SENAI CIMATEC

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL E TECNOLOGIA INDUSTRIAL

Mestrado em Modelagem Computacional e Tecnologia Industrial

Dissertação de mestrado

MOFI: Modelo Computacional para Recuperação de Informação baseado em Ontologias, Folksonomia e Indexação Automática de Conteúdo

Apresentada por: Uedson Santos Reis
Orientador: Hernane Borges de Barros Perreira

Fevereiro de 2011

Uedson Santos Reis

MOFI: Modelo Computacional para Recuperação de Informação baseado em Ontologias, Folksonomia e Indexação Automática de Conteúdo

Dissertação de mestrado apresentada ao Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial, Curso de Mestrado em Modelagem Computacional e Tecnologia Industrial do SENAI CIMATEC, como requisito parcial para a obtenção do título de **Mestre em Modelagem Computacional e Tecnologia Industrial**.

Área de conhecimento: Interdisciplinar

Orientador: Hernane Borges de Barros Perreira
SENAI CIMATEC

Salvador
SENAI CIMATEC
2011

Nota sobre o estilo do PPGMCTI

Esta dissertação de mestrado foi elaborada considerando as normas de estilo (i.e. estéticas e estruturais) propostas aprovadas pelo colegiado do Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial e estão disponíveis em formato eletrônico (*download* na Página Web http://ead.fieb.org.br/portal_faculdades/dissertacoes-e-teses-mcti.html ou solicitação via e-mail à secretaria do programa) e em formato impresso somente para consulta.

Ressalta-se que o formato proposto considera diversos itens das normas da Associação Brasileira de Normas Técnicas (ABNT), entretanto opta-se, em alguns aspectos, seguir um estilo próprio elaborado e amadurecido pelos professores do programa de pós-graduação supracitado.

SENAI CIMATEC

Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial

Mestrado em Modelagem Computacional e Tecnologia Industrial

A Banca Examinadora, constituída pelos professores abaixo listados, leram e recomendam a aprovação [com distinção] da Dissertação de mestrado, intitulada "MOFI: Modelo Computacional para Recuperação de Informação baseado em Ontologias, Folksonomia e Indexação Automática de Conteúdo", apresentada no dia (dia) de (mês) de (ano), como requisito parcial para a obtenção do título de **Mestre em Modelagem Computacional e Tecnologia Industrial**.

Orientador:

Prof. Dr. Hernane Borges de Barros Perreira
SENAI CIMATEC

Membro da Banca:

Prof. Dr. Josemar Rodrigues de Souza
SENAI CIMATEC

Membro externo da Banca:

Profa. Dra. Sonia Limoeiro Monteiro
Laboratório Nacional de Computação Científica

Dedico este trabalho a meu Pai (Edson Reis), que não está mais entre nós, mas que sempre se empenhou por meus estudos e que, enquanto vivo, demonstrou o verdadeiro amor e dedicação que um pai deve ter por cada um de seus filhos.

Agradecimentos

Em primeiro lugar, gostaria de agradecer à minha família, pela minha educação, pela compreensão, e pelo amor e apoio que sempre me deram. A minha noiva, Lívia, pelo amor, carinho e paciências pelas horas de estudo em sua casa.

Agradeço também ao meu orientador Hernane Pereira pelos ensinamentos, pelo incentivo e pela atenção sempre prestada. Assim como agradeço a Eduardo Jorge, que compartilhou comigo seus conhecimentos no tema estudado e que me fez lembrar sempre da importância maior de uma pesquisa, que é a qualidade de uma contriuíção deixada para a comunidade.

Agradeço também a todos os professores que tive nesse mestrado, que contribuíram para o meu crescimento pessoal e profissional. Assim como a meus amigos, principalmente Ronaldo Evangelista, que me ajudaram durante o processo de pesquisa e desenvolvimento do projeto.

Por fim, agradeço aos meus colegas de turma, os quais compartilhei grandes experiências durante o curso.

Salvador, Brasil — Uedson Santos Reis

24 de Fevereiro de 2011

Resumo

A maior parte do conteúdo disponível na web só tem significado para os seres humanos, não podendo ser entendido pela máquina em um contexto semântico. Como reflexo disso, a maioria dos motores de busca na web leva a sintaxe em consideração, e não a semântica, no momento de realizar uma busca, o que nem sempre traz um resultado satisfatório. Uma das formas de ampliar a semântica, nesse contexto, é a incorporação de sistemas de representação do conhecimento, como por exemplo, as ontologias, que possam ser manipulados e entendidos por sistemas computacionais. A utilização da Folksonomia (i.e. classificação feita por pessoas) nesse contexto também pode auxiliar na ampliação da semântica desses dados, principalmente se for utilizada de forma integrada com as ontologias. Esta dissertação é fruto de um projeto que propõe a construção de um modelo computacional que permita a construção de bases de dados semi-estruturados pautadas em ontologias, folksonomia e indexação automática de arquivos, de modo à facilitar a classificação e recuperação de conteúdos armazenados nesta base. Para construção deste projeto foram feitos levantamentos bibliográficos, coleta e analise de dados na web e o desenvolvimento de um sistema web seguindo o modelo incremental de desenvolvimento de software. Como contribuíção temos a proposta do uso de um sistema de representação do conhecimento, como as ontologias, para aperfeiçoamento das buscas.

Abstract

Most of the content available on the web only has meaning for human beings, which can not be understood in a semantic context by the machine. As a reflect, most of the search engines on the web take the syntax, not semantics, at consideration when performing a search, which not always produce a satisfactory result. One way to extend the semantics, in this context, is to incorporate systems that represents the knowledge, such as ontologies that can be manipulated and understood by computer systems. The use of Folksonomy (e.g. classification made by the people) in this context can also assist the expantion of the data semantics, especially if it is used in an integrated manner with the ontologies. This master's thesis is the result of a project that attempts to build a computational model that allows the construction of semi-structured data bases guided by ontologies, folksonomy and automatic files indexing, whose purpose is to contribute to classification and mapping of all content stored in that base. The elaboration of this project, it was done several bibliographical research, collect and analyze of the data in the web and developing a web system which implements an incremental model of software developing. Our contribution lies on the purpose of using a knowledge representation system, such as ontologies, to refine the search.

Sumário

Lista de Tabelas

Lista de Figuras

List of Algorithms

Lista de Siglas

DAML	DARPA Agent Markup Language
HTML	HyperText Markup Language
OIL	Ontology Inference Layer
OWL	Web Ontology Language
PPGMCTI ..	Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial
RDF	Resourse Description Framework
XML	eXtensible Markup Language
W3C	World Wide Web Consortium
WWW	World Wide Web

Capítulo Um

Introdução

Neste capítulo será feita uma introdução aos temas abordados nesse trabalho, além de definir pontos importantes dessa pesquisa, como problema, objetivo, metodologia e estrutura do documento.

1.1 Definição do problema

Pessoas ao redor do mundo vêm alimentando bases de conhecimento na *web*, principalmente através de diversos serviços (e.g. sites de relacionamento, redes sociais, enciclopédias virtuais, *blogs* e fóruns de discussão) que facilitam a publicação de conteúdo por parte do usuário, aumentando a quantidade de dados armazenados e potencializando a capacidade do usuário de difundir conteúdo na *web*.

A maior parte desse conteúdo é manipulada em páginas web não podem ser interpretados por algoritmos, mesmo os mais robustos baseados em inteligência artificial. Desta forma, torna-se difícil a extração de conhecimento na web. Como reflexo disso a maioria dos motores de busca na web acaba levando a sintaxe, e não a semântica, em consideração no momento de realizar uma busca, o que nem sempre traz um resultado satisfatório (BREITMAN, 2005). Também fica comprometida a realização de tarefas na web por parte de agentes lógicos, que são softwares baseados em inteligência artificial (GRUBER, 1993). A montagem de um roteiro de visitas médicas, para um determinado paciente, é uma tarefa que pode ser citada como exemplo, as páginas web geralmente contém informações sobre a localização das clínicas e ainda podem provêr o serviço de agendamento de consultas.

Tentando solucionar esse problema, a Web Semântica foi idealizada, por Berners-Lee, Hendler e Lassila (2001), como a evolução da web onde o seu conteúdo tem significado bem definido. A ideia dessa nova web é permitir o entendimento desse conteúdo pelos computadores, para assim aperfeiçoar o funcionamento dos motores de busca e viabilizar a realização de algumas tarefas por parte das máquinas (BERNERS-LEE; HENDLER; LASSILA, 2001), como a citada no parágrafro anterior.

Sistemas de Representação do conhecimento, como por exemplo, Ontologias, podem auxiliar a atribuir mais semântica aos conteúdos. Ontologias, no âmbito da Ciência da Computação, são estruturas formais de conceitos e seus relacionamentos que especificam um determinado domínio do conhecimento. Por ser uma estrutura formal, a ontologia

pode ser manipuladas e entendidas por sistemas computacionais (RA; GA; RA, 2004). Os conceitos de um sistema como este podem auxiliar na classificação e recuperação de dados, assim como seus relacionamentos podem indicar a semântica correta para a busca.

Utilizar um sistema de representação do conhecimento, como a ontologia, agregado à técnicas de classificação e recuperação de informação, como a folksonomia, também pode aperfeiçoar a recuperação de informação na web (REIS et al., 2009). Folksonomia é uma técnica já popular na web, nela o próprio usuário classifica seu conteúdo, atribuindo uma ou mais tags ao mesmo, o que facilita a recuperação desse conteúdo (PRIMO, 2006). Entende-se neste trabalho, tag como etiqueta, rótulo ou palavra para fins de classificação de conteúdo.

A ontologia pode fornecer termos formais para classificação do conteúdo, esses termos também podem ser mais difundidos, pois a ontologia pode ser utilizada por diversos usuários. Já a folksonomia pode agregar conhecimento coletivo obtido com as diversas classificações dos usuários, pois permite que o mesmo utilize um termo não existente na ontologia para classificar seu conteúdo (REIS et al., 2009).

Portanto, o problema estudado nessa dissertação é como utilizar as técnicas da Web Semântica (ontologia e folksonomia) para aperfeiçoar as técnicas convencionais de Recuperação de Informação que é estudada no Cápitulo 2.

1.2 Objetivo

O objetivo geral desse trabalho é projetar um modelo computacional que integre Ontologias, Folksonomia e Técnicas de Indexação Automática para aperfeiçoar os processos de armazenamento e recuperação de dados semi-estruturados. Esse modelo computacional será denominado MOFI.

Para atingir este objetivo faz-se necessário as seguintes metas:

1. Alterar o algoritmo *IterativeCloud*, no componente *Ontology Tagging*, para converter uma ontologia de domínio em uma metáfora visual, mais especificamente, a nuvem de tags. Essa nuvem de tags resultante, apresenta a tag selecionada no centro, tendo as outras dispostas ao seu redor, podendo ser alterada de acorda a nova tag escolhida pelo usuário.
2. Especificar e implementar um componente de indexação automática, que utilize um motor de busca baseado em indexação automática de arquivos para realização de busca sintática, como feito em sites de busca como Google, Yahoo e AltaVista.

3. Especificar e implementar um componente de indexação manual, que utilize a folksonomia para a indexação manual dos arquivos, permitindo ao usuário classificar o conteúdo com termos inseridos livremente ou com os conceitos da ontologia de domínio, seguindo a técnica sugerida por Reis et al. (2009).
4. Especificar e implementar um motor de busca, que gerencie a recuperação de conteúdo, consultando ontologias de domínio para refinar as buscas. Este irá tentar contextualizar as buscas e também localizar termos sinônimos aos termos buscados. Após esse refinamento o componente realizará a busca nos dois componentes de indexação, no automático e no manual.
5. Desenvolver um sistema web para armazenamento e recuperação de dados semi-estruturados que implemente as especificações do MOFI.

1.3 Importância da pesquisa

O modelo proposto tem como finalidade melhorar o compartilhamento e otimizar a recuperação de dados disponíveis na web, assim como permitir às máquinas, por intermédio de agentes lógicos, processar melhor esse conteúdo, facilitando algumas tarefas desenvolvidas pelos seres humanos.

A difusão do conhecimento também será facilitada nesse contexto, uma vez que os motores de busca poderão entender melhor a semântica de cada dado manipulado no momento da busca, podendo ter um retorno mais preciso.

1.4 Limitações

O uso da folksonomia é um ponto crítico do projeto, pois requer um espaço de armazenamento muito extenso para as tags. Por ser uma classificação feita de forma livre, o usuário pode atribuir quantos termos desejar o que, no longo prazo, pode aumentar muito o número de registros no banco de dados. Por esse motivo, recomenda-se uma manutenção periódica no banco de dados (e.g. a exclusão das tags que não estiverem sendo utilizadas).

1.5 Aspectos Metodológicos

O desenvolvimento da pesquisa ocorreu em 4 (quatro) fases, de acordo ao diagrama da Figura 1.1.

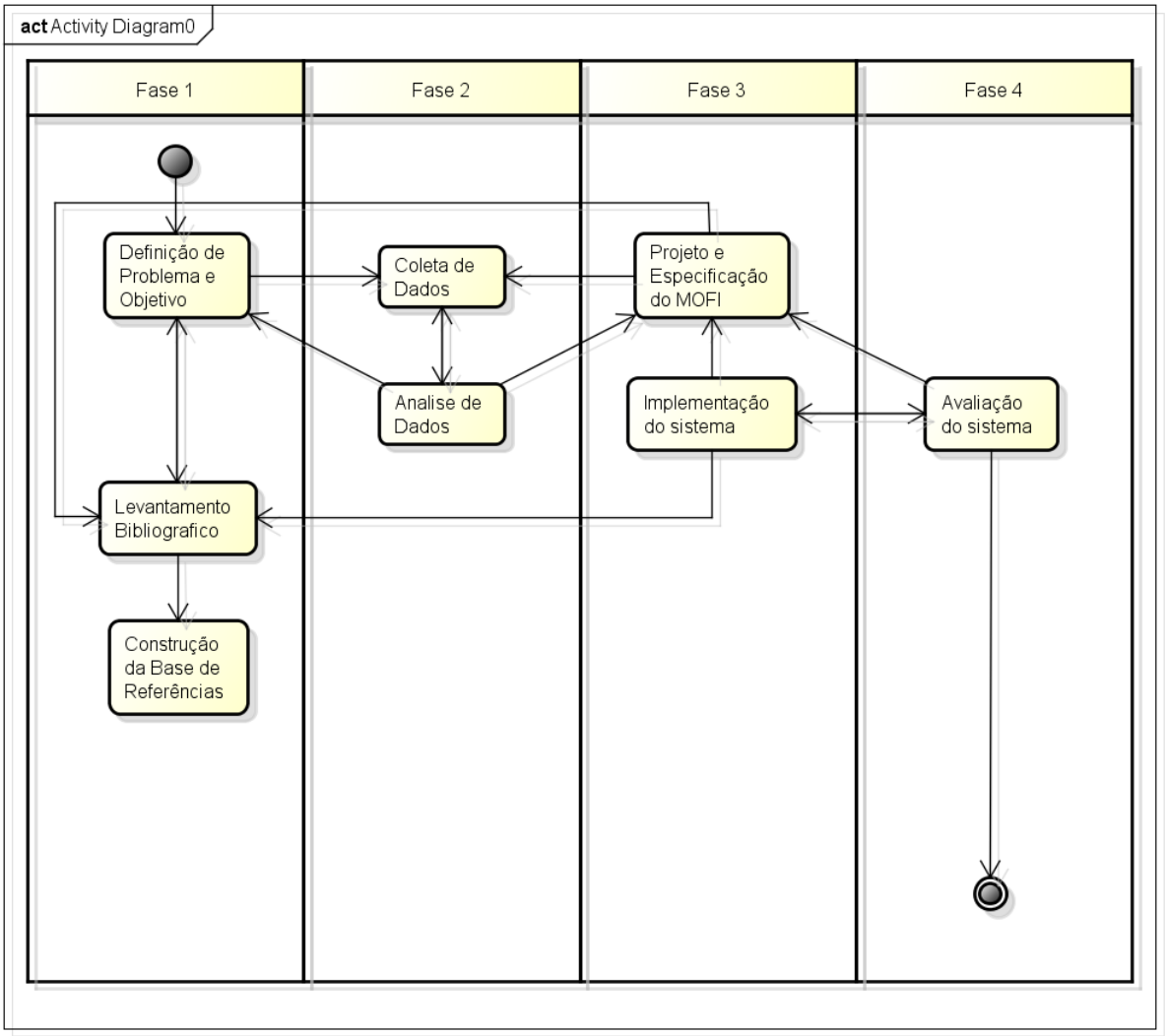


Figura 1.1: Arquitetura do modelo MOFI.

Para a definição do problema foi necessário um levantamento bibliográfico sobre os temas abordados, web semântica, folksonomia, recuperação de informação e principalmente ontologia. Ainda para a definição do problema também foi necessária uma coleta e análise de dados, que nesta pesquisa foi feita através de testes de busca na web e de uso e estudo de tecnologias relevantes aos temas abordados, como por exemplo a anotação semântica.

A partir da definição do problema, da base bibliográfica levantada, e da analise de dados ou tecnologias foi possível iniciar a específicação do MOFI. Em alguns momentos durante a especificação do modelo foi necessário uma nova coleta e análise de dados, assim como um novo levantamento bibliográfico o que permitiu o aperfeiçoamento do modelo.

Pouco antes de concluir a especificação do modelo o desenvolvimento do software foi iniciado. A especificação do MOFI só foi concluída durante o desenvolvimento do sistema, pois este guiou os ajustes do modelo. Também foram necessários novos levantamentos bi-

bliográficos nesta fase para solucionar problemas técnicos no desenvolvimento e esclarecer dúvidas quanto à especificação do modelo.

O sistema desenvolvido para validar o Modelo Computacional proposto foi feito de acordo com a metodologia Incremental, uma vez que no início do projeto não se tinha ciência de todos os requisitos funcionais e não-funcionais, do sistema e do modelo.

Na metodologia Incremental cada fase é desenvolvida como uma funcionalidade e no final da fase, esta será integrada ao restante do projeto. Desta forma, caso haja algum problema, este poderá ser detectado e tratado antes de se aproximar o prazo de conclusão do projeto de software. Com a metodologia Incremental é possível o refinamento progressivo dos requisitos do projeto que estará sendo desenvolvimento gradativamente (SOMMERVILLE, 2003). Cada um dos componentes, descritos nos objetivos específicos, foram desenvolvidos de acordo com as seguintes fases: analise de requisitos, projeto e implementação, e testes e implantação.

1.6 Organização da Dissertação de mestrado

Esta Dissertação de mestrado apresenta *6* capítulos e está estruturada da seguinte forma:

- **Capítulo 1 - Introdução**: Apresenta tópicos importantes da pesquisa, como o problema, o objetivo e a metodologia;
- **Capítulo 2 - Da Web Colaborativa para a Web Semântica**: apresenta um panorama da Web, mostrando sua evolução, suas características, benefícios e deficiências, assim como alguns porquês do surgimento da Web Semântica;
- **Capítulo 3 - Sistema de Representação do Conhecimento - Ontologias**: estuda as técnicas de Representação do Conhecimento com foco em Ontologia;
- **Capítulo 4 - MOFI**: apresentação do modelo computacional proposto, do sistema desenvolvido para validar o modelo, e a avaliação do modelo;
- **Capítulo 5 - Considerações Finais**: apresenta as conclusões, contribuições e algumas sugestões de atividades de pesquisa a serem desenvolvidas no futuro.

Capítulo Dois

Da Web Colaborativa para a Web Semântica

Este capítulo discute a evolução da web e de suas técnicas para armazenar e recuperar informação. Inicialmente é estudada a origem da web. Características como as redes sociais e o conhecimento coletivo são explorados e uma análise de seus métodos de armazenamento e recuperação é feita, assim como é apresentada a técnica de indexação de arquivos. Também são tratados alguns porquês do surgimento da Web Semântica, juntamente com sua definição. Por fim, a Folksonomia será apresentada como uma iniciativa para viabilizar a Web Semântica.

2.1 Surgimento da Web

A internet foi desenvolvida em 1969 pelos pesquisadores da área de computação, J. C. R. Licklider e Robert Taylor, para manter a comunicação das bases militares dos Estados Unidos durante a guerra fria. Com o fim da Guerra fria a *ArphaNet*, como era chamada, foi cedida à universidades norte-americanas, e seu acesso permitido aos cientistas e estudantes da área de computação. Técnicos da área de computação passaram a utilizar a internet para compartilhar informações, o que a tornou popular entre eles até se tornar a *World Wide Web* (www) ou simplesmente Web.

Por ter uma forma simples de utilização, usuários comuns também começaram a criar suas páginas e disponibilizá-las na web. Desde então, o número de usuários vem aumentando e uma cultura de serviços, comunicação e entretenimento vem se popularizando nessa rede mundial de computadores (BREITMAN, 2005).

A web evolui para o desenvolvimento de serviços *online*, como webmails e redes sociais, potencializando as formas de publicação, compartilhamento, criação e organização de conteúdo, além de ampliar as formas de relacionamento entre seus usuários. Por essas características, foi denominada por pesquisadores de Web Colaborativa, também chamada de Web 2.0 (PRIMO, 2006).

Na visão técnica da Web 2.0, descrita por O'Reilly (2005), pode-se dizer que o fator mais importante é desenvolver aplicações que aproveitem os recursos da rede para permitir ao usuário não só criar, publicar e acessar conteúdo como também editar ou comentar o conteúdo criado e publicado por outros usuários, criando o chamado conhecimento coletivo, ou inteligência coletiva. Esta expressão é usada para definir todo conteúdo criado

pela contribuição coletiva das pessoas que utilizam os recursos da Web 2.0 (GRUBER, 2007).

2.2 Conhecimento Coletivo

A criação e o compartilhamento de conteúdo, como artigos científicos ou textos escritos livremente, podem ser encontrados em sites como *Wikipedia*[1], *Google Knol*[2] ou em alguns *Blogs*. São serviços que contribuem para a formação de bases de conhecimento na web. E são denominados de Sistemas de Conhecimento Coletivo e fazem parte da Web Colaborativa, que tem como um de seus aspectos principais, a possibilidade de criação e disseminação de conteúdo pelos usuários (HENDLER; GOLBECK, 2007).

Apesar dos sistemas de conhecimento coletivo serem bastante utilizados e úteis, não se pode garantir a veracidade, ou a qualidade, de suas informações. Como seu conteúdo está disponível para criação e edição por qualquer usuário, nem sempre são profissionais da área que disponibilizam informações sobre determinadas áreas do conhecimento. Além disso, não há como impedir que fontes fraudulentas ou não apropriadas disponibilizem dados equivocados na rede (GRUBER, 2007).

Os sistemas de conhecimento coletivo podem contribuir para a difusão do conhecimento também dentro das organizações. Já é comum empresas ou instituições fazerem uso de *wikis* coorporativos ou fóruns de discussão com o objetivo de tornar explícito parte do conhecimento tácito de seus colaboradores, facilitando a aprendizagem de novos colaboradores e mantendo seu conhecimento organizacional atualizado e robusto.

2.3 Armazenamento e Recuperação de Informação na Web

A estrutura da web nem sempre facilita a recuperação ou localização de um conteúdo disponibilizado nela, já que ficam armazenados geralmente em uma estrutura centralizada de hierarquia de pastas e subpastas. Para localizar um determinado conteúdo o usuário tem que saber o endereço do site e qual o caminho exato do conteúdo dentro do site, o que na maioria das vezes não é cômodo.

Alguns pesquisadores e colaboradores da rede mundial de computadores estão tentando facilitar o acesso a essa grande quantidade de informação que está disponível e descentralizada na web. Estudos como a Inteligência Artificial, a Web Semântica e Algoritmos de

[1] http://pt.wikipedia.org/ acessado em 13/11/2010 às 10:00.
[2] http://knol.google.com/ acessado em 13/11/2010 às 10:00.

Busca estão sendo desenvolvidos com o objetivo de aprimorar as técnicas de Recuperação de Informação na web (HATCHER; GOSPODNETIC, 2005).

Para Beppler et al. (2005), a Recuperação de Informação é a ciência da busca por informação em conteúdos diversos, como documentos digitais, em dados que descrevem documentos ou, ainda, dados armazenados em bases relacionais na Internet ou Intranet.

Os sistemas de Recuperação de Informação permitem armazenar, indexar e recuperar informação sobre uma base de conteúdo (SILVEIRA, 2003). A ideia central é armazenar toda informação relevante e recuperar somente a parcela de conteúdo desejado (BEPPLER, 2002). A extração precisa de conteúdo relevante depende da interpretação da informação. A grande quantidade de informações atualmente na web torna essa tarefa de recuperar informação relevante complexa, o que pode ser facilitada com a utilização de mecanismos de software que automatizem o processo.

Dentro desse contexto, surgiram os motores de busca para internet (ex.: *Google*, *Yahoo*), desenvolvidos para facilitar a localização de conteúdo em toda a web, assim como para intranet (e.g. XQOM proposto por Silvestre (2005)). Esses motores extraem os termos mais freqüentes dentro de um texto, denominados de palavras-chave, para indexar e buscar arquivos e páginas web, facilitando a vida de estudantes, pesquisadores e outros usuários em todo o mundo.

O processo de recuperação de informação é alimentado por uma *string* de busca (*query*), montadas pelos usuários para realizar as consultas. Essas *strings* de busca são compostas de termos que são comparados com os índices de documentos em uma ou mais bases de conteúdo com o intuito de identificar o grau de relevância e similaridade entre o termo e os índices de busca.

Para melhor entendimento da técnica de indexação de textos, imagina-se o seguinte exemplo. Quando se deseja ler um assunto específico abordado em um livro, torna-se mais fácil localizar a página onde o assunto é tratado no índice deste livro. O índice de um livro descreve sua estrutura organizacional que compreende tópicos e sub-tópicos que expressam os assuntos e temas abordados nesse livro. A indexação de arquivos é uma técnica bastante similar, que visa facilitar a localização de arquivos dentro de um arcevo muito grande. Seguindo o exemplo citado, um algoritmo de indexação de arquivos lê a estrutura sintática dos arquivos a serem indexados e extrai deles as palavras mais relevantes, seguindo algum critério pré-definido, para formar o índice de cada um deles (HATCHER; GOSPODNETIC, 2005).

Para localizar arquivos indexados, basta que os motores de busca leiam os índices dos arquivos e retornem os documentos que tiverem em seus índices as palavras-chave bus-

cadas, geralmente a ordenação desse resultado leva em consideração o número de vezes que as palavras-chave aparecem juntas no arquivo, quanto maior for sua ocorrência mais relevante será o arquivo.

Alguns modelos de recuperação de informação são baseadas em indexação de documentos. Souza (2006) define a seguir algumas dessas técnicas.

- **Modelo Booleano**: são recuperados apenas os documentos que possuem os termos especificados na *string* de busca do usuário. Podem ser utilizados na *string* de busca os operadores booleanos OR, AND e NOT para definir ocorrências específicas das palavras-chave, especificando os documentos a serem recuperados.
- **Modelo Vetorial**: os documentos são manipulados como "sacos de palavras"(*bags of words*), sendo representados como vetores no espaço *n-dimensional*, onde *n* é o total de termos (palavras) indexados de todos os documentos armazenados. Nesse modelo pode-se calcular um grau de similaridade entre os documentos que forem considerados relevantes (e.g. apartir de 5 palavras em comum), construindo um ranking de similaridade.
- **Modelo Probabilístico**: a busca inicial feita neste modelo utiliza técnicas de outros modelos, porém busca-se aproximar cada vez mais este conjunto de resultados através do feedback do usuário em sucessivas interações. A interação continua com usuário é considerada nesse modelo para refinar o resultado continuamente.

Um exemplo prático e interessante a ser citado é o algoritmo utilizado no mecanismo de busca da Google[3] o *PageRank*. Esse algoritmo foi criado pelos fundadores da empresa, Sergey Brian e Larry Page, durante o curso de doutorado que ambos fizeram na Universidade de Stanford nos Estados Unidos.

O processo de busca do Google consiste em indexar o maior número de páginas web possível e agrupa-lás em ordem de relevância. Essa relevância é determinada pelo algoritmo *PageRank*, que classifica como mais relevânte o site que tiver um número maior de *links*, em outros sites da web, apontando para ele. Além disso, quanto mais relevante for uma página web que possui um *link* apontado para outra página maior será a pontuação que essa segunda página receberá (FRIEDMAN, 2007). Desta forma, o *PageRank* tenta retornar em seus busca a importância que a web atribuiu a elas.

Apesar dos benefícios alcançados, a técnica de indexação de arquivos ainda não fornece a melhor solução para localização de conteúdo na web. Os resultados obtidos pela busca quase sempre trazem uma quantidade muito grande de dados, e que em sua maioria não

[3] http://www.google.com/

tem a informação desejada. Isso ocorre por causa da falta de semântica na busca, que é baseada em comparação sintática entre textos, já que esses motores de busca só entendem a sintaxe de uma palavra-chave. O retorno de uma busca pela palavra artigo, por exemplo, trará resultados ligados a artigos acadêmicos e a livros de português com definições e exemplos sobre artigos definidos e indefinidos. A Web Semântica é uma abordagem em desenvolvimento que tenta solucionar esse e outros problemas na estrutura da web.

2.4 Web Semântica

O número de pessoas utilizando a web esta aumentando, porém o conteúdo da maioria das páginas continua seguindo o contexto inicial de disponibilizar informação apenas para humanos. Por este motivo a maioria dos motores de busca, dificilmente retorna resultados satisfatórios, uma vez que não podem entender a informação, esses agentes lógicos fazem a indexação das páginas e seus conteúdos por meio de palavras-chave e a busca não leva em consideração o sentido, senão a sintaxe das palavras-chave (BREITMAN, 2005).

Figura 2.1: Busca pelos termos *Aluno Pedro*. Fonte: Autor.

Na Figura 2.1, temos uma busca exemplo feita na web onde um professor, hipotético, está

procurando por um ex-aluno seu. A busca foi realizada em uma das ferramentas de busca mais utilizadas na web, que é baseada em comparação sintática de textos, ou palavras.

Ao digitar os termos *Aluno Pedro* no motor de busca foram retornados inúmeros resultados com o termo *Aluno*, porém o termo *Discente* não teve nenhuma ocorrência. Os termos são sinônimos, porém como essas ferramentas realizam a busca de forma sintática, elas não conseguem encontrar a maioria dos termos sinônimos.

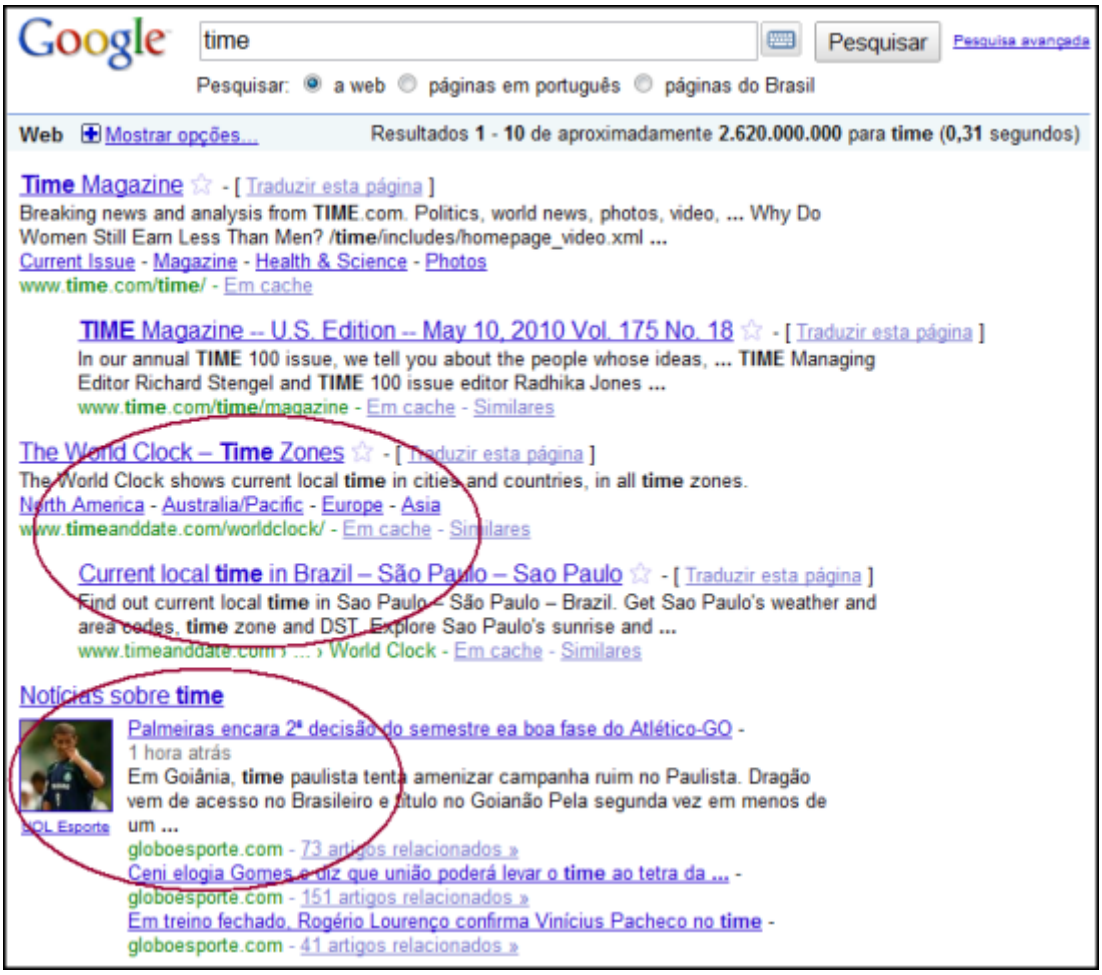


Figura 2.2: Busca pelo termo *Time*. Fonte: Autor.

Outra busca feita como exemplo foi a que utilizou o termo *Time*. Na Figura 2.2 é possível identificar dois gêneros diferentes para o mesmo termo, *Time* em inglês que significa tempo e *Time* de Futebol, encontrado em textos escritos em português.

