que foram utilizados neste trabalho e comentar as vantagens e desvantagens que cada novo critério pode trazer.

- Similaridade de textos

Similaridade de textos resume-se a procurar por textos iguais em diferentes itens, ou seja, procurar publicações que tenham o mesmo título ou então autores que tenham o mesmo nome, por exemplo. Esse é o caso mais óbvio de duplicidade mas na prática é o critério que menos as encontra pois qualquer diferença de caracteres entre itens fará com que eles sejam considerados diferentes e portanto erros simples de digitação já não serão contemplados por esse critério. Além disso, a quantidade de comparações na busca pode crescer muito se a busca por similaridades for feita em todos os outros itens previamente inseridos (independente do tipo de informação que seja) sem que seja utilizado algum critério para que não sejam feitas comparações desnecessárias.

- Busca por similaridades apenas dentro de um mesmo grupo de itens

Segue o mesmo padrão da busca genérica por similaridades, mas apresenta a vantagem de se

procurar similaridades em um grupo menor de itens, o que diminui drasticamente o número de comparações das buscas.

A desvantagem de se limitar um grupo de itens é que não serão encontradas similaridades entre itens que estejam em grupos distintos, ou seja, se um mesmo item for inserido em classificações distintas (por exemplo, se um item for inserido por um autor como Artigo e por outro Autor como Trabalho em Evento), a limitação por grupos não permitirá que essa duplicidade seja encontrada.

- Busca por textos de tamanhos próximos

Uma das formas mais eficazes de diminuir a complexidade do número de comparações entre textos é somente comparar textos de comprimentos próximos, ou seja, textos cuja diferença de comprimentos seja menor do que um valor pré-estabelecido. Dessa forma, eliminamos comparações inúteis de textos com comprimentos completamente diferentes, que certamente não serão considerados iguais na comparação usando a distância de Levenshtein. A desvantagem deste critério é que eliminamos a possibilidade de encontrar duplicidades entre trabalhos que possuam subtítulos, ou seja, se um autor utilizar somente o título para identificar uma publicação e outro autor utilizar título e subtítulo, certamente essa duplicidade não será encontrada quando utilizamos o critério de tamanhos próximos de comprimentos de *strings*

- Busca por características similares

Além de buscar por similaridades de textos, pode-se também verificar outras características como por exemplo quantidade de autores de uma produção bibliográfica ou então comparar se o ano de publicação é o mesmo para dois itens candidatos a representarem o mesmo objeto.

- Combinação de critérios

A combinação de alguns dos critérios citados acima parece ser a melhor solução para a busca de duplicidades. Não existe uma combinação perfeita de critérios, uma vez que alguns critérios, quando adotados, beneficiaram alguns casos mas prejudicaram outros. O que buscamos é tentar maximizar a qualidade e quantidade de duplicidades encontradas, sem que sejam encontradas falsas duplicidades. Para mensurar a qualidade do sistema de acordo com estes objetivos,

utilizaremos nos testes os conceitos de Precisão e Cobertura, que serão explicados mais adiante. (ver seção 6.2)

5.5 Testes com os algoritmos e os critérios de similaridade

Com o início do funcionamento básico do script, ou seja, a partir do momento que ele conseguia ler os arquivos de origem e gerar a base de conhecimento, começamos a detectar as co-referências e a escolher os critérios a serem usados para encontrar e retirar as duplicidades.

O desenvolvimento dos algoritmos iniciou-se com o foco de retirar, principalmente, duplicidades com as Produções Bibliográficas. As duplicidades das Produções são comuns pois a grande maioria delas são escritas por mais de um autor e portanto é comum que sejam inseridas várias instâncias representando mesmos objetos.

No caso das Produções Bibliográficas, começamos o processo utilizando a busca por similaridades entre os títulos das produções. Além de buscar por títulos iguais, buscamos também por títulos que tivessem pequenas diferenças. Para isso, utilizamos o algoritmo de Levenshtein [16]. Esse algoritmo retorna o número de remoções, inserções e substituições de caracteres necessários para se chegar de uma *string* A para uma *string* B.

Após implementar o uso do algoritmo de Levenshtein dentro do script, foi necessário definir qual seria a distância limite (DIFMAX) para que considerássemos duas *strings* “iguais”, ou seja, dadas as *strings* A e B, vamos considerar iguais as *strings* quando a distância de Levenshtein entre A e B for menor que a diferença DIFMAX e considerar diferentes quando a distância for maior ou igual a DIFMAX.

Utilizando somente o critério acima, foram feitos os primeiros testes mas logo foi necessário buscar novas soluções porque o número de comparações que estava sendo feito entre textos estava crescendo rapidamente e portanto foi necessário encontrar maneiras de otimizar as comparações de textos.

A primeira forma de minimizar as comparações foi limitando as chamadas ao algoritmo de Levenshtein e para isso, usamos o critério da diferença de tamanhos entre *strings*. Somente eram comparadas *strings* cujos comprimentos tivessem uma diferença máxima próxima. O valor dessa diferença escolhido para ser usado neste trabalho, após testes com uma base de 7 currículos, foi de 5%, com um valor mínimo de 3.

Em seguida, adicionamos um novo critério: a comparação de itens somente dentro de seu próprio grupo de itens, ou seja, somente procuramos similaridades de produções classificadas da mesma forma. Para exemplificar, vamos supor que exista uma Produção Bibliográfica escrita por 2 autores. Um deles classificou essa Produção como um Artigo de Revista e o outro como Capítulo de Livro. Dentro deste contexto, o nosso novo critério utilizado não irá considerar essas Produções como sendo referentes ao mesmo trabalho. Inicialmente, esse critério pode parecer apenas uma limitação do sistema, evitando que Produções de grupos diferentes sejam identificadas como duplicidades. Entretanto, usamos esse critério com o intuito de evitar que falsas duplicidades sejam encontradas, ou seja, evitando que o sistema identifique erroneamente uma duplicidade entre itens de dois grupos diferentes. Com isso, estamos tentando efetivamente diminuir o número de falsas duplicidades, mesmo que isso faça o sistema deixar de encontrar duplicidades verdadeiras. Além disso, limitando as comparações dentro de um mesmo grupo, diminuimos mais ainda a quantidade de comparações feitas no algoritmo, melhorando seu desempenho.

Fazendo alguns testes simples, obtivemos bons resultados quando fixamos o valor de DIFMAX com 5% do tamanho do título das publicações. O valor mínimo para DIFMAX foi colocado como 3, ou seja, se os 5% do tamanho fossem menor que 3, o valor ficaria fixado em 3. Com esse valor de DIFMAX, os resultados obtidos para testes feitos com uma base de 7 currículos foram muito bons (Ver resultados no capítulo 6). Quase todas as duplicidades dos currículos foram detectadas corretamente.

O algoritmo 2 apresenta o algoritmo de detecção de duplicidades na inserção de uma nova Produção Bibliográfica realizando busca por todos os itens previamente inseridos dentro da mesma categoria de Produção Bibliográfica.

Algoritmo 2: Busca de Duplicidades em Produções - Busca em todos os itens

```

foreach ( item: cada uma das Produções Bibliográficas ) do
  dif_comprimento  $\leftarrow \lfloor (\text{comp\_titulo}(\text{item}) * 0.05) \rfloor$ ;
  if ( dif_comprimento < 3 ) then
    dif_comprimento  $\leftarrow$  3;
  end
  if ( ( min_comprimento  $\leftarrow$  comp_titulo(item) - dif_comprimento ) < 1 ) then
    min_comprimento  $\leftarrow$  1;
  end
  max_comprimento  $\leftarrow$  comp_titulo(item) + dif_comprimento;
  distancia_levenshtein_limite  $\leftarrow \lfloor (\text{comp\_titulo}(\text{item}) * 0.05) \rfloor$ ;
  if ( distancia_levenshtein_limite < 3 ) then
    distancia_levenshtein_limite  $\leftarrow$  3;
  end
  producao_repetida  $\leftarrow$  0;
  for ( comprimento_atual  $\leftarrow$  min_comprimento; comprimento_atual <= max_comprimento;
  comprimento_atual  $\leftarrow$  (comprimento_atual + 1)) do
    foreach ( producao_existente: cada uma das Produções já Inseridas na Base de conhecimento
    com comprimento igual a comprimento_atual ) do
      if ( dist_levenshtein( titulo(item), titulo(producao_existente)) <
      distancia_levenshtein_limite ) then
        producao_repetida  $\leftarrow$  1;
        Sair do for ;          /* Duplicidade encontrada. Parar comparações */
      end
    end
  end
  if ( producao_repetida = 0 ) then
    inserir_item_na_base_de_conhecimento(item) ;
  end
end

```

Mesmo atingindo bons resultados, o número de chamadas do algoritmo de Levenshtein (comparações entre *strings*) foi alto. Antes das otimizações, o número de comparações para um teste com 4 currículos, estava acima de 10 mil. Depois das otimizações, o número de comparações caiu para menos de 2 mil. Entretanto, os resultados foram altos para um número pequeno de currículos utilizados e esse foi um dos principais motivos para que uma nova abordagem fosse desenvolvida, utilizando critérios diferentes e de forma diferente. Vejamos, a seguir, como funciona a nova abordagem.