Berners-Lee, Hendler e Lassila (2001) definiram a Web Semântica como a evolução da Web Colaborativa, onde o conteúdo publicado nela tem significado bem definido, permitindo que computadores e seres humanos entendam-no e trabalhem em cooperação. Na Figura 2.3 temos um panorama que ilustra a evolução da web desde o seu surgimento.

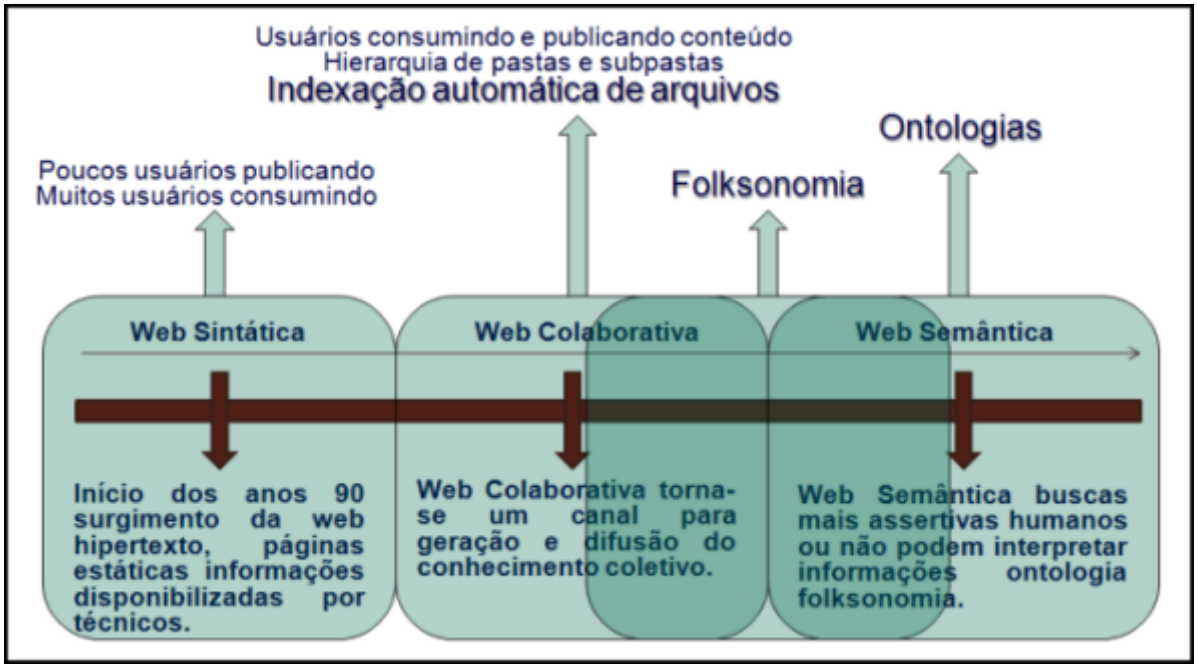


Figura 2.3: Panorama de Evolução da Web.

A ideia é que as páginas web, além do seu conteúdo, possuam uma camada de metadados que relacionem as informações em um determinado domínio. Metadados são informações sobre um determinado conteúdo vinculadas ao mesmo, com o intuito de classificá-lo, descrevê-lo ou localizá-lo e não precisam ser necessariamente digitais (BREITMAN, 2005). Esta abordagem seria capaz de solucionar os problemas citados acima, melhorando a eficiência e a eficácia dos motores de busca.

Outra vantagem proporcionada pela web semântica será viabilizar a execução de tarefas na web, como agendamentos ou pesquisas, atualmente feitas apenas pelos seres humanos, por intermédio de agentes lógicos. Esses agentes serão capazes de identificar a utilidade do *site*, podendo classificá-lo como uma loja de roupas, por exemplo, e extrair informações sobre o site, como endereço ou razão social, caso seja o *site* de uma empresa. Desse modo, um usuário poderia utilizar um desses agentes lógicos para encontrar todos os *websites* de lojas de roupas que tenham lojas físicas localizadas em um determinado bairro de sua cidade (BERNERS-LEE; HENDLER; LASSILA, 2001).

Berners-Lee, em uma conferência sobre XML (*eXtensible Markup Language*) em 2000, propos também uma arquitetura para a Web Semântica em sete camadas (BREITMAN, 2005). Essas camadas são ilustradas na Figura 2.4 e definidas abaixo:

- Camada 1: A base da hierarquia é composta pelos componentes **URI** (Universal Resource Identifier) e **UNICODE**. O URI é um padrão para identificar de forma única recursos ou objetos na web. O padrão Unicode refere-se à forma de como

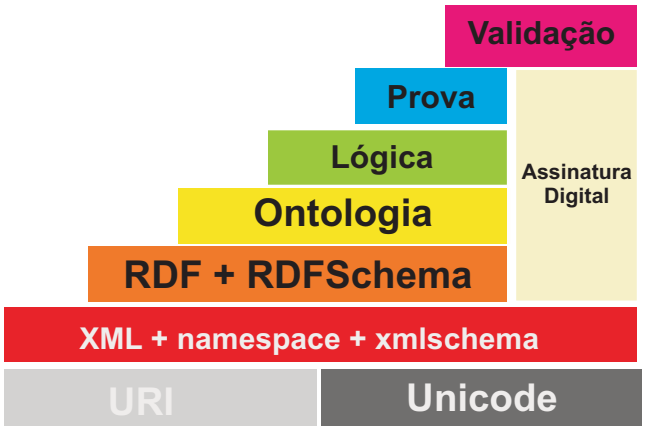


Figura 2.4: Arquitetura da Web Semântica. Fonte: Breitman (2005).

representar textos em várias línguas.

- Camada 2: É composta por **XML**, **namespace** e **xmlschema**. XML é uma linguagem de marcação para estruturar documentos. Namespace é uma coleção de nomes, identificados unicamente por uma URI. XML Schema é uma linguagem que descreve a estrutura de um documento XML, possibilitando a definição dos tipos de dados, regras de agrupamento e restrições;

- Camda 3: É a base da ontologia e é composta por **RDF** (Resource Description Framework) e **RDF Schema**. RDF é uma linguagem constituída para descrever semanticamente um domínio, possuindo três elementos básicos: recursos, propriedades e declarações. Já o **RDF Schema** (RDFS) é uma evolução do RDF, permitindo a utilização de conceitos como Classes e fornecendo um vocabulário básico de RDF para criar relações mais poderosas;

- Camada 4: É a **Ontologia** que estende o RDFS adicionando mecanismos mais avançados para descrever um domínio semântico;

- As camadas 5, 6 e 7 são de **Lógica, Prova e Validação** e têm como objetivo obter derivações usando provas lógicas no processo dedutivo, como por exemplo: "se um homem tem algum filho, logo ele é pai".

2.5 Anotações Semânticas

Seguindo a ideia da Web Semântica algumas técnicas foram desenvolvidas para aumentar a semântica dos dados na web. As anotações semânticas surgiram para estruturar textos em páginas *web*, relacionando formalmente termos desse texto com os elementos de uma Ontologia (FONTES et al., 2010).

Essas anotações seguem a estrutura de uma linguagem RDF para representar informações associadas a recursos. Recursos são objetos como arquivos e serviços presentes na web que possuem um URI. Uma das utilidades do RDF é a representação de metadados propiciando integração entre sistemas, catalogando conteúdo e compartilhando conhecimento. Sites como o *eBay* e *Amazon*, fazem uso do RDF em sua estrutura de pesquisa (MANOLA; MILLER, 2004).

A representação de informações em RDF se baseia em uma tripla (estrutura com três elementos): recurso, propriedade e valor da propriedade. A Figura 2.5 exibe um exemplo dessa estrutura, onde o recurso é o endereço web *http://www.w3c.org*, a propriedade é representada por *data-catalogação* e o valor da propriedade é representado pela data *12/10/2005*. O que significa que o site *http://www.w3c.org* foi catalogado no dia 12/10/2005.

Com base nessas triplas, o modelo RDF é capaz de expressar o significado das informações, da mesma forma que a relação entre sujeito, verbo e predicado de uma frase.

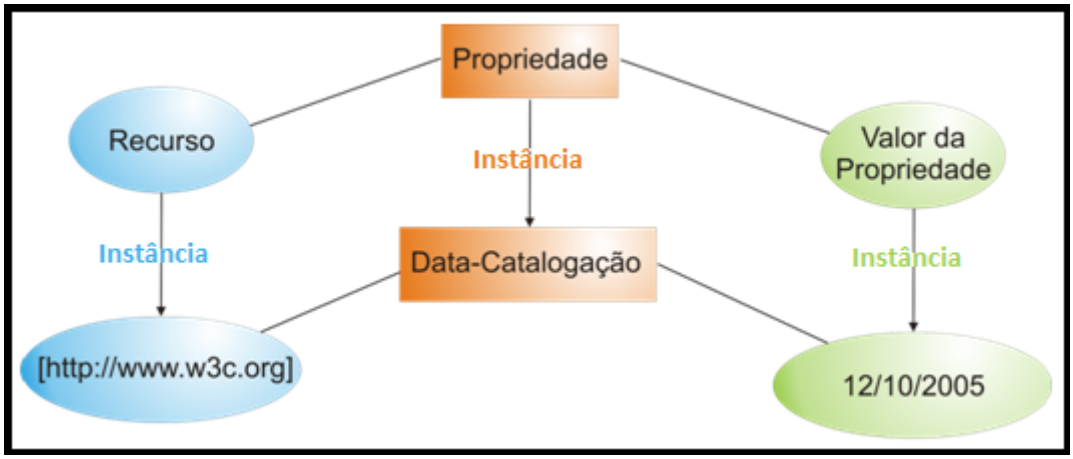


Figura 2.5: Exemplo da estrutura de um RDF.

O RDF é um dos pilares, como mecanismo de compartilhamento e interpretação de dados, para linguagens que constroem e manipulam ontologias. O conceito Ontologia e sua linguagem padrão são definidos e detalhados no Capítulo 3.

Em uma anotação semântica, o recurso pode ser um conceito da Ontologia assim como o predicado pode ser uma propriedade da mesma. O objeto, ou valor da propriedade, é o texto anotado. No Capítulo 3 esses e outros elementos de uma Ontologia são definidos.

Um dos padrões utilizados para implementar uma anotação semântica é o RDFa (RDF Anotation), com ele pode-se adicionar atributos aos marcadores HTML (*Hyper Text Multi Language*) para anotar os termos dentro de uma págnia *web*. Na Figura 2.6 temos um exemplo de texto anotado em RDFa.

```
1<div typeof="Aluno" about="#Celso">
2    Olá, meu nome é <span propert="temNome">Celso Fontes</span>
3</div>
```

Figura 2.6: Exemplo de texto anotado em RDFa. Fonte: Fontes et al. (2010)

Usando essas anotações, agentes de software poderiam extrair mais semântica das páginas web, ou realizar buscas através do vínculo estabelecido entre os termos anotados. Alguns sites na web, como o Twitter[4] e o Wikipédia[5], já estruturam seus dados com anotação semântica. Porém esta não é a única iniciativa da Web Semântica. Uma técnica mais conhecida é utilizada em alguns serviço na web para auxiliar a recuperação de conteúdo, através do uso de metadados, essa técnica é denominada Folksonomia.

2.6 Folksonomia

Sites de compartilhamento de conteúdo, como *Flickr*[6] e *Del.icio.us*[7], e *webmails*, como o *Gmail*, permitem que seus usuários aumentem a semântica de seus conteúdos com as tags (ou rótulos). O usuário atribui uma ou mais tags a um conteúdo específico, que são anexadas em um metadado. Para recuperar esse conteúdo, um mecanismo baseado em busca por essas tags é utilizado, essa técnica é denominada Folksonomia (WAL, 2007).

A palavra Folksonomia tem origem nas palavras *folk* e *taxonomia*. Onde *folk* significa povo e a *taxonomia* é o estudo da classificação das coisas. A ideia da Folksonomia é que o usuário classifique seus dados de forma livre, viabilizando uma melhor eficiência no armazenamento e na recuperação desses dados ante a ação de motores de busca (WAL, 2007).

Além dos serviços web o sistema operacional *Windows Vista* e os aplicativos que compõem o *Office 2007*, conjuntamente, fornecem uma estrutura baseada em Folksonomia como uma das formas de armazenamento e recuperação de dados. Conforme pode ser visto na Figura 2.7, os aplicativos do $Office$ (e.g. $Word$) permitem que os usuários atribuam uma ou mais tags (denominadas no aplicativo de *marcas*) aos arquivos. A estrutura de recuperação é fornecida pelo sistema operacional *Windows Vista*, que tem entre suas opções de busca as *marcas* atribuídas aos arquivos pelos aplicativos do $Office$, conforme visto na Figura 2.8.

No contexto da Folksonomia, as tags são atribuídas pelo usuário, que tem total liberdade

[4] http://twitter.com/
[5] http://www.wikipedia.org.br/
[6] http://www.flickr.com/
[7] http://www.delicious.com/

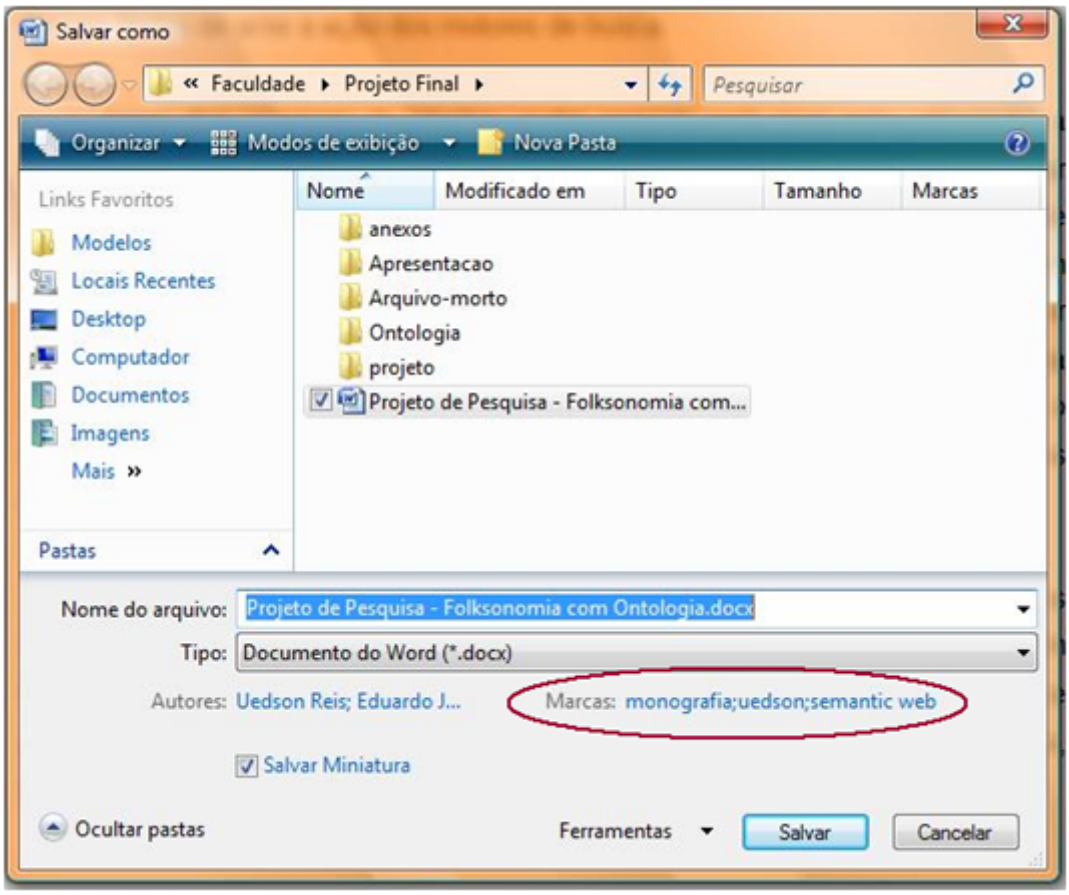


Figura 2.7: Tela do Microsoft Word 2007.

no momento de classificar seu conteúdo. Entretanto, além dos erros de sintaxe, cada pessoa classifica seus conteúdos de uma maneira própria, seguindo conceitos que para ela têm sentido ou significado. Isso facilita a recuperação desse conteúdo por parte da pessoa que o classificou. Porém a mesma facilidade não será encontrada se outra pessoa, que não participou da classificação, tentar recuperar esse conteúdo. Esse ponto pode ser considerado uma desvantagem da Folksonomia quando utilizada sozinha para o armazenamento e recuperação de conteúdo. Além disso, a Folksonomia possui uma desvantagem tecnologica, a quantidade de registros armazenados pode aumentar muito com o passar do tem. Por esse motivo, recomenda-se uma manutenção periódica (e.g. a exclusão das tags que não estiverem sendo utilizadas) no banco de dados ou no sistema de armazento utilizado.

Um aspecto que pode ser considerado positivo na Folksonomia é a capacidade de cruzamento das próprias tags, uma vez que isto facilita a extração de informações armazenadas implicitamente nos dados rotulados. Um exemplo bem comum é quando um documento recebe a mesma tag várias vezes de pessoas diferentes, isso na maioria dos casos pode ser considerada uma classificação conceitualmente apropriada, uma vez que o conteúdo tem a mesma representatividade para vários outros usuários. Da mesma forma, quando uma tag é atribuída a um documento por apenas dois ou três usuários isso provavelmente será

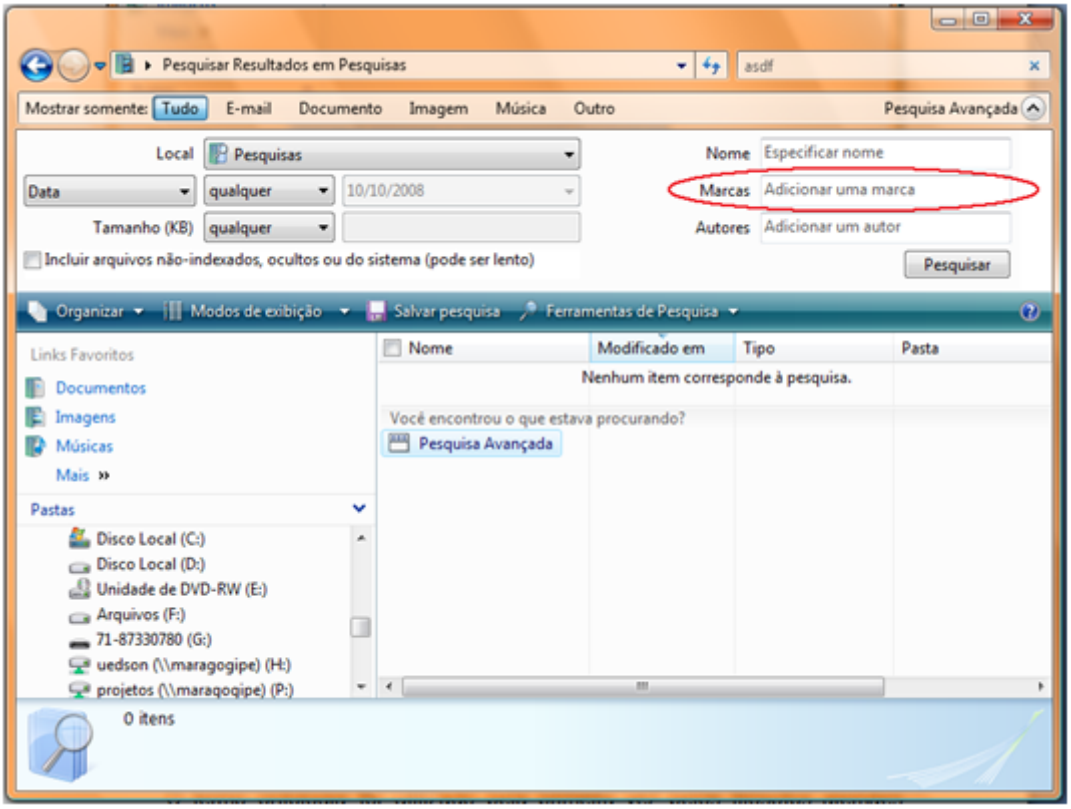


Figura 2.8: Tela do Mecanismo de busca do Windows Vista

considerada uma classificação específica dessas pessoas.

A Ontologia presente na arquitetura da Web Semântica pode aprimorar a técnica da Folksonomia, como discutido por Reis et al. (2009). No próximo capítulo o termo Ontologia será definido e seus elementos serão detalhados. A linguagem para construção e manutenção de uma Ontologia será apresentada, assim como exemplos de aplicabilidade da mesma.

Capítulo Três

Sistemas de Representação do Conhecimento

Este capítulo define a Representação do Conhecimento e mostra alguns de seus exemplos. A Ontologia é apresentada como um sistema de representação do conhecimento que viabiliza a Web Semântica. A linguagem OWL padrão para construção e manutenção de Ontologias na web também é apresentada. Por fim, um estudo sobre Metafóras Visuais com destaque para a Nuvem de Tags, bastante utilizada na web.

3.1 Representação do Conhecimento

O conhecimento é uma informação valiosa da mente humana. Ele inclui reflexão, síntese, contexto e inferência (DAVENPORT, 1998). Inferência é um método baseado na Semiótica que indica a adoção de uma crença como consequência de um outro conhecimento (PEIRCE, 1975). Quando se tem conhecimento de um determinado assunto pode-se realizar inferências para extrair informação e realimentar o próprio conhecimento.

Por ser frequentemente tácito a um humano, o conhecimento é de díficil transferência e captura em máquinas (DAVENPORT, 1998). Para ser utilizado computacionalmente o conhecimento precisa ser representado e estruturado. Jonassen, Beissner e Yacci (1993), apresentam três classificações para o conhecimento, são elas:

- o **conhecimento declarativo** representa as diretrizes, estruturas ou jurisdições de um determinado objeto, evento ou ideia. É um conhecimento do tipo *sabendo que* por exemplo, é quando uma pessoa sabe que ela pode descrever o seu conhecimento, mas não necessariamente pode fazer uso deste conhecimento;
- o **conhecimento processual** é a aplicação do conhecimento declarativo, exemplificando seria o *saber como*. Conhecimento processual implica a utilização do conhecimento declarativo, utilizando sua estrutura ou suas diretrizes, para executar uma ação (e.g. realizar uma tarefa, resolver um problema ou elaborar um texto);
- o **conhecimento estrutural** representa as relações entre conceitos internos de um domínio de conhecimento declarativo. Ele torna explícita as inter-relações do conhecimento declarativo a fim de permitir sua manipulação pelo conhecimento processual. Este tipo de conhecimento seria o *saber por que*.

Na Web Semântica, o conhecimento declarativo e o estrutural podem ser a semântica

que falta aos dados. Com o conhecimento declarativo os conceitos de um determinado domínio podem se tornar explicitos para uso na web, como as tags da folksonomia por exemplo, classificando o conteúdo. O conhecimento estrutural pode definir e compartilhar as relações entre esses conceitos, que são processadas pelo conhecimento processual. Desta forma, o conhecimento processual desse domínio pode ser implementado em motores de busca, que poderiam melhorar a eficiência das buscas, ou em agentes lógicos de software que poderiam realizar tarefas consideradas complexas na web, como marcação de consulta médica ou elaboração de um roteiro de visitas de um vendedor aos seus clientes.

Um tipo de conhecimento estrutural que pode ser utilizado na Web Semântica é a Rede Semântica. Esta divide o conhecimento em partes, definidas como conceitos, e interliga estes conceitos, formando uma rede. Cada conceito é um ponto, ou vértice, que se conecta com outro conceito através de uma associação ou ligação semântica (aresta), a fim de expressar uma ideia ou um domínio de conhecimento (JONASSEN; BEISSNER; YACCI, 1993). Na Figura 3.1 temos um exemplo de Rede Semântica.

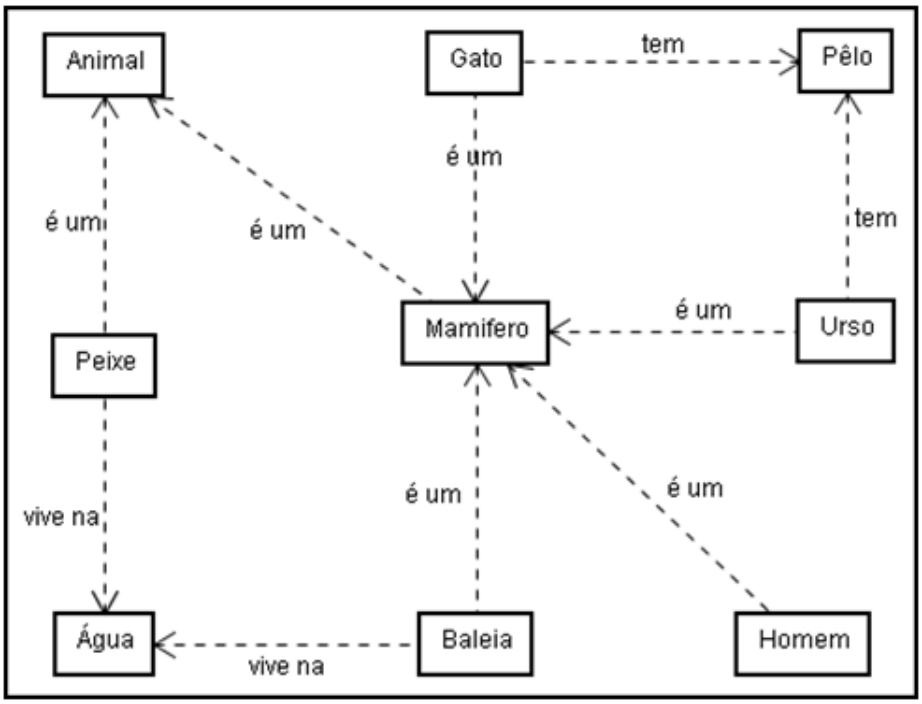


Figura 3.1: Exemplo de uma Rede Semântica adaptado do site Wikipédia.

Essa estrutura de conceitos e ligações da rede semântica pode ser encontrada em uma ontologia, a diferença é que as relações entre os conceitos de uma ontologia possuem algumas restrições (características) o que permite a sua estrutura um maior poder de representação do que o encontrado em uma rede semântica. Por exemplo, em uma ontologia pode-se definir que uma relação só poderá associar um conceito a 5 outros conceitos (e.g. uma mão possui cinco dedos). Essas restrições serão explicadas detalhadamente na Seção 3.4.

3.2 Ontologia

O termo Ontologia foi cunhado pelos filósofos alemães Rudolf Goclenius e Jacob Lorhard, para designar o estudo do *ser enquanto ser* no ramo da filosofia, entre os séculos XVII e XVIII, apesar deste estudo ter sido feito por Aristóteles em seus estudos sobre Metafísica. A palavra vem do grego onde *ontos* significa *ser* e *logos* significa *palavra* (RA; GA; RA, 2004).

Várias definições para o termo ontologia podem ser apresentadas a depender do contexto utilizado. Essas definições podem ser distintas ou complementares entre si, para um mesmo conceito ou domínio, e o seu significado tende a variar de acordo com o objetivo de seu uso (BREITMAN, 2005).

O uso do termo Ontologia foi incorporado na Ciência da Computação objetivando o estudo de mecanismos para organização da informação. Em 1991, o termo Ontologia foi proposto pelo grupo de pesquisa *DARPA Knowledge Sharing Effort*, que apresentou a seguinte definição: "Uma Ontologia define os termos básicos e as relações que compreendem um vocabulário de um domínio, bem como as regras para combinar termos e as relações para definir extensões do vocabulário".

Na ciência da computação o termo ontologia começou a ser utilizado na área da Inteligência Artificial, em projetos para formação de grandes bases de conhecimento (RA; GA; RA, 2004). Dentro do contexto da Web Semântica, Ontologia foi definida por Thomas Gruber (1993) como uma estrutura de conceitos e seus relacionamentos que especifica a conceitualização compartilhada de um determinado domínio em âmbito semântico. Em outras palavras, uma Ontologia define formalmente conceitos e restrições de um determinado domínio, através de uma estrutura de relacionamentos. Isso permite que a ontologia seja processada pela máquina e também entendida pelos seres humanos (BREITMAN, 2005).

Apesar do uso de Ontologias em sistemas computacionais ser um campo emergente, algumas organizações estão percebendo a importância de se desenvolver projetos envolvendo este conceito. Um deles é o Gene Ontology[1], projeto de bioinformática de amplitude mundial que objetiva uniformizar as descrições sobre o gene, garantindo que vários grupos de pesquisa espalhados pelo mundo trabalhem sobre o mesmo vocabulário. A Petrobrás também desenvolve um projeto com a finalidade de padronizar um vocabulário para os seus grupos de pesquisa. Com isso, se torna possível que esses grupos trabalhem em cooperação na construção de uma base de informações sobre arenitos e carbonatos, para ser utilizada na avaliação da qualidade de reservatórios de petróleo (ONTOBRAS, 2009).

[1] *www.geneontology.com*, acessado em 06/10/2010

Outra iniciativa importante é a das organizações Globo, com um projeto que visa estruturar o seu portal através do uso de Ontologia. A ideia é utilizar uma Ontologia que trate de assuntos gerais, como pessoas ou localidades, e mais quatro ou cinco Ontologias especializadas, abordando domínios como Esporte, Política ou Economia. Através dessas Ontologias, as buscas feitas no portal *Globo.com* poderão ser mais específicas (ONTOBRAS, 2009).

Utilizando o poder das relações entre conceitos de uma Ontologia, o portal *Globo.com* poderia realizar uma busca por todos os gols perdidos por um determinado jogador retornando um resultado bem satisfatório, ao invés de realizar uma busca sintática, como feita normalmente pelos motores de busca, com os termos *gols perdidos* de *Paulo*, onde o resultado, certamente, conteria também alguns gols feitos por *Paulo*. Para isso seria necessária a utilização de uma Ontologia de Futebol que contenha a propriedade *gols perdidos*, vinculando o conceito *Gol* ao conceito *Jogador*. Mesmo a solução de associar *tags* (palavras-chave) às informações do site é limitada neste caso, pois as *tags* não se relacionam. Isto é, mesmo que as *tags* *gols*, *perdidos* e *Paulo* estejam vinculadas ao vídeo, a busca seria feita sem considerar uma relação entre elas.

Esta dissertação de mestrado foi desenvolvida com o auxílio de um grupo de pesquisa, o qual este autor participa. Este grupo publicou na Revista Java Magazine (JORGE et al., 2010), edição 85, um artigo sobre como desenvolver Ontologias utilizando linguagem de programação Java. O artigo tem como objetivo auxiliar desenvolvedores de software que queiram trabalhar com ontologias na linaguagem Java.

O estudo sobre ontologia, nesta dissertação, se inicia na sua classificação de acordo ao seu grau de especialização, Manhães, Santos e Farias (2006) o fizeram da seguinte forma:

- **Ontologia de Alto Nível** é a ontologia generalista, dentre as quatro classificadas, descrevendo conceitos gerais como tempo, matéria, objeto, ação, entre outros. Por isso não dependem de um domínio específico de conhecimento;
- **Ontologia de Domínio** descrevem os conceitos de um determinado domínio do conhecimento, como odontologia, análise de sistemas, técnicas de vendas, entre outras. Esta naturalmente herda as especialidades da ontologia de alto nível;
- **Ontologia de Tarefa** descreve os conceitos de uma determinada tarefa ou serviço, como manutenção, diagnósticos e etc, também herda os termos da ontologia de Alto Nível;
- **Ontologia de Aplicação** é a ontologia especialista, dentre as quatro classificadas. Descreve conceitos de um domínio específico juntamente com os de uma tarefa específica, herdando os termos e definições da ontologia de domínio e da de tarefas.

Uma Ontologia de Domínio, escrita em uma linguagem formal, possui os elementos descritos abaixo (BREITMAN, 2005).

- **Classes (conceitos)**: definem as subáreas ou subgrupos de um domínio de interesse, como *Pessoa*, *Veículo* ou *Médico*;
- **Propriedades (Relacionamentos)**: são formas de interligar os conceitos. Por exemplo, um possível relacionamento entre os conceitos *criação* e *criador* poder ser definido como *criado_por* outro exemplo é o de interligar um conceito a um atributo (tipo de dados) como o conceito *pessoa* e o atributo texto *endereço* que pode ser definido como *mora_em*;
- **Restrições**: são características, ou regras, definidas logicamente para restringir a inserção de futuros indivíduos e as formas de relacionamento entre eles, e possibilitar a descoberta de novas classificações desses indivíduos a partir de inferências (este tópico será mais detalhado na seção 3.4);
- **Indivíduos (instâncias)**: é a unidade materializada de uma classe, como João da Silva que é uma pessoa única, ou um carro específico que possui uma placa e um chassi, identificando-o unicamente;
- **Axiomas**: determinam verdades sobre um determinado domínio. Como por exemplo, todo *pai* tem no mínimo um $filho$.

Ontologias de domínio podem ser utilizadas como metadados para viabilizar a Web Semântica, sendo anexada, ou alguns de seus elementos, ao conteúdo pelos usuários. Desta forma, pode-se atribui semântica ao dado, permitindo a manipulação e o compartilhamento dessas informações na web. Isso pode ser feito, utilizando os conceitos (classes) de uma ontologia como tags para classificar conteúdo, como feito na folksonomia, neutralizanado possíveis erros de digitação ou ortográficos do usuário, além de aumentar o acerto das buscas.

Como uma Ontologia é compartilhável e usa termos de um determinado domínio de conhecimento, um usuário utilizará termos comuns à todos os outros usuários que utilizam a mesma Ontologia entretanto a classificação de conteúdo que utiliza apenas *tags* oriundas de uma Ontologia pode se tornar incompleta, além de não atribuir sentido particular para um usuário. Isso porque a Ontologia tem uma quantidade limitada de termos, o que restringe a abrangência na escolha de *tags* por parte de um usuário (REIS et al., 2009).

A utilização das tags oriundas de uma Ontologia, sem a restrição de outras tags inseridas livremente pelo usuário, minimiza os pontos negativos apresentados pelo uso isolado tanto de tags livres quanto das tags restritas a Ontologia. Ademais, realça os pontos

positivos de ambas as técnicas. O usuário pode utilizar a tag da ontologia para atribuir um conceito mais formal ao seu conteúdo e uma tag digitada por ele mesmo para atribuir uma semântica própria (REIS et al., 2009). Por esses motivos, essa forma de utilização da ontologia também é sugerida e implementada no MOFI.

Para viabilizar o anexo de uma Ontologia em arquivos na web ou em outros ambientes computacionais, faz se necessária a sua representação física, que pode ser feita em uma linguagem de marcação estruturada como o XML (*eXtensibleMarkupLanguage*), por exemplo. O W3C[2], recomenda a utilização da linguagem *OWL* para o desenvolvimento de Ontologias (LIMA; CARVALHO, 2005).

3.3 OWL - Web Ontology Languege

A Linguagem de Ontologia para Web (*OWL - Web Ontology Language*) é a linguagem padrão recomendada pela W3C para construir e processar ontologias no ambiente web. Ela é baseada em *RDF* (*Resourse Description Framework*) e *RDF Schema*, e utiliza sintaxe *XML*. A *OWL* pode viabilizar o processamento do conteúdo e não apenas disponibilizar a visualização do mesmo (LIMA; CARVALHO, 2005).

A vantagem da *OWL* é que ela não é apenas uma linguagem para definir formatos de mensagem ou especificação de protocolos. Ela consegue representar conhecimento a partir dos vínculos e relacionamentos existentes entre os conceitos da ontologia e da semântica formal *OWL* que especifica fatos lógicos não presentes explicitamente em uma Ontologia, mas que são extraídos de seus indivíduos por inferência (SMITH; WELTY; MCGUINNESS, 2004). Por exemplo, para identificação de indivíduos, que podem ser classificados como *Pai* seriam verificados os classificados como *Homem* e que possuissem uma relação *ter_filho* com algum outro indivíduo.

Smith, Welty e McGuinness (2004) define três sub-linguagens, da *OWL*, para grupos específicos de usuários e desenvolvedores de Ontologias. Elas são classificadas da seguinte maneira:

- *OWL Lite* fornece uma simples forma de construção para ontologias *OWL*. Por exemplo, as restrições de cardinalidades, que na OWL Lite só pode se utilizar 0 ou 1, não podendo relacionar um indivíduo a dois ou três outros indivíduos. Ela é recomendada para usuários iniciantes que utilizarão apenas algumas características da linguagem;

[2] *World Wide Web Consortium* - organização que padroniza as tecnologias da web

- *OWL DL* fornece todas as construções da OWL, porém tem limitações, uma classe não pode ser um indivíduo, ou um indivíduo não pode ser uma propriedade;
- *OWL Full* também fornece todas as construções da OWL, porém sem restrição quanto a uma classe ser um indivíduo ou uma propriedade.

3.4 Restrições na Linguagem OWL

A arquitetura da Web Semântica, apresentada na Figura 2.4, possui uma camada que representa a linguagem OWL, denominada Ontologia. Esta camada, como já vimos, é estruturada e montada a partir das linguagens RDF e RDF Schema. A OWL se diferencia dessas linguagens por possuir elementos com maior poder de representação semântica, como as restrições de propriedades ou de classes.

Uma das restrições da linguagem OWL são os grupos de propriedades nativas para o relacionamento entre classes (Restrições de Classes) que estão definidas abaixo:

- **Herança**: relação entre uma classe generalista e outra classe especialista, por exemplo, se a classe *Pediatra* é uma especialização da classe *Médico*, então todo *Pediatra* é um *Médico*, porém nem todo *Médico* é um *Pediatra* ($Pediatra \subset Medico$);
- **Equivalência**: relação que indica igualdade entre duas classes, por exemplo, se a classe *Aluno* equivale a classe *Discente*, então todo *Aluno* é um *Discente*, e todo *Discente* é um *Aluno* ($Discente \subset Aluno \nleftarrow Aluno \subset Discente$);
- **Disjunção**: relação entre classes que indica que estas não podem possuir indivíduos em comum, por exemplo, se a classe $Homem$ é disjunta da classe $Macaco$, então nenhum $Homem$ pode ser um $Macado$, e nenhum $Macaco$ pode ser um $Homem$ ($Macaco \cap Homem = \emptyset$).

Outro grupo de Restrições é o das Propriedades. As propriedades são como ligações de um conceito a outro conceito, ou a um atributo (tipo de dados). O primeiro lado da propriedade, aquele que é lido inicialmente, é chamado *domínio*, e o segundo, o qual é lido após a propriedade, é denominado *range*. Juntos, *domínio* e *range* indicam o sentido semântico da relação, ou como esta deve ser lida, e limitam quais os tipos de Classe ou *Datatype* podem ser relacionados por esta propriedade. Por exemplo, na relação ilustrada pela Figura 3.2 Jogador *joga_no* Time, Jogador é a classe *domínio* da prorpiedade *joga_no*, e Time é a classe *range*. As Restrições de Propriedade, descritas abaixo, são baseadas no *domínio* e no *range* das propriedades.

Figura 3.2: Exemplo ilustrativo de Domínio e *Range*.

- **Funcional**: Sendo P uma propriedade funcional, e x, y e z indivíduos de um dado domínio, se *P(x,y)* e *P(x,z)* então $y = z$. Isso indica que a propriedade só pode ter 1 (um) indivíduo associado ao seu *range*, por exemplo, se a propriedade *treinado_por* é dita funcional, então um time só é *treinado_por* um técnico;

- **Inversa**: Sendo PI a propriedade inversa de P, e X e Y classes de um dado domínio, se *P(X, Y)* então *PI(Y,X)*. Isso indica que uma propriedade tem, associada a ela, uma propriedade inversa, onde a Classe definida no *domínio* de uma é o *range* da outra e vice versa, por exemplo, se a propriedade *tem* for dita como inversa da propriedade *pertence_a*, então se uma Pessoa *tem* uma Coisa logo uma Coisa *pertence_a* uma Pessoa;

- **Inversa Funcional**: Sendo P inversa funcional, e x, y e z indivíduos de um dado domínio, se *P(x,y)* e *P(z,y)* então $x = z$. Isso indica que a propriedade só pode ter 1 indivíduo associado ao seu *domínio* e que sua propriedade inversa (se houver) é funcional, por exemplo, sendo *treina* propriedade inversa de *ser_treinado*, e dita como inversa funcional, então um time só pode *ser_treinado* por um técnico;

- **Simétrica**: Sendo P uma propriedade simétrica, e x e y indivíduos de um dado domínio, se *P(x,y)* então *P(y,x)*. Indica que a inversa de uma propriedade é ela mesma, por exemplo, se a propriedade *faz_fronteira_com* é dita simétrica, implica que se Bahia *faz_fronteira_com* Minas Gerais então Minas Gerais também *faz_fronteira_com* Bahia;

- **Transitiva**: sendo P uma propriedade transitiva, e x, y e z indivíduos de um dado domínio, se $P(x, y)$ e $P(y, z)$ então $P(x, z)$, esta propriedade pode ser utilizada para modelar o seguinte exemplo, se o país *Brasil tem* o estado *Bahia* e a *Bahia tem* a cidade *Salvador*, logo o país *Brasil tem* a cidade *Salvador*.

Para ser processada pela máquina uma Ontologia precisar ser escrita em uma linguagem formal como a OWL. Porém, quanto mais formal uma linguagem for, mais difícil torna-se o entendimento para um ser humano. Por este motivo, o modelo, proposto nesta pesquisa,

sugere a conversão de uma Ontologia em uma Metáfora Visual para melhor interação do usuário. No próximo capítulo é explicado o termo "Metáfora Visual"e exemplificado o uso de Nuvem de Tags.

3.5 Metáfora Visual

Uma metáfora visual é utilizada para apresentar informações a partir de dados brutos. Em geral, os seres humanos têm facilidade em reconhecer padrões (ALMEIDA, 2003). Esta característica torna-se mais evidente quando a percepção se dá pela visão. Da mesma forma que a identificação e o entendimento, por parte dos seres humanos, de informações mostradas graficamente como metáforas visuais, geralmente, ocorrem mais facilmente do que quando essas mesmas informações são mostradas em modo de texto ou em simples tabelas (ALMEIDA, 2003).

Um ser humano não teria dificuldades em solucionar uma conta de multiplicação se esta for entre dois números relativamente pequenos (e.g. 9 x 3). Porém, se aumentarmos os valores consideravelmente (e.g. 15.936 x 7.834) a dificuldade aumentará quase que proporcionalmente. Isso ocorre porque os seres humanos têm um poder restrito de memorização. Tendo dificuldade em memorizar partes de um resultado para posterior utilização. Enquanto que se lhe for permitido o uso de um lápis e um papel para solucionar a segunda conta esta será feita com mais facilidade. Da mesma forma, o cérebro humano utiliza sua grande capacidade de identificar padrões visuais para extrair a informação mais rápido (ALMEIDA, 2003).

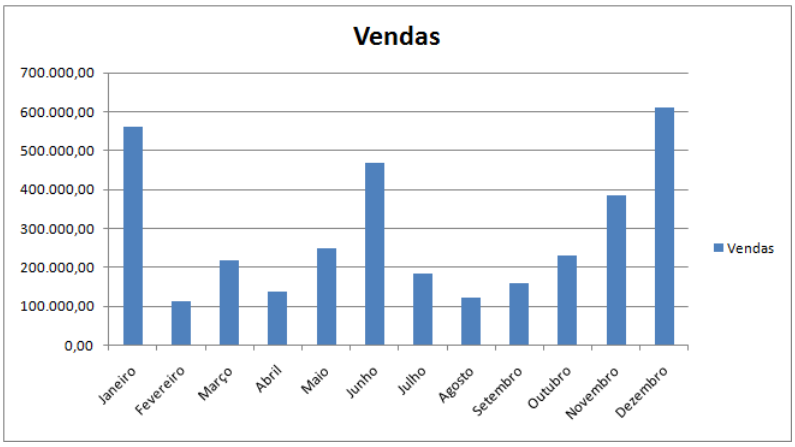
As figuras abaixo, ilustram um exemplo onde os valores das vendas de cada mês são disponibilizados em forma de tabela (Figura 3.3(a)) e de gráfico (Figura 3.3(b)). A identificação visual dos meses do ano que tiveram um volume maior nas vendas será mais fácil, para a maioria das pessoas, observando o gráfico da Figura 3.3(b).

Entender uma informação mostrada graficamente, como na Figura 3.3(b), geralmente, é mais simples do que entender a mesma informação mostrada de forma mais simples, como na Figura 3.3(a).

Neste exemplo, os demonstrativos de volume de vendas registradas anualmente, quando apresentados graficamente, mostram sua evolução ao longo dos meses de forma mais explícita do que quando apresentados em uma tabela com seus dados brutos. Geralmente, no gráfico o entendimento é mais rápido, do que na tabela, que nos períodos de fim de ano e no meio do ano o volume de vendas foi maior que no restante do ano.

Período	Volume de Vendas
Janeiro	560.000,00
Fevereiro	113.000,00
Março	218.000,00
Abril	137.000,00
Maio	249.000,00
Junho	470.000,00
Julho	183.000,00
Agosto	122.000,00
Setembro	159.000,00
Outubro	231.000,00
Novembro	385.000,00
Dezembro	610.000,00

(a) Volume de vendas registrados em 2010 de uma empresa hipotética.



(b) Gráfico dos dados mostrados na tabela da Figura 3.3(a).

Figura 3.3: Comparação de dados representados em uma Tabela e em um Gráfico.

Outro exemplo interessante pode ser extraído da ferramenta web Google Agenda[3]. Nessa ferramenta o usuário pode registrar vários eventos do seu dia a dia, como uma agenda digital, e ainda acompanhar em modos de visualização diferentes. Dentre elas temos a forma mais simples mostrada na Figura 3.4, e outras visões utilizando a metáfora visual de um calendário, como na Figura 3.5 e na Figura 3.6.

Figura 3.4: Eventos registrados no calendário do Google Agenda.

Nas visões de mês e semana (Figuras 3.5 e 3.6) identificam-se mais facilmente os eventos registrados para aquele período da agenda do que na amostragem mais simples da Figura 3.4.

[3]http://www.google.com/calendar/

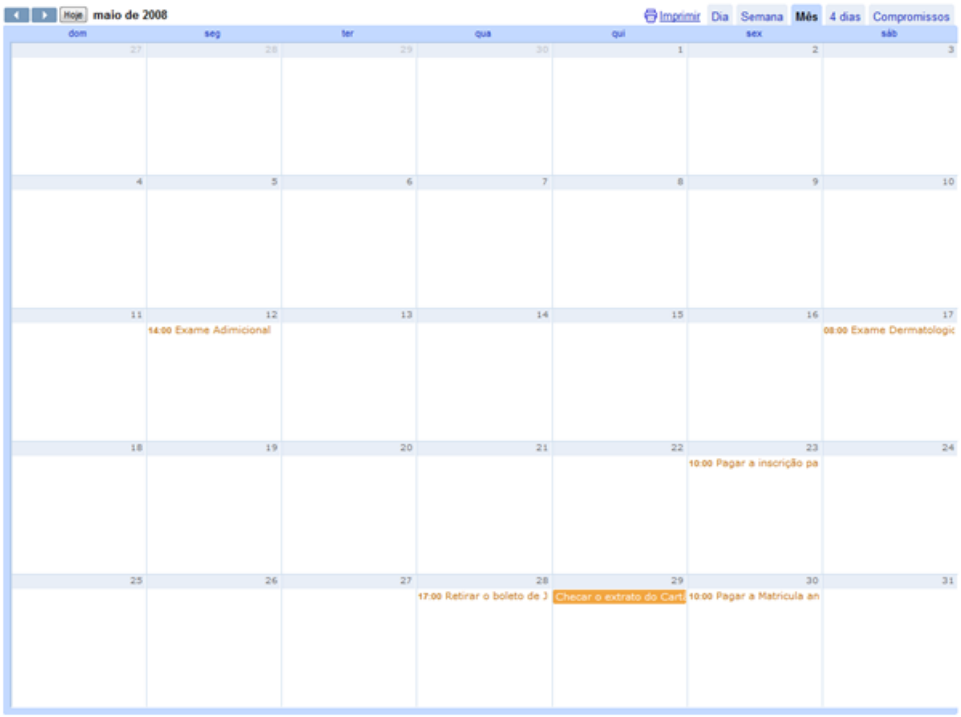


Figura 3.5: Eventos de calendário mostrados numa visão de mês.

Na Figura 3.5, que mostra a visão de mês, ainda identifica-se mais facilmente o evento do dia 29 de maio, pois este aparece em destaque, com uma cor de fundo acentuada. Isso ocorre porque este evento está registrado para o dia inteiro. A ideia é que esse tipo de evento (i.e. eventos que duram o dia inteiro) sejam mais destacados que os demais. Já na visão de semana (Figura 3.6) a mesma idéia não é passada ao usuário. O evento de dia inteiro aparece no alto, fora do alcance da tabela de horas e com menos destaque que os outros eventos que duram parte de um dia.

O exemplo mostra uma falta de coerência da ferramenta quanto à definição dos critérios de relevância para exibição da metáfora visual.

A metáfora visual utilizada na estrutura de armazenamento e recuperação da web apresenta um modelo onde pastas e subpastas são utilizadas para armazenar e recuperar os arquivos. Esse modelo segue o adotado pela maioria dos Sistemas Operacionais, como Windows, Linux e Mac OS.

Conforme dito anteriormente, os seres humanos tem dificuldade de memorizar resultados parciais, para utilização em um segundo momento. Isso indica que o modelo utilizado pode não ser o ideal para os humanos, uma vez que se faz necessária a memorização da seqüência de pastas percorridas para salvar um arquivo no momento em que se vai recuperá-lo.

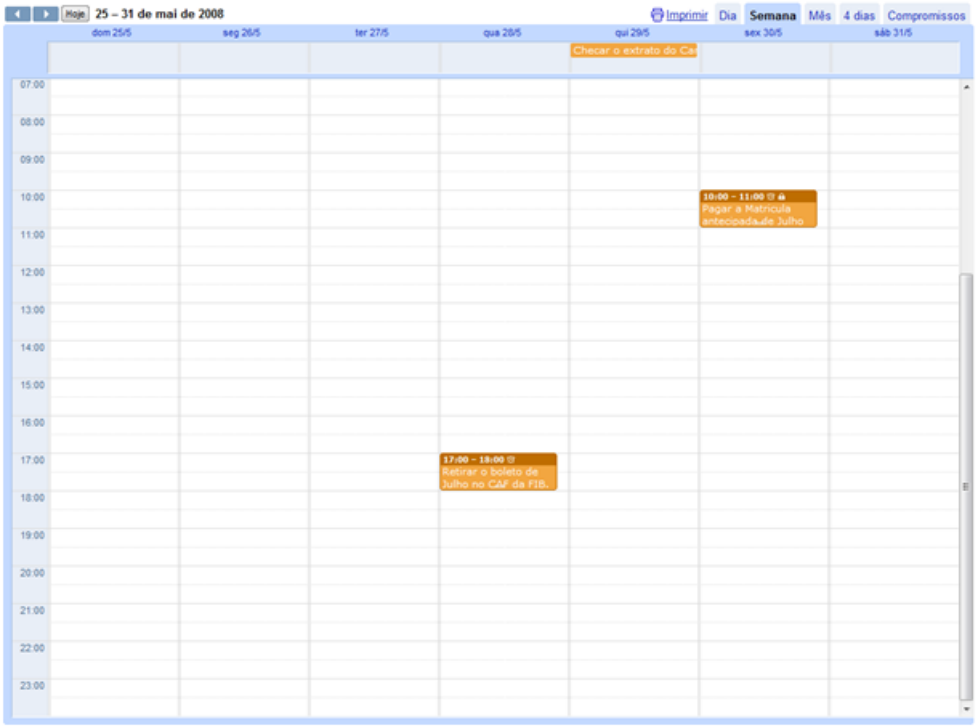


Figura 3.6: Eventos de calendário mostrados numa visão de semana.

3.6 Nuvem de Tags

A nuvem de tags é uma metáfora visual muito utilizada, dentro do contexto de folksonomia, na web para prover uma estrutura de classificação, armazenamento e recuperação de dados. Esta metáfora visual se assemelha com um histograma, porém normalmente histogramas representam poucos itens e uma nuvem de tags geralmente representam uma variedade maior de itens, além da possibilidade de interação que a nuvem de tags fornece (KASER; LEMIERE, 2007).

A freqüência de interação por parte do usuário, normalmente, é usada para modificar o formato ou disposição da nuvem de tags. A estratégia utilizada pela maioria dos sites, é que quanto mais uma tag é escolhida para ser atribuída a um arquivo ou para recuperar uma informação, mais ela se torna relevante ao contexto e por conseqüência aparece na nuvem com um tamanho maior (e.g. Figura 3.7). Sendo assim, uma nuvem de tags geralmente apresenta palavras com textos grandes e pequenos, onde a tag que tem o maior texto é a tag mais relevante, ou a mais utilizada, e a tag de menor texto é a menos relevante (KASER; LEMIERE, 2007).

A nuvem de tags pode ser utilizada para rotular ou classificar um conteúdo. As tags selecionadas pelo usuário são atribuídas ao dado, de acordo com o conceito de folksonomia. Esse processo permitirá a recuperação posterior desse dado a partir das tags atribuídas,

seja pela nuvem de tags ou por outro meio. Entretanto, isso não significa que a recuperação a partir das tags só possa ser feita para conteúdos rotulados. A recuperação a partir de tags pode ser feita levando em consideração o nome de uma foto ou ainda o texto encontrado dentro de um documento. A recuperação dos dados não está atrelada somente à nuvem, uma nova palavra pode ser informada para realizar a busca, podendo inclusive fazer parte de uma formação futura da nuvem de tags.

Alguns sites na web já utilizam a nuvem de tags para viabilizar a recuperação de conteúdo. Por exemplo, é apresentada na Figura 3.7 a nuvem de tags extraída do site da Globo.

Figura 3.7: Imagem da nuvem de *tags* do portal Globo.com.

A Figura 3.8 mostra a nuvem de tags do site Quintura que é utilizada para auxiliar a busca dentro do portal. Ao clicar em alguma tag da nuvem a palavra referente a tag será incluída na *string* de busca. A nuvem é montada de acordo com as palavras escolhidas durante o processo de busca, envolvendo termos relacionados a essas palavras.

Cada site tem uma forma de exibir sua nuvem, geralmente as tags vêm em ordem alfabética e com o tamanho de cada uma relacionada a quantidade de vezes que aquela palavra foi requisitada. Desta forma quando o usuário for buscar algo ele saberá o que está sendo mais procurado naquele site.

As técnicas e conceitos estudadas até aqui são a base do MOFI. Esse modelo consiste na utilização de técnicas consolidadas e promissoras da web para aperfeiçoar buscas na rede mundial de computadores. No capítulo seguinte o MOFI é apresentado e detalhado.

Figura 3.8: Imagem do portal Quintura.com.

Capítulo Quatro

MOFI - Modelo Computacional para Recuperação de Informação baseado em Ontologias, Foksonomia e Indexação Automática de Conteúdo

Pesquisar informações na web é uma atividade útil a diversas pessoas, seja para estudar um assunto ou por lazer. Alguns pesquisadores vêm desenvolvendo tecnologias com o intuito de facilitar este tipo de tarefa na web. São técnicas como Folksonomia, Algoritmos de Busca e Indexação automática de arquivos (HATCHER; GOSPODNETIC, 2005).

Apesar de representarem um avanço na recuperação de informações na *web* as técnicas citadas ainda não atendem plenamente os usuários. Termos sinônimos não são localizados em *sites* de buscas baseados nessas tecnologias, assim como termos relacionados nem sempre são incrementados na *string* de busca. Por exemplo, uma pesquisa feita pelo termo *aluno* dificilmente retorna documentos que contenham o termo *discente* (quando o termo discente não acompanha o termo *aluno*), assim como uma busca pela palavra *manga* pode retornar documentos de diversos gêneros, como a revista japonesa *mangá*, a *manga* de camisa e a fruta *manga*.

A Web Semântica surge como solução para esses problemas (BERNERS-LEE; HENDLER; LASSILA, 2001). Uma das iniciativas da Ciência da Computação para viabilizar a Web Semântica é a incorporação de um sistema de representação do conhecimento a Ontologia, principalmente devido a sua interoperabilidade semântica. Uma Ontologia possui estrutura formal e elementos de representação do conhecimento. O uso de Folksonomia associado à Ontologias de Domínio, sugerido por este autor no artigo *Ontology Tagging - Um componente para Integração de Ontologia e Folksonomia* (REIS et al., 2009), e a construção de uma Ontologia de Folksonomia para interoperar sistemas, sugerida por Gruber (2007), são alguns exemplos de utilização de uma Ontologia para optimizar a recuperação de informação na *web*.

Relacionado com esse contexto, esta dissertação tem como objetivo especificar e construir um modelo computacional, denominado MOFI, que utilize uma Ontologia de Domínio para aperfeiçoar a recuperação de informação, feita através de folksonomia e de técnicas de indexação automática de arquivos, em bases de dados semi-estruturados.

Um mecanismo de busca baseado no MOFI, deve possuir dois módulos para execução da busca. O primeiro é o módulo de indexação automática, que possui o índice dos termos encontrados nos conteúdos armazenados, por meio desse módulo pode-se localizar textos

existentes nos conteúdos de arquivos, como documentos texto ou planilhas eletrônicas. O segundo é o módulo de indexação manual, que tem seu funcionamento baseado na técnica Folksonomia, o que permite ao usuário localizar seus conteúdos através de tags previamente atribuídas aos conteúdos, como fotos, vídeos, arquivos textos e planilhas eletrônicas.

Em 2010 esta dissertação gerou um artigo (REIS; JORGE; PEREIRA, 2010) que foi publicado no III Seminário de Pesquisa sobre Ontologia no Brasil (OntoBras), este é o seminário mais importante dentre os que abordam o tema de Ontologias no Brasil. O artigo publicado no OntoBras é um resumo de alguns tópicos abordados nesta dissertação.

Visando o entendimento do MOFI este capítulo apresenta primeiramente, um trabalho correlato a esta pesquisa. Posteriormente, o diagrama de caso de uso e os requisitos do sistema desenvolvido para atender o MOFI. Em seguida, a arquitetura do modelo é apresentada e discutida detalhadamente. Os quatro tipos de buscas que foram desenvolvidos com base em uma Ontolgoia de Domínio também serão apresentados e discutidos. O componente *Ontology Tagging* e o método proposto para a conversão de uma Ontologia em uma Nuvem de Tags são explicados. Também é apresentado o sistema Goon, desenvolvido com base no modelo MOFI. Por fim, é feita a avaliação do modelo e do sistema.

4.1 Trabalhos Correlatos

Um trabalho similar a este foi o apresentado na Conferência Internacional de Tecnologia da Informação e Comunicação ocorrido na Índia em 2010. Saruladha, Aghila e Penchala (2010), propôs em seu artigo o mapeamento de termos ou palavras de um texto para determinados conceitos de uma ontologia. Desta forma os termos de um documento poderiam ser definidos pela ontologia, de acordo ao conceito vinculado. Problemas como ambiguidades seriam solucionados, pois o contexto dessas palavras seria definido pela ontologia, também seria possível identificar termos sinônimos ao dos termos mapeados.

A ideia é que alguns termos de um texto sejam definidos pela ontologia, através de algum de seus conceitos. Desta forma esse termo não só pode ser classificado, pelo conceito que está vinculado, e também pelas relações que este conceito possui. Para exemplificar melhor, imagina-se um texto onde aparece o termo *Paulo*, vinculando este termo ao conceito *Pai* dentro de uma ontologia de Pessoas, pode-se inferir que Paulo é um homem e que tem filhos, pois para ser Pai o sujeito deve ser um homem e também possuir filhos.

4.2 Diagrama de Caso de uso

O diagrama de caso de uso do sistema Goon foi desenhado para auxiliar na extração dos requisitos do software. A Figura 4.1 ilustra o caso de uso desenhado.

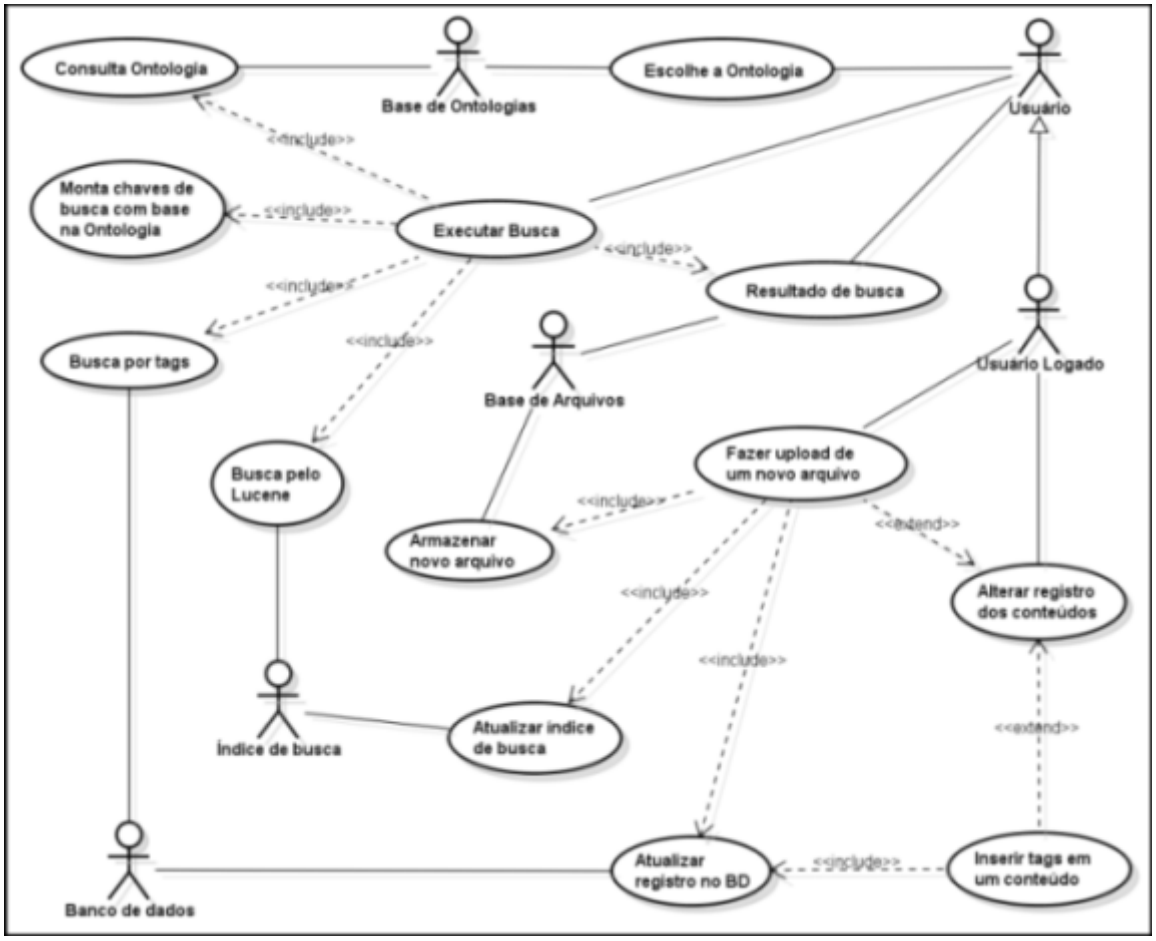


Figura 4.1: Diagrama de Caso de Uso do MOFI.

De acordo com este caso de uso, o Usuário pode escolher uma ontologia, dentre as armazenas na Base de Ontologia, Executar a Busca e acessar os Resultados da Busca, que estão armazenados na Base de Arquivos. Ao Executar a Busca o sistema aciona necesseriamente as funções de Consulta a Ontologia, Montar Chaves de Busca com Base na Ontologia, Busca por Tags, que utiliza o banco de dados, e Busca pelo Lucene. Já um Usuário Logado, além dessas funções pode também, Alterar o Registro dos Conteúdos que consistêm em Fazer Upload de um Novo Arquivo, que será incluído no índice de busca do Lucene, ou Inserir Tags em um Conteúdo.

4.3 Requisitos do Sistema

Para validar o MOFI, foi necessária sua implementação em sistema web capaz de armazenar e recuperar dados semi-estruturados, como arquivos textos, fotos ou videos de diferentes formatos. Para isso foi desenvolvido o sistema *Goon* que tem os seus requisitos descritos a seguir.

Requisitos Funcionais (RQF):

- **RQF1** - gerir uma base de dados semi-estruturados onde o usuário poderá armazenar seus arquivos através de um sistema web;
- **RQF2** - permitir ao usuário recuperar seus documentos através de um sistema web;
- **RQF3** - prover indexação manual com base em Folksonomia;
- **RQF4** - prover indexação automática com base em uma Técnicas de Indexação de Conteúdos (arquivos ou textos);
- **RQF5** - gerir uma base de dados estruturados para armazenar o endereço físico de cada arquivo e seus vínculos com as tags da folksonomia e o índice de busca da indexação;
- **RQF6** - converter uma Ontologia de Domínio em nuvem de tags, facilitando a utilização da Ontologia por parte do usuário, e permitindo que o mesmo classifique e busque conteúdo a partir de tags oriundas de uma Ontologia;
- **RQF7** - a nuvem de tags deve ser gerada a partir das classes e propriedades da Ontologia de Domínio, e também deve ser interativa, destacando o conceito (ou classe) escolhido pelo usuário no centro da nuvem e ao redor dele os outros conceitos que com ele estejam vinculados (via alguma propriedade).
- **RQF8** - utilizar Ontologias para realizar buscas contextualizadas, e por sinônimos e hipônimos;
- **RQF9** - prover um motor de busca que gerencie o fluxo da busca, a montagem da *string* de busca, o acionamento da consulta nos dois módulos de indexação, e consolidação e retorno dos resultados;
- **RQF10** - gerir uma base de arquivos OWL para serem utilizados pelos usuários na montagem da nuvem de tags, e para classificação e recuperação dos documentos.

Requisitos Não-Funcionais (RNF):

- **RNF1** - ser um sistema web;

- **RNF2** - possuir divisão de camadas;
- **RNF3** - ser portavél entre sistemas operacionais;
- **RNF4** - utilizar frameworks para auxiliar o desenvolvimento.

4.4 *Arquitetura*

Seguindo o RNF2, a arquitetura do MOFI consiste em três camadas, visão, controle e modelo (Figura 4.2).