A idéia da nova abordagem é diminuir o número de comparações sem que tenhamos queda na qualidade das remoções de duplicidades. Para isso, vamos mudar a forma de realizar as comparações entre Produções. Ao invés de comparar cada nova Produção com todas as outras Produções inseridas na base de conhecimento dentro de um mesmo grupo (ou seja, ao invés de comparar um novo Artigo com todos os outros Artigos anteriores), vamos buscar os autores de cada nova Produção e procurar somente nas publicações dos autores para verificar se a Produção já foi inserida. Pelo método anterior, após já termos inserido um bom número de Produções, uma nova Produção iria ser comparada com todas as anteriores que tivessem comprimento do título próximo ao comprimento do título da Produção em questão. No novo método, somente faremos essa busca dentro das Produções dos autores envolvidos. Isso faz com que o número de comparações diminua drasticamente, mas cria um novo problema: fazer comparações de nomes de autores pois assim como títulos de Produções podem ser escritos de formas diferentes, nomes de autores também aparecem escritos de maneiras diferentes e portanto iremos adotar padrões para tentar identificar similaridades nos nomes de autores e tentar evitar duplicidades por culpa de nomes mal escritos.

Algoritmo 3: Busca de Duplicidades em Produções - Busca em itens de autores citados

```

foreach ( item: cada uma das Produções Bibliográficas ) do
  dif_comprimento ← [( comp_titulo(item) *0.05)];
  if ( dif_comprimento < 3) then
    dif_comprimento ← 3;
  end
  if (( min_comprimento ← comp_titulo(item) – dif_comprimento ) < 1) then
    min_comprimento ← 1;
  end
  max_comprimento ← comp_titulo(item) + dif_comprimento;
  distancia_levenshtein_limite ← [( comp_titulo(item) *0.05)];
  if ( distancia_levenshtein_limite < 3) then
    distancia_levenshtein_limite ← 3;
  end
  producao_repetida ← 0;
  for ( comprimento_atual ← min_comprimento; comprimento_atual <= max_comprimento;
  comprimento_atual ← ( comprimento_atual +1)) do
    foreach ( autor_atual: cada um dos autores do item atual. ) do
      foreach ( producao_existente: cada uma das Produções do autor_atual, de comprimento
      igual a comprimento_atual, já inseridas na Base de conhecimento ) do
        if ( dist_levenshtein( titulo(item), titulo(producao_existente)) <
        distancia_levenshtein_limite ) then
          producao_repetida ← 1;
          Sair do for ;          /* Duplicidade encontrada. Parar comparações */
        end
      end
    end
  end
  if ( producao_repetida = 0) then
    inserir_item_na_base_de_conhecimento(item) ;
  end
end

```

Os nomes de autores de Produções, em geral, vêm na forma de citação bibliográfica, ou seja, com o último nome escrito por inteiro e os primeiros nomes representados apenas pelas iniciais. Seguindo esse padrão, faremos então uma transformação em todos os nomes de autores que utilizaremos dentro da estrutura do script, para que utilize as iniciais dos primeiros nomes e somente o último nome

completo.

Logo nos primeiros testes, notamos que essa padronização funciona mas tem problemas com nomes comuns. Seguindo esses critérios, se temos os autores João Silva e José Silva, ambos tem o mesmo nome em citações bibliográficas que é Silva, J. Se considerarmos todas as produções de Silva, J sendo de um mesmo autor, estaremos cometendo um erro e portanto é necessário ter um cuidado especial com esses casos. Esse é o principal problema desse novo método de buscas de duplicidades nas Produções Bibliográficas.

O novo método de busca de duplicidades, como dito anteriormente, diminuiu as comparações entre títulos de Produções, mas trouxe um grande número de comparações entre nomes de autores. Se somarmos o número de comparações entre títulos e o número de comparações entre nomes de autores, a quantidade total de comparações aumentou, mas o tempo de execução em alguns casos até diminuiu porque as chamadas do algoritmo agora são feitas com textos de entrada menores. Entretanto, a imprecisão dos resultados inicialmente aumentou por conta do maior número de falhas ao encontrar nomes de autores duplicados ou então ao encontrar falsas duplicidades. Foi necessário recorrer a interação humana para que as duplicidades encontradas fossem validadas corretamente para algumas situações.

Para tentar diminuir as falsas duplicidades, mais uma vez tentamos usar outras propriedades das Produções Bibliográficas para que tenhamos mais informações de comparação. Incluir novas propriedades na comparação traz maior segurança para evitar falsas duplicidades, mas também diminui a chance de que verdadeiras duplicidades sejam encontradas, uma vez que um autor pode ter incluído informações que outro autor não incluiu e, portanto, comparar mais propriedades pode fazer com que a busca falhe mais.

No fim, optamos por não utilizar propriedades adicionais visto que em pequenos testes feitos, essas propriedades não ajudaram a melhorar os resultados das buscas por duplicidades. Somente vamos utilizar similaridades entre os nomes dos autores das Produções.

Capítulo 6

Análise dos Resultados

No capítulo anterior, vimos como são processadas as informações que são inseridas na ontologia e discutimos alguns dos problemas encontrados, destacando principalmente o problema das co-referências, propondo soluções distintas para resolvê-lo.

Mostraremos, neste capítulo, alguns testes comparativos com as abordagens sugeridas, apresentando os resultados obtidos e analisando seu desempenho.

6.1 Introdução

Neste trabalho, foram desenvolvidos métodos para permitir que um sistema automaticamente consiga ler informações, identificá-las corretamente e inserí-las dentro de uma ontologia. No decorrer deste processo, foi identificado um problema de grande relevância: as co-referências. Foram propostas duas soluções com abordagens distintas visando solucionar o problema das co-referências, especificamente das Produções Bibliográficas.

O principal objetivo dos testes realizados é identificar o número de co-referências (duplicidades) encontradas dentro do universo de duplicidades existente. Também estamos interessados em saber se falsas duplicidades foram encontradas, ou seja, se os sistemas identificaram duplicidades que na verdade não representavam um mesmo objeto.

Diante desse cenário, para verificar e mensurar a qualidade dos resultados obtidos, vamos utilizar alguns conceitos de medidas amplamente difundidos em sistemas que trabalham com Recuperação de Informações (RI) [28] [14]. Estes conceitos serão apresentados na seção 6.2.

6.2 Medidas em Sistemas de Recuperação de Informações

Para avaliarmos a qualidade dos resultados obtidos nos nossos testes, utilizaremos duas medidas padrão em sistemas de RI: Precisão e Cobertura. Estas duas medidas são baseadas em relações entre os conjuntos de objetos corretos e incorretos e também conjuntos de objetos encontrados e não encontrados [28]. A figura 6.1 apresenta as relações entre esses conjuntos.

	Corretos	Incorretos	
Encontrados	$A \cap B$	$\bar{A} \cap B$	B
Não Encontrados	$A \cap \bar{B}$	$\bar{A} \cap \bar{B}$	\bar{B}
	A	\bar{A}	

Figura 6.1: Relações entre os conjuntos de objetos corretos/incorretos e encontrados/não encontrados.

Baseado nesses conjuntos, apresentamos as definições:

- **Precisão (P):** Obtém-se a Precisão de um sistema calculando a razão entre o número de objetos corretamente encontrados pelo número total de objetos encontrados.

$$P = \frac{|A \cap B|}{|B|}$$

No contexto que estamos trabalhando, podemos redefinir a Precisão como sendo o número de duplicidades corretamente encontradas dividido pelo número de duplicidades encontradas.

- **Cobertura (C):** Definimos a Cobertura como sendo a razão entre o número de objetos corretamente encontrados pelo número total de objetos corretos existentes.

$$C = \frac{|A \cap B|}{|A|}$$

Dentro do nosso contexto, dizemos que o conceito de Cobertura representa o número de duplicidades corretamente encontradas dividido pelo número total de duplicidades corretas existentes.

- **Medida-F (F):** As medidas de Precisão e Cobertura vistas separadamente podem dar uma falsa impressão nos resultados. Um sistema pode apresentar 100% de Precisão se ele retornar apenas 1 objeto e esse objeto for correto. No entanto, esse mesmo sistema pode ter uma Cobertura muito pequena pois o número de objetos corretos pode ser bem maior.

Da mesma forma, um sistema pode apresentar 100% de Cobertura se retornar todos os objetos do domínio. Entretanto, a Precisão pode ser baixa se grande parte desses objetos não forem corretos.

Para combinar os números de Precisão e Cobertura, foi criada a Medida-F [28] [17]:

$$F = \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{C}}$$

, onde $0 < \alpha < 1$.

O valor de α indica a importância relativa da precisão para o usuário. O valor de $\alpha = 0,5$, significa que a Precisão e a Cobertura possuem a mesma importância na Medida-F. Quando aumentamos α , aumentamos a importância da Precisão e quando diminuimos α , aumentamos a importância da Cobertura.

As medidas de Precisão e Cobertura nem sempre podem ser obtidas visto que, dependendo do domínio que estiver sendo feito o teste, o número total de duplicidades existente pode ser difícil de se conseguir (o que atrapalharia a medida de Cobertura) ou até mesmo o número de falsas duplicidades também pode ser difícil de se descobrir (atrapalhando a medida de Precisão).

6.3 Testes do Sistema

Para realizar os testes do sistema, tanto em relação a qualidade como em relação a quantidade, montamos alguns cenários com o intuito de verificar o desempenho do script desenvolvido em diferentes situações. Variamos a quantidade de currículos na entrada para verificar como o script comporta-se com o aumento de informações na entrada. Vamos analisar a Precisão, a Cobertura e também o desempenho do sistema.

Os currículos utilizados como entrada do teste foram currículos de professores do departamento de Ciência da Computação do IME-USP. Foram utilizados currículos que possuem produções bibliográficas em comum para que exista um bom número de produções duplicadas presentes nos arquivos de entrada e que possamos efetivamente verificar o comportamento do sistema diante das duplicidades.