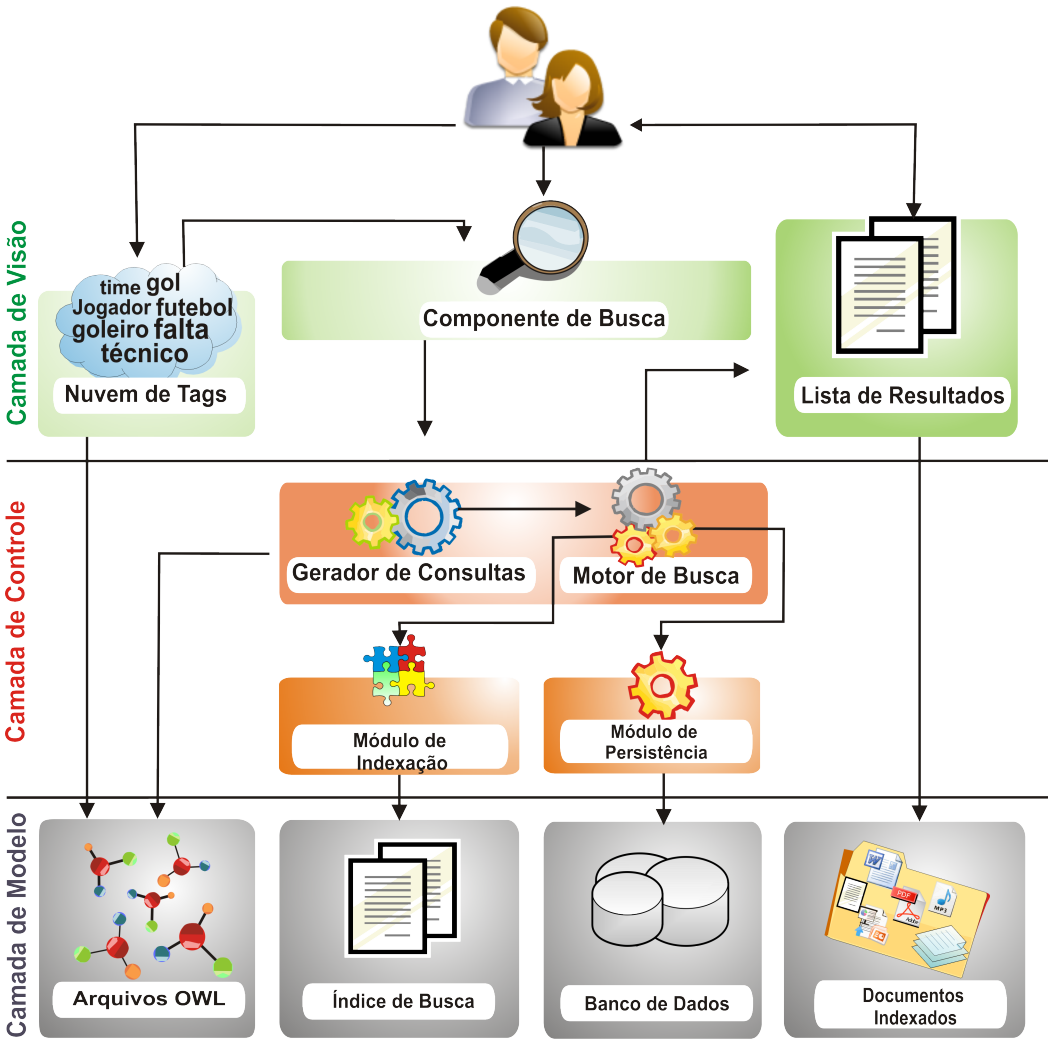


Figura 4.2: Arquitetura do modelo MOFI.

O usuário poderá interagir com a camda de visão para realizar suas buscas e ter acesso aos seus documentos e aos documentos públicados por outros usuários. Os objetos da camada de visão são definidos a seguir.

- **Componente de Busca** (RQF2) - É o componente de entrada das buscas e permite ao usuário fornecer termos para realização da busca digitados livremente ou através do componente de Nuvem de Tags, que serão termos oriundos de uma Ontologia de Domínio;
- **Nuvem de Tags** (RQF6 e RQF7) - Componente que provê a nuvem de tags montada pelo *Ontology Tagging* (REIS et al., 2009). Este componente permite ao usuário escolher uma Ontologia, para contextualizar a busca, e navegar por ela escolhendo os termos (clicando nas classes) que serão utilizados na consulta. As buscas realizadas, tendo uma Ontologia escolhida neste componente, serão refinadas no *Gerador de Consultas*;
- **Lista de Resultados** (RQF2) - Componente visual que exibe o resultado das buscas para o usuário. Cada documento, ou item, encontrado deve exibir no seu resultado dados relevantes, como nome, data de publicação e usuário que o publicou.

Já os objetos da camada de controle são responsáveis pelo processamento das informações. É nesta camada que estão implementadas as principais regras do software, conforme a descrição dos componentes a seguir.

- **Gerador de Consultas** (RQF8) - Componente responsável pelo refinamento da busca, e é baseado nas classes da Ontologia de Domínio, e em suas propriedades (relações) e restrições. O Gerador de Consultas recebe os termos informados pelo usuário e, caso exista uma Ontologia selecionada, verifica a existência na mesma de alguns desses termos. Caso exista, o componente montará a *string* de busca (*query*) com base nas relações e restrições desse termo encontrado. Se não encontrar nenhum dos termos da busca, ou se nenhuma Ontologia tiver sido escolhida pelo usuário, montará uma *query* sem nenhum refinamento. Este componente será mais detalhado na Seção 4.6;
- **Motor de Busca** (RQF9) - Engrenagem responsável por interligar o Gerador de Consultas, o Módulo de Indexação e o de Persistência, a fim de realizar todo o processo de busca. Esta engrenagem repassa a *query* de busca, construída no Gerador de Consultas, aos módulos de Indexação e de Persistência, consolida os resultados e os repassa para a *Lista de Resultados* na camada de visão;
- **Módulo de Indexação Automática** (RQF4) - Módulo responsável pela indexação automática e busca dos documentos. A técnica implementada se baseia na extração dos termos mais freqüentes de um texto, denominados palavras-chave, criando um índice de palavras. Esse índice agiliza a busca pelos arquivos que foram indexados (Seção 2.3);

- **Módulo de Indexação Manual** (RQF3) - Módulo responsável pela indexação manual e busca dos mesmos documentos. Este módulo é baseado na classificação do usuário (folksonomia). Onde este atribui tags (rótulos) aos arquivos e as buscas são realizadas através dessas tags (Seção 2.6).

Por fim, os objetos da camada de modelo ou persistência, armazenam as informações inerentes ao sistema. São os arquivos OWL, o banco de dados, o índice de busca e os documentos indexados.

- **Base de Arquivos OWL** (RQF10) - São as Ontologias, escritas em arquivos OWL, que serão utilizadas pelo sistema, tanto para prover a nuvem de tags e refinar as buscas;
- **Banco de Dados** (RQF5) - Utilizado para registrar os arquivos armazenados pelo sistema e os vínculos deles com as tags;
- **Índice de Busca** (RQF4) - Índice das palavras-chave dos arquivos gerado pelo módulo de indexação automática;
- **Base de Conteúdo** (RQF1) - Diretório onde o sistema armazena os arquivos ou documentos indexados.

Visando um maior entendimento da arquitetura, estão detalhados nas seções seguintes (4.5 e 4.6) o componente *Ontology Tagging* juntamente com o método proposto para conversão de uma Ontologia de Domínio em uma nuvem de tags (RQF6 e RQF7), e os 4 (quatro) tipos de Consultas baseados em uma Ontologia que foram desenvolvidos durante esta pesquisa (RQF8).

4.5 Ontology Tagging

O componente *Ontology Tagging* foi desenvolvido por este autor em seu trabalho de conclusão de curso de graduação (monografia) (REIS et al., 2009), e sofreu algumas alterações durante o desenvolvimento desta disseertação de mestrado para atender a conversão de uma ontologia de domínio em uma nuvem de tags proposta no MOFI (RQF7). Porém, a ideia central do *Ontology Tagging* continua sendo a mesma, a de prover a outras aplicações ou sites uma nuvem de tags, formada a partir de uma ontologia de domínio, para utilização de seus usuários (RQF6).

Este componente provê ao usuário a metáfora visual de uma nuvem de tags, onde os conceitos de uma ontologia são utilizados como tags na composição da nuvem. Esses

conceitos podem ser utilizados como tags sugestivas para classificar, armazenar e recuperar documentos, juntamente com outras tags, que poderão ser inseridas manualmente pelo usuário.

Para converter uma ontologia de domínio em uma nuvem de tags, o componente *Ontology Tagging* extrai o nome de cada classe da ontologia transformando-o em uma tag, além de replicar as relações (propriedades) de cada conceito (classe) em sua respectiva *tag* (de mesmo nome). Desta fomra, as ligações entre as tags desta nuvem serão espelhadas nos relacionamentos (propriedades) que interligam os conceitos.

Na pesquisa (REIS et al., 2009) foi utilizada uma Ontologia de Pizza para os testes do componente. Esta Ontologia foi desenvolvida no software Protégé a partir do tutorial *A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools* (HORRIDGE et al., 2004). A Figura 4.3 mostra a hierarquia dessa Ontologia.

Figura 4.3: Hierarquia da Ontologia de Pizza ilustrada pelo software Protégé. (REIS et al., 2009).

O trabalho publicado inicialmente (REIS et al., 2009) ordenava essas tags segundo o seu critério de relevância disponibilizando a tag mais relevante no centro da nuvem e as outras tags ao seu redor, sempre em ordem de relevância.

A Tabela 4.1 ilustra o cálculo de relevância para cada tag da Ontologia de Pizza. A tag mais relevante é aquela que possuiu o maior número de relações. Para calcular a relevância das demais tags também é levada em consideração a sua proximidade com a tag mais relevante. Uma base de relevância é utilizada para representar a proximidade de uma tag com a tag central. A tag central central terá automaticamente um valor igual a 100 na Base de Relevância e as outras terão valor 10% menor a cada iteração. Quanto menor a Base de Relevância menor a proximidade de uma tag com a tag central. Desta forma, a quantidade de relações de cada tag é somada a sua base de relevância para determinar o seu grau de relevância (REIS et al., 2009).

Tabela 4.1: Ilustrando os dados utilizados no calculo de relevância. Fonte (REIS et al., 2009).

***Tags* Conceitos**	**Relações**	**Base**	**Relevância**
VegetableTopping	7	100	107
MeatTopping	5	90	95
PepperTopping	4	90	94
SeafoodTopping	4	90	94
CheeseTopping	3	90	93
AnchovyTopping	1	90	91
CaperTopping	1	90	91
HamTopping	1	90	91
MozzarellaTopping	1	90	91
MushroomTopping	1	90	91
OliveTopping	1	90	91

Na Figura 4.4 é possível observar a nuvem resultante onde são exibidos os conceitos da ontologia já convertidos em tags. A tag mais relevante aparece no centro da nuvem e as demais são dispostas ao seu redor utilizando-se do seu grau de relevância em relação à tag central.

O problema identificado durante essa pesquisa de mestrado é que essa nuvem resultante é estática, o que dificulta que algum termo seja destacado e permite que alguma tag possa não aparecer na tela caso a Ontologia seja muito extensa. Por esses motivos, é proposto no MOFI o uso de um algoritmo de conversão de Ontologia de Domínio em uma Nuvem de Tags que sejá interativa, que se modifique reorganizando a posição das tags e destacando os termos escolhidos pelo usuário.

A geração dessa nova estrutura continua disponibilizando a tag mais relevante no centro da nuvem. Essa tag central, agora, será a classe (conceito) mais alta na hierarquia da Ontologia. Caso existam mais de uma classe no topo da hierarquia da Ontologia, a tag central será gerada pelo nome do arquivo OWL da Ontologia, além disso, também é criada uma propriedade de composição para relacionar essa tag central com as tags das classes mais altas na hierarquia da Ontologia. O restante da nuvem será formado pelas tags que se relacionam com a tag central, que serão postas ao seu redor.

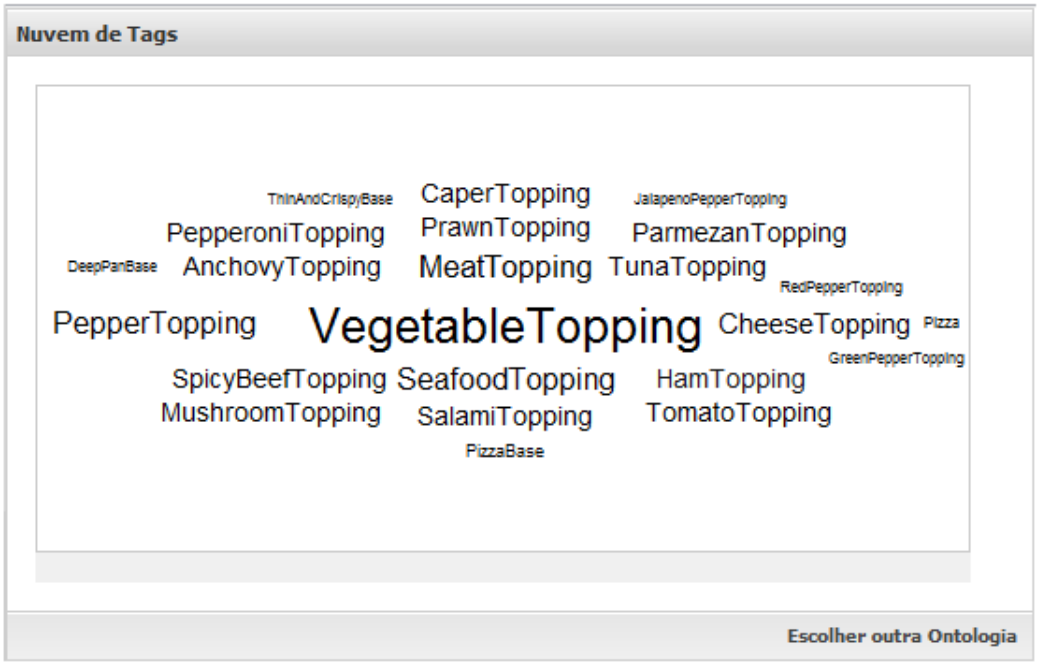


Figura 4.4: Nuvem de Tags gerada inicialmente pelo *Ontology Tagging*. Fonte: (REIS et al., 2009).

Essa nuvem de tags poderá se modificar de acordo com a escolha do usuário, que poderá escolher outra tag para torna-se a tag central da nuvem. Desta forma o usuário pode navegar na nuvem de tags, através das propriedade da ontologia, visualizando suas classes.

A Figura 4.5 ilustra o esquema de montagem e de navegação da Nuvem de Tags gerada a partir de uma Ontologia. O quadro A da Figura 4.5 demonstra os conceitos de uma ontologia de domínio e seus relacionamentos de herança e composição. No quadro B é apresentada a ontologia anterior convertida em um formato de nuvem de tags. A tag central desta nuvem é o conceito mais alto na hierarquia da ontologia, ou o nome do arquivo OWL que armazena a ontologia. O quadro C mostra que é possível navegar por esta nuvem através das tags que possuem o sinal "+"(soma), que são tags que possuem outros relacionamentos além do relacionamento com a tag central. Ao clicar em uma dessas tags, uma nova nuvem será gerada tendo como destaque central a tag escolhida, e seus relacionamentos definirão as tags ao seu redor. O quadro D ilustra que após expandir um conceito a tag central anterior, a do quadro C, é apresentada na nuvem com o sinal de "+"(soma), o que torna possível retornar a visualização da nuvem anterior. Para melhor entendimento ver o algoritmo implementado, para a montagem desta nuvem de tags, no Apêndice A.2.

A Figura 4.6 mostra como ficou o novo formato da nuvem de tags para ontologia de Pizza, gerada pelo novo algoritmo do *Ontology Tagging*.

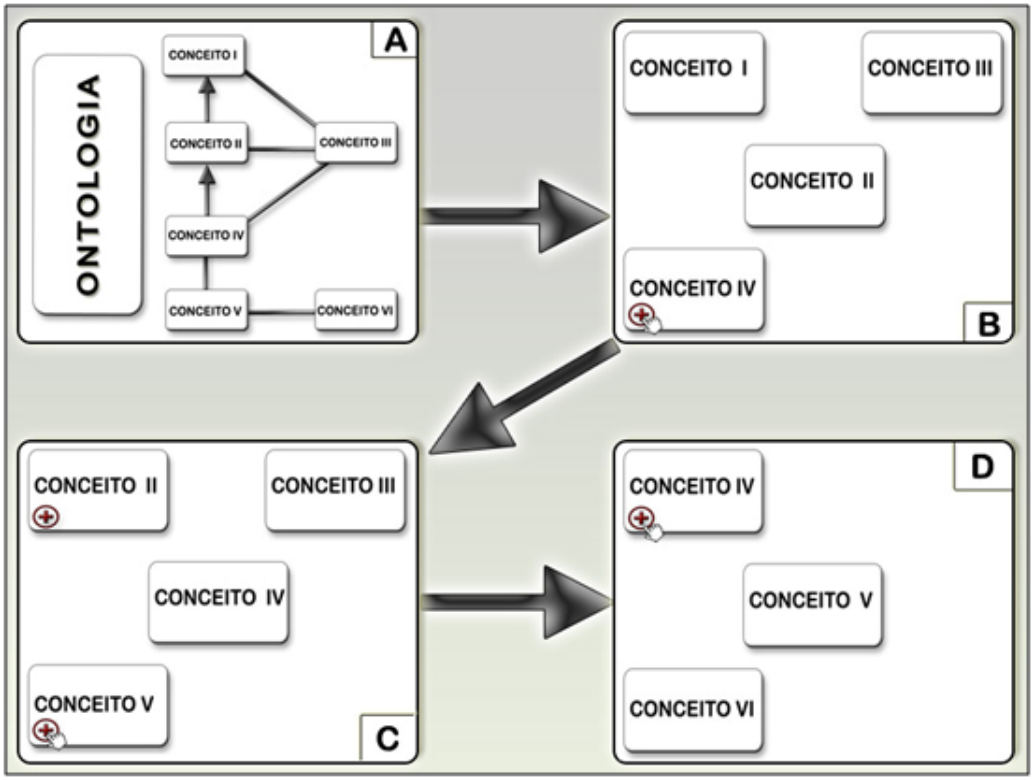


Figura 4.5: Esquema de montagem da Nuvem de *Tags*.

4.6 Consultas baseadas em Ontologia

O motor de busca do MOFI, através do Gerador de Consultas, tenta localizar na Ontologia alguma das palavras-chave informada pelo usuário para a realização da busca. Caso encontre alguma, montará as chaves de busca de forma a implementar consultas baseadas nas propriedades e restrições do termo encontrado na Ontologia. Os termos que não forem encontrados na ontologia são mantidos na *string* de busca sem alteração.

O principal diferencial do MOFI está nos seus 4 (quatro) tipos de processamento de busca construídos a partir das propriedades (relações) e restrições de uma Ontologia. Detalha-se cada tipo a seguir através de um exemplo ilustrativo no contexto de futebol (RQF8).

1. **Busca por sinônimos** - Caso alguma das palavras-chave, que foram encontradas na Ontologia, tenham uma relação de equivalência com algum outro termo, essa será incluída na string de busca com o operador lógico *OR*, de modo que também será procurado. Um exemplo dessa string de busca seria *Aluno OR Discente*. O objetivo desse tipo de busca é localizar termos sinônimos não encontrados em buscas baseadas em estrutura sintática;
2. **Busca por hipônimos** - Caso alguma das palavras-chave, que foram encontradas na Ontologia, tenha uma relação de hierarquia, sendo ela a superclasse de algum outro

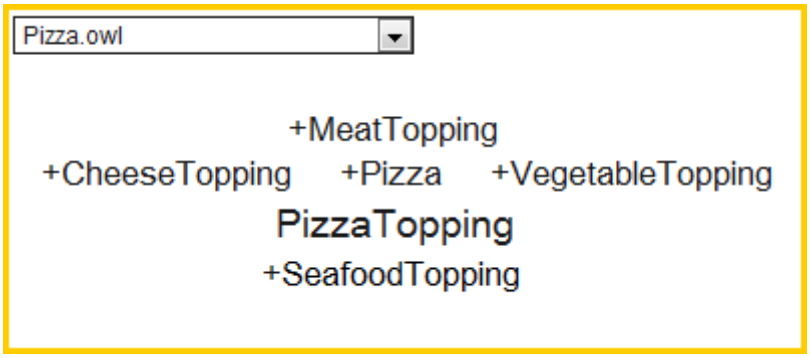


Figura 4.6: Novo formato da nuvem gerada pelo *Ontology Tagging*.

termo, essa seráo incluída na chave de busca com o operador lógico OR, de modo a também ser procurado. Por exemplo, *Jogador OR Goleiro OR Jogador_de_Linha*, sendo Jogador o termo buscado. O objetivo desse tipo de busca é encontrar conceitos que não são sinônimos, mas que são um tipo do conceito procurado;

3. **Busca contextualizada com o termo buscado** - Caso alguma das palavras-chave, que foram encontradas na Ontologia, tenha outros tipos de relação, exceto equivalência e herança (filhos ou especializações) com algum outro termo, esses termos, juntamente com os termos mais acima na hierarquia, serão incluídos na *string* de busca com os operadores lógicos AND e OR. Por exemplo, *Jogador AND (Futebol OR Time OR impedimento)*, sendo Jogador o termo a ser recuperado. O objetivo desse tipo de busca é filtrar os resultados, focando no contexto definido pela Ontologia escolhida.

4. **Busca contextualizada sem o termo buscado** - Caso alguma das palavras-chave, que foram encontradas na Ontologia, tenha outros tipos de propriedades, exceto equivalência e herança (filhos ou especializações) com algum outro termo, esses formarão, sem o termo buscado, a string de busca com o operador lógico AND. Um exemplo dessa chave seria *(Jogador AND Time AND impedimento AND Gol)*, sendo Futebol o termo a ser recuperado. Vale ressaltar que nesse tipo de busca os termos mais altos da hierarquia da ontologia não entram, apenas os termos ligados diretamente ao termo buscado contextualizam essa busca. O objetivo desse tipo de busca é retornar conteúdos relacionados com o contexto definido pela Ontologia escolhida, mesmo que o termo buscado não esteja presente. Já que é possível um texto falar de futebol sem conter a palavra futebol.

Um dos diferenciais do modelo MOFI, em relação à busca apenas sintática, é que existe uma diminuição do retorno de conteúdos sobre futebol americano, aumentando o retorno de conteúdos que falem sobre o futebol tradicional (*soccer*), por exemplo. E ainda permite ao motor de busca encontrar textos que falem de futebol sem ter a palavra futebol, especificamente, como um texto que fale de um *gol feito pelo jogador A do time B em*

impedimento aos 45 minutos do segundo tempo.

Utilizando a propriedade de equivalência, existente em uma ontologia OWL, ainda podem ser obtidos termos equivalentes (ou sinônimos) a outros termos. Desta forma, ao definir uma propriedade de equivalência da classe *Arbitro* para a classe *Juiz* em uma ontologia de domínio de Futebol, o sistema *Goon* pode realizar a busca pelo termo sinônimo de árbitro. Isso não seria possível em um motor de busca apenas sintático. A busca por hierarquia de conceitos é realizada quando a palavra-chave que se deseja buscar está no contexto da Ontologia e tem conceitos abaixo em sua hierarquia. Por exemplo, as palavras Jogador e Goleiro. Elas não significam a mesma coisa, não são sinônimos, mas o Goleiro é um tipo de Jogador, portanto se procuramos por um Jogador esse pode ser um Goleiro, o que não ocorre no caso contrário, quando procuro um Goleiro não posso retornar qualquer Jogador, tem que ser um Goleiro e não um Jogador de Linha.

Abaixo um exemplo de uma *string* de busca resultante nesse modelo de consulta:

- (time AND (selecao club)) **OR** ((time selecao club) AND (evento arbitro tecnico gol jogo dirigente impedimento falta membro_comissao_tecnica auxiliar_tecnico medico pessoa futebol partida preparador_fisico jogador)) **OR** (futebol AND dirigente AND tecnico AND auxiliar_tecnico AND medico AND futebol AND (jogo partida) AND (preparador_fisico) AND (jogador goleiro jogador_linha))

Para elucidar melhor a *string* de busca montada pelo Goon, vamos dividi-lá em blocos, a divisão de cada bloco é onde aparece o conectivo *OR*, sendo assim, o primeiro bloco termina antes do primeiro *OR*, além disso, onde não existir conectivo entre uma palavra e outra funcionará como se fosse o conectivo*OR*. Esse primeiro bloco é responsável pelas buscas por sinônimo e hipônimo, sendo que todo termo contido antes do conectivo *AND* é o proprio termo buscado e os termos contidos depois do conectivo *AND* são seus equivalêntes ou filhos na hierarquia de classes. O segundo bloco implementa a busca contextualizada com o termo buscado, que está antes do conectivo *AND* e é composto não só pelo termo buscado como também pelos seus termos sinônimos e hipônimos. O último bloco implementa a busca contextualizada sem o termo buscado. Em alguns casos pode-se perceber que existem termos juntos entre parênteses dentro da *string* de busca, como no último bloco (*(jogador goleiro jogador_linha)*), isso ocorre porque nesse lugar teria apenas um termo, neste exemplo o *jogador* mas como existem termos que são sinônimo ou equivalentes a ele esses também compõem a *string*. Todas as demais *strings* de busca seguem esse mesmo padrão.

Caso o motor de busca do modelo proposto não encontre nenhuma das palavra-chave solicitadas pelo usuário na Ontologia ou o termo encontrado não possua ligações que

permitam realizar os quatro tipos de busca, as strings de busca serão montadas de forma convencional, onde os termos solicitados são pesquisados sem a inclusão de novos termos ou operadores lógicos. Ou seja, a string de busca inserida pelo usuário será passada diretamente aos módulos de indexação (automática e manual), sem alterações. Nesse caso, a qualidade do retorno da busca será influenciada pela classificação feita pelo usuário no módulo de indexação manual ou pela técnica utilizada dentro do módulo de indexação automática.

Para melhor entendimento do processo de busca ver o algoritmo implementado no Apêndice A.1.

Para validar o modelo proposto foi desenvolvido um sistema para armazenamento e recuperação de dados na web, seguindo as diretrizes do modelo MOFI. A seguir é apresentado o sistema desenvolvido, assim como os resultados obtidos nos experimentos e testes dessa ferramenta.

4.7 Goon - Aplicação do MOFI

O software Goon está disponível no endereço web *http://goon.no-ip.org/*. O Goon permite que usuários não logados realizem buscas e escolham a Ontologia que irá contextualizar as mesmas. Na Figura 4.7, apresentamos a imagem da página inicial do Goon (sem um usuário logado).

Para usuários logados o Goon permite acesso a tela de edição dos dados armazenados. Nesta tela, o usuário pode não só fazer a busca, como também incluir ou remover arquivos da base, associar ou desassociar tags dos documentos e indexar todos os documentos da base automaticamente, atualizando o índice de busca do Lucene. Na Figura 4.8 temos a imagem da tela de edição de dados do Goon.

Para realizar o *upload* de um arquivo o usuário deve clicar no botão *Novo* dentro do *menu*, e depois em *Arquivo*. Antes de iniciar o *upload* o usuário pode atribuir suas *tags* ao arquivo. Após o processo de *upload* ser concluído o arquivo é indexado automáticamente pelo Lucene. Além disso o usuário pode utilizar o botão *Indexar* para indexar todos os arquivos da base, essa opção pode ser utilizada caso algum documento seja inserido na base diretamente, sem passar pelo software.

A escolha de uma Ontologia para contextualizar a busca não é um fator obrigatório. O Goon realiza buscas sem ter uma Ontologia selecionada, levando em consideração apenas os termos localizados pelo Lucene ou nas tags associadas aos arquivos. Essa abordagem permite ao usuário realizar buscas não contextualizadas, caso o assunto desejado seja

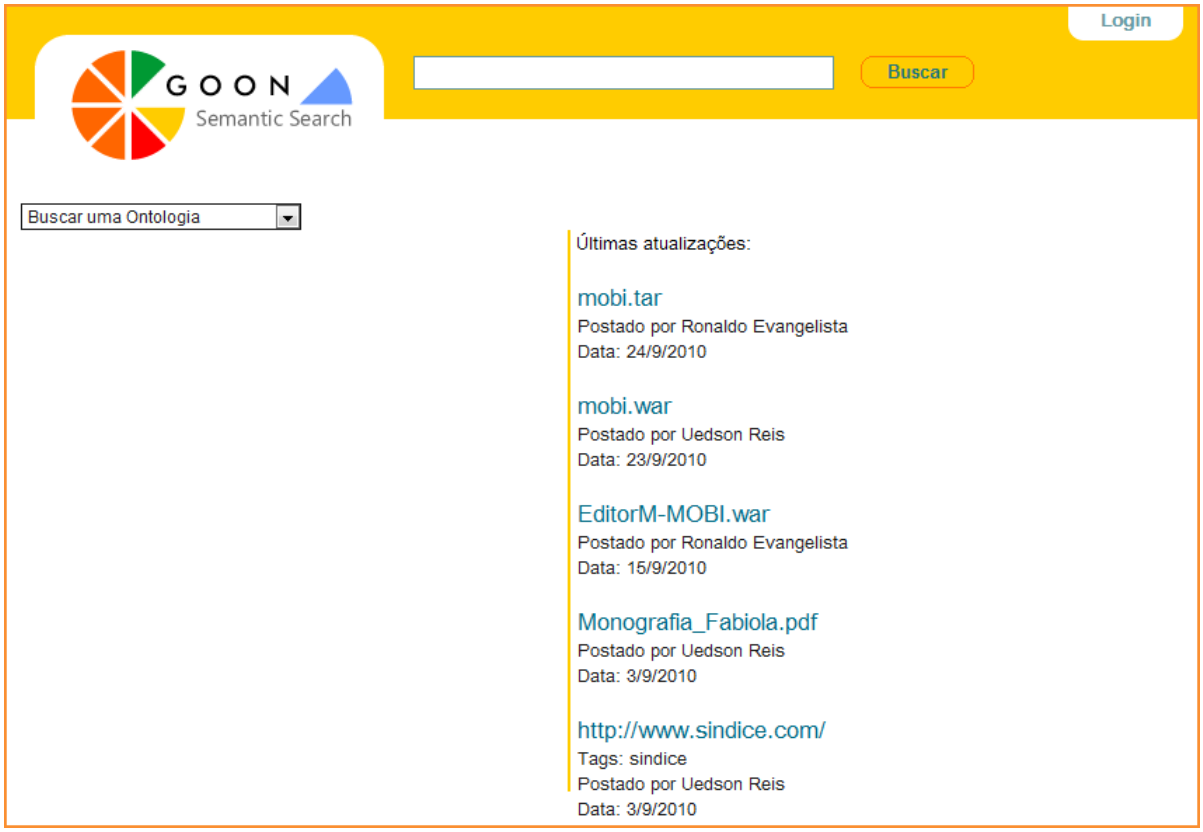


Figura 4.7: Tela inicial do Goon.

muito abrangente ou ainda não esteja disponível uma Ontologia da área ou assunto em questão.

Para atender aos seus requisitos, o sistema *Goon* foi desenvolvido como uma aplicação web (RNF1), seguindo o padrão MVC (*Model View Control*) (RNF2). A linguagem de programação Java versão 1.6, foi escolhida por fornecer uma maior portabilidade ao software (RNF3), além de ser uma das mais utilizadas para o desenvolvimento de aplicações web.

4.7.1 Camada de Visão

Para prover a camada de interface (*view*) com usuário, o GWT (*Google Web Toolkit*) é utilizado por permitir a implementação de AJAX em aplicações Java com transparência, sem a necessidade que o desenvolvedor tenha conhecimento de código *JavaScript*. O desenvolvedor escreve toda a sua camada de interface em código Java e o *framework* a transforma em código HTML (*Hyper Text Multi Language*) e JavaScript.

Como dito anteriormente, o componente *Ontology Tagging* é utilizado para conversão de

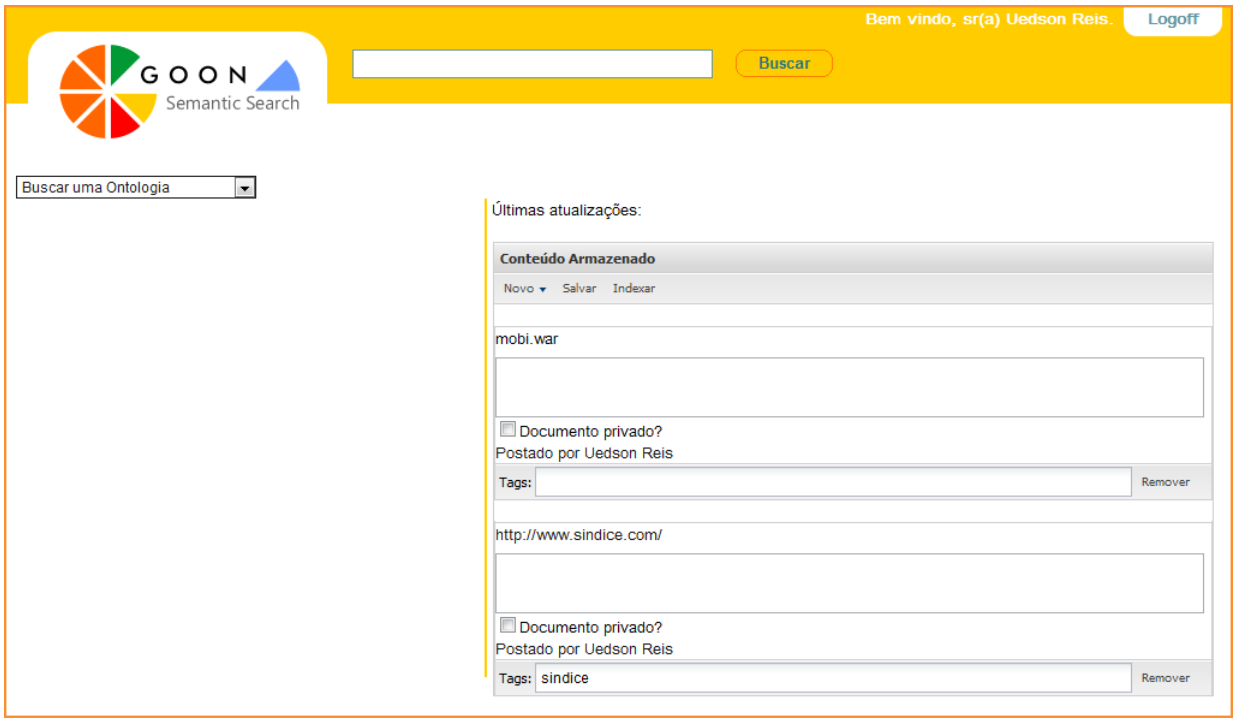


Figura 4.8: Tela de Edição do Goon.

uma Ontologia em uma nuvem de tags e com a ajuda do *framework* GWT esta nuvem é disposta na camada de interface e pode ser manipulada pelo usuário. Para manipular as ontologias no *Ontology Tagging* e durante a busca, o *framework* JENA é utilizado, pois é de código aberto e escrito em linguagem de programação Java. Este *framework* permite acesso aos elementos de uma Ontologia escrita em linguagem OWL, como classes, propriedades ou indivíduos, e as restrições desses elementos (e.g. hierarquia ou equivalência) (JORGE, 2010).

A nuvem de tags criada pelo *Ontology Tagging*, é posicionada do lado direito da tela (esquerdo do usuário) sempre a baixo do componente utilizado para selecionar uma ontologia, onde está escrito *Buscar uma Ontologia*. Ao lado do componente de seleção da ontologia, estão os últimos arquivos armazenados no Goon. Este posicação também é reservada ao resultado das buscas que substituem as *Últimas Atualizações* após a realização de uma consulta. Na parte central superior da tela, está a caixa de texto onde os termos que se desejam buscar são digitados (ver Figura 4.7).

4.7.2 Camada de Modelo

A base utilizada para gerir a persistência dos dados foi o banco de dados MySQL versão 5.0 (RQF5). Por ser um software livre, menos burocrático em termos de implantação e

gratuito para uso acadêmico.

Para gerir a comunicação entre o banco de dados MySQL e a linguagem de programação Java foi utilizado o *framework* Hibernate (RNF4). É através dessas tecnologias que são armazenadas as informações sobre os arquivos e as tags, e os vínculos entre arquivos e tags.

Um diagrama de classes foi desenhado para mostrar quais informações são persistidas e como os dados de usuários, arquivos e tags estão vinculados (Figura 4.9).

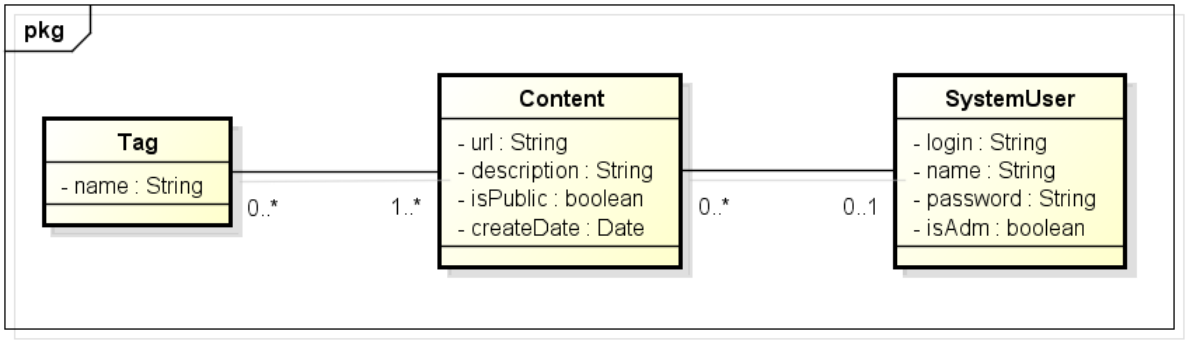


Figura 4.9: Diagrama de Classes do Modelo de Dados.

A classe *Tag* (etiqueta) tem os seguintes atribuitos:

- *name* é o nome ou texto atribuído a tag;
- além do atributo *name* esta classe possui um vínculo com a classe *Content* que permite a uma tag classificar mais de um conteúdo.

A classe *Content* (conteúdo) tem os seguintes atribuitos:

- *id* é o identificador do arquivo no banco de dados;
- *url* é o endereço fisíco dos arquivos;
- *description* é um texto que descreve o conteúdo;
- *isPublic* é a variável que indica se o arquivo é publico, ou seja, se pode ser visto por outros usuários além do usuário que o publicou;
- *createDate* é a data de armazenamento do arquivo na base;
- além desses atributos, esta classe possui um vínculo com a classe *Tag* que permite a um conteúdo ser classificado por mais de uma tag;

- outro vínculo existente nesta classe é com a classe *SystemUser* que permite a um conteúdo ser publicado por um usuário do sistema.

A classe *SystemUser* (usuário de sistema) tem os seguintes atribuitos:

- *login* é o identificador do usuário no sistema;
- *name* é o nome do usuário no sistema;
- *password* é a senha do usuário para acesso ao sistema;
- além desses atributos esta classe possui um vínculo com a classe *Content* que permite a um usuário do sistema publicar mais de um conteúdo;
- *isAdm* identifica, ou não, o usuário como administrador do sistema.

4.7.3 Camada de Controle

Na camada de controle o processo de busca se inicia através de uma chamada assíncrona feita a partir da camada de visão, gerida pelo GWT. De acordo com o Diagrama de Sequência ilustrado na Figura 4.10, o processo passa pelas classes *GreetingService* (fachada de comunição entre as camadas de Visão e Aplicação), *CenterBO* (Objeto de Negócio Central da aplicação), *SearchEngine* (Motor de Busca - RQF9) e termina no *BasicDao* (responsável pela persistência - RQF3) e no *Index* (responsável pela indexação e busca - RQF4).

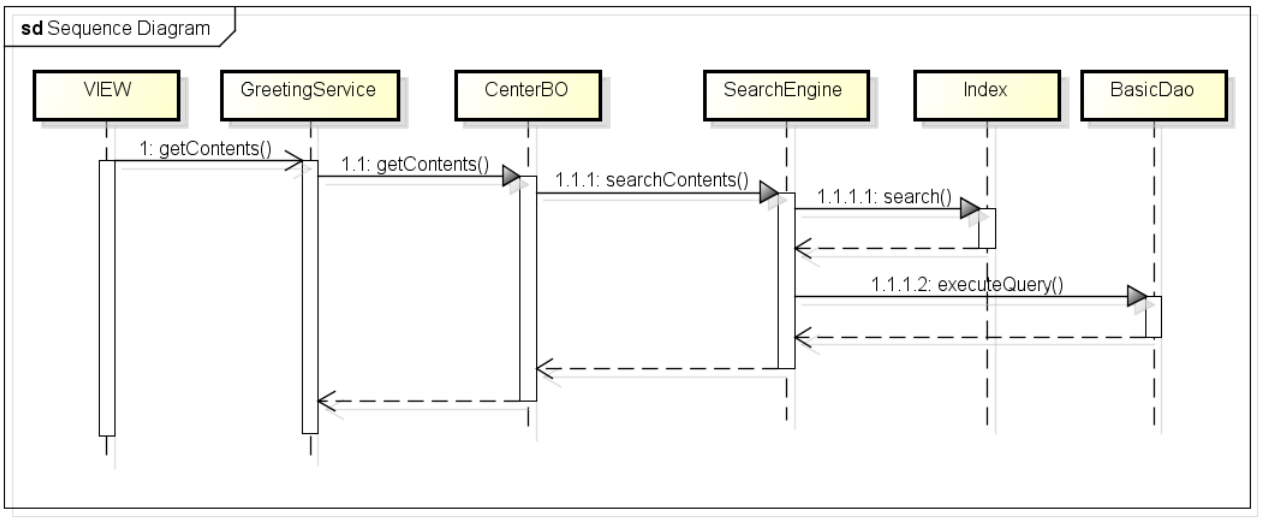


Figura 4.10: Diagrama de Sequência do processo de Busca.

O processo de inclusão de conteúdo no sistema segue um fluxo parecido, neste caso a classe *SearchEngine* não participa do processo, pois a regra de negócio esta implementada na classe *CenterBO*. A Figura 4.11 mostra o diagrama de sequência desenhado para esse processo.

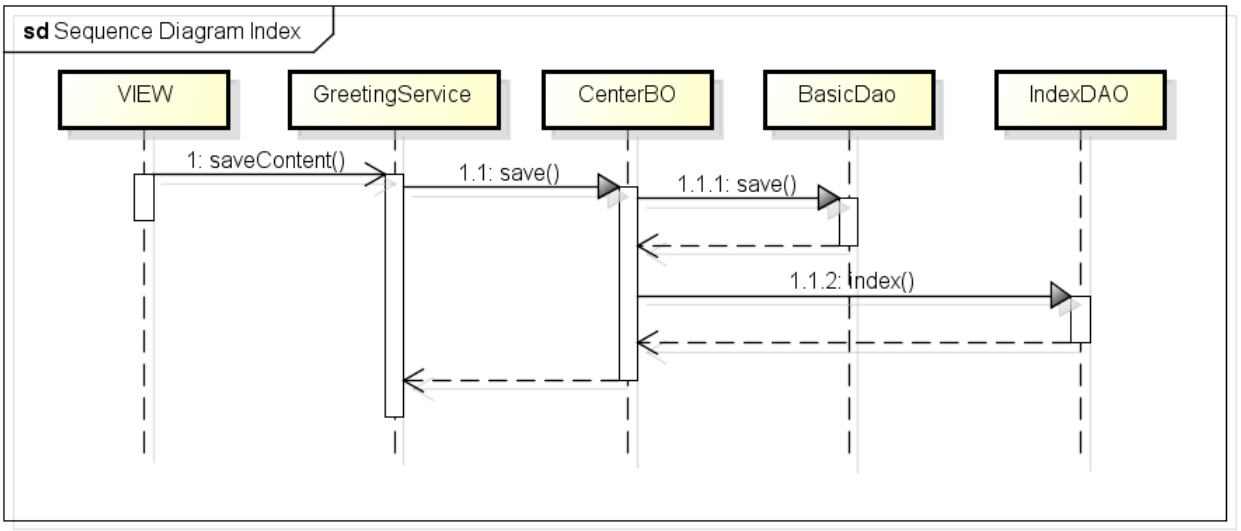


Figura 4.11: Diagrama de Sequência do processo de armazenamento e indexação dos conteúdos.

Para melhor entendimento sobre funcionamento de cada uma dessas classes, um diagrama de classes resumido da camada de aplicação foi desenhado, ver Figura 4.12. Os métodos descritos não são necessariamente todos os métodos que estas classes possuem, mas apenas os métodos mais importantes, acionados durante o processo de armazenamento e recuperação.

A classe *GreetingService* é uma fachada da camada de aplicação, ela abstrai da camada de visão as outras classes da camada de aplicação, provendo apenas as funções pertinentes a camada de visão. Esta classe foi implementada seguindo os princípios do padrão de projeto *Façade* (GAMMA et al., 2000). Abaixo seguem os seus métodos que participam do processo de busca.

- *getContents*: este método recebe duas variáveis texto como parâmetros, a primeira é o nome da ontologia selecionada no *Ontology Tagging* e a segunda são os termos digitados no campo de busca do Goon, o retorno desse método será uma lista dos conteúdos (classe *Content*) encontrados na busca.
- *getSessionUser*: método responsável por recuperar o usuário (classe *SystemUser*) logado, armazenado na sessão de usuário. Este método não recebe parâmetros.
- *saveContent*: método responsável por salvar os conteúdos na base de conteúdos.

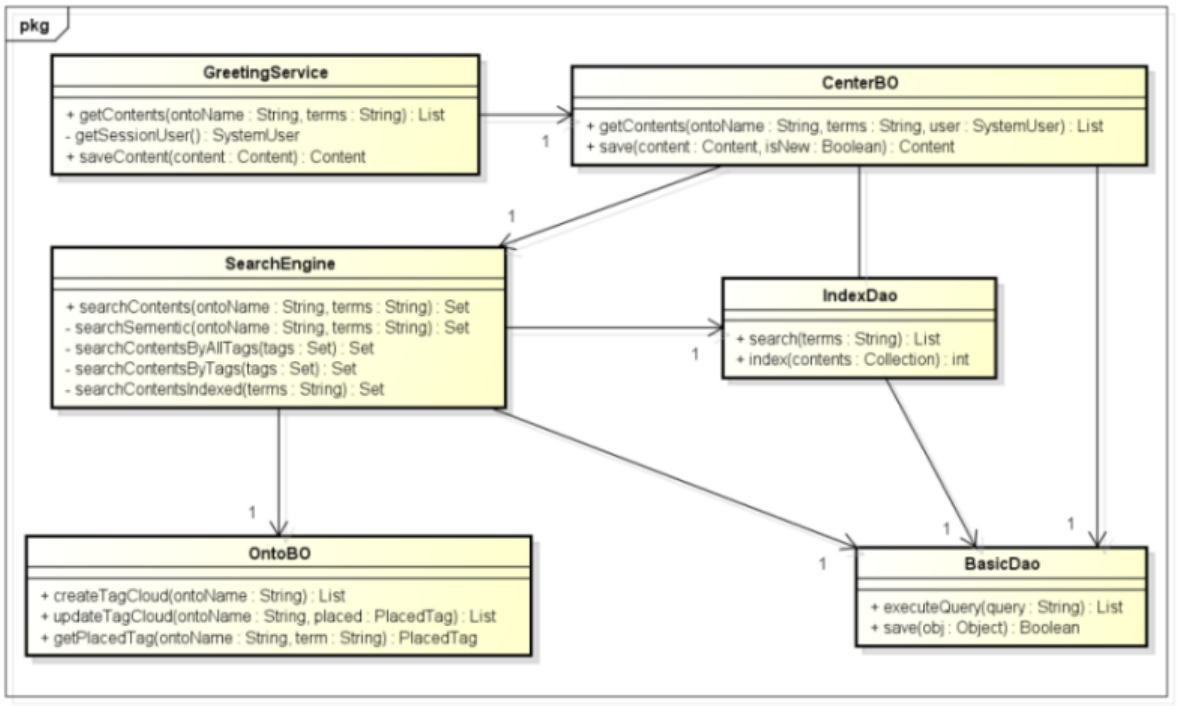


Figura 4.12: Diagrama de Classes da camada de Aplicação.

A classe *CenterBO* é o Objeto de Negócio (Business Object) central da aplicação, nele estão implementadas as regras de funcionamento do software como fluxo para execução dos métodos do *Ontology Tagging* ou dos métodos necessários para indexar e salvar os novos arquivos adicionados a base. Seus métodos utilizados no processo de busca são:

- o método público *getContents* que recebe 3 (três) parâmetros como entrada é utilizado no processo convencional de busca, onde o usuário está tentando localizar algo, os 2 (dois) primeiros parâmetros são os mesmos do método *getContents* na classe *GreetingService*, já o terceiro é um objeto da classe *SystemUser* utilizado para verificar se o usuário está logado ou não e se este é um administrador.
- o método *save* é utilizado para savar os conteúdos e caso algum desses sejam novos também são indexados.

A classe *OntoBO* é o Objeto de Negócio responsável pela manipulação das ontologias e do componente *Ontology Tagging*. É através desta classe que o motor de busca acessa a ontologia selecionada pelo usuário. Seus métodos mais importantes estão descritos abaixo.

- *createTagCloud*: método responsável por acionar a montagem da nuvem de tags no componente *Ontology Tagging*. Recebe como parâmetro o nome da ontologia selecionada e retorna uma lista de tags e suas posições de inserção na tela.

- *updateTagCloud*: com base uma outra tag contida na nuvem já montada, este método modifica o formato da nuvem posicionando esta mesma tag no centro e suas tags relacionadas a seu redor.

- *getPlacedTag*: este método recebe como parâmetro 2 (duas) váriaveis texto, onde a primeira indica o nome da ontologia selecionada e a segunda um termo, referente a uma tag, que se deseja recuperar na ontologia, é através desse método que os termos de busca são consultados na ontologia para que no motor de busca sejam extraídos os termos sinônimos, hipônimos ou os que contextualizarão a busca.

A classe *SearchEngine* é o motor de busca da aplicação (RQF9). Esta é responsável por montar a *string* de busca baseada na ontologia, acionar a busca no Lucene com base nessa *string* e acionar também a busca através do hibernate no banco de dados pelas tags dos conteúdos (ver Apêndice A.1). Abaixo os principais métodos desta classe que são utilizados no processo de busca.

- *searchContents*: neste método é verificada a existência de conteúdo na variável *ontoName*, caso exista o método *searchSementic* é acionado e caso não exista os métodos *searchContentsByAllTags*, *searchContentsByTags* e *searchContentsIndexed* são executados sem o refinamento da ontologia na *string* de busca.

- *searchSementic*: este é método responsável pela busca baseada em ontologia, nele cada termo informado pelo usuário para a busca é comparado com as classes da ontologia selecionada, cada termo encontrado na ontologia é substituído por uma expressão de acordo a explicação da Seção 4.6, essas expressões são unidas aos termos não pertencentes a ontologia através de conectivos *OR* para formar a *string* de busca que será executada nos métodos *searchContentsIndexed*, *searchContentsByAllTags* e *searchContentsByTags*.

- *searchContentsIndexed*: este método executa a busca pelos conteúdos através do *framework* Lucene.

- *searchContentsByTags*: este método executa a busca pelos conteúdos através das tags vínculadas no banco de dados.

- *searchContentsByAllTags*: este método executa a busca pelos conteúdos através das tags vínculadas no banco de dados, porém só retorna os conteúdos que tiverem vínculos com todas as tags recebidas como parâmetro.

Por fim, as classes que provêm o armazenamento e recuperação dos conteúdos do sistema. A classe *BasicDao* é Objeto de Acesso a Dados basico do software, é ele quem gere a persistência dos dados através do hibernate e é ele também que realiza a busca pelas tags,

baseada em Folksonomia. Já a *IndexDao* é a responsável pela manipulação do *framework* Lucene. É através dela que os arquivos são indexados antes de serem armazenados na base de arquivos e que as buscas baseadas em indexação de arquivos são realizadas. Vale ressaltar que esta classe também utiliza uma instância da classe *BasicDao*, pois após a realização da busca no Lucene os documentos são recuperados na base de dados para que suas informações apareçam no resultado de busca que vai para o usuário.

Na Figura 4.13 temos um complemento do diagrama de classes anterior onde o foco é o componente *Ontology Tagging*. Nesse diagrama a classe OntoBO reaparece, pois é a responsável pela manipulação da ontologia escolhida pelo usuário e pela manipulação do *Ontology Tagging*.

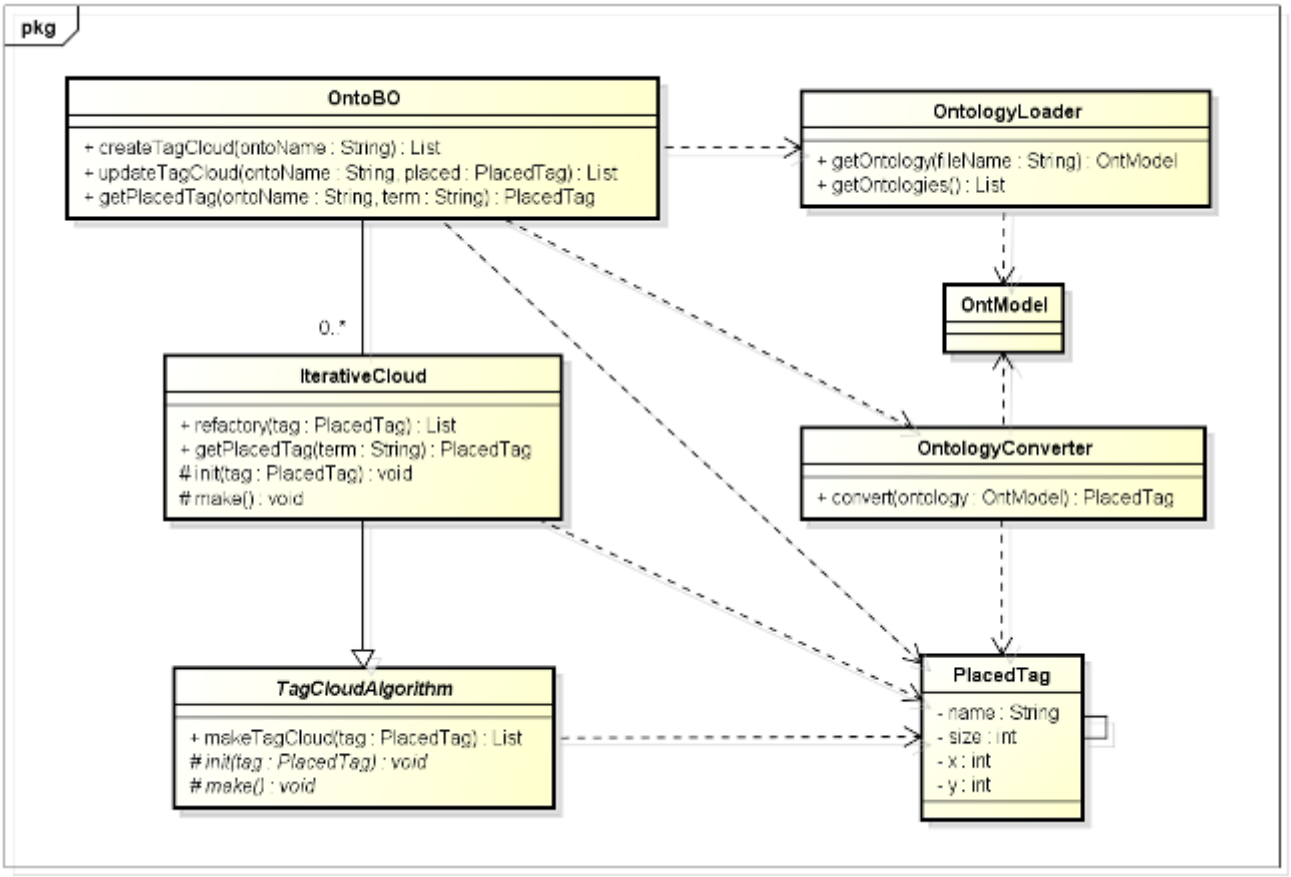


Figura 4.13: Diagrama de classes do componente *Ontology Tagging*.

A classe *OntModel* é a classe do *framework Jena* que representa uma ontologia. É através dessa classe que os elementos da ontologia escolhida serão acessados. Já a classe *PlacedTag* representa as tags que serão dispostas na nuvem de tags. Ela possui atributos como nome (*name*), tamanho (*size*), posição (x, y) e um auto-relacionamento para indicar quais conceitos da ontologia estão relacionadas diretamente, já que é esta classe que representa um conceito da ontologia na nuvem de tags.

A classe *OntologyLoader* é responsável pela base de ontologias, informando a lista de ontologias armazenadas na base ou carregando, via *framework Jena*, a Ontologia selecionada. Seus métodos são:

- *getOntology* tem como função carregar a ontologia escolhida pelo usuário, ele recebe o nome da ontologia como parâmetro e retorna uma instância da classe OntModel;
- *getOntologies* tem como função informar a lista de nomes das ontologias armazenadas na base de ontologias.

A classe *OntologyConverter* é a responsável por converter uma ontologia de domínio em uma lista de tags presevando o relacionamento entre os conceitos da ontologia. Para cada conceito convertido uma instância da classe PlacedTag é gerada. Isso é feito através do método *convert* que recebe uma instância da classe *OntModel* como parâmetro e retorna uma lista de instâncias da classe *PlacedTag*.

A *TagCloudAlgorithm* é uma classe abstrata utilizada para implementação do padrão de projeto *Template Method*, que determina os passos a serem seguidos para construção da nuvem de tags (ver o Apêndice A.2). Para se implementar um novo algoritmo dentro do componente *Ontology Tagging* é necessário implementar uma nova classe que herde da *TagCloudAlgorithm*. Os seus métodos principais são os que compõem o padrão *Template Method*.

- *init*: é um método abstrato onde devem ser implementadas as ações necessárias antes do ínicio da montagem da nuvem de tags, como instânciar a lista de resultados, ou qualquer outro procedimento necessário antes da montagem da nuvem. Ele recebe como parâmetro uma instância da classe *PlacedTag* e não retorna nada.
- *make*: é um método abstrato onde devem ser implementado o processo de montagem da nuvem, não recebe parâmetros e não retorna nada, apenas deve adicionar as tags da nuvem numa lista de resultados, que é uma lista de instâncias da classe *PlacedTag*.
- *makeTagCloud*: é o responsável por executar os dois métodos anteriores (*init* e *make*), sendo uma implementação do padrão de projeto *Template Method* (GAMMA et al., 2000), este padrão define a estrutura de executação de um algoritmo deixando para a subclasse alguns passas da execução, neste caso os métodos *init* e *make* são implementados na subclasse (*IterativeCloud*), porém a ordem de sua execução é definida neste método (*makeTagCloud*). Este método recebe uma instância da classe *PlacedTag* como parâmetro e retorna a lista de resultados pronta, já com as tags e suas posições atribuídas.

A classe *IterativeCloud* é a nova implementação da classe abstrata *TagCloudAlgorithm* feita nessa pesquisa para gerar a nuvem de tags iterativa, que se modifica de acordo a escolha do usuário (ver o Apêndice A.2). Seus principais métodos, além dos dois abstratos exigidos na classe pai, são:

- *refactory*: é responsável por remontar a nuvem tendo como base uma outra tag escolhida pelo usuário, recebe um termo (*string*) como parâmetro e retorna uma lista de *PlacedTag*;
- *getPlacedTag*: é responsável por verificar a existência de um conceito dentro da ontologia, recebe um termo (*string*) e retorna uma instância de *PlacedTag* caso o termo passado seja um conceito da ontologia.

A ferramenta Apache Lucene, escolhida para inserir a indexação automática de arquivos na aplicação, é uma biblioteca de Recuperação de Informação de alto desempenho. Ela é um projeto livre de código aberto, escrito em Java e mantido pela fundação Apache. O Lucene utiliza técnicas de indexação de arquivos para realizar suas buscas. Esse tipo de técnica é utilizada na maioria dos sites de busca atualmente na web (HATCHER; GOSPODNETIC, 2005).

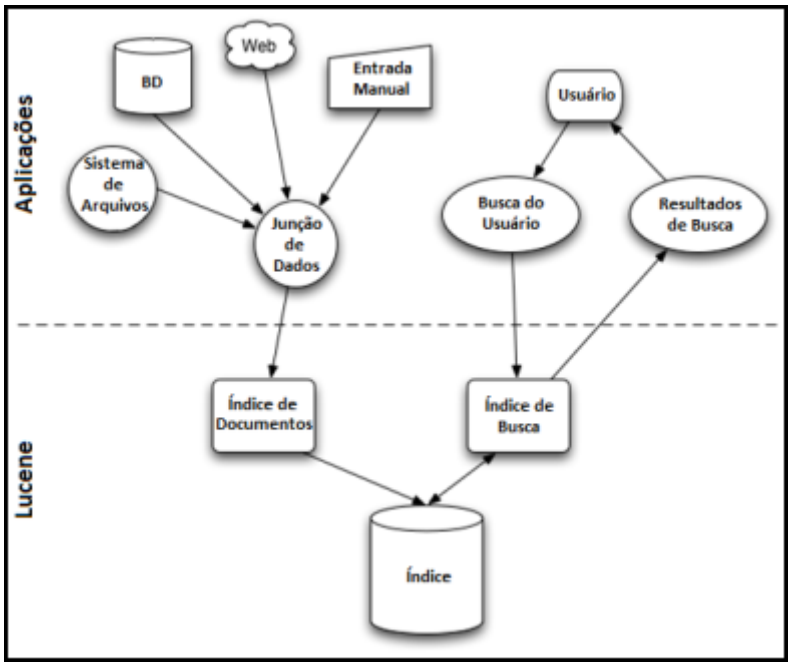


Figura 4.14: Arquitetura do Lucene. Fonte: adaptada de HATCHER e GOSPODNETIC (2005).

O Lucene indexa os dados armazenados pela aplicação que podem estar em bases de dados distintas, gerando uma base de índices onde serão feitas as buscas do usuário (Figura 4.14). A gerência dos dados e a visualização do conteúdo retornado para o usuário ficam a cargo da aplicação que estiver utilizando o Lucene.

O Algoritmo A.3 foi implementado na classe *SearchEngine* para montar a *string* de busca necessária à realização dos 4 (quatro) tipos de busca baseados em Ontologia, apresentados na Seção 4.6, através do método *searchSementic*. Vale ressaltar que caso não exista uma ontologia selecionada esse método não é executado, nesse caso a busca é feita sem o tratamento empregado nele.

O método *searchSementic* recebe como parâmetro o nome do arquivo OWL, referente a ontologia selecionada pelo usuário, e os termos digitados pelo usuário no campo de busca da aplicação. Esse método retorna uma lista dos conteúdos encontrados na busca.

Nas linhas 2,3,4 e 6 o algoritmo instância ou inicializa as variáveis necessários para a sua execução. A variável *result* se refere ao resultado da busca, é a lista de conteúdos encontrados. A variável *tags* se refere às tags que serão buscadas no banco de dados, a variável *query* é a query montada para a busca feita no *framework* Lucene. Já a variável *array* é a separação, a partir dos espaços, dos termos digitados pelo usuário para a busca.

O primeiro comando **for**, que se inicia na linha 7 e vai até a linha 102, serve para executar o bloco de comandos (de 8 à 101) uma vez para cada termo pedido pelo usuário. A *string* de busca é montada dentro dessas iterações a medida que cada termo é analisado. A variável *terms*, declarada na linha 8, é utilizada para armazenar a *string* de busca parcial montada para cada termo da variável *array* (termos solicitados na busca). A variável *term* (linha 10) armazena apenas o termo da busca ou algum sinônimo ou hipônimo seu, por serem considerados equivalentes nesse tipo de busca. O método *treatString* (linha 10) remove todos os acentos de uma palavra e converte todas as suas letras em minúsculas. Nas linhas 13 e 14, é verificada a existência do termo de busca dentro da ontologia e caso não exista a *string* de busca parcial, a variável *terms*, será igual ao termo buscado, a variável *term*. Caso exista, alguns dos 4 tipos de buscas seão implementados.

A variável *relateds*, declarada na linha 17, armazenará todos conceitos que possuem uma relação com termo buscado, a variável*term*, mais o conceito mais alto na hierarquia da ontologia e os conceitos que possuam uma relação com este, a variável *directs*, declarada na linha 18, armazenará os conceitos que possuem uma relação com termo buscado porém, sem os equivalêntes e os filhos na hierarquia. O preenchimento das variáveis *relateds* e *directs* é feito com base nas relações do termo buscado (linhas 21 à 59), a variável *relateds* será utilizada para a busca contextualizada **com** o termo buscado (linhas 74 à 82), já a variável *directs* será utilizada para a busca contextualizada **sem** o termo buscado (linhas 50 à 59). Da linha 48 a linha 58, é adicionado na variável *relateds* o conceito raíz da ontologia e os conceitos relacionados a ele. A partir da linha 62 a *string* de busca parcial, feita para cada termo buscado, começa a ser montada.

Nas linhas 62 à 72, o primeiro bloco da *string* de busca começa a ser montado, o termo

buscado é contextualizado por seus termos equivalêntes e por seus filhos na hierarquia da ontologia (eg.: termo AND (equivalente filho)). Da linha 74 à 82, é feito o sengundo bloco, onde o termo buscado juntamente com seus termos equivalêntes e seus filhos são contextualizados pelos termos armazenados na variável *relateds* (eg.: (termo equivalente filho) AND (relacionado1 relacionado2 relacionado3)). Nas linhas 85 à 98, é montado o bloco da busca contextualizada **sem** o termo buscado, nesse bloco os termos armazenados na variável *directs* unidos pelo conectivo *OR* formam a *string* (eg.: directs1 AND directs2 AND directs3).

Para montar a *string* de busca final, as *strings* parciais são unidas pelo conectivo *OR* e a busca é executada através dos métodos *searchContentsByAllTags*, *searchContentsByTags* e *searchContentsIndexeds* (linhas 99 à 114).

```
1  private Set<Content> searchSementic(String ontoName, final String termsQuery) {
2     final Set<Content> result = new LinkedHashSet<Content>();
3     final Set<Tag> tags = new HashSet<Tag>();
4     String query = "";
5
6     final String[] array = termsQuery.split(" ");
7     for (int i = 0; i < array.length; i++) {
8        String terms = "";
9
10       String term = this.treatString(array[i]);
11       tags.add(new Tag(term));
12
13       PlacedTag tag = this.onto.getPlacedTag(ontoName, term);
14       if (tag == null) {
15          terms += term;
16       } else {
17          Set<PlacedTag> relateds = new HashSet<PlacedTag>();
18          Set<PlacedTag> directs = new HashSet<PlacedTag>();
19
20          // Preenchendo as variáveis relateds e directs com as relações do termo
21          if (tag.getRelateds() != null) {
22             Set<PlacedTag> equivalents = tag.getRelateds().get(PlacedTag.
                  EQUIVALENT);
23             Set<PlacedTag> subClasses = tag.getRelateds().get(PlacedTag.
                  superClassOf);
24
25             for (Set<PlacedTag> set : tag.getRelateds().values()) {
26                for (PlacedTag pt : set) relateds.add(pt);
27                directs.addAll(relateds);
28             }
29
30             if (equivalents != null) {
31                for (PlacedTag pt : equivalents) {
32                   directs.remove(pt);
33                   String t = this.treatString(pt.getName());
34                   tags.add(new Tag(t));
35                   term += " "+ t;
```

```
36                  }
37              }
38              if (subClasses != null) {
39                  for (PlacedTag pt : subClasses) {
40                      directs.remove(pt);
41                      String t = this.treatString(pt.getName());
42                      tags.add(new Tag(t));
43                      term += " "+ t;
44                  }
45              }
46          }
47
48          // Preenchendo a variável relateds com as relações do conceito raíz da
                ontologia
49          PlacedTag rootTag = this.onto.getPlacedTag(ontoName, ontoName.substring
                (0, ontoName.length()-4));
50          if (rootTag != null) {
51              relateds.add(rootTag);
52              if ((rootTag.getRelateds() != null) && (rootTag.getRelateds().size() >
                    0)) {
53                  for (Set<PlacedTag> set : rootTag.getRelateds().values()) {
54                      for (PlacedTag pt : set) {
55                          relateds.addAll(this.getTagEquivalents(pt));
56                      }
57                  }
58              }
59          }
60
61          // Busca contextualizada COM o termo buscado
62          String[] array2 = term.split(" ");
63
64          String closeEnd = "";
65          if (array2.length > 1) {
66              terms = "(("+array2[0]+") AND (";
67              for (int y = 1; y < array2.length; y++) {
68                  terms += " "+array2[y];
69              }
70              terms += ")) OR (";
71              closeEnd = ")";
72          }
73
74          terms += "(("+term+") AND (";
75
76          if ((relateds != null) && (relateds.size() > 0)) {
77              for (PlacedTag pt : relateds) {
78                  String name = this.treatString(pt.getName());
79                  if (!term.contains(name)) terms += " "+ name;
80              }
81          }
82          terms += "))"+closeEnd;
83
84          // Busca contextualizada SEM o termo buscado
```

```
85            if ((directs != null) && (directs.size() > 1)) {
86               if (!rootTag.getName().equalsIgnoreCase(term)) {
87                  terms += " OR (" + "("+this.getStringEquivalents(rootTag)+")";
88               } else {
89                  PlacedTag pt = (PlacedTag) directs.iterator().next();
90                  terms += " OR (" + "("+this.getStringEquivalents(pt)+")";
91                  directs.remove(pt);
92               }
93               for (PlacedTag pt : directs) {
94                  String d = "("+this.getStringEquivalents(pt)+")";
95                  terms += " AND "+ d;
96               }
97               terms += ")";
98            }
99            terms = "("+ terms +")";
100        }
101        query += " "+ terms +" OR";
102     }
103
104     query = query.trim();
105     if (query.endsWith(" OR")) query = query.substring(0, query.length()-3);
106     System.out.println(query +" = "+ tags);
107
108     result.addAll(this.searchContentsByAllTags(tags));
109     result.addAll(this.searchContentsByTags(tags));
110     result.addAll(this.searchContentsIndexed(query));
111     return result;
112  }
```

Listing 4.1: Algoritmo para Montagem da Busca baseada em Ontologia

Para ver o código fonte da classe *SearchEngine* na integra verificar o Apêndice A.1.