Chamaremos de *BuscaEmProduções* o script que usa o algoritmo que utiliza o método de busca de duplicidades diretamente nas produções bibliográficas, ou seja, para cada nova Produção, o algoritmo procura por similares em todas as produções anteriores que estejam dentro da mesma categoria (artigos somente são comparados com artigos, livros somente são comparados com livros, etc), sem comparar nomes de autores, apenas utilizando os critérios de diferença máxima de comprimentos e diferença de texto. Chamaremos de *BuscaEmAutores* o script que utiliza o método de busca de duplicidades em itens de autores presentes na citação da Produção. Neste segundo método, para cada nova Produção, o algoritmo verifica quem são os autores e busca por duplicidades em produções somente nos itens dos autores. Estes dois métodos foram explicados com mais detalhes na seção 5.5.

Os testes foram feitos em ambiente Windows, sendo que no momento da execução do script, todos os outros processos que pudessem atrapalhar as medições de tempo de execução foram parados.

A medida de Cobertura exige que tenhamos conhecimento do número total de duplicidades existente entre Produções dos arquivos de entrada. Esse número foi obtido manualmente, ou seja, verificando item por item das Produções de cada currículo. Como quantidade total de duplicidades que deveriam ser encontradas, vamos considerar somente as Produções duplicadas entre os autores que possuem o currículo na entrada de dados. Se estamos utilizando como entrada os currículos dos pesquisadores A, B e C, somente nos interessa, para esta análise, encontrar as Produções comuns entre A, B e C. Quaisquer informações de duplicidades para outros autores não serão levadas em conta.

O fato de ser necessário analisar manualmente as duplicidades para verificar a quantidade total existente a ser utilizada no cálculo da Cobertura acaba sendo um limitante no tamanho dos testes realizados, pois torna-se inviável analisar manualmente um número grande de currículos. Além de levar muito tempo para realizar a análise, é praticamente impossível que a contagem final seja correta. Por essa razão, os testes com quantidades maiores de currículos não contam com medidas de Cobertura e Precisão.

A tabela a seguir apresenta os resultados obtidos, variando a quantidade de currículos utilizados na entrada:

Algoritmo	Quantidade de currículos	Tempo de Execução (segundos)	Número de comparações ^a	Precisão	Cobertura	Medida-F
<i>BuscaEmProduções</i>	2	2	223	1	0,78	0,88
<i>BuscaEmProduções</i>	3	6	530	1	0,8	0,89
<i>BuscaEmProduções</i>	4	18	1458	1	0,74	0,85
<i>BuscaEmProduções</i>	7	40	3185	1	0,83	0,91
<i>BuscaEmProduções</i> ^b	14	149	10692	-	-	-
<i>BuscaEmAutores</i>	2	4	3426	1	0,78	0,88
<i>BuscaEmAutores</i>	3	7	8701	1	0,8	0,89
<i>BuscaEmAutores</i>	4	18	23762	1	0,74	0,85
<i>BuscaEmAutores</i>	7	34	52576	1	0,83	0,91
<i>BuscaEmAutores</i> ^b	14	128	236201	-	-	-

^aNúmero de chamadas do algoritmo de Levenshtein comparando duas *strings*.

^bAs medidas de Precisão, Cobertura e Medida-F não foram calculadas para essa quantidade de currículos por conta do elevado número de informações a serem verificadas para encontrar o conjunto total de duplicidades.

Tabela 6.1: Resultados dos testes de execução do script usando os dois diferentes algoritmos.

Podemos notar nos testes que as medidas de Precisão e Cobertura são iguais para os dois sistemas. Isso ocorre porque os critérios de similaridade utilizados são os mesmos. A diferença de abordagem afeta apenas o desempenho e o número de comparações. Para que pudéssemos ter uma diferença clara entre os dois sistemas nas medidas de Cobertura e Precisão, seria necessário processar currículos de pesquisadores com nomes semelhantes, como João Silva e José Silva, por exemplo. Um caso deste tipo iria fazer o *BuscaEmAutores* falhar em muitas duplicidades e daria vantagem ao *BuscaEmProduções*. Entretanto, se imaginarmos que o número de nomes semelhantes entre pesquisadores que são co-autores de produções é pequeno, então este problema existente no *BuscaEmAutores* passa a ter menor importância.

Analisando o número de comparações, o *BuscaEmAutores* possui um número muito maior de chamadas do algoritmo de Levenshtein, mas o tempo de execução é semelhante ao *BuscaEmProduções*. Isso ocorre porque no *BuscaEmAutores* existem muitas chamadas do algoritmo com entradas de tamanho pequeno, o que torna a execução do algoritmo de Levenshtein bem mais rápida e justifica os resultados.

Levando em conta que os resultados obtidos para ambos os métodos foram bastante semelhantes, não seria correto afirmar que algum deles é melhor que o outro sem levar em conta outros aspectos.

tos. Para realizar essa análise, é necessário verificar também as características de cada método. Se considerarmos importante, por exemplo, que o script associe automaticamente produções de outros autores que por alguma razão não incluíram determinada Produção no seu currículo, então a abordagem do *BuscaEmAutores* passa a ser melhor. Entretanto, essa funcionalidade do *BuscaEmAutores* também pode gerar citações duplicadas quando o sistema procura por duplicidade e não a encontra, mesmo ela já existindo. Neste caso, o sistema irá considerar que a Produção em questão não existe no currículo do outro autor e irá inserir a Produção para ele, o que fará com que esse autor esteja relacionado duas vezes com o que deveria ser uma mesma Produção. Já o *BuscaEmProduções* não associa produções para outros autores, ou seja, ele apenas limita-se a buscar pelas duplicidades, sem realizar ações de inserção da Produção em outros currículos. Isso acaba sendo uma limitação do sistema mas evita que falsas duplicidades sejam inseridas. Se desejamos um sistema com poucas falsas duplicidades, então a abordagem do *BuscaEmProduções* é melhor.

Capítulo 7

Consultas na Ontologia

Até agora, descrevemos todo o processo de extração de informações dos arquivos HTML dos currículos e inserção dessas informações na ontologia.

Vamos apresentar neste capítulo algumas consultas que podem ser realizadas na ontologia e analisar seus resultados.

7.1 Introdução

Desde o princípio do desenvolvimento deste trabalho, um dos principais objetivos que buscamos atingir foi o de realizar consultas e extrair relatórios que pudessem de alguma forma auxiliar os pesquisadores a verificar se as informações inseridas por eles estava correta ou se existiam dados errados, faltando ou até mesmo duplicados.

Através da linguagem SPARQL e o seu esquema de consultas baseado em triplas, uma ontologia pode ser utilizada como um banco de dados inteligente, onde podemos extrair relatórios sobre todas as informações inseridas na ontologia.

Primeiramente, vamos mostrar através de consultas simples como podemos extrair as informações que estavam contidas nos arquivos HTML e que foram inseridas na ontologia.

Em seguida, faremos algumas consultas consolidadas, ou seja, utilizando simultaneamente informações de vários currículos.

7.2 Consultas Simples

Nestas primeiras consultas, nosso objetivo será de mostrar que as informações dos currículos estão armazenadas dentro da ontologia. Para isso, faremos algumas consultas simples em SPARQL que vão retornar as informações que desejamos.

Vamos tomar como referência as informações do currículo da figura [7.1](#).

Dados pessoais	
Nome	Renata Wassermann
Nome em citações bibliográficas	WASSERMANN, R.
Sexo	Feminino
Endereço profissional	Universidade de São Paulo, Instituto de Matemática e Estatística. Rua do Matao 1010 Cidade Universitaria 05508-090 - Sao Paulo, SP - Brasil Telefone: (011) 30919687 Fax: (011) 30916134 URL da Homepage: www.ime.usp.br/~renata
Endereço eletrônico	renata@ime.usp.br

Formação acadêmica/Titulação	
2005	Livre-docência. Universidade de São Paulo, USP, Brasil. <i>Título: Lógica e Representação do Conhecimento, Ano de obtenção: 2005.</i>
2000 - 2001	Pós-Doutorado. Universidade de São Paulo, USP, Brasil. <i>Bolsista do(a): Fundação de Amparo à Pesquisa do Estado de São Paulo, FAPESP, Brasil. Grande área: Ciências Exatas e da Terra / Área: Ciência da Computação / Subárea: Teoria da Computação / Especialidade: Inteligencia Artificial.</i>
1995 - 1999	Doutorado em Ciência da Computação. University of Amsterdam, AMSTERDAM, Holanda. <i>Título: Resource Bounded Belief Revision, Ano de Obtenção: 2000. Orientador: Hans Rott . Bolsista do(a): Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, CAPES, Brasil.</i>
1993 - 1995	Mestrado em Matemática Aplicada. Universidade de São Paulo, USP, Brasil. <i>Título: A Lógica das Estruturas de Features e suas Aplicações, Ano de Obtenção: 1995. Orientador: Flávio Soares Corrêa da Silva. Bolsista do(a): Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, CAPES, Brasil.</i>
1988 - 1991	Graduação em Bacharelado em Ciencia da Computacao. Universidade de São Paulo, USP, Brasil.

Atuação profissional	
Universidade de São Paulo, USP, Brasil.	
Vínculo institucional	
2006 - Atual	Vínculo: Servidor Público, Enquadramento Funcional: Professor associado, Carga horária: 40, Regime: Dedicção exclusiva.
Vínculo institucional	
2001 - 2006	Vínculo: Servidor Público, Enquadramento Funcional: Professor doutor, Carga horária: 40, Regime: Dedicção

Figura 7.1: Tela de um Currículo Lattes visto na Web.

Para cada uma das consultas que serão mostradas a seguir, mostraremos em destaque o trecho do currículo original HTML, de onde as informações retornadas foram retiradas.

A primeira consulta que vamos realizar será para buscar os dados básicos do pesquisador. Segue primeiro a figura 7.2 com o arquivo original HTML e a figura 7.3 com a consulta realizada e os resultados obtidos.

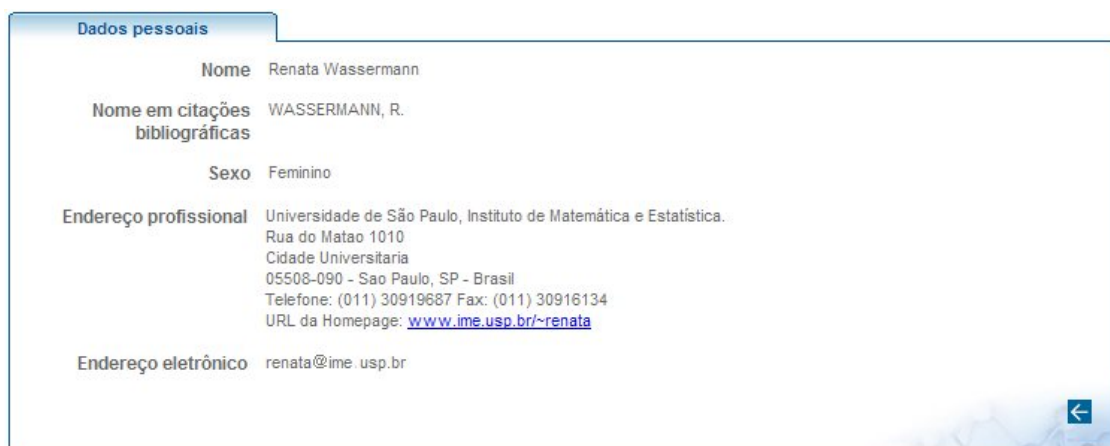


Figura 7.2: Dados Pessoais do pesquisador - original HTML.

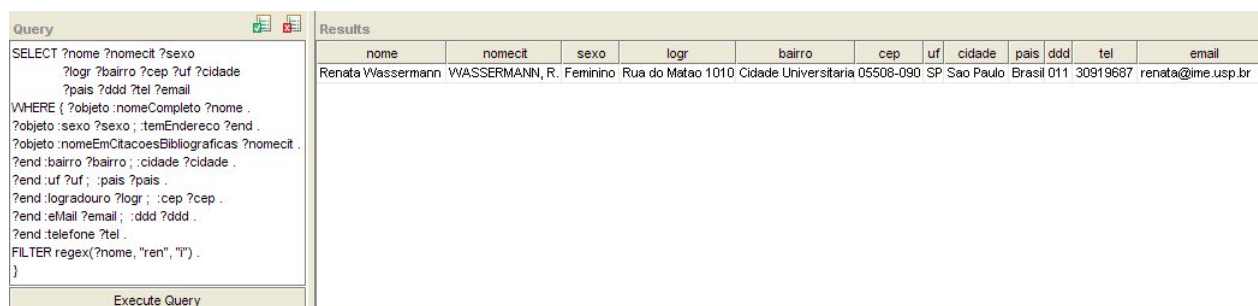


Figura 7.3: Consulta SPARQL buscando Dados Pessoais do pesquisador.

Agora vamos fazer outra consulta que buscará as informações de Formação Acadêmica do pesquisador.

Formação acadêmica/Titulação

2005 Livre-docência.
Universidade de São Paulo, USP, Brasil.
Título: Lógica e Representação do Conhecimento, *Ano de obtenção:* 2005.

2000 - 2001 Pós-Doutorado.
Universidade de São Paulo, USP, Brasil.
Bolsista do(a): Fundação de Amparo à Pesquisa do Estado de São Paulo, FAPESP, Brasil.
Grande área: Ciências Exatas e da Terra / *Área:* Ciência da Computação / *Subárea:* Teoria da Computação / *Especialidade:* Inteligência Artificial.

1995 - 1999 Doutorado em Ciência da Computação.
University of Amsterdam, AMSTERDAM, Holanda.
Título: Resource Bounded Belief Revision, *Ano de Obtenção:* 2000.
Orientador: Hans Rott .
Bolsista do(a): Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, CAPES, Brasil.

1993 - 1995 Mestrado em Matemática Aplicada.
Universidade de São Paulo, USP, Brasil.
Título: A Lógica das Estruturas de Features e suas Aplicações, *Ano de Obtenção:* 1995.
Orientador: Flávio Soares Corrêa da Silva.
Bolsista do(a): Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, CAPES, Brasil.

1988 - 1991 Graduação em Bacharelado em Ciencia da Computacao. Universidade de São Paulo, USP, Brasil.

Figura 7.4: Formação Acadêmica do pesquisador - original HTML.

Query	Results																														
<pre> SELECT ?nome ?tipo ?nomeinst ?anoini ?anoconc WHERE { ?objeto :temFormacaoAcademicaTitulacao ?form . ?form a ?tipo . ?form :realizadoNainstituicao ?inst . ?inst :nomeInstituicaoEmpresa ?nomeinst . FILTER regex(?nome, "ren", "i") . FILTER ((?tipo = :Graduacao) (?tipo = :Doutorado) (?tipo = :Mestrado) (?tipo = :PosDoutorado) (?tipo = :LivreDocencia)) . OPTIONAL { ?form :anoDeInicio ?anoini } OPTIONAL { ?form :anoDeConclusao ?anoconc } } </pre>	<table border="1"> <thead> <tr> <th>nome</th> <th>tipo</th> <th>nomeinst</th> <th>anoini</th> <th>anoconc</th> </tr> </thead> <tbody> <tr> <td>Renata Wassermann</td> <td>Graduacao</td> <td>Universidade de São Paulo, USP, Brasil.</td> <td>1988</td> <td>1991</td> </tr> <tr> <td>Renata Wassermann</td> <td>Doutorado</td> <td>University of Amsterdam, AMSTERDAM, Holanda.</td> <td>1995</td> <td>1999</td> </tr> <tr> <td>Renata Wassermann</td> <td>PosDoutorado</td> <td>Universidade de São Paulo, USP, Brasil.</td> <td>2000</td> <td>2001</td> </tr> <tr> <td>Renata Wassermann</td> <td>LivreDocencia</td> <td>Universidade de São Paulo, USP, Brasil.</td> <td>2005</td> <td></td> </tr> <tr> <td>Renata Wassermann</td> <td>Mestrado</td> <td>Universidade de São Paulo, USP, Brasil.</td> <td>1993</td> <td>1995</td> </tr> </tbody> </table>	nome	tipo	nomeinst	anoini	anoconc	Renata Wassermann	Graduacao	Universidade de São Paulo, USP, Brasil.	1988	1991	Renata Wassermann	Doutorado	University of Amsterdam, AMSTERDAM, Holanda.	1995	1999	Renata Wassermann	PosDoutorado	Universidade de São Paulo, USP, Brasil.	2000	2001	Renata Wassermann	LivreDocencia	Universidade de São Paulo, USP, Brasil.	2005		Renata Wassermann	Mestrado	Universidade de São Paulo, USP, Brasil.	1993	1995
nome	tipo	nomeinst	anoini	anoconc																											
Renata Wassermann	Graduacao	Universidade de São Paulo, USP, Brasil.	1988	1991																											
Renata Wassermann	Doutorado	University of Amsterdam, AMSTERDAM, Holanda.	1995	1999																											
Renata Wassermann	PosDoutorado	Universidade de São Paulo, USP, Brasil.	2000	2001																											
Renata Wassermann	LivreDocencia	Universidade de São Paulo, USP, Brasil.	2005																												
Renata Wassermann	Mestrado	Universidade de São Paulo, USP, Brasil.	1993	1995																											

Figura 7.5: Consulta SPARQL buscando Formação Acadêmica do pesquisador.

Em seguida, veremos outro exemplo que retorna as proficiências do pesquisador (de fala, de compreensão, de leitura e de escrita) em idiomas:

Idiomas	
Compreende	Inglês (Bem), Alemão (Bem), Francês (Bem), Espanhol (Bem), Italiano (Bem), Holandês (Bem).
Fala	Inglês (Bem), Alemão (Razoavelmente), Francês (Razoavelmente), Espanhol (Bem), Italiano (Razoavelmente), Holandês (Bem).
Lê	Inglês (Bem), Alemão (Bem), Francês (Bem), Espanhol (Bem), Italiano (Bem), Holandês (Bem).
Escreve	Inglês (Bem), Alemão (Razoavelmente), Francês (Pouco), Espanhol (Pouco), Italiano (Pouco), Holandês (Bem).

Figura 7.6: Proficiência em Idiomas do pesquisador - original HTML.