4.8 Avaliação MOFI

Para validar a implementação do MOFI, o software Goon, foram feitas algumas consultas a uma base com 43 (quarenta e três) arquivos de tipos diversos, pdf, zip, doc, imagens e escritos nas línguas portuguesa e inglesa. Alguns dos arquivos inseridos na base foram inseridos propositalmente, como dois textos que falavam sobre futebol que foram extraídos do portal *A Tarde*[1]. O primeiro texto, falava sobre a última partida do Esporte Clube Vitória[2], e o segundo sobre a saída de um dos goleiros do Esporte Clube Bahia[3].

Foram feitas buscas comparativas no sistema *Goon* com e sem uma ontologia selecionanda.

[1] http://www.atarde.com.br/ - Acessado em 29/11/2010.
[2] http://www.atarde.com.br/esporte/noticia.jsf?id=5656365 - Acessado em 29/11/2010.
[3] http://www.atarde.com.br/esporte/noticia.jsf?id=5656423 - Acessado em 29/11/2010.

Sem a ontologia selecionada as *strings* de busca são passadas diretamente aos módulos de indexação, manual e automático, sem a contribuição semântica de uma ontologia. Já com a ontologia selecionada, o *Goon* consegue realizar buscas com um maior poder semântico (e.g. buscas contextualizadas e buscas por sinônimo e por hipônimo).

(a) Classes da Ontologia. (b) Propriedades da Ontologia.

Figura 4.15: Hierarquia de classes e propriedades da Ontologia de Futebol.

A ontologia de domínio utilizada nos experimentos foi desenvolvida nesta pesquisa, e aborda o domínio do Futebol. A Figura 4.15 mostra a hierarquia de classes e de propriedades da ontologia.

Dentre as caracteristicas mais importantes dessa ontologia de futebol, para esses testes, podemos destacar a de equivalência entre *Partida* e *Jogo*, a classe Jogador, que é a classe pai da hierarquia de tipos de jogadores, abaixo dela estão classes como *Jogador_Linha*, *Goleiro*, *Zagueiro* ou *Atacante*, as classes que contextualizaram o termo Futebol como as classes *Evento*, *Jogo*, *Partida*, *Pessoa* e *Time* por serem as classes mais altas da hierarquia da Ontologia, desconsiderando a classe nativa *Thing*, e por fim, as classes que se relacionam com a classe *Time* através de propriedades como as classes *Evento*, *Arbitro*, *Tecnico Gol* ou *Jogo*. O código fonte dessa ontologia está no Apêndice A.3.

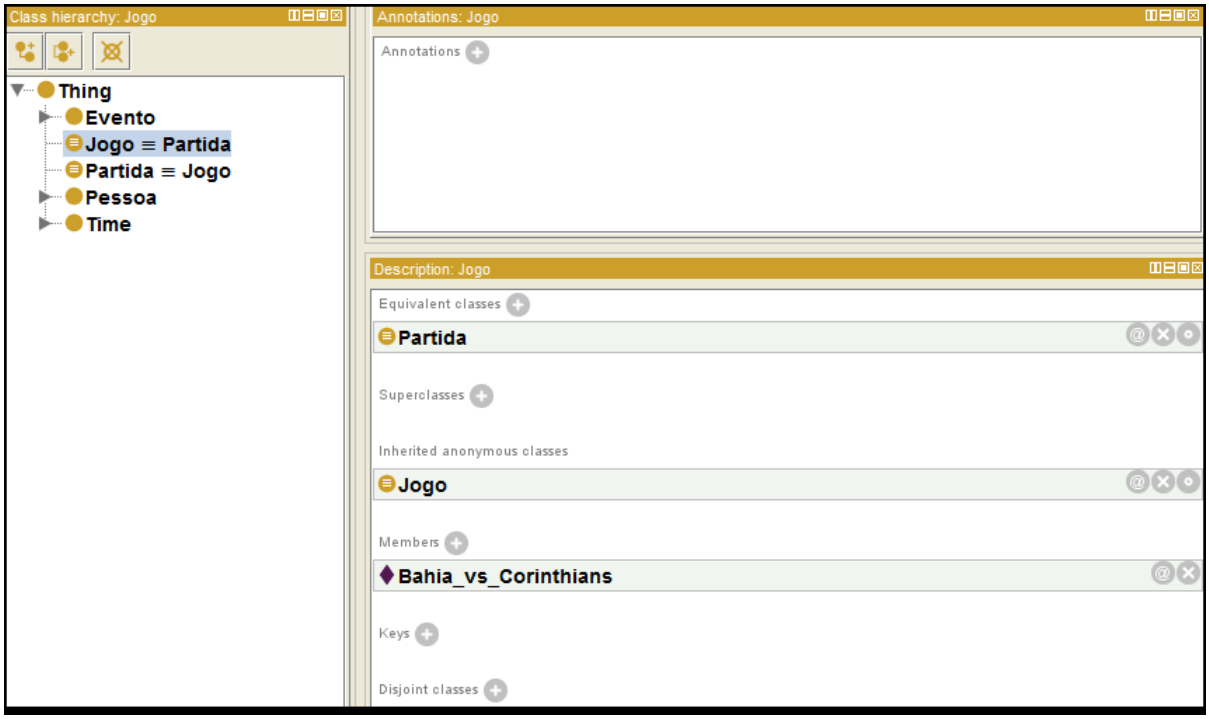


Figura 4.16: Relação de Equivalência entre as classes *Partida* e *Jogo*.

A avaliação foi feita com base nos 4 tipos de busca apresentados na Seção 4.6. Foram feitas 2 (duas) buscas para cada um desses tipos no sistema *Goon*, com e sem uma ontologia seleciona, e em alguns casos foram feitas comparações na ferramenta de busca mais utilizada da web atualmente, o *Google*[4].

Para avaliar o tipo de **busca contextualizada com o termo buscado** foram feitas duas pesquina no sistema *Goon* pela palavra *time*. Sem uma ontologia selectionada (Figura 4.17) o *Goon* generalizou as buscas e retornou mais de 10 (dez) documentos, sendo que os primeiros lugares eram arquivos de relatórios escritos em lingua inglesa, e que têm a palavra *time* (tempo em inglês) e no fim da lista alguns textos sobre times de futebol. Um resultado parecido foi encontrato no exemplo ilustrado pela Figura 2.2 na Seção 2.4. Onde foram retornados alguns sites escritos em ligua inglesa, e outros escritos em português se referindo a *time* no contexto do futebol.

Na segunda busca realizada pela palavra *time* foi selecionada a ontologia de futebol no Goon e resultados foram diferentes (Figura 4.18). Apenas 4 (quatro) documentos foram retornados sendo que os 3 (três) primeiros da lista têm textos escritos sobre times de futebol.

Os 2 primeiros arquivos, *brasil.txt* e *gol anulado.txt*, foram texto escritos livremente por este autor para testar a ferramenta e contêm os seguintes textos:

[4]http://www.google.com.br/ - acessado em 29/11/2010

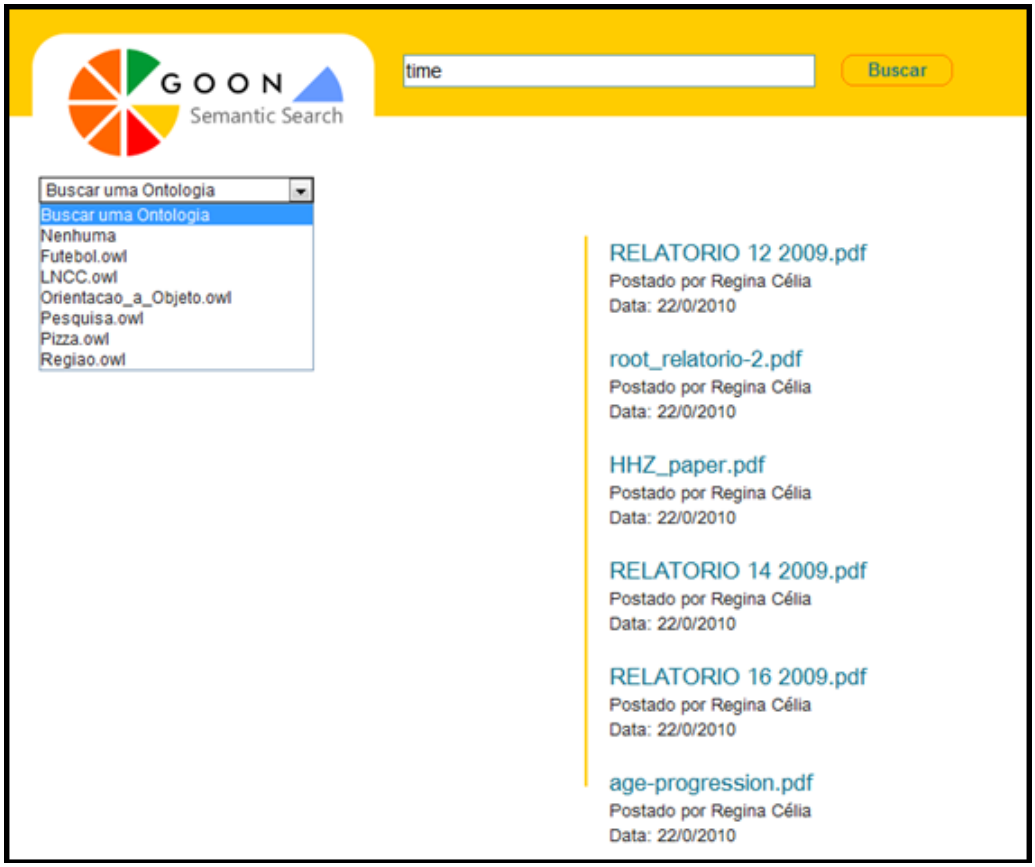


Figura 4.17: Busca feita no Goon pela palavra time sem uma ontologia selecionada.

- **brasil.txt**: *Ronaldo é jogador de futebol do time do corinhtians e pretende ser tecnico quando encerrar a carreira.*
- **gol anulado.txt**: *O Corithians, na pessoa do seu vice presidente, informou que vai entrar com um processo contra os arbitros que apitaram o último jogo do time, por terem anulado um gol legítimo do jogador Elias.*

O terceiro arquivo encontrado é um dos textos extraídos do portal *Atarde.com*, que é a notícia sobre o último jogo do time E. C. Vitória:

- **noticia_vitoria.txt** *Em partida complicada, o Vitória empatou em 1 a 1 com o Internacional neste domingo, 28, no Beira-Rio, pela 37ª rodada do Brasileirão, e depende apenas de um triunfo contra o Atlético-GO na última rodada para não cair à Série B. O atacante Adailton marcou o gol do rubro-negro e Rafael Sobis empatou para a equipe gaúcha.*

  Com o resultado, o Leão chegou aos 41 pontos, mas permanece em 17º lugar, na zona de rebaixamento, pois o Atlético-GO empatou com o São Paulo e também atingiu o mesmo número de pontos, superando o time baiano na quantidade de triunfos (11 a

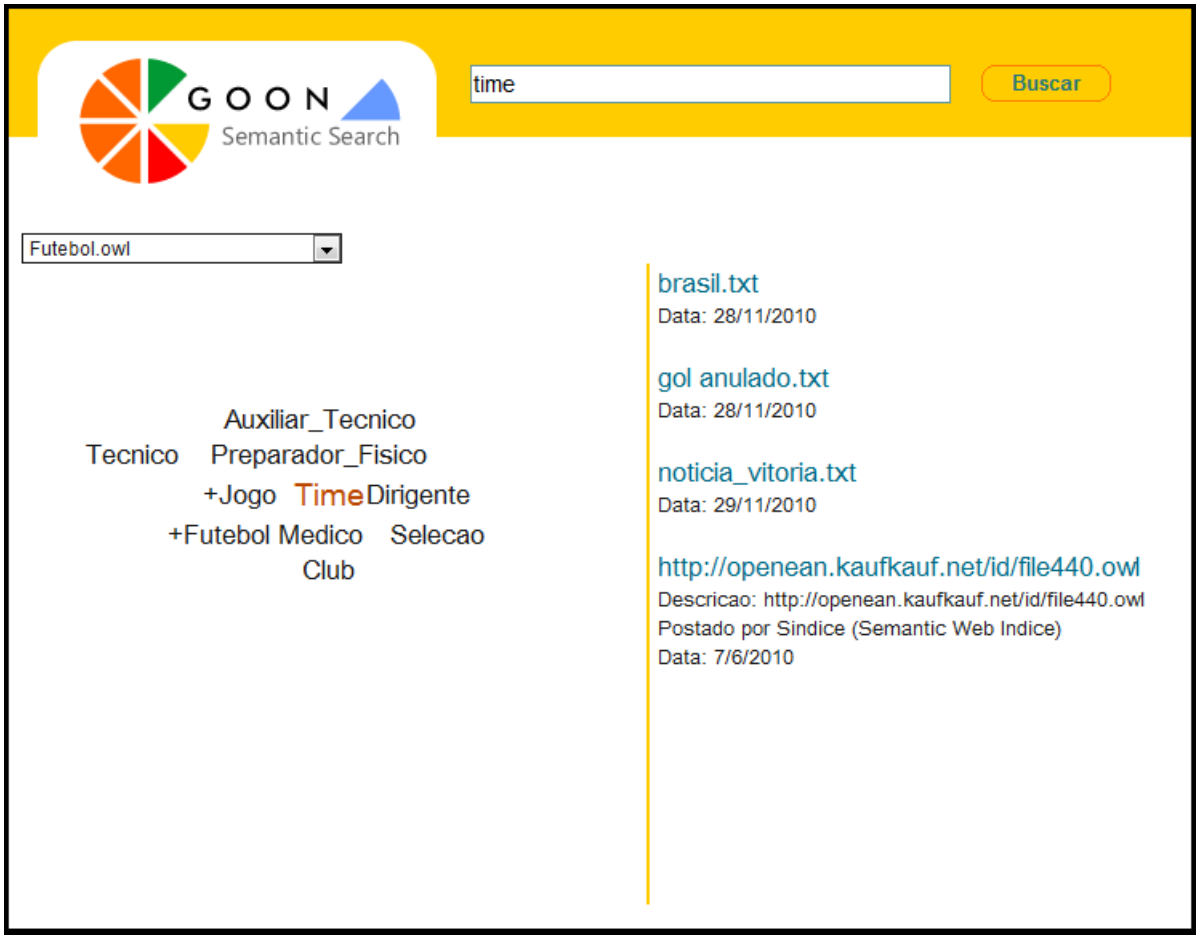


Figura 4.18: Busca feita no Goon pela palavra time com uma ontologia selecionada.

9).

Avaí, Atlético-MG e o Flamengo, que perdeu para o Cruzeiro, estão livres da degola. Quem não teve a mesma sorte foi o Guarani, rebaixado após a derrota para o Grêmio. Além do time paulista, Prudente e Goiás também estão na Série B de 2011.

A última vaga do Z4 será decidida no próximo domingo, 5, quando o rubro-negro enfrenta o Atlético-GO no Barradão e precisa vencer a partida para rebaixar a equipe goiana e se livrar da queda.

O jogo — O Internacional teve uma maior posse de bola durante todo o primeiro tempo, mas não conseguiu criar jogadas perigosas de ataque. O Vitória jogou bem na defesa e soube equilibrar a partida.

Com isso, o Inter não achou espaços para atacar em velocidade e tentou pressionar com jogadas aéreas. Em escanteio cobrado por D'Alessandro ainda no início do jogo, o zagueiro Índio desviou de cabeça para o gol, mas Viáfara estava bem colocado para efetuar a defesa.

O único lance perigoso de ataque do Colorado na primeira etapa só aconteceu aos

40 minutos, quando o atacante Alecsandro chutou forte de fora da área e exigiu uma bela defesa de Viáfara, que espalmou para escanteio.

No intervalo, o técnico Antônio Lopes colocou o volante Fernando no lugar de Elkeson e o Vitória voltou melhor para a segunda etapa. Logo aos cinco minutos, o atacante Adailton recebeu livre no lado esquerdo da área e chutou de bico, no ângulo direito de Renan para marcar um belo gol.

Após o gol do Leão, o técnico Celso Roth colocou Andrezinho e Giuliano nos lugares de Tinga e D'Alessandro, respectivamente, para deixar a equipe gaúcha mais ofensiva. Aos 16, as alterações deram resultado e o Inter empatou a partida com Rafael Sobis. O atacante recebeu bola do lado esquerdo da área e chutou forte para marcar um belo gol.

A equipe colorada foi com tudo para o ataque e pressionou o rubro-negro. No último lance do jogo, Rafael Sobis chutou cruzado da direita e o volante Vanderson, que entrou no lugar de Bida, afastou o perigo na pequena área para garantir o empate.

Utilizando os termos encontrados na ontologia de domínio o *Goon* pode contextualizar a busca e retornou documentos mais ligados ao contexto especificado de times de Futebol. Para atingir esse resultado a *string* de busca montada pelo *Goon* foi a seguinte:

- *(time AND (selecao club)) OR ((time selecao club) AND (evento arbitro tecnico gol jogo dirigente impedimento falta membro_comissao_tecnica auxiliar_tecnico medico pessoa futebol partida preparador_fisico jogador)) OR (futebol AND dirigente AND tecnico AND auxiliar_tecnico AND medico AND futebol AND (jogo partida) AND (preparador_fisico) AND (jogador goleiro jogador_linha))*

Apesar de existirem alguns termos unidos por um *underline* (_), esse caracter não é considerado na realização da busca, está presente desta forma na *string* de busca porque assim foi definida na ontologia.

Para testar o tipo de **busca contextualizada sem o termo buscado** também foram feitas duas buscas no Goon. Sem uma ontologia selecionada (Figura 4.19) os resultados foram mais variados do que na busca com a ontologia de futebol selecionada que fez um filtro maior dos documentos.

Na busca com a ontologia selecionada (Figura 4.20) foi retornado um texto sobre futebol que não tinha sido retornado na busca anterior (sem a ontologia selecionada), o arquivo (*gol anulado.txt*) não contém em seu texto a palavra futebol, porém fala sobre o assunto mensionando palavras como *time*, *jogador* e *gol*. O auxilio à ontologia permitiu ao Goon contextualizar a busca mesmo sem utilizar a palavra buscada pelo usuário na *string* de

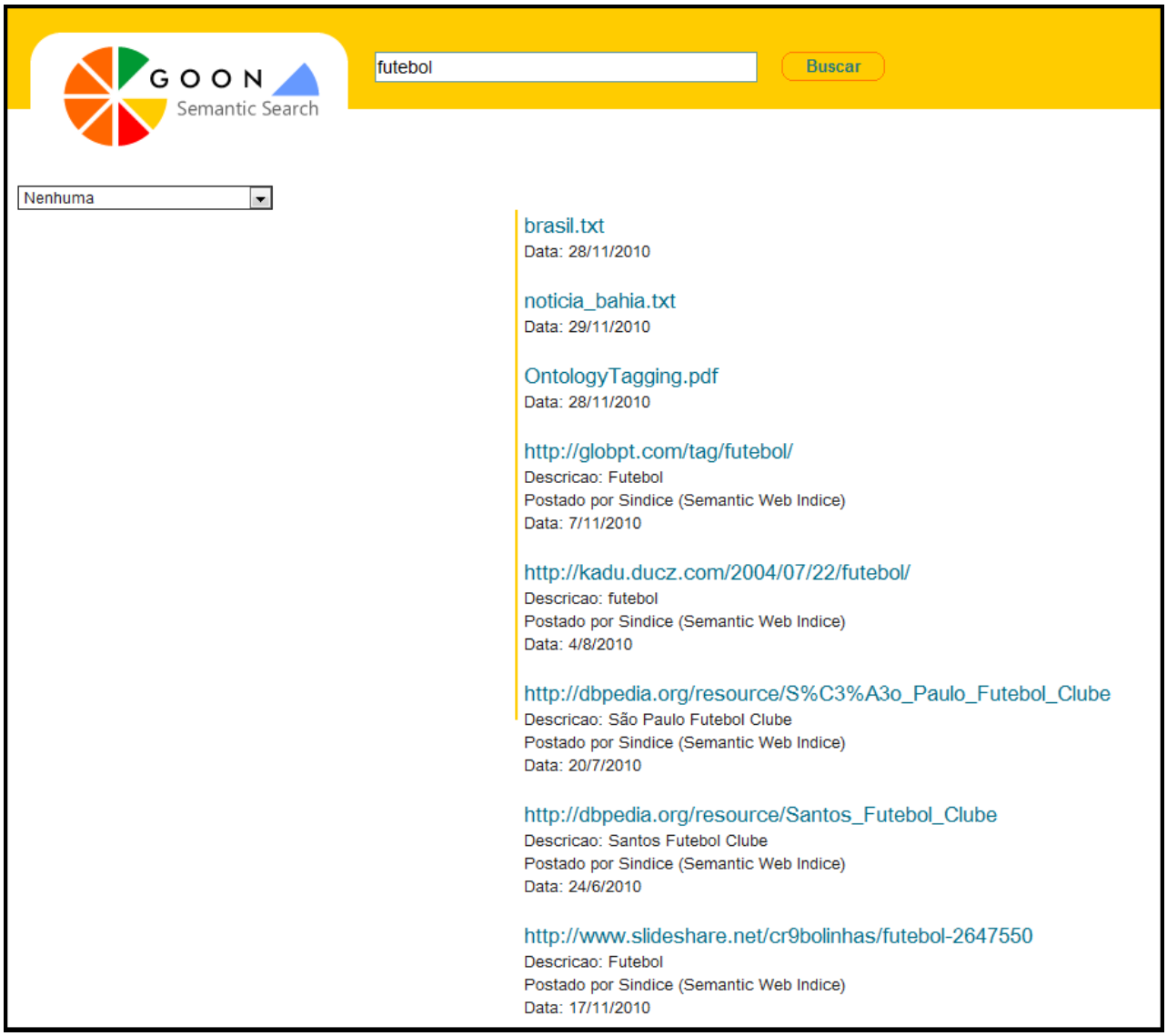


Figura 4.19: Busca feita no Goon pela palavra futebol sem uma ontologia selecionada.

busca passada para os componentes de indexação. Abaixo a *string* de busca montada pelo Goon para essa busca:

- (futebol AND (evento arbitro gol club jogo time impedimento dirigente falta membro_comissao_tecnica selecao pessoa partida jogador)) OR ((evento impedimento falta gol) AND (pessoa dirigente arbitro membro_comissao_tecnica jogador) AND (partida jogo) AND (jogo partida) AND (time selecao club))

Para avaliar a **busca por sinônimo** foram feitas buscas pela palavra *partida*. Sem uma ontologia selecionada os resultados também passaram dos 10 (dez) arquivos dentre eles textos sobre futebol como o *noticia_vitoria.txt*.

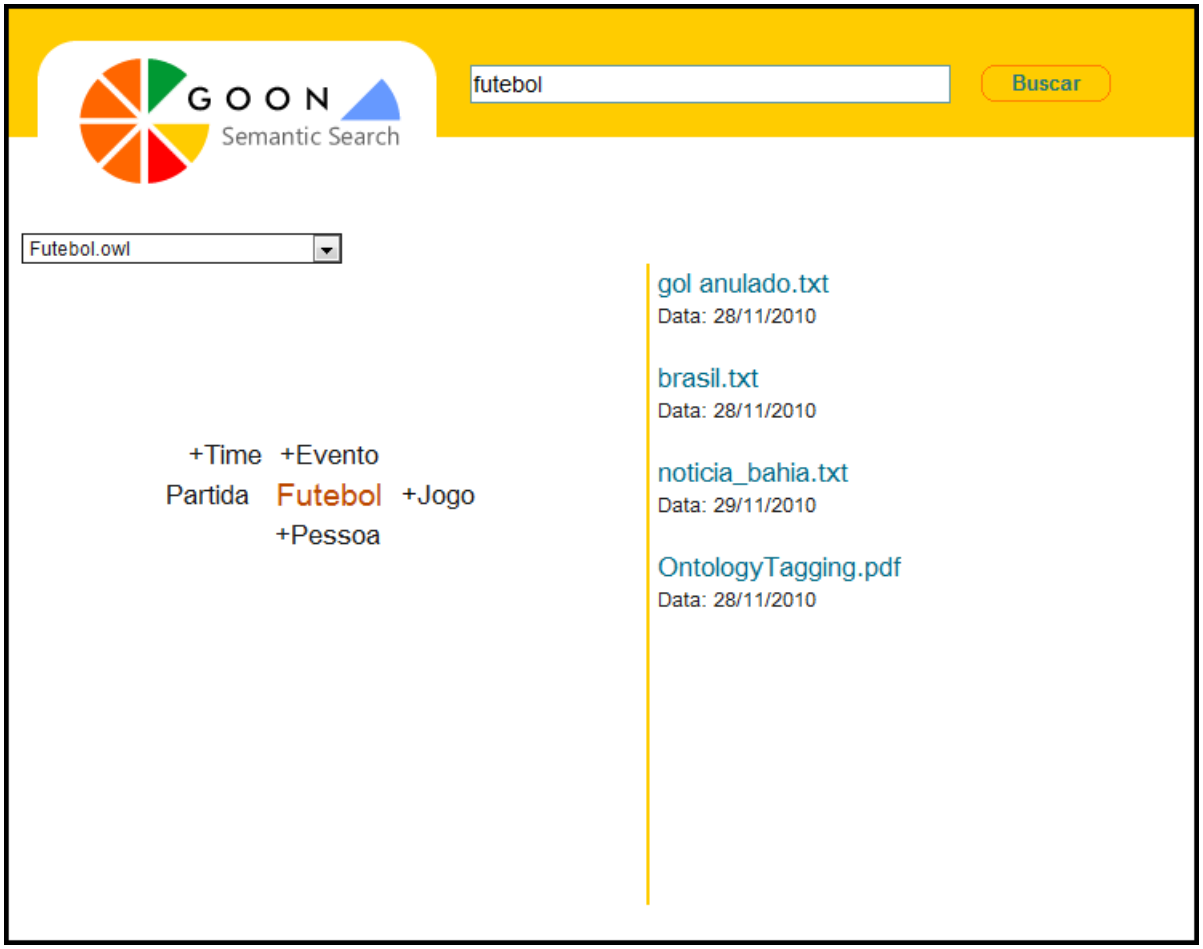


Figura 4.20: Busca feita no Goon pela palavra futebol com uma ontologia selecionada.

Selecionando a ontologia de futebol os resultados novamente foram menores, apenas 4 arquivos retornados, sendo que 3 deles não foram retornados na busca anterior, sem a ontologia selecionada. Estes arquivos, dentre eles o *gol anulado.txt*, não têm a palavra *partida*, mas têm textos sobre futebol como a palavra *jogo* que no contexto de futebol equivale a *partida*.

O arquivo *julgamento.txt* é um arquivo que foi escrito livremente (por este autor) para testes do software, e o arquivo *noticia_bahia.txt* contém um texto que também foi extraído do portal *Atarde.com*, e fala sobre o time do Bahia. Abaixo os textos contidos em cada um deles:

- **julgamento.txt**: *O goleiro Renê foi julgado nesta terça-feira, 19, pelo Superior Tribunal de Justiça Desportiva (STJD), e foi suspenso por um ano após ter sido pego no exame antidoping após o triunfo do seu time, o Bahia, no jogo diante da Portuguesa, válido pela 17ª rodada da competição.*
- **noticia_bahia**: *Ainda falta mais de um mês para a virada do ano e a Série B*

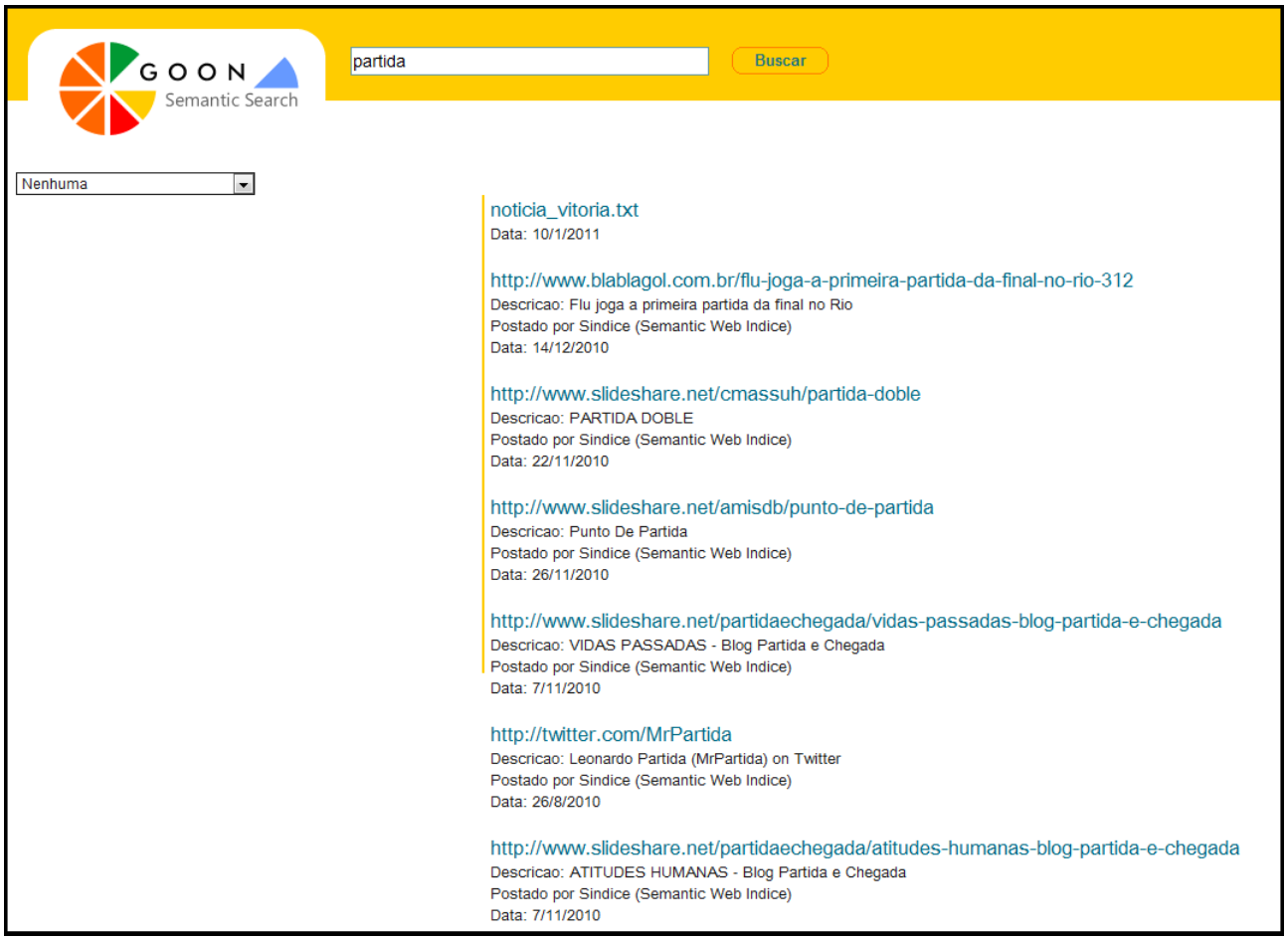


Figura 4.21: Busca feita no Goon pela palavra partida sem uma Ontologia selecionada.

acabou no último sábado, 27. Porém, o torcedor já está preocupado com o destino do Bahia na próxima temporada, em que tenta finalizar jejum de 10 anos sem título baiano, passar pela primeira vez das quartas-de-final da Copa do Brasil e fazer uma campanha digna na volta à Primeira Divisão do Campeonato Brasileiro.

Nesta segunda-feira, 29, às 11 horas da manhã, no Fazendão, o gestor de futebol Paulo Angioni concede sua primeira entrevista coletiva desde que acertou sua permanência. Neste domingo, porém, procurado pela reportagem de A TARDE, não foi encontrado para explicar o motivo para a dispensa do goleiro Fernando, que participou com destaque da reta final da Série B.

Na última sexta, 26, Angioni chamou o ex-paredão tricolor e comunicou a decisão. "Ele disse que o clube não tinha interesse em renovar comigo, pois a folha do início do ano seria pequena", relata Fernando. Discurso que não combina com a promessa de montar um elenco de primeira já para o Estadual. "E meu salário é um dos menores do grupo", acrescenta o goleiro.

Bode expiatório — Sem conseguir encontrar motivo técnico para sua demissão, Fernando acredita que influenciou a confusão por conta do prêmio da subida, que culmi-

Figura 4.22: Busca feita no Goon pela palavra partida com uma Ontologia selecionada.

> *nou com acusações, feitas por parte da imprensa, de corpo mole no jogo contra o Santo André.*
>
> *Ele desabafa: "Tentaram me colocar de bode expiatório depois daquela conversa fiada toda. Desafio qualquer um a provar isso. Peço que me julguem apenas como profissional".*

A *string* de busca deste teste foi a seguinte:

- (partida AND jogo) OR ((partida jogo) AND (evento arbitro gol club time impedimento dirigente falta membro_comissao_tecnica futebol pessoa selecao jogador)) OR (futebol AND (evento impedimento falta gol) AND (time selecao club))

Também foi feita essa mesma busca, pela palavra *partida*, no *Google* e não foram encontrados na primeira página retornos contendo a palavra *jogo*, além da grande quantidade de resultados retornados (Figura 4.23).