Query	Results																																										
<pre> SELECT ?nome ?nomeidioma ?profCompreensao ?profFala ?profLeitura ?profEscrita WHERE { ?objeto :nomeCompleto ?nome . ?objeto :temIdiomas ?profidioma . ?profidioma :idioma ?idioma . ?profidioma :proficienciaDeLeitura ?profLeitura ; :proficienciaDeEscrita ?profEscrita ; :proficienciaDeCompreensao ?profCompreensao ; :proficienciaDeFala ?profFala . ?idioma :nomeDoIdioma ?nomeidioma . FILTER regex(?nome, "Renata", "i") . } </pre>	<table border="1"> <thead> <tr> <th>nome</th> <th>nomeidioma</th> <th>profCompreensao</th> <th>profFala</th> <th>profLeitura</th> <th>profEscrita</th> </tr> </thead> <tbody> <tr> <td>Renata Wassermann</td> <td>Alemão</td> <td>Bem</td> <td>Razoavelmente</td> <td>Bem</td> <td>Razoavelmente</td> </tr> <tr> <td>Renata Wassermann</td> <td>Holandês</td> <td>Bem</td> <td>Bem</td> <td>Bem</td> <td>Bem</td> </tr> <tr> <td>Renata Wassermann</td> <td>Italiano</td> <td>Bem</td> <td>Razoavelmente</td> <td>Bem</td> <td>Pouco</td> </tr> <tr> <td>Renata Wassermann</td> <td>Espanhol</td> <td>Bem</td> <td>Bem</td> <td>Bem</td> <td>Pouco</td> </tr> <tr> <td>Renata Wassermann</td> <td>Francês</td> <td>Bem</td> <td>Razoavelmente</td> <td>Bem</td> <td>Pouco</td> </tr> <tr> <td>Renata Wassermann</td> <td>Inglês</td> <td>Bem</td> <td>Bem</td> <td>Bem</td> <td>Bem</td> </tr> </tbody> </table>	nome	nomeidioma	profCompreensao	profFala	profLeitura	profEscrita	Renata Wassermann	Alemão	Bem	Razoavelmente	Bem	Razoavelmente	Renata Wassermann	Holandês	Bem	Bem	Bem	Bem	Renata Wassermann	Italiano	Bem	Razoavelmente	Bem	Pouco	Renata Wassermann	Espanhol	Bem	Bem	Bem	Pouco	Renata Wassermann	Francês	Bem	Razoavelmente	Bem	Pouco	Renata Wassermann	Inglês	Bem	Bem	Bem	Bem
nome	nomeidioma	profCompreensao	profFala	profLeitura	profEscrita																																						
Renata Wassermann	Alemão	Bem	Razoavelmente	Bem	Razoavelmente																																						
Renata Wassermann	Holandês	Bem	Bem	Bem	Bem																																						
Renata Wassermann	Italiano	Bem	Razoavelmente	Bem	Pouco																																						
Renata Wassermann	Espanhol	Bem	Bem	Bem	Pouco																																						
Renata Wassermann	Francês	Bem	Razoavelmente	Bem	Pouco																																						
Renata Wassermann	Inglês	Bem	Bem	Bem	Bem																																						

Figura 7.7: Consulta SPARQL buscando Proficiência em Idiomas do pesquisador.

7.3 Consultas Consolidadas

Primeiro, mostraremos uma consulta relacionando informações de proficiência em idiomas de alguns pesquisadores. Este é um bom exemplo a ser mostrado, pois facilmente é possível localizar e visualizar as informações que dão origem ao resultado.

Na figura 7.8, temos trechos dos arquivos originais dos idiomas dos pesquisadores.

Idiomas **Renata - Idiomas**

Compreende Inglês (Bem), Alemão (Bem), Francês (Bem), Espanhol (Bem), Italiano (Bem), Holandês (Bem).

Fala Inglês (Bem), Alemão (Razoavelmente), Francês (Razoavelmente), Espanhol (Bem), Italiano (Razoavelmente), Holandês (Bem).

Lê Inglês (Bem), Alemão (Bem), Francês (Bem), Espanhol (Bem), Italiano (Bem), Holandês (Bem).

Escreve Inglês (Bem), Alemão (Razoavelmente), Francês (Pouco), Espanhol (Pouco), Italiano (Pouco), Holandês (Bem).

Idiomas **Flávio - Idiomas**

Compreende Inglês (Bem), Espanhol (Bem), Francês (Razoavelmente), Catalão (Razoavelmente).

Fala Inglês (Bem), Espanhol (Bem), Francês (Razoavelmente), Catalão (Razoavelmente).

Lê Inglês (Bem), Espanhol (Bem), Francês (Bem), Catalão (Razoavelmente).

Escreve Inglês (Bem), Espanhol (Razoavelmente), Francês (Razoavelmente), Catalão (Pouco).

Idiomas **Fábio - Idiomas**

Compreende Espanhol (Bem), Francês (Razoavelmente), Inglês (Bem).

Fala Espanhol (Razoavelmente), Francês (Razoavelmente), Inglês (Bem).

Lê Espanhol (Bem), Francês (Razoavelmente), Inglês (Bem).

Escreve Espanhol (Pouco), Francês (Pouco), Inglês (Bem).

Idiomas **Leliane - Idiomas**

Compreende Inglês (Bem), Espanhol (Bem).

Fala Inglês (Bem), Espanhol (Pouco).

Lê Inglês (Bem), Espanhol (Bem).

Escreve Inglês (Bem), Espanhol (Pouco).

Figura 7.8: Proficiência em Idiomas dos pesquisadores - original HTML.

Na figura 7.9, apresentamos a consulta que retorna as informações de proficiência dos pesquisadores no idioma “Espanhol”.

nome	nomeidioma	profCompreensao	profFala	profLeitura	profEscrita
Renata Wassermann	Espanhol	Bem	Bem	Bem	Pouco
Leliane Nunes de Barros	Espanhol	Bem	Pouco	Bem	Pouco
Fabio Kon	Espanhol	Bem	Razoavelmente	Bem	Pouco
Flavio Soares Correa da Silva	Espanhol	Bem	Bem	Bem	Razoavelmente

Figura 7.9: Consulta SPARQL buscando Proficiência em Espanhol dos pesquisadores.

Mostraremos agora uma consulta mais complexa. Vamos buscar na ontologia os Artigos que foram publicados entre os anos de 2003 e 2006, escritos por 2 ou mais autores. A figura 7.10 mostra toda a consulta em SPARQL e os resultados obtidos. Podemos notar como a estrutura da consulta SPARQL vai se tornando mais complexa.

Query	Results														
<pre> SELECT DISTINCT ?titulo ?ano WHERE { ?objeto :nomeCompleto ?nome . ?objeto2 :nomeCompleto ?nome2 . ?cur :temDadosGerais ?objeto ; :temProducaoBibliografica ?form . ?cur2 :temDadosGerais ?objeto2 ; :temProducaoBibliografica ?form . ?form a ?tipo . ?form :temDadosBasicos ?dados . ?dados :titulo ?titulo . ?dados :ano ?ano . } FILTER (?ano > 2002 && ?ano < 2007) . FILTER (?tipo = :ArtigosPublicados) . FILTER (?cur != ?cur2) . } ORDER BY ?ano ?titulo </pre>	<table border="1"> <thead> <tr> <th>titulo</th> <th>ano</th> </tr> </thead> <tbody> <tr> <td>Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines.</td> <td>2003</td> </tr> <tr> <td>The Universe of Approximations.</td> <td>2003</td> </tr> <tr> <td>Approximate and Limited Reasoning: Semantics, Proof Theory, Expressivity and Control.</td> <td>2004</td> </tr> <tr> <td>InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines.</td> <td>2004</td> </tr> <tr> <td>Designing Logic-based Robots.</td> <td>2006</td> </tr> <tr> <td>The Universe of Propositional Approximation.</td> <td>2006</td> </tr> </tbody> </table>	titulo	ano	Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines.	2003	The Universe of Approximations.	2003	Approximate and Limited Reasoning: Semantics, Proof Theory, Expressivity and Control.	2004	InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines.	2004	Designing Logic-based Robots.	2006	The Universe of Propositional Approximation.	2006
titulo	ano														
Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines.	2003														
The Universe of Approximations.	2003														
Approximate and Limited Reasoning: Semantics, Proof Theory, Expressivity and Control.	2004														
InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines.	2004														
Designing Logic-based Robots.	2006														
The Universe of Propositional Approximation.	2006														
Execute Query															

Figura 7.10: Consulta SPARQL buscando Artigos Publicados entre os anos de 2003 e 2006 que foram escritos por 2 ou mais autores.

7.4 SPARQL vs SQL vs XQuery

Nas seções anteriores, vimos alguns exemplos de consultas realizadas em uma ontologia OWL através da linguagem SPARQL. Os resultados poderiam também ser obtidos se utilizássemos consultas em arquivos XML através de XQuery¹ ou então fazendo consultas SQL em um banco de dados relacional [18] que contivesse as informações de alguma forma equivalente ao conteúdo presente na ontologia.

Um grafo RDF pode ser transformado para uma estrutura relacional de tabelas e as consultas em SPARQL podem ser convertidas para SQL [18]. Entretanto, consultas que necessitam relacionar dados diferentes tendem a ser bem mais complexas em SQL do que em SPARQL. A linguagem SQL exige que tabelas sejam relacionadas explicitamente através de *joins* enquanto que na linguagem SPARQL, as relações são implícitas. Quanto maior for o número de informações relacionadas, maior será a vantagem do SPARQL diante do SQL na facilidade de criação de consultas.

Além disso, as ontologias OWL, que usam os grafos RDFs, possuem motores de inferência que

¹<http://www.w3.org/TR/xquery/>

detectam relações entre classes que não são explícitas dentro do grafo, o que facilita ainda mais as consultas SPARQL que necessitam buscar informações em um conjunto de classes que possuem superclasses em comum.

Assim como é possível realizar uma série de consultas em SQL equivalentes a consultas SPARQL, também podemos realizar consultas utilizando a linguagem XQuery buscando o mesmo tipo de informação. Basicamente, podemos dizer que as vantagens do SPARQL em relação ao SQL também valem em relação ao XQuery, ou seja, a linguagem SPARQL é mais intuitiva e simples de se realizar consultas que relacionam muitas informações.

Vamos agora mostrar um exemplo de uma mesma consulta sendo realizada nas 3 linguagens. Através desse exemplo, poderemos ilustrar as vantagens e desvantagens de cada linguagem de consulta.

A consulta a ser realizada irá buscar as seguintes informações: “Mostrar o nome completo do pesquisador e as suas proficiências em fala, compreensão, leitura e escrita em cada idioma, mostrando também o nome do idioma.”.

Primeiro, vamos mostrar essa consulta na linguagem SPARQL e também a estrutura de dados acessada por essa consulta:

Exemplo 1:

```
SELECT ?nome ?nomeidioma ?profCompreensao
      ?profFala ?profLeitura ?profEscrita
WHERE {
  ?objeto :nomeCompleto ?nome .
  ?objeto :temIdiomas ?profidioma .
  ?profidioma :idioma ?idioma .
  ?profidioma :proficienciaDeLeitura ?profLeitura ;
              :proficienciaDeEscrita ?profEscrita ;
              :proficienciaDeCompreensao ?profCompreensao ;
              :proficienciaDeFala ?profFala .
  ?idioma :nomeDoIdioma ?nomeidioma .
}
```

Classes da ontologia acessadas por essa consulta:



Figura 7.11: Diagrama contendo as classes da ontologia do Currículo Lattes envolvidas na consulta de proficiências de idioma de um pesquisador.

Agora, buscaremos obter os mesmos resultados utilizando a linguagem SQL. Para isso, primeiro temos que criar a estrutura de dados em um banco SQL, compatível com a estrutura de dados que existe na ontologia, ou seja, respeitando as relações entre as triplas que existiam antes na ontologia.

Segue abaixo a estrutura de dados SQL criada, gerada a partir da estrutura da ontologia do Currículo Lattes:

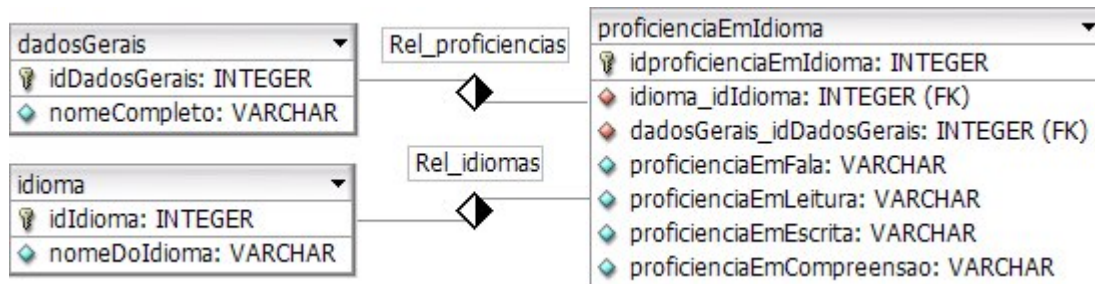


Figura 7.12: Modelo da estrutura de dados de um banco SQL contendo as mesmas informações e relações existentes na ontologia para o exemplo das proficiências em línguas.

Utilizando essa estrutura de dados SQL, mostraremos agora a consulta SQL que retorna os

mesmos resultados retornados pela consulta SPARQL:

```
SELECT dg.nomeCompleto, i.nomeDoIdioma, pei.proficienciaEmCompreensao,
       pei.proficienciaEmFala, pei.proficienciaEmLeitura, pei.proficienciaEmEscrita
FROM dadosGerais AS dg
JOIN proficienciaEmIdioma AS pei
     ON dg.idDadosGerais = pei.dadosgerais_idDadosGerais
JOIN idioma AS i
     ON pei.idioma_idIdioma = i.idIdioma
```

Nossa próxima consulta será com a linguagem XQuery. Neste caso, utilizaremos a linguagem XQuery nos arquivos XML originais do Currículo Lattes gerados pelos pesquisadores, com todas as suas informações. Apresentaremos uma possível consulta que traz os resultados desejados.

A consulta a ser realizada é:

```
<Resultados>
{
for $currículo in ($currículos/*)
return
  <LinhaResultado>
  {
    for $dg in $currículo/DADOS-GERAIS
    return
      $dg/@NOME-COMPLETO
  }
  {
    for $idioma in $currículo/DADOS-GERAIS/IDIOMAS/*
    return
      <Idioma>
      {$idioma/@DESCRICAO-DO-IDIOMA}
      {$idioma/@PROFICIENCIA-DE-LEITURA}
      {$idioma/@PROFICIENCIA-DE-FALA}
```

```

        {$idioma/@PROFICIENCIA-DE-ESCRITA}
        {$idioma/@PROFICIENCIA-DE-COMPREENSAO}
    </Idioma>
  }
</LinhaResultado>
}
</Resultados>

```

Agora, vamos apresentar outro exemplo de consulta que utilizará a facilidade da linguagem SPARQL em buscar informações facilmente em uma árvore de classes e subclasses. A busca que será realizada, em linguagem natural é: “Listar as orientações em andamento e concluídas dos pesquisadores, mostrando os dados da orientação como nome do orientando, título do trabalho e tipo de orientação (Mestrado, Doutorado, Iniciação Científica, etc)”.

Mais uma vez, começaremos mostrando essa consulta na linguagem SPARQL

Exemplo 2:

```

SELECT ?nome ?tipoDeOrientacao ?titulo ?nomeOrientando
WHERE {
  ?curriculo :temDadosGerais ?dg .
  ?dg :nomeCompleto ?nome .
  ?curriculo ?opoudc ?orientacao .
  ?orientacao a ?tipoDeOrientacao .
  ?orientacao :temDetalhamento ?detalhe .
  ?orientacao :temDadosBasicosDeOrientacoes ?db .
  ?detalhe :nomeDoOrientando ?nomeOrientando .
  ?db :tituloDoTrabalho ?titulo .
  ?tipoDeOrientacao rdfs:subClassOf ?superclasse .
  FILTER ( (?superclasse = :OrientacoesConcluidas) ||
           (?superclasse = :OrientacaoEmAndamento) ) .
  FILTER ( (?opoudc = :temOutraProducao) ||
           (?opoudc = :temDadosComplementares) ) .
}

```

Agora, mostraremos uma possível forma de armazenar os dados referentes a orientações dos pesquisadores em um banco de dados SQL. A estrutura apresentada não é a melhor modelagem possível. A idéia dessa modelagem é apresentar uma estrutura semelhante a ontologia, ou seja, com tabelas representando classes da ontologia sem utilizar regras de normalização na estrutura.

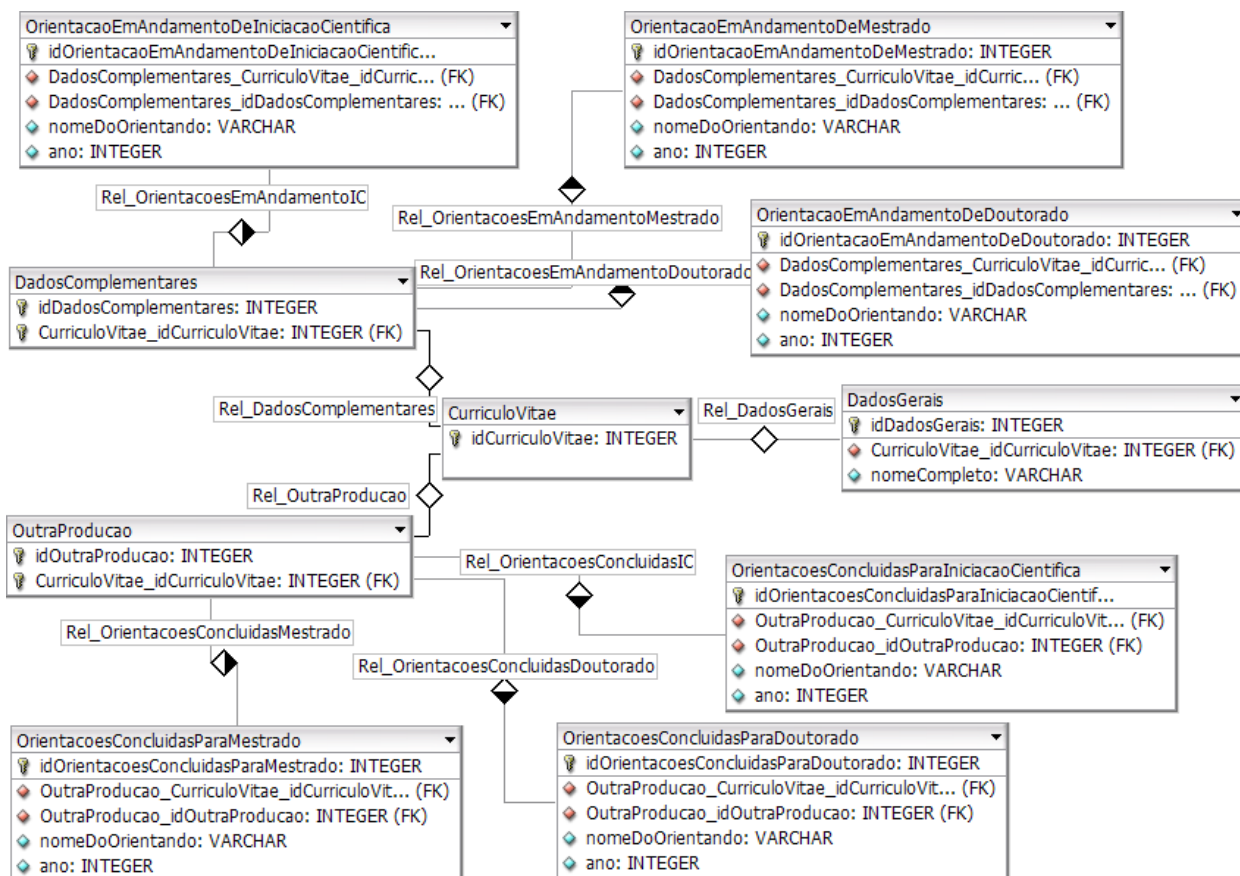


Figura 7.13: Modelo da estrutura de dados de um banco SQL contendo orientações de Mestrado, Doutorado e Iniciação Científica, baseado na estrutura de classes da ontologia do Currículo Lattes.