No caso do tipo de **busca por hipônimo** os resultados também foram satisfatórios. Na busca feita no *Goon* sem a ontologia selecionada (Figura 4.24) os resultados mais uma vez foram extensos, assim como no *Google* (Figura 4.25).

Os resultados da mesma busca, pela palavra jogador, feita no *Goon* tendo a ontologia

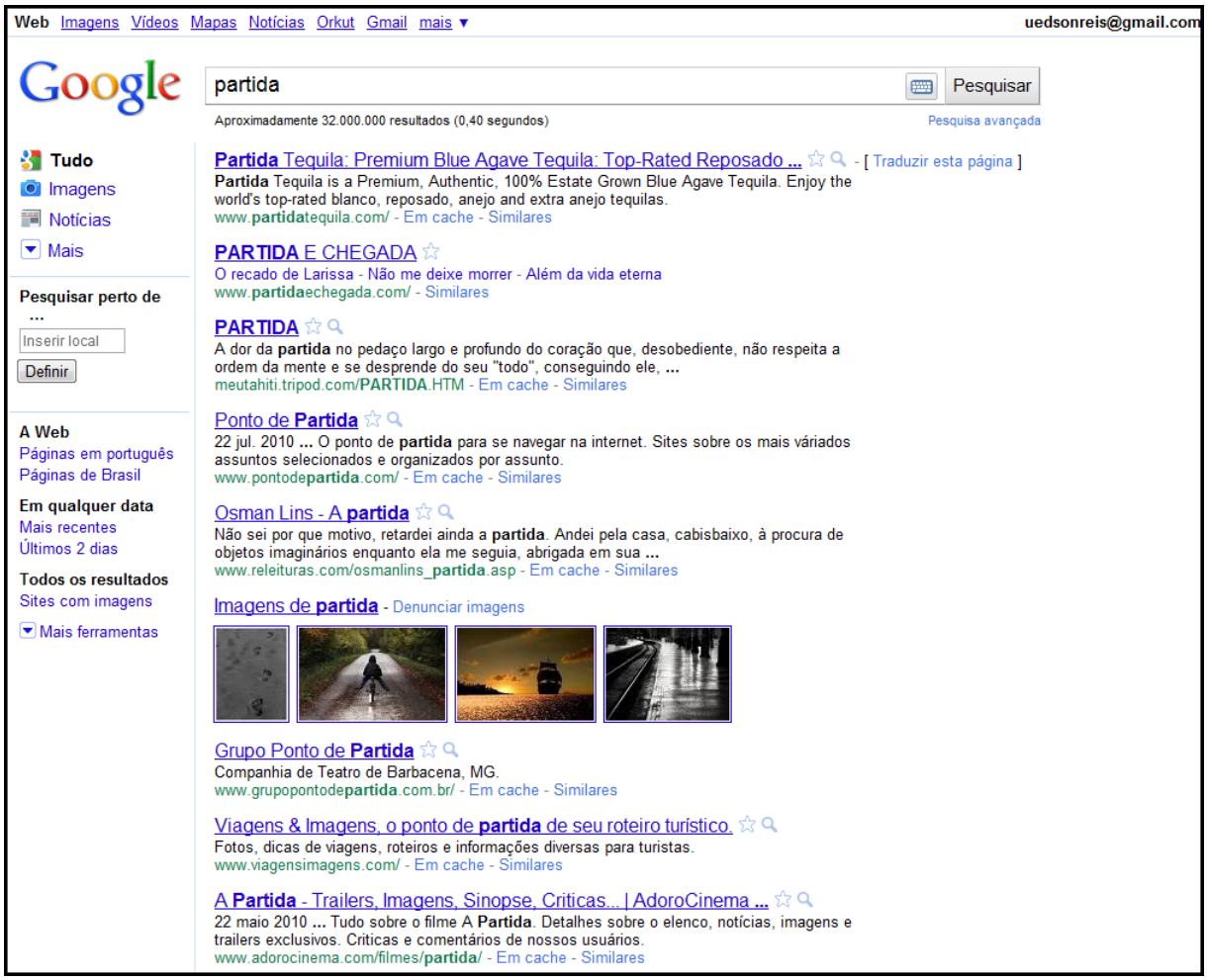


Figura 4.23: Busca feita no Google pela palavra partida.

de futebol selecionada, mais uma vez retornou arquivos não encontrados na busca sem a contextualização de uma ontologia (Figura 4.26). Os arquivos *noticia_bahia.txt* e *julgamento.txt* não têm a palavra *jogador* em seus textos, porém possuem o termo *goleiro* que é um tipo de jogador. Abaixo a *string* de busca montada pelo Goon para essa consulta:

- (jogador AND (goleiro jogador_linha)) OR ((jogador goleiro jogador_linha) AND (evento arbitro club jogo time impedimento dirigente falta membro_comissao_tecnica selecao pessoa futebol partida)) OR (futebol AND (evento impedimento falta gol) AND (pessoa dirigente arbitro membro_comissao_tecnica jogador) AND (time selecao club))

Nesses testes, as buscas feitas no *Goon*, com a ontologia de futebol selecionada, foram mais enxutas e exatas. Consultando a ontologia de futebol o *Goon* foi capaz de filtrar melhor os resultados, acrescentando termos relevantes na *string* de busca focando as buscas em

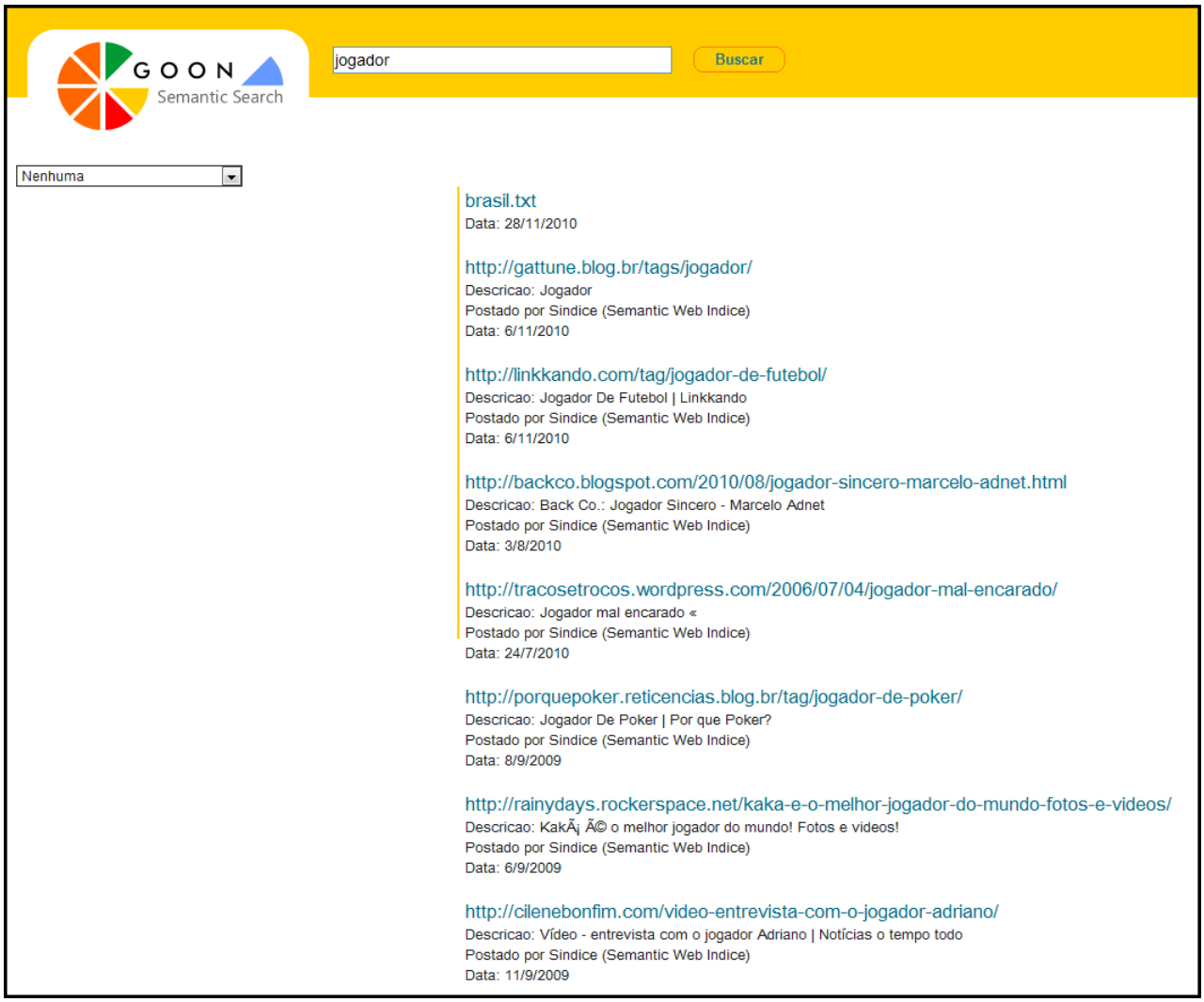


Figura 4.24: Busca feita no Goon pela palavra jogador sem uma ontologia selecionada.

arquivos específicos. Também foi possível, para o *Goon*, utiliza-se das restrições de classes (Seção 3.4) para inferir relações semânticas entre termos do domínio de futebol retornando arquivos que não tinham os termos pedidos na busca, mas que eram semanticamente ligados ao domínio em questão.

Esses resultados foram possíveis por causa da ontologia utilizada que previa tais relações. Caso esta não possuísse a relação de equivalência entre os termos *jogo* e *partida*, por exemplo, os arquivos *noticia_bahia.txt*, *julgamento.txt* e *gol anulado.txt* não seriam encontrados, como mostrado na mesma busca sem uma ontologia selecionada (Figura 4.21). Isso evidência que a qualidade dos resultados do MOFI está relacionada a qualidade da ontologia utilizada. Por esse motivo, a nível de comparação, outras buscas foram feitas no software *Goon* utilizando uma ontologia do domínio Futebol diferente da desenvolvida nesta pesquisa. A ontologia utilizada foi a Futologia, desenvolvida por (MAíRA; SILVA, 2007) em seu projeto de graduação (monografia) apresentado a Universidade Católica do

Figura 4.25: Busca feita no Google pela palavra jogador.

Salvador (UCSal). Na Figura 4.27 é ilustrada a hierarquia de classes dessa ontologia.

Seguindo a ordem dos testes anteriores, essa nova rodada de buscas começa pela palavra *time*, ver Figura 4.28. A *string* de busca montada a partir dessa ontologia só contemplou a busca contextualizada com o termo buscado. O conceito *time*, na ontologia Futologia, não possui filhos em sua hierarquia e também não possui relação de equivalência com outro conceito o que não viabiliza a busca por sinônimo e nem por hipônimo (Figura 4.27). Os conceitos utilizados para contextualizar essa busca são os conceitos mais altos na hierarquia da ontologia, uma vez que o conceito *time* só possui relação com um conceito, conforme visto na Figura 4.29. Por esse mesmo motivo a busca contextualizada sem o termo buscado não foi realizada (esta só é realizada quando existêm mais de 1 (um) conceitos relacionados diretamente com o conceito do termo buscado).

- (time) AND (equipamento_outro tempo arbitragem espectador acao_arbitragem espaço_fisico segundo_tempo morte_subita equipamento pessoa futebol intervalo acao_partida posicao area linha acao_jogador lance prorrogacao goleiro acrescimo equipamento_obrigatorio_jogad do_time primeiro_tempo posicao_campo)

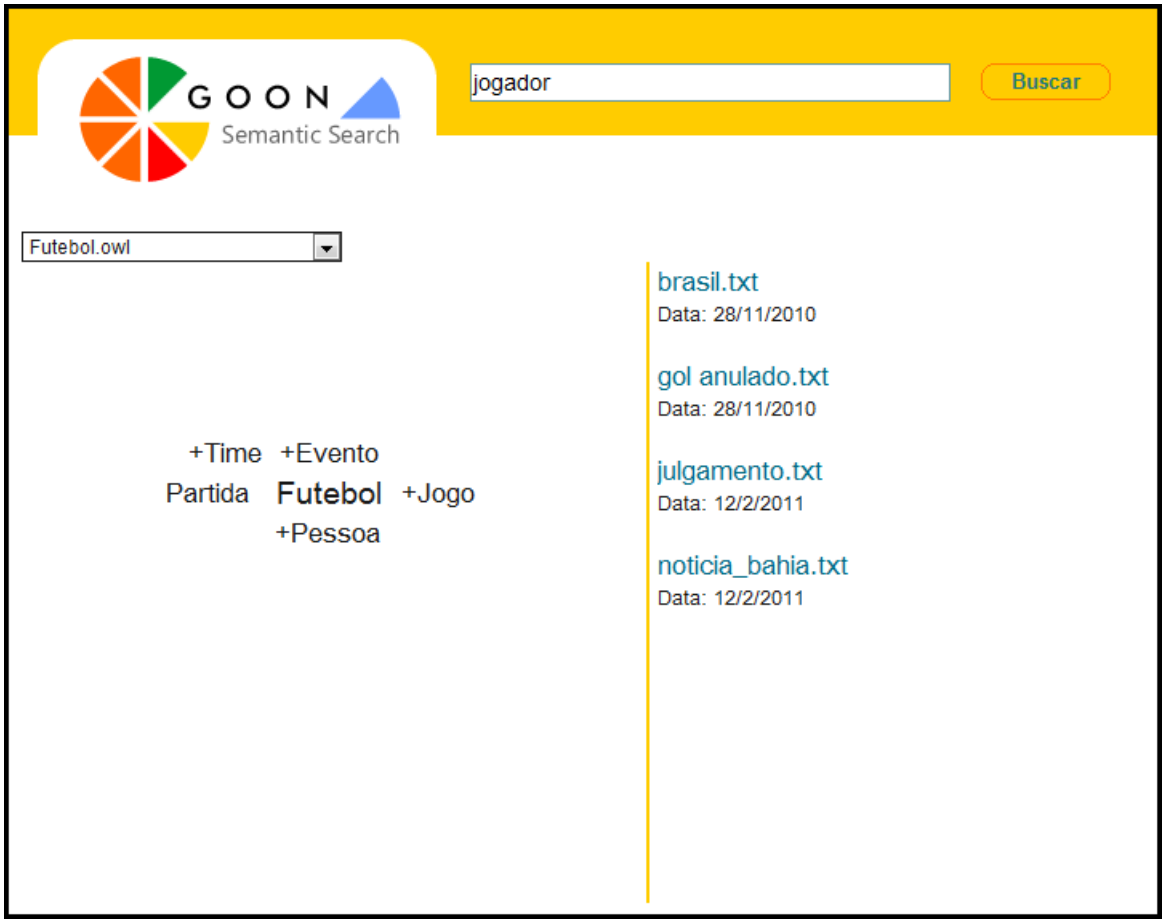


Figura 4.26: Busca feita no Goon pela palavra *jogador* com uma ontologia selecionada.

Neste caso, foram 18 (dezoito) os arquivos retornados, uma quantidade extensa se for considerado o resultado na ontologia anterior, que foram 4 (quatro). Isso ocorreu porque os termos que contextualizaram a palavra *time*, usando a ontologia Futologia, foram mais extensos que na busca anterior. Quanto mais abrangente for a contextualização maior será a quantidade de resultados.

Assim como a *string* de busca montada através da ontologia anterior, a *string* montada através da ontologia Futologia não contemplou a busca por sinônimo nem por hipônimo, uma vez que também neste caso o conceito *Futebol* não possui relação de equivalência com outro conceito e também não possui filhos em sua hierarquia.

- ((futebol) AND (equipamento_outro tempo arbitragem espectador espaço_fisico acao_arbitragem segundo_tempo morte_subita equipamento intervalo pessoa acao_partida posicao area linha lance acao_jogador time prorrogacao goleiro acrescimo equipamento_obrigatorio_jogador do_time primeiro_tempo posicao_campo)) OR ((espaço_fisico area linha) AND (equipamento equipamento_outro equipamento_obrigatorio_jogador) AND (pessoa arbitragem espectador do_time) AND (lance acao_arbitragem acao_jogador acao_partida)

Figura 4.27: Hierarquia de classes da ontologia Futologia.

AND (posicao goleiro posicao_campo) AND (time) AND (tempo prorrogacao acrescimo segundo_tempo morte_subita primeiro_tempo intervalo))

O resultado da busca foi similar, na busca com a ontologia anterior foram retornados 4 (quatro) arquivos, e nesta apareceram 3 (três). Os conceitos da ontologia Futologia não foram eficiêntes no tipo de busca sem termo buscado, pois o arquivo que faltou foi justamente o que não possui o termo *futebol* (*gol anulado.txt*), que foi localizado na busca através da ontologia anterior (Figura 4.20).

A ontologia Futologia não possui o termo *jogo* e por consequência, também não possui uma relação de equivalência entre esse termo e o termo *partida*. Por esses motivos, os arquivos *gol anulado.txt* e *julgamento.txt* nesta busca não foram retornados. Por possuir mais conceitos a ontologia de Futologia mais uma vez teve resultados mais extensos, foram 21 (vinte e um) arquivos contra 4 (quatro) na busca com a ontologia desenvolvida para essa avaliação. Abaixo a *string* montada nesta busca.

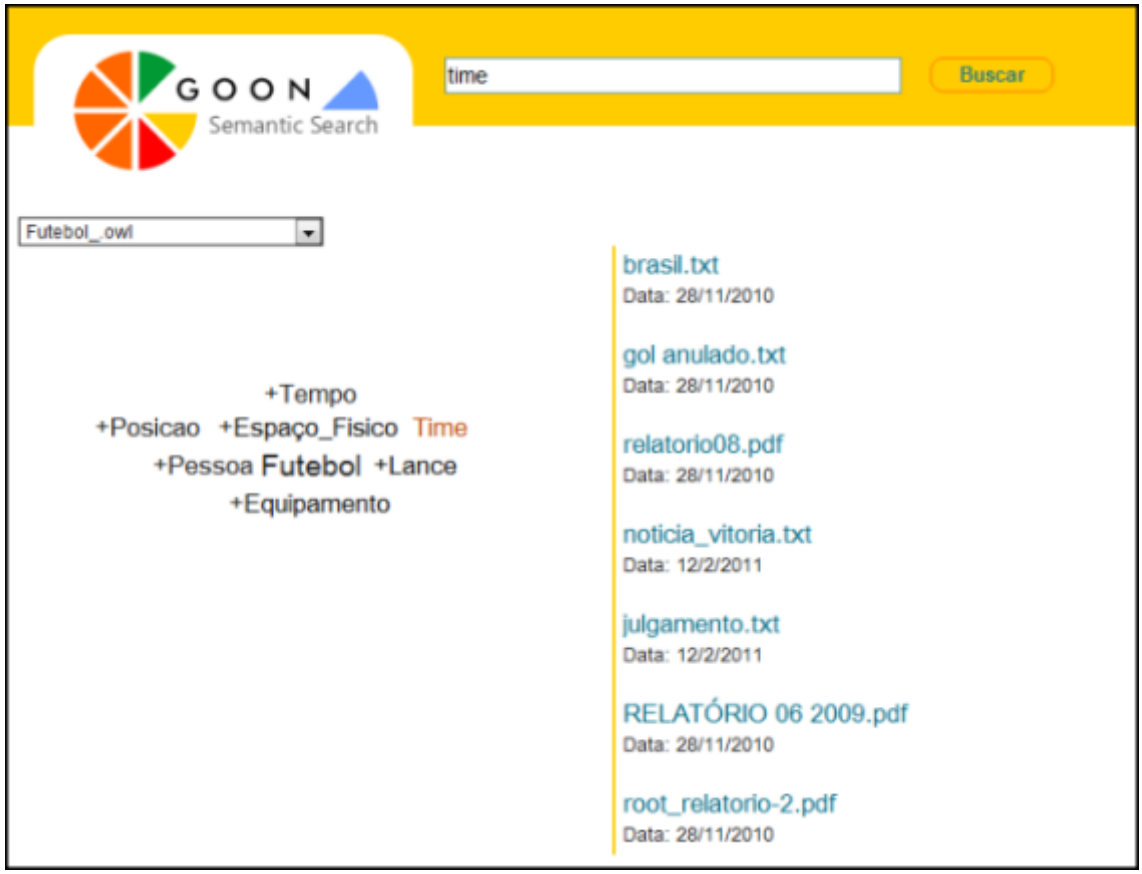


Figura 4.28: Busca feita no *Goon* pela palavra *time* com a ontologia Futologia selecionada.

- ((partida) AND (final inicio)) OR (((partida final inicio) AND (equipamento_outro tempo arbitragem espectador espaço_fisico acao_arbitragem segundo_tempo morte_subita equipamento intervalo futebol pessoa acao_partida posicao area linha lance acao_jogador time prorrogacao goleiro acrescimo equipamento_obrigatorio_jogador do_time primeiro_tempo posicao_campo)))

A busca feita pela palavra *jogador*, foi a única que teve o mesmo resultado da mesma busca feita na ontologia de futebol testada anteriormente. A Futologia também possuia a relação de herança entre os conceitos *jogador* e *goleiro* o que possibilitou a realização da busca por hipônimo e que, consecutivamente, localizou os arquivos *noticia_bahia.txt* e *julgamento.txt*. Esses arquivos não possuem a palavra *jogador* em seus textos, mas possuem a palavra *goleiro* que é um tipo de jogador, por isso a necessidade da busca por hipônimo para localizar esses arquivos. Abaixo a *string* montada para esta busca.

- ((jogador) AND (jogador_goleiro jogador_campo)) OR ((jogador jogador_goleiro jogador_campo) AND (equipamento_outro tempo arbitragem espectador espaço_fisico acao_arbitragem segundo_tempo morte_subita equipamento intervalo pessoa futebol acao_partida posicao area linha acao_jogador lance time prorrogacao acrescimo equi-

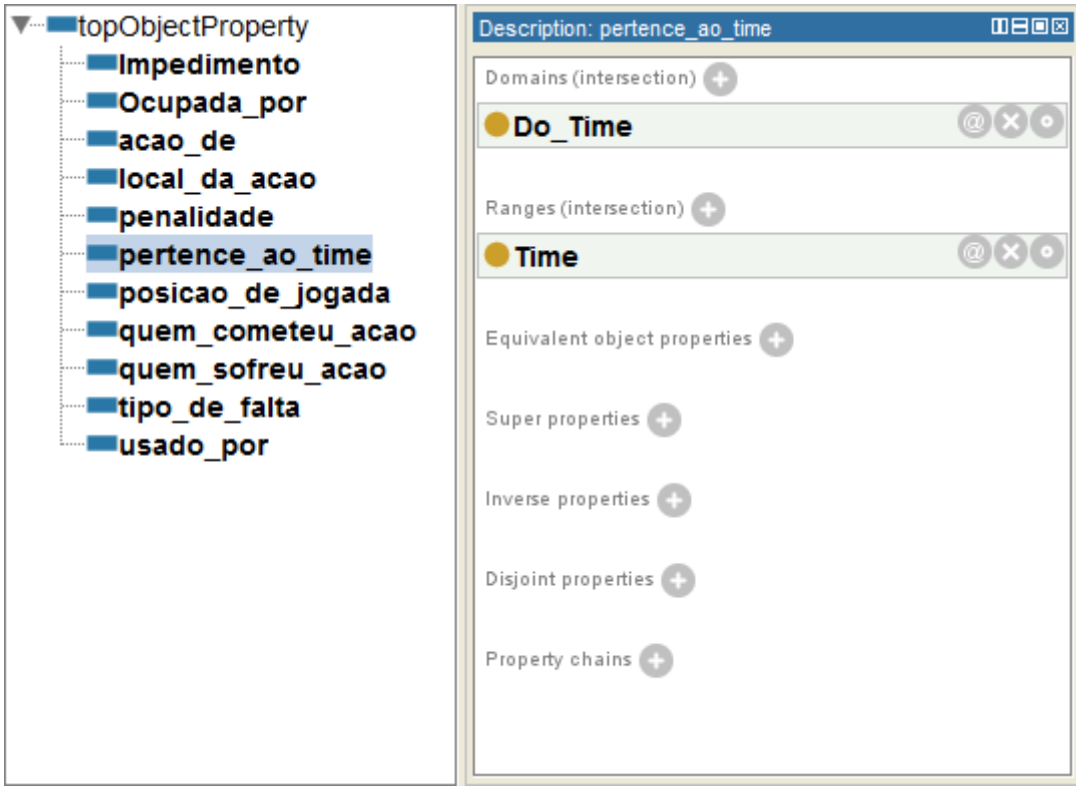


Figura 4.29: Propriedades da ontologia Futologia.

pamento_obrigatorio_jogador do_time primeiro_tempo posicao_campo))

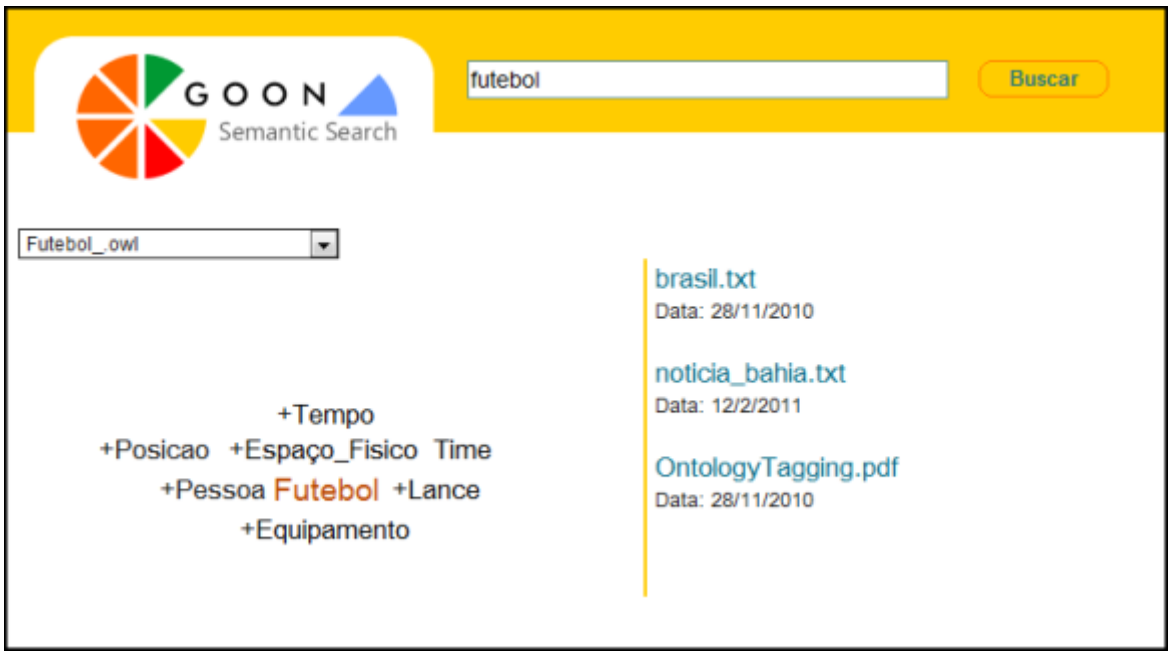


Figura 4.30: Busca feita no *Goon* pela palavra *futebol* com a ontologia Futologia selecionada.

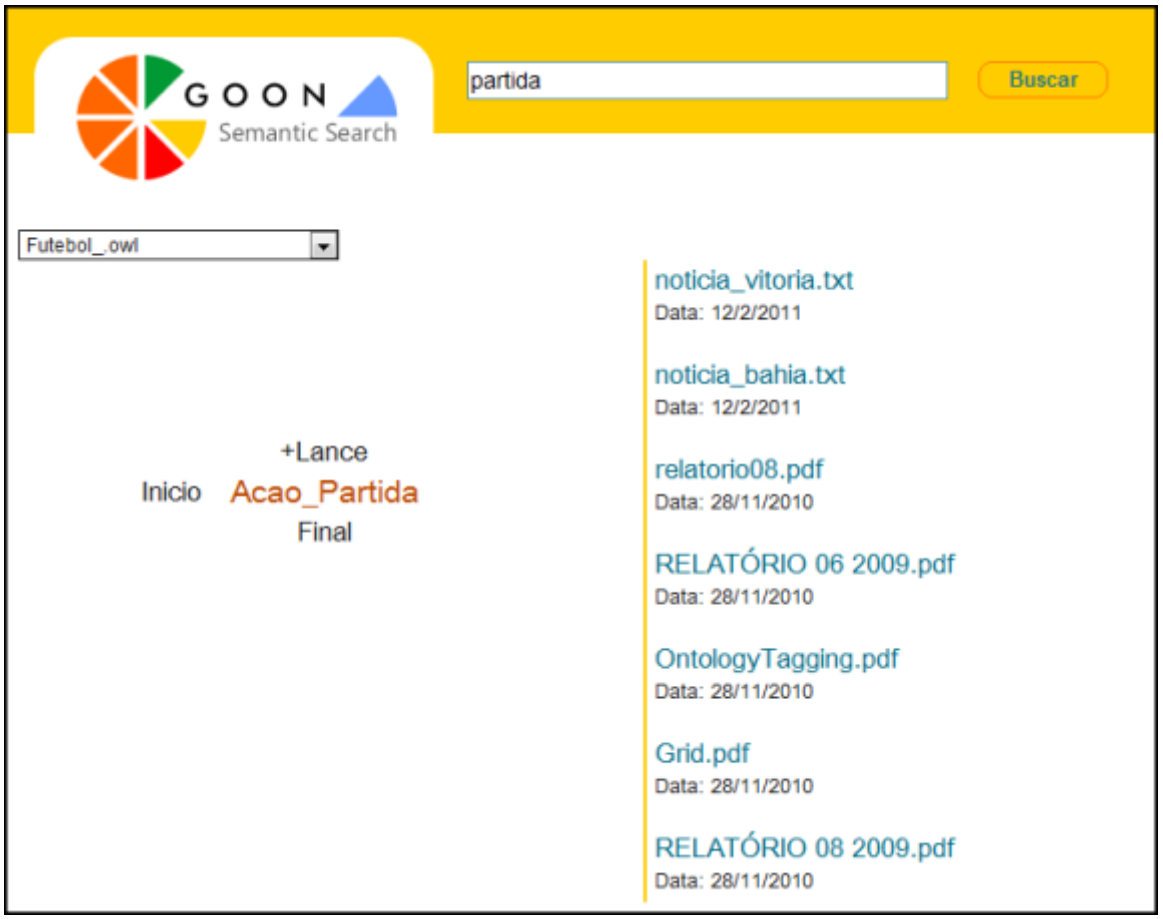


Figura 4.31: Busca feita no *Goon* pela palavra *partida* com a ontologia Futologia selecionada.

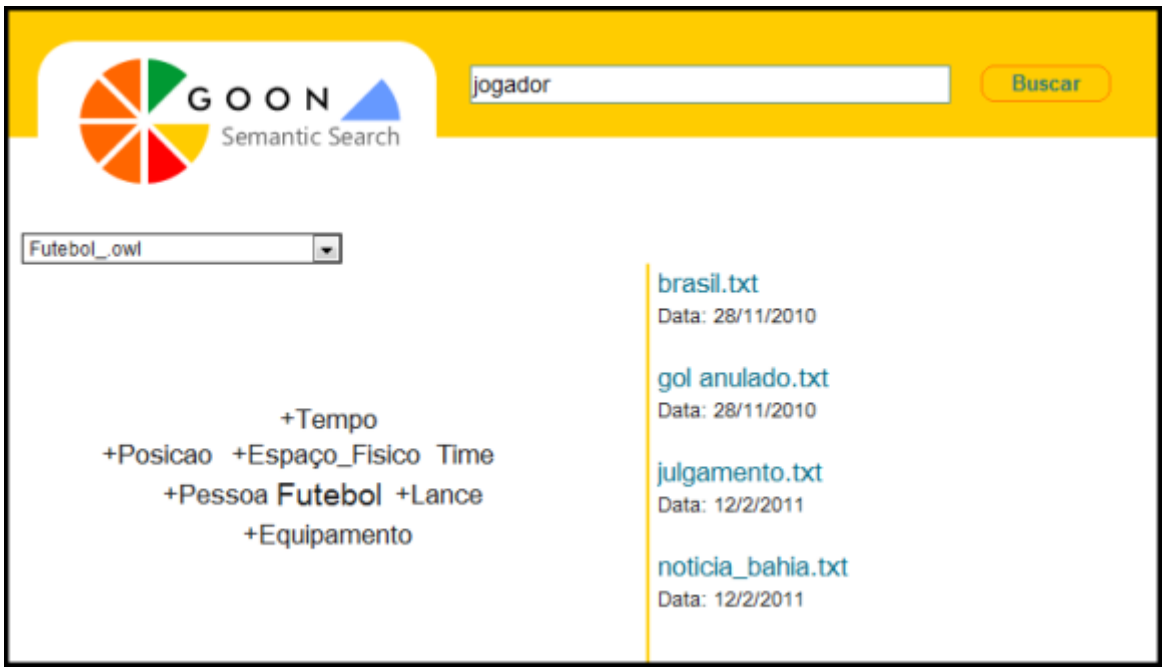


Figura 4.32: Busca feita no *Goon* pela palavra *jogador* com a ontologia Futologia selecionada.

Capítulo Cinco

Considerações Finais

Nesta dissertação foi apresentada a proposta do MOFI que tem como principal característica a utilização conjunta das técnicas Folksonomia e Indexação de Automática Arquivos, além da utilização de uma Ontologia para aperfeiçoamento das busca.

5.1 Conclusões

O modelo foi proposto para solucionar o problema de pouca semântica nas buscas realizadas por softwares atualmente e nos dados disponíveis na web. Para isso as seguintes metas foram cumpridas:

1. o algoritmo para conversão de uma ontologia de domínio em uma nuvem de tags foi modificado para permitir que a nuvem montada pelo componente *Ontology Tagging* fosse modificada de acordo a escolha do usuário, facilitando o uso da ontologia pelo usuário;
2. o componente de indexação automática do sistema *Goon* foi implementado permitindo ao software indexar automaticamente os arquivos armazenados em sua base facilitando a identificação de termos relevantes dentro do arquivo;
3. o componente de indexação manual, baseado em folksonomia, também foi implementado no *Goon* permitindo ao usuário classificar especificamente seus documentos;
4. o motor de busca responsável pelo refinamento das consultas também foi implementado sendo repensado a cada iteração do ciclo de desenvolvimento, o resultado final foi a realização de buscas não realizadas pela maioria dos sites de busca atualmente, a busca contextualizada, a busca por sinônimos e por hipônimos;
5. por fim o sistema web para armazenamento e recuperação de dados semi-estruturados foi desenvolvido seguindo todas as especificações do MOFI.