A consulta SQL para buscar as informações de orientações dos pesquisadores é:

```

SELECT nomeCompleto, tipo, nomeDoOrientando, ano
FROM
( (
  SELECT cv.idCurriculoVitae, 'OrientacaoEmAndamentoDeMestrado' tipo,
        o.nomeDoOrientando, o.ano
  FROM CurriculoVitae cv
  INNER JOIN OrientacaoEmAndamentoDeMestrado o
  ON o.DadosComplementares_CurriculoVitae_idCurriculoVitae = cv.idCurriculoVitae
) UNION (
  SELECT cv.idCurriculoVitae, 'OrientacaoEmAndamentoDeDoutorado' tipo,
        o.nomeDoOrientando, o.ano
  FROM CurriculoVitae cv
  INNER JOIN OrientacaoEmAndamentoDeDoutorado o
  ON o.DadosComplementares_CurriculoVitae_idCurriculoVitae = cv.idCurriculoVitae
) UNION (
  SELECT cv.idCurriculoVitae, 'OrientacaoEmAndamentoDeIniciacaoCientifica' tipo,
        o.nomeDoOrientando, o.ano
  FROM CurriculoVitae cv
  INNER JOIN OrientacaoEmAndamentoDeIniciacaoCientifica o
  ON o.DadosComplementares_CurriculoVitae_idCurriculoVitae = cv.idCurriculoVitae
) UNION (
  SELECT cv.idCurriculoVitae, 'OrientacoesConcluidasParaIniciacaoCientifica' tipo,
        o.nomeDoOrientando, o.ano
  FROM CurriculoVitae cv
  INNER JOIN OrientacoesConcluidasParaIniciacaoCientifica o
  ON o.OutraProducao_CurriculoVitae_idCurriculoVitae = cv.idCurriculoVitae
) UNION (
  SELECT cv.idCurriculoVitae, 'OrientacoesConcluidasParaMestrado' tipo,
        o.nomeDoOrientando, o.ano
  FROM CurriculoVitae cv
  INNER JOIN OrientacoesConcluidasParaMestrado o
  ON o.OutraProducao_CurriculoVitae_idCurriculoVitae = cv.idCurriculoVitae
) UNION (
  SELECT cv.idCurriculoVitae, 'OrientacoesConcluidasParaDoutorado' tipo,
        o.nomeDoOrientando, o.ano
  FROM CurriculoVitae cv
  INNER JOIN OrientacoesConcluidasParaDoutorado o
  ON o.OutraProducao_CurriculoVitae_idCurriculoVitae = cv.idCurriculoVitae
) ) as orientacoes
INNER JOIN DadosGerais dg
ON dg.CurriculoVitae_idCurriculoVitae = orientacoes.idCurriculoVitae

```

E, finalmente, mostraremos a consulta em XQuery buscando as orientações dos pesquisadores:

```

<Resultados>
{
for $curriculo in ($curriculos/*)
return
  <LinhaResultado>
    {
      for $dg in $curriculo/DADOS-GERAIS
      return
        $dg/@NOME-COMPLETO
    }
    {
      for $orien_concl in $curriculo/OUTRA-PRODUCAO/ORIENTACOES-CONCLUIDAS/*
      return
        <OrientacaoConcluida>
          {$orien_concl/*/@TITULO}
          {$orien_concl/*/@NOME-DO-ORIENTADO}
          <tipoDeOrientacao>
            {fn:node-name($orien_concl)}
          </tipoDeOrientacao>
        </OrientacaoConcluida>
    }
    {
      for $orien_andam in $curriculo/DADOS-COMPLEMENTARES/ORIENTACOES-EM-ANDAMENTO/*
      return
        <OrientacaoEmAndamento>
          {$orien_andam/*/@TITULO}
          {$orien_andam/*/@NOME-DO-ORIENTANDO}
          <tipoDeOrientacao>
            {fn:node-name($orien_andam)}
          </tipoDeOrientacao>
        </OrientacaoEmAndamento>
    }
  </LinhaResultado>
}
</Resultados>

```

Através desses exemplos, mostramos a facilidade que a linguagem SPARQL proporciona para o desenvolvimento de consultas que relacionam muitas informações. No exemplo 1, notamos com

a linguagem SPARQL torna mais intuitiva a ligação entre informações diferentes. O tamanho das consultas SPARQL e SQL é semelhante mas a consulta em SPARQL é bem mais simples de ser desenvolvida.

No exemplo 2, vemos como facilmente conseguimos realizar uma busca que acessa informações que estão em diferentes classes mas que possuem uma superclasse em comum, ou seja, informações que representam entidades diferentes, mas que possuem características comuns. Essa característica do SPARQL mostra uma grande vantagem diante do SQL que necessita que as informações sejam explicitamente relacionadas na busca. A linguagem XQuery apresenta uma certa facilidade para realizar buscas de diferentes informações que tenham uma entidade pai em comum, mas apresenta algumas dificuldades no momento que queremos selecionar somente alguns tipos de propriedades a serem apresentadas e também certos problemas para unificar os resultados, ou seja, para retornar propriedades diferentes de informações diferentes.

Em relação ao desempenho das linguagens, a linguagem SPARQL consegue trazer respostas rápidas para consultas simples mas apresenta certa demora quando estamos tratando de muitas informações. Certamente a linguagem SQL leva vantagem no desempenho em relação ao SPARQL pois já é uma linguagem amplamente estudada e com implementações bastante otimizadas. Porém, é provável que nos próximos anos as pesquisas relacionadas com SPARQL se intensifiquem e a tendência é que sejam desenvolvidas otimizações, assim como aconteceu com o SQL, de maneira que o desempenho deixe de ser uma desvantagem do SPARQL.

De modo geral, podemos dizer que o uso de SPARQL e ontologias facilita o desenvolvimento de consultas pois as consultas apresentam uma intuitividade muito maior e, portanto, o tempo necessário para desenvolver consultas em SPARQL tende a ser menor do que para desenvolver consultas SQL. Para o tipo de informação que estamos tratando, as consultas SQL podem ser bastante complexas, conforme mostramos nos exemplos. Temos, portanto, de um lado o SPARQL e as ontologias apresentando sua maior facilidade nas consultas e de outro lado o SQL e os bancos de dados relacionais, apresentando melhor desempenho no tempo de retorno dos resultados.

Neste trabalho não incluímos estudos com bancos de dados orientados a objetos nas comparações mas acreditamos que seria um estudo de grande valia por conta de suas características de modelagem se assemelharem as características dos modelos de ontologias e por essa razão esse estudo fica como sugestão para trabalhos futuros.

Capítulo 8

Conclusões e trabalhos futuros

O processo de população de uma ontologia a partir de informações em arquivos HTML e posterior extração de relatórios foi a principal motivação deste trabalho desde o princípio e utilizamos o caso dos currículos da Plataforma Lattes como base deste estudo.

Com a grande popularização da linguagem HTML, já encontramos hoje muitos trabalhos e algoritmos que tentam auxiliar a manipulação desses arquivos mas neste trabalho a dificuldade maior não foi de somente extrair a informação do HTML e sim de extraí-la e transformá-la em uma base de conhecimento, mapeando corretamente suas informações. Acreditamos que uma das principais contribuições deste trabalho seja o de mostrar todo o processo de criação de instâncias dentro de uma ontologia a partir de scripts automáticos e descrever as dificuldades envolvidas neste processo, dificuldades essas que tomaram grande parte do trabalho.

Os scripts desenvolvidos no trabalho possuem conteúdo bastante específico para tratar do caso do Currículo Lattes. Entretanto, a forma como as informações são tratadas e inseridas na base de conhecimento certamente pode ser aplicada a outros tipos de problemas. Este trabalho teve como foco um estudo de caso, mas o objetivo maior desse estudo foi trabalhar com uma ontologia e mostrar alguns dos problemas relacionados com o tema. Os problemas apontados e as soluções propostas podem ser adaptados a casos semelhantes e servir como base de soluções ainda mais interessantes.

Dentro do processo de população da ontologia foram encontrados grandes desafios mas podemos destacar em especial dois deles:

- Identificar corretamente os textos dentro dos arquivos originais para que fosse possível mapear a ontologia com a semântica correta dos termos.

- Identificar e retirar as duplicidades de instâncias que se referem a um mesmo objeto

O problema da co-referência que existe dentro do processo de população automática da ontologia pode ser tratado de várias maneiras diferentes. Neste trabalho, utilizamos duas abordagens diferentes na busca por similaridades e mostramos suas principais características.

Cada abordagem utilizada apresenta suas vantagens e desvantagens e é difícil simplesmente afirmar que uma abordagem é melhor que a outra. Essa avaliação depende da realização de testes baseados nos objetivos que se deseja atingir. Podemos, em determinado momento, ter interesse em encontrar o maior número de verdadeiras duplicidades, mesmo que isso tenha como efeito o aparecimento de falsas duplicidades. Em outro momento, podemos desejar encontrar duplicidades mas sem que nenhuma falsa duplicidade seja encontrada.

Levando em conta os números obtidos nos testes realizados, podemos dizer que os dois métodos desenvolvidos atingem bons resultados ao buscar duplicidades. Dificilmente um sistema conseguiria obter 100% na medida Cobertura para este problema de duplicidades e portanto os resultados obtidos, que foram próximos a 80%, parecem ser bastante satisfatórios.

Alguns métodos utilizados para otimizar a busca por duplicidades, como a busca apenas por títulos de comprimento próximo, são certamente bastante eficazes neste caso e possivelmente em problemas similares. Entretanto, outros critérios que foram adotados podem não ser necessariamente a melhor solução para outros casos. Somente quando se sabe o tipo de informação que está sendo tratada e o contexto dessas informações, torna-se possível definir quais critérios utilizar ou até mesmo definir se é vantajoso utilizar muitos critérios simultaneamente. Podemos dizer que a escolha da melhor abordagem somente pode ser bem feita se acompanhada de testes para avaliar a qualidade dos resultados e também em quanto tempo esses resultados estão sendo gerados.

É importante ressaltar que o problema das co-referências foi um dos temas principais deste trabalho por conta de sua complexidade e grande importância por aparecer em vários tipos de situações dentro da Computação. Apresentamos neste trabalho dois tipos de soluções tratando especificamente do caso do Currículo Lattes, mas podemos afirmar que as idéias contidas no trabalho podem ser usadas para tratar de outros casos relacionados com co-referências. Procuramos, neste trabalho, apresentar o problema e mostrar formas de tentar minimizar seu impacto negativo. Esperamos que o estudo de caso realizado possa facilitar o tratamento para este tipo de problema, mesmo que em outros casos ele esteja inserido em contextos distintos.

Neste trabalho, conseguimos realizar a população automática da ontologia do Currículo Lattes, ontologia esta que havia sido criada por Ailton Sergio Bonifacio e depois convertida de DAML+OIL para OWL por Marcos Yoshinori Nakashima. Essa automatização claramente não é um processo simples e uma mostra disso é ver cada um dos trabalhos que já foram relacionados com esse tema. Podemos considerar que este trabalho é apenas mais um passo no processo visto que melhorias ainda podem ser feitas na ontologia que está sendo gerada.

Para estudos futuros relacionados com o que foi estudado neste projeto, podemos citar algumas idéias:

- Realizar algum tipo de treinamento no processo de busca de similaridades, tornando o algoritmo mais inteligente na tomada de decisão para dizer se dois itens referem-se a um mesmo objeto.
- Testar algoritmos mais eficientes na comparação de itens similares, possivelmente com algumas condições que façam o algoritmo detectar itens diferentes com mais rapidez.
- Criar ferramentas que permitam que os usuários tenham mais facilidade para interagir com o script de população da ontologia, permitindo que se tenha mais flexibilidade sem que seja necessário modificar o código-fonte.
- Fazer comparações entre SPARQL, SQL e XQuery. Verificar com mais profundidade as vantagens e desvantagens de cada uma das linguagens e sugerir melhorias para a linguagem SPARQL, que ainda é uma linguagem nova.
- Comparar o uso de SPARQL em bases de conhecimento com consultas em bancos de dados orientados a objetos.

Os arquivos desenvolvidos neste trabalho podem ser encontrados no endereço:
<http://www.ime.usp.br/~casado/Mestrado/Arquivos/>.

Referências Bibliográficas

- [1] Harith Alani, Srinandan Dasmahapatra, Nicholas Gibbins, Hugh Glaser, Steve Harris, Yannis Kalfoglou, Kieron O'Hara, and Nigel Shadbolt, *Managing reference: Ensuring referential integrity of ontologies for the semantic web*, 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02), Lecture Notes in Artificial Intelligence, 2002, pp. 317–334.
- [2] Franz Baader, Ian Horrocks, and Ulrike Sattler, *Description logics as ontology languages for the semantic web*, Mechanizing Mathematical Reasoning: Essays in Honor of Jörg Siekmann on the Occasion of His 60th Birthday (Dieter Hutter and Werner Stephan, eds.), Lecture Notes in Artificial Intelligence, no. 2605, Springer, 2005, pp. 228–248.
- [3] Tim Berners-Lee, James Hendler, and Ora Lassila, *The semantic web*, Scientific American **284** (2001), no. 5, 34–43.
- [4] Ailton Sergio Bonifacio, *Ontologias e consulta semântica: Uma aplicação ao caso Lattes*, Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, 2002.
- [5] Thomas M. Breuel, *Information extraction from HTML documents by structural matching*, Second International Workshop on Web Document Analysis (Edinburgh, UK), 2003, <http://www.csc.liv.ac.uk/~wda2003/>.
- [6] Stefan Decker, Sergey Melnik, Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks, *The semantic web: The roles of XML and RDF*, IEEE Internet Computing **04** (2000), no. 5, 63–74.
- [7] Frederico Luiz Gonçalves De Freitas, *Sistemas multiagentes cognitivos para a recuperação, classificação e extração integradas de informação da web*, Tese de Doutorado, Universidade Federal de Santa Catarina, 2002.
- [8] Johannes Goller, *Stan: Structural analysis for web documents*, Second International Workshop on Web Document Analysis (Edinburgh, UK), 2003, <http://www.csc.liv.ac.uk/~wda2003/>.
- [9] Grigoris Antoniou and Frank van Harmelen, *A Semantic Web Primer*, MIT Press, 2004.

- [10] T. R. Gruber, *Ontolingua: A mechanism to support portable ontologies*, Technical report, Knowledge Systems Laboratory, Stanford University, Stanford, United States (1992).
- [11] T. R. Gruber, *Towards principles for the design of ontologies used for knowledge sharing*, Formal Ontology in Conceptual Analysis and Knowledge Representation (Deventer, The Netherlands) (N. Guarino and R. Poli, eds.), Kluwer Academic Publishers, 1993.
- [12] Thomas R. Gruber, *A translation approach to portable ontology specifications*, Knowl. Acquis. **5** (1993), no. 2, 199–220.
- [13] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen, *The making of a web ontology language*, Journal of Web Semantics **1** (2003), no. 1, 7–26.
- [14] P. Jackson and I. Moulinier, *Natural language processing for online applications: Text retrieval, extraction, and categorization (natural language processing, 5)*, John Benjamins Publishing Co, Junho 2002.
- [15] Ah-Hwee Tan Lakshmi Vijjappu and Chew-Lim Tan, *Web structure analysis for information mining*, First International Workshop on Web Document Analysis (Seattle, Washington, USA), 2001, <http://www.csc.liv.ac.uk/~wda2001/>.
- [16] Vladimir Levenshtein, *Binary codes capable of correcting deletions, insertions and reversals*, Original in Russian in Dokl. Akad. Nauk (1966), 845–848.
- [17] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel, *Performance measures for information extraction*, In Proceedings of DARPA Broadcast News Workshop (Virginia, EUA), Fevereiro 1999.
- [18] Jim Melton, *SQL, XQuery, and SPARQL*, XTech 2006: “Building Web 2.0” (Amsterdam, Holanda), Maio 2006.
- [19] Marcos Yoshinori Nakashima, *Currículo Lattes e Web Semântica, Projeto de Iniciação Científica, IME - USP*, <http://www.linux.ime.usp.br/~cef/mac499-04/monografias/rec/myn/>, 2004.
- [20] Gonzalo Navarro, *Binary codes capable of correcting deletions*, ACM Comput. Surv. **33** (2001), no. 1, 31–88.
- [21] Natalya F. Noy and Deborah L. McGuinness, *Ontology development 101: A guide to creating your first ontology*, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880 (2001).
- [22] Jeff Z. Pan and Ian Horrocks, *Introducing customised datatypes and datatype predicates into OWL*, Proc. of the First OWL Experiences and Directions Workshop, CEUR (<http://ceur-ws.org/>), vol. 188, 2005.

- [23] DAML+OIL Reference, *Reference description of the DAML+OIL ontology markup language*, Disponível em: <http://www.daml.org/2001/03/reference> (2001).
- [24] Tércio De Moraes Sampaio Silva, *Extração de informação para busca semântica na web baseada em ontologias*, Dissertação de Mestrado, Universidade Federal de Santa Catarina, 2003.
- [25] Giorgos Stoilos, Giorgos Stamou, Vassilis Tzouvaras, Jeff Z. Pan, and Ian Horrocks, *Fuzzy OWL: Uncertainty and the semantic web*, Proc. of the First OWL Experiences and Directions Workshop, CEUR (<http://ceur-ws.org/>), vol. 188, 2005.
- [26] Manabu Okumura Tomoyuki Nanno, Suguru Saito, *Structuring web pages based on repetition of elements*, Second International Workshop on Web Document Analysis (Edinburgh, UK), 2003, <http://www.csc.liv.ac.uk/~wda2003/>.
- [27] G. van Heijst, A. Th. Schreiber, and B. J. Wielinga, *Using explicit ontologies in KBS development*, Int. J. Hum.-Comput. Stud. **46** (1997), no. 2-3, 183–292.
- [28] C. J. Van Rijsbergen, *Information retrieval, 2nd edition*, Dept. of Computer Science, University of Glasgow, 1979.
- [29] Katherine Wolstencroft, Andy Brass, Ian Horrocks, Phil Lord, Ulrike Sattler, Robert Stevens, and Daniele Turi, *A little semantic web goes a long way in biology*, Proc. of the 4th International Semantic Web Conference (ISWC 2005), Lecture Notes in Computer Science, vol. 3729, Springer, 2005, pp. 786–800.