A metodologia incremental de desenvolvimento de software foi importante nesta dissertação. O fato de desenvolver o software *Goon* sempre em ciclo permitiu ao autor refletir sobre o modelo e sobre como utilizar mais do poder de uma Ontologia. Desta forma os 4 (quatro) tipos de busca foram surgindo entre algumas das iterações de implementação do software, estando presentes na conclusão do projeto.

A metodologia incremental também possibilitou a descoberta do problema causado pelo aumento exagerado das tags no banco de dados. Resultado de classificações de arquivos utilizando muitas tags e de exclusão de arquivos que deixavam tags sem vínculos no banco de dados. Este problema foi solucionado por meio de acompanhamento constante do banco de dados e remoções de tempos em tempos das tags sem vínculos existentes no banco.

Este modelo propõe uma integração de tecnologias presentes na realidade da *web* com um sistema de representação do conhecimento que ainda é considerado dúvida por alguns pesquisadores, a Ontologia. Porém a utilização conjunta dessas tecnologias é que permitiu ao software Goon se beneficiar da especificação formal de uma área do conhecimento, como o Futebol. E as Ontologias aparecerem como principal diferencial do modelo proposto na recuperação de informação, uma vez que permitem a realização de buscas com maior poder semântico.

O uso de uma Ontologia pelo software *Goon* permitiu o refinamento das buscas. Termos que não eram exatamente iguais aos termos solicitados na busca eram localizados, pois tinham uma ligação semântica com algum dos mesmos. Através da lógica computacional tornou-se possivél a extração de relações semânticas existêntes na ontologia para refinar as buscas. Como no caso dos arquivos que foram retornados mesmo sem possuir, em seu texto, a palavra solicitada na busca, pois possuiam termos relacionados a palavra buscada.

Na primeira Ontologia utilizada, a de Futebol que foi desenvolvida para avaliar as buscas feitas pelo software *Goon*, existe uma relação de equivalência entre os conceitos *partida* e *jogo*, por essa relação ser escrita em linguagem OWL ela pode ser interpretada e utilizada pelo software. Desta forma o termo *jogo* foi colocado na *string* de busca aumentando a semântica e a eficiência na recuperação da informação.

Para que um ser humano identifique o domínio abordado em um texto, que não tenha título, é necessário que este leia o texto. A medida que for lendo saberá, através da interpretação das palavras contidas ali, sobre o que se trata o texto. Com base nessa observação surgiu a ideia de se implementar no Goon as buscas contextualizadas. Apesar de não utilizar um algoritmo robusto que lê e interpreta texto o MOFI propõe a comparação das palavras do texto com termos da Ontologia que tenham relação com as palavras solicitadas na busca, desta forma pode-se filtrar o número de resultados irrelevantes da busca. Esta implementação não elimina todos os resultados que não sejam relevantes a busca, assim como também pode eliminar um resultado que seja relevante, porém ela reduz bastante essas possibilidades por não trazer um arquivo que possua apenas a palavra buscada, o algoritmo só retorna um arquivo se este possuir a palavra buscada e pelo menos um termo da ontologia, que esteja ligado a ela. No caso da busca contextualizada **sem** o termo buscado, o algoritmo tenta localizar arquivos que, não contenham a palavra

buscada pelo usuário e, tenham os termos ligados a ela.

As buscas contextualizadas poderiam ter outras variações, como por exemplo, não ser necessário a localização de todos os termos ligados a palavra buscada para retornar o arquivo na busca sem o termo buscado, poderiam ser retornados os arquivos que possuissem pelo menos 80% dos termos relacionados a ela na ontologia. Isso iria depender do nível de contextualização desejado, quanto menor o nível de contextualização maior a possibilidade de arquivos irrelevantes, e quanto maior o nível de contextualização maior a possibilidade de eliminação de arquivos relevantes.

Um ponto importante identificado nesta pesquisa é que o software *Goon*, desenvolvido com base no MOFI, é altamente dependente do nível de qualidade da ontologia utilizada para a busca. Se esta ontologia tiver erros lógicos em sua implementação, como relações de equivalência equivocadas ou relações que deveriam existir mas não foram definidas, fatalmente o resultado da busca será falho. Nos testes de busca feitos comparado a utilização da Ontologia de Futebol, desenvolvida nesta pesquisa, e a Futologia (MAíRA; SILVA, 2007) foram detectadas diferenças nos resultados das buscas. Por possuir um maior número de conceitos a Futologia geralmente retornava uma quantidade maior de resultados, fruto de uma contextualização mais abrangente, ou com mais palavras. Como a *string* de busca solicita o termo buscado e apenas um dos termos usados para contextualizar, quanto mais palavras utilizadas na contextualização maior a probabilidade de um resultado com mais retornos. Outra diferença notada foi pela falta do conceito *Jogo* com um relacionamento de equivalência com o conceito *partida*, existênte na primeira ontologia utilizada.

Por outro lado, o *Goon* pode ser utilizado sem uma Ontologia, desta forma o usuário pode se beneficiar de técnicas já existêntes na web como a folksonomia, que irá permitir a classificação pessoal do usuário sobre seus arquivos e o uso da indexação que permite a busca pelo texto contido nos arquivos. Geralmente essas técnicas são utilizadas separadas, porém serviços com o do Gmail[1] já utilizam técnicas conjuntamente.

5.2 Contribuições

Utilizar uma metáfora visual como uma nuvem de *tags* para facilitar a interação do usuário com uma Ontologia, que é uma linguagem formal, é uma contribuição importante. A metáfora visual permite que o usuário navegue pelas relações da Ontologia visualizando suas classes e utilizando-as como termos em suas buscas. Geralmente, as ferramentas de visualização de ontologia provêem uma visão hierárquica dos conceitos, o algoritmo desenvolvido permite ao usuário visualizar um conceito da ontologia com todos os outros

[1] http://www.gmail.com/

conceitos que tem relação direta com ele ao seu redor, uma navegação através dos conceitos da ontologia.

Para solucionar o problema de pouca semântica nos dados e nas buscas da web essa dissertação tem como uma de suas principais propostas o uso da folksonomia, para permitir ao usuário marcar o conteúdo de acordo a sua necessidade, atribuindo-lhe uma palavra que para si tenha sentido ou relevância. Um conteúdo classificado como *entretenimento* pode não ter esse sentido para todos os usuário, mas o tem para quem o classificou desta forma (REIS et al., 2009). Da mesma maneira que se várias pessoas classificam um mesmo conteúdo com a mesma tag, provavelmente, esta classificação é pertinente.

Outra proposta importante para a solução do problema é o uso de uma ontologia de domínio para especificar as buscas feitas. Utilizando uma ontologia de determinado domínio o usuário pode utilizar termos (conceitos) comuns a área em questão que são usados por muitas outras pessoas (REIS et al., 2009). Além disso, uma vez sendo escolhida pelo usuário a ontologia de domínio pode ser utilizada pelo software para inferir relações entre os termos utilizados realizando buscas com mais semântica.

Nesta dissertação, foram desenvolvidos 4 (quatro) tipos de busca baseados em uma ontologia. A **busca contextualizada com o termo buscado** que permite específicar o resultado da busca acrescentando termos relacionados aos termos pedidos pelo usuário. A **busca contextualizada sem o termo buscado** que modifica a *string* de busca removendo para encontrar documentos que possuam apenas os termos relacionados ao termo pedido pelo usuário (sem o termo pedido pelo usuário). A **busca por sinônimos** que utiliza a propriedade de equivalência para identificar termos sinônimos ao termo pedido pelo usuário (e.g. Jogo é uma Partida). E por fim a **busca por hipônimos** que utiliza a hieraquia das classes de uma ontologia para identificar termos hipônimos ao termo solicitado pelo usuário (e.g. Goleiro é um tipo de Jogador).

Foi constatado nessa pesquisa que é possível a utilização de uma ontologia de domínio para aperfeiçoar os métodos de recuperação de informação existêntes, e que a ontologia deve ser desenvolvida com a melhor qualidade possível, pois as funcionalidades de sistemas como o *Goon*, que se beneficiam de uma ontologia, são dependentes dessa. Essas constatações reforçam a necessidade de metodologias de desenvolvimento de ontologias e de estudos sobre o tema, como feito por Eduardo Jorge (2010) em sua qualificação de doutorado. Eduardo Jorge também faz parte do grupo de pesquisa que este autor participa junto com seu orientador. Ele propõe o *M-MOBI* (Método de Modelagem de Ontologia Baseado em Instâncias) que visa a redução no esforço inicial da curva de aprendizado dos conceitos de modelagem de Ontologia, abordando o desenvolvimento de ontologia a partir das instâncias.

5.3 Atividades Futuras de Pesquisa

Como trabalhos futuros, sugere-se o desenvolvimento de um algoritmo de indexação automática de conteúdo que indexe arquivos e páginas web, e guarde as referências e os índices desses arquivos no banco de dados do *Goon*. Com esse algoritmo, o *Goon* poderia realizar buscas na web o que permitiria uma melhor comparação de seus resultados com o de portais de busca mais conhecidos, como *Google* e *Yahoo*.

A evolução da ferramenta, possibilitando a utilização de outros elementos da ontologia, como os axiomas e indivíduos, para aperfeiçoar mais as buscas, também é sugerida. Os axiomas são regras lógicas contidas nas ontologias que definem as classificações dos novos indivíduos. A realização de inferência nos axiomas ou restrições da ontologia também pode aperfeiçoar as buscas. Encontrando novas relações entre os conceitos da ontologia no caso de propriedades transitivas (Seção 3.4), por exemplo.

Por fim, a criação de métodos de busca baseados em Ontologia, ou a adequação dos métodos propostos, para a realização de buscas por páginas web anotadas semanticamente, seguindo a técnica apresentada na Seção 2.5. Essa técnica permite que os criadores de sites na web anotem semânticamente suas páginas, montando vínculos entre os textos da página. Esses vínculos são como as propriedades de uma ontologia de domínio e também pode ser utilizados para aumentar a semântica das páginas web e das buscas realizadas em páginas web.

O MOFI é um exemplo de como as relações semânticas de uma ontologias podem ser trabalhadas para melhorar o processamento de informações realizado nos softwares. Por ser baseada em lógica descritva a ontologia permite outras formas, mais robustas, de utilização das relações semânticas, como as inferências que permitem a descoberta de novas relações com base nas já pré-definidas.

Apêndice A

Código Fonte do software Goon

Este apêndice contém o código, feito em linguagem Java, das classes mais importantes do projeto. O código da classe do algoritmo de busca implementado. O código das classes de montagem da nuvem de tags oriunda da ontologia.

A.1 Algoritmo de Busca baseada em Ontologia

Classe *SearchEgine*:

```
1  package br.org.uedsonreis.ontologytagging.server.bo.search;
2
3  import java.io.File;
4  import java.util.HashSet;
5  import java.util.LinkedHashSet;
6  import java.util.List;
7  import java.util.Set;
8
9  import org.apache.lucene.document.Document;
10
11 import br.org.uedsonreis.ontologytagging.bean.PlacedTag;
12 import br.org.uedsonreis.ontologytagging.model.bean.dto.Content;
13 import br.org.uedsonreis.ontologytagging.model.bean.dto.SystemUser;
14 import br.org.uedsonreis.ontologytagging.model.bean.dto.Tag;
15 import br.org.uedsonreis.ontologytagging.model.dao.BasicDao;
16 import br.org.uedsonreis.ontologytagging.model.dao.index.IndexDAO;
17
18 import com.sindice.Sindice;
19 import com.sindice.SindiceException;
20 import com.sindice.result.SearchResult;
21 import com.sindice.result.SearchResults;
22
23 public class SearchEngine {
24
25    private final OntoBO onto;
26    private final BasicDao dao;
27    private final IndexDAO index;
28
29    private Sindice sindice;
30
31    public SearchEngine(IndexDAO index, OntoBO onto, BasicDao dao) {
32       this.index = index;
33       this.onto = onto;
34       this.dao = dao;
35    }
```

```
36
37    private String treatString(String terms) {
38       terms = PlacedTag.removeAccent(terms);
39       terms = terms.trim().toLowerCase();
40       return terms;
41    }
42
43    public Set<Content> searchContents(String ontoName, String terms) {
44       Set<Content> result = new LinkedHashSet<Content>();
45       if ((terms == null) || (terms.trim().equals(""))) return result;
46
47       terms = this.treatString(terms);
48
49       if ((ontoName != null) && (!ontoName.trim().equals("")) && (this.onto.
             getCloudMap().containsKey(ontoName))) {
50          result.addAll(this.searchSementic(ontoName, terms));
51       } else {
52          String[] array = terms.split(" ");
53          Set<Tag> tags = new LinkedHashSet<Tag>();
54          for (int i = 0; i < array.length; i++) tags.add(new Tag(array[i]));
55
56          result.addAll(this.searchContentsByAllTags(tags));
57          result.addAll(this.searchContentsByTags(tags));
58          result.addAll(this.searchContentsIndexed(terms));
59       }
60       return result;
61    }
62
63    private Set<Content> searchSementic(String ontoName, final String termsQuery) {
64       final Set<Content> result = new LinkedHashSet<Content>();
65       final Set<Tag> tags = new HashSet<Tag>();
66       String query = "";
67
68       final String[] array = termsQuery.split(" ");
69       for (int i = 0; i < array.length; i++) {
70          String terms = "";
71
72          String term = this.treatString(array[i]);
73          tags.add(new Tag(term));
74
75          PlacedTag tag = this.onto.getPlacedTag(ontoName, term);
76          if (tag == null) {
77             terms += term;
78          } else {
79             Set<PlacedTag> relateds = new HashSet<PlacedTag>();
80             Set<PlacedTag> directs = new HashSet<PlacedTag>();
81
82             // Preenchendo as variáveis relateds e directs com as relações do
                   termo
83             if (tag.getRelateds() != null) {
84                Set<PlacedTag> equivalents = tag.getRelateds().get(PlacedTag.
                      EQUIVALENT);
```

```
85                Set<PlacedTag> subClasses = tag.getRelateds().get(PlacedTag.
                      superClassOf);
86
87                for (Set<PlacedTag> set : tag.getRelateds().values()) {
88                   for (PlacedTag pt : set) relateds.add(pt);
89                   directs.addAll(relateds);
90                }
91
92                if (equivalents != null) {
93                   for (PlacedTag pt : equivalents) {
94                      directs.remove(pt);
95                      String t = this.treatString(pt.getName());
96                      tags.add(new Tag(t));
97                      term += " "+ t;
98                   }
99                }
100               if (subClasses != null) {
101                  for (PlacedTag pt : subClasses) {
102                     directs.remove(pt);
103                     String t = this.treatString(pt.getName());
104                     tags.add(new Tag(t));
105                     term += " "+ t;
106                  }
107               }
108            }
109
110            // Preenchendo a variável relateds com as relações do conceito raíz da
                  ontologia
111            PlacedTag rootTag = this.onto.getPlacedTag(ontoName, ontoName.
                  substring(0, ontoName.length()-4));
112            if (rootTag != null) {
113               relateds.add(rootTag);
114               if ((rootTag.getRelateds() != null) && (rootTag.getRelateds().size
                      () > 0)) {
115                  for (Set<PlacedTag> set : rootTag.getRelateds().values()) {
116                     for (PlacedTag pt : set) {
117                        relateds.addAll(this.getTagEquivalents(pt));
118                     }
119                  }
120               }
121            }
122
123            // Busca contextualizada COM o termo buscado
124            String[] array2 = term.split(" ");
125
126            String closeEnd = "";
127            if (array2.length > 1) {
128               terms = "(("+array2[0]+") AND (";
129               for (int y = 1; y < array2.length; y++) {
130                  terms += " "+array2[y];
131               }
132               terms += ")) OR (";
```

```
133                 closeEnd = ")";
134             }
135 
136             terms += "(("+term+") AND (";
137 
138             if ((relateds != null) && (relateds.size() > 0)) {
139                 for (PlacedTag pt : relateds) {
140                     String name = this.treatString(pt.getName());
141                     if (!term.contains(name)) terms += " "+ name;
142                 }
143             }
144             terms += "))"+closeEnd;
145 
146             // Busca contextualizada SEM o termo buscado
147             if ((directs != null) && (directs.size() > 1)) {
148                 if (!rootTag.getName().equalsIgnoreCase(term)) {
149                     terms += " OR (" + "("+this.getStringEquivalents(rootTag)+")";
150                 } else {
151                     PlacedTag pt = (PlacedTag) directs.iterator().next();
152                     terms += " OR (" + "("+this.getStringEquivalents(pt)+")";
153                     directs.remove(pt);
154                 }
155                 for (PlacedTag pt : directs) {
156                     String d = "("+this.getStringEquivalents(pt)+")";
157                     terms += " AND "+ d;
158                 }
159                 terms += ")";
160             }
161             terms = "("+ terms +")";
162         }
163         query += " "+ terms +" OR";
164     }
165 
166     query = query.trim();
167     if (query.endsWith(" OR")) query = query.substring(0, query.length()-3);
168     System.out.println(query +" = "+ tags);
169 
170     result.addAll(this.searchContentsByAllTags(tags));
171     result.addAll(this.searchContentsByTags(tags));
172     result.addAll(this.searchContentsIndexed(query));
173     return result;
174 }
175 
176 private String getStringEquivalents(PlacedTag tag) {
177     Set<PlacedTag> equivalents = tag.getRelateds().get(PlacedTag.EQUIVALENT);
178     Set<PlacedTag> subClasses = tag.getRelateds().get(PlacedTag.superClassOf);
179 
180     String equiv = this.treatString(tag.getName());
181     if (equivalents != null) {
182         for (PlacedTag pt : equivalents) {
183             String t = this.treatString(pt.getName());
184             equiv += " "+ t;
```

```
185             }
186         }
187         if (subClasses != null) {
188             for (PlacedTag pt : subClasses) {
189                 String t = this.treatString(pt.getName());
190                 equiv += " "+ t;
191             }
192         }
193         return equiv;
194     }
195     private Set<PlacedTag> getTagEquivalents(PlacedTag tag) {
196         Set<PlacedTag> equivalents = tag.getRelateds().get(PlacedTag.EQUIVALENT);
197         Set<PlacedTag> subClasses = tag.getRelateds().get(PlacedTag.superClassOf);
198 
199         Set<PlacedTag> equiv = new HashSet<PlacedTag>();
200         equiv.add(tag);
201 
202         if (equivalents != null) {
203             for (PlacedTag pt : equivalents) equiv.add(pt);
204         }
205         if (subClasses != null) {
206             for (PlacedTag pt : subClasses) equiv.add(pt);
207         }
208         return equiv;
209     }
210 
211     private Set<Content> searchContentsByAllTags(Set<Tag> tags) {
212         Set<Content> result = new LinkedHashSet<Content>();
213         Set<Content> contentsAll = this.searchContentsByTags(tags);
214 
215         for (Content c : contentsAll) {
216             if (c.getTags().containsAll(tags)) result.add(c);
217         }
218         return result;
219     }
220 
221     private Set<Content> searchContentsByTags(Set<Tag> tags) {
222         Set<Tag> listTags = new LinkedHashSet<Tag>();
223         for (Tag t : tags) {
224             Tag tag = (Tag) this.dao.get(t.getName().trim().toLowerCase(), Tag.class)
                    ;
225             if (tag != null) listTags.add(tag);
226         }
227 
228         Set<Content> contentsAll = new LinkedHashSet<Content>();
229         for (Tag t : listTags) contentsAll.addAll(t.getContents());
230         return contentsAll;
231     }
232 
233     private Set<Content> searchContentsIndexed(String terms) {
234 
235         Set<Document> docs = this.index.search(terms);
```

```
236         Set<Content> result = new LinkedHashSet<Content>();
237 
238         if ((docs != null) && (docs.size() > 0)) {
239             for (Document doc : docs) {
240                 File file = new File(doc.get("filename"));
241 
242                 String query = "SELECT DISTINCT c FROM content c "
243                         +"WHERE UPPER(c.url) LIKE UPPER('"+ file.getName().trim()+"')
                            ";
244 
245                 List<Content> list = (List<Content>) this.dao.executeQuery(query);
246 
247                 Content c = null;
248                 if ((list != null) && (list.size() > 0)) c = list.get(0);
249 
250                 if (c == null) {
251                     if ((file.isFile()) && (file.canRead())) {
252                         c = new Content();
253                         c.setUrl(file.getName().trim());
254                         c.setDirectory(file.getParent());
255                         c.setIsFile(Boolean.TRUE);
256                         result.add(c);
257                     }
258                 } else {
259                     result.add(c);
260                 }
261             }
262         }
263         return result;
264     }
265 }
```

Listing A.1: Classe para Montagem da Busca baseada em Ontologia

A.2 Algoritmos de Montagem da Nuvem de Tags

Classe *TagCloudAlgorithm*:

```
1  package br.org.uedsonreis.ontologytagging.algorithm.cloud.impl;
2 
3  import java.awt.Font;
4  import java.util.ArrayList;
5  import java.util.List;
6 
7  import sun.font.FontDesignMetrics;
8  import br.org.uedsonreis.ontologytagging.algorithm.cloud.iface.ITagCloudAlgorithm;
9  import br.org.uedsonreis.ontologytagging.bean.PlacedTag;
10 
11 public abstract class TagCloudAlgorithm implements ITagCloudAlgorithm {
12 
```

```
13    protected static final int whiteSpace = 10;
14
15    protected static final int relevanceBase = 100;
16    protected static final double FATOR = 0.9;
17
18    protected int maxRelevance = 0, minRelevance = 0;
19    protected int panelWidth, panelHeight;
20    protected int maxFont, minFont;
21
22    protected PlacedTag onto;
23    protected PlacedTag centerTag;
24    private String fontType = "Arial";
25
26    private final List<PlacedTag> tagsCloud = new ArrayList<PlacedTag>();
27
28    public TagCloudAlgorithm(int minFontSize, int maxFontSize, int sizeWidth, int
          sizeHeight) {
29       this.minFont = minFontSize;
30       this.maxFont = maxFontSize;
31
32       this.panelWidth = sizeWidth;
33       this.panelHeight = sizeHeight;
34    }
35
36    // Template Method
37    public final List<PlacedTag> makeTagCloud(PlacedTag placed) {
38       if (placed == null) return null;
39
40       this.tagsCloud.clear();
41       this.init(placed);
42       this.make();
43
44       return this.tagsCloud;
45    }
46
47    protected int getWidth(String name, int size) {
48       Font font = new Font(fontType, Font.PLAIN, size);
49       FontDesignMetrics metrics = FontDesignMetrics.getMetrics(font);
50       return metrics.stringWidth(name);
51    }
52    protected int getHeight(String name, int size) {
53       return size;
54    }
55
56    protected int calculateFont(int relevance) {
57       if (maxRelevance == minRelevance) return this.maxFont;
58       Double result = ((relevance - minRelevance)*(this.maxFont - this.minFont) +
59                   this.minFont*(maxRelevance - minRelevance))/(maxRelevance -
                        minRelevance)+0.5;
60
61       return result.intValue();
62    }
```

```
63
64     protected void addTagPlaced(PlacedTag placed, int x, int y) {
65         placed.setX(x); placed.setY(y);
66         this.tagsCloud.add(placed);
67     }
68
69     public void setFontType(String fontType) {
70         this.fontType = fontType;
71     }
72     protected boolean isPlaced(PlacedTag placed) {
73         return this.tagsCloud.contains(placed);
74     }
75     protected List<PlacedTag> getCloud() {
76         return this.tagsCloud;
77     }
78
79     // Template Method
80     protected abstract void make();
81     protected abstract void init(PlacedTag placed);
82
83 }
```

Listing A.2: Classe para Abstrata que provê os métodos básicos para montagem da nuvem.

Classe *IterativeCloud*:

```
1  package br.org.uedsonreis.ontologytagging.algorithm.cloud.impl;
2
3  import java.util.ArrayList;
4  import java.util.Collection;
5  import java.util.HashMap;
6  import java.util.HashSet;
7  import java.util.LinkedHashSet;
8  import java.util.List;
9  import java.util.Set;
10
11 import br.org.uedsonreis.ontologytagging.bean.Coordinated;
12 import br.org.uedsonreis.ontologytagging.bean.PlacedTag;
13
14 public class IterativeCloud extends TagCloudAlgorithm {
15
16     private static final int RELEVANCE = 8;
17
18     private final List<Coordinated> freePoints = new ArrayList<Coordinated>();
19     private final Collection<Coordinated> busyPoints = new LinkedHashSet<
           Coordinated>();
20     private final List<PlacedTag> unPlaceds = new ArrayList<PlacedTag>();
21
22     private List<PlacedTag> listTags;
23     private Coordinated middle;
24
```

```
25    public IterativeCloud(int minFontSize, int maxFontSize, int widthSize, int
          heightSize) {
26       super(minFontSize, maxFontSize, widthSize, heightSize);
27       super.maxRelevance = 10;
28       super.minRelevance = 1;
29    }
30
31    protected void init(PlacedTag placed) {
32       if (this.listTags == null) {
33          this.listTags = new ArrayList<PlacedTag>();
34          this.makeList(placed);
35          this.listTags = new ArrayList<PlacedTag>(new LinkedHashSet<PlacedTag>(
                this.listTags));
36       }
37       this.centerTag = placed;
38    }
39    private void makeList(PlacedTag placed) {
40       this.listTags.add(placed);
41       if ((placed.getRelateds() != null) && (placed.getRelateds().size() > 0)) {
42          for (Set<PlacedTag> placeds : placed.getRelateds().values()) {
43             for (PlacedTag pt : placeds) {
44                if (!this.listTags.contains(pt)) this.makeList(pt);
45             }
46          }
47       }
48    }
49
50    @Override
51    protected void make() {
52       this.freePoints.clear();
53       this.busyPoints.clear();
54       this.unPlaceds.clear();
55       this.middle = null;
56
57       this.makeTagCenter(super.centerTag);
58
59       for (Set<PlacedTag> tags : super.centerTag.getRelateds().values()) {
60          for (PlacedTag t : tags) {
61             if (super.isPlaced(t)) continue;
62             t.setSize(super.calculateFont(RELEVANCE));
63             this.setPoint(t);
64          }
65       }
66
67       for (PlacedTag unPlaced : this.unPlaceds) {
68          if (super.isPlaced(unPlaced)) continue;
69          this.setPoint(unPlaced);
70       }
71    }
72
73    private void setPoint(PlacedTag placed) {
74       if (this.freePoints.size() > 0) {
```

```
75          Coordinated point = this.freePoints.remove(0);
76          point.setTagPlaced(placed);
77          point.setWidth(super.getWidth(placed.symbol()+placed.getName(), placed.
                getSize()));
78          point.setHeight(super.getHeight(placed.getName(), placed.getSize()));
79
80          if (this.isBusy(point)) {
81             this.settingBusy(point);
82          } else {
83             this.setting(point);
84          }
85       }
86    }
87
88    private void settingBusy(Coordinated point) {
89       int eixoX = point.getX() - this.middle.getX();
90
91       if (eixoX < 0) {
92          point.setX(point.getX() - point.getWidth()/4);
93          if (isBusy(point)) {
94             point.setX(point.getX() - point.getWidth()/4);
95             if (isBusy(point)) return;
96          }
97       } else if (eixoX > 0) {
98          point.setX(point.getX() + point.getWidth()/4);
99          if (isBusy(point)) {
100            point.setX(point.getX() + point.getWidth()/4);
101            if (isBusy(point)) return;
102         }
103      } else {
104         return;
105      }
106      this.addTagPlaced(point);
107   }
108
109   private void setting(Coordinated point) {
110      int eixoX = point.getX() - middle.getX();
111
112      if (eixoX < 0) {
113         this.settingRight(point);
114      } else if (eixoX > 0) {
115         this.settingLeft(point);
116      }
117      this.addTagPlaced(point);
118   }
119
120   private void settingLeft(final Coordinated point) {
121      while (!this.isBusy(point)) {
122         point.setX(point.getX() - point.getWidth()/4);
123         if (point.getX() < 0) return;
124      }
125      point.setX(point.getX() + point.getWidth()/4 + whiteSpace/2);
```

```
126     }
127
128     private void settingRight(final Coordinated point) {
129         while (!this.isBusy(point)) {
130             point.setX(point.getX() + point.getWidth()/4);
131             if (point.getX() > super.panelWidth) return;
132         }
133         point.setX(point.getX() - point.getWidth()/4 - whiteSpace/2);
134     }
135
136     private void makeTagCenter(PlacedTag placed) {
137         placed.setSize(super.maxFont);
138
139         this.middle = new Coordinated();
140         this.middle.setX(super.panelWidth/2);
141         this.middle.setY(super.panelHeight/2);
142
143         this.middle.setTagPlaced(placed);
144         this.middle.setWidth(super.getWidth(placed.getName(), placed.getSize()));
145         this.middle.setHeight(super.getHeight(placed.getName(), placed.getSize()));
146         this.addTagPlaced(this.middle);
147     }
148
149     private boolean isBusy(Coordinated point) {
150         if ((this.busyPoints == null) || (this.busyPoints.size() <= 0)) return false
                ;
151
152         for (Coordinated eixo : this.busyPoints) {
153             if (eixo.isBusy(point)) return true;
154         }
155         return false;
156     }
157
158     private boolean isInPanel(Coordinated point) {
159         if ((point.getX() + point.getWidth()/2) > super.panelWidth) return false;
160         if ((point.getX() - point.getWidth()/2) < 0) return false;
161
162         if ((point.getY() + point.getHeight()/2) > super.panelHeight) return false;
163         if ((point.getY() - point.getHeight()/2) < 0) return false;
164
165         return true;
166     }
167
168     private void addTagPlaced(Coordinated point) {
169         if (this.isInPanel(point)) {
170             this.busyPoints.add(point);
171
172             int x = point.getX() - (point.getWidth()/2);
173             int y = point.getY() - (point.getHeight()/2);
174
175             super.addTagPlaced(point.getTagPlaced(), x, y);
176             this.addFreePointsOf(point);
```

```
177         } else {
178             this.setPoint(point.getTagPlaced());
179             point.setWidth(0);
180             point.setHeight(0);
181             point.setTagPlaced(null);
182             this.freePoints.add(point);
183         }
184     }
185     private void addFreePointsOf(Coordinated point) {
186         Coordinated up = new Coordinated(point.getX(), point.getY() - point.
                getHeight());
187         Coordinated left = new Coordinated(point.getX() - (point.getWidth()), point.
                getY());
188         Coordinated right = new Coordinated(point.getX() + (point.getWidth()), point
                .getY());
189         Coordinated down = new Coordinated(point.getX(), point.getY() + point.
                getHeight());
190 
191         if (point.getX() > this.middle.getX()) {
192             left = null;
193         } else if (point.getX() < this.middle.getX()) {
194             right = null;
195         }
196 
197         if (point.getY() > this.middle.getY()) {
198             up = null;
199         } else if (point.getY() < this.middle.getY()) {
200             down = null;
201         }
202 
203         if (point.getY() == this.middle.getY()) {
204             if (up != null) {
205                 up.setY(up.getY() - whiteSpace/2);
206                 this.freePoints.add(up);
207             }
208             if (down != null) {
209                 down.setY(down.getY() + whiteSpace/2);
210                 this.freePoints.add(down);
211             }
212             if (left != null) this.freePoints.add(left);
213             if (right != null) this.freePoints.add(right);
214         } else {
215             if (left != null) this.freePoints.add(left);
216             if (right != null) this.freePoints.add(right);
217             if (up != null) {
218                 up.setY(up.getY() - whiteSpace/2);
219                 this.freePoints.add(up);
220             }
221             if (down != null) {
222                 down.setY(down.getY() + whiteSpace/2);
223                 this.freePoints.add(down);
224             }
```

```
225         }
226     }
227
228     public List<PlacedTag> refactory(PlacedTag placed) {
229         if ((!placed.equals(super.centerTag)) && (placed.
                getRelatedSizeWithoutSuperClass() > 1)) {
230             if (super.centerTag.getRelateds().get(PlacedTag.superClassOf) == null) {
231                 super.centerTag.getRelateds().put(PlacedTag.superClassOf, new HashSet<
                        PlacedTag>());
232             }
233             if (super.centerTag.getRelateds().get(PlacedTag.superClassOf).contains(
                    placed)) {
234                 if (placed.getRelateds().get(PlacedTag.subClassOf) == null) {
235                     placed.getRelateds().put(PlacedTag.subClassOf, new HashSet<
                            PlacedTag>());
236                 }
237                 placed.getRelateds().get(PlacedTag.subClassOf).add(super.centerTag);
238             }
239
240             return super.makeTagCloud(placed);
241         }
242         return null;
243     }
244
245     public PlacedTag getPlacedTag(String name) {
246         PlacedTag term = new PlacedTag(name);
247
248         if (this.listTags.contains(term)) {
249             term = this.listTags.get(this.listTags.indexOf(term));
250         } else {
251             term = this.getContained(this.listTags, term);
252         }
253
254         if (term != null) {
255             PlacedTag superTag = this.getSuperTag(term);
256             if (superTag != null) {
257                 if (term.getRelateds() == null) term.setRelateds(new HashMap<String,
                        Set<PlacedTag>>());
258                 if (term.getRelateds().get(PlacedTag.subClassOf) == null) {
259                     term.getRelateds().put(PlacedTag.subClassOf, new HashSet<PlacedTag
                            >());
260                 }
261                 term.getRelateds().get(PlacedTag.subClassOf).add(superTag);
262             }
263             return term;
264         }
265         return null;
266     }
267     private PlacedTag getSuperTag(PlacedTag placed) {
268         for (PlacedTag pt : this.listTags) {
269             if ((pt.getRelateds() != null) && (pt.getRelateds().get(PlacedTag.
                    superClassOf) != null)) {
```

```
270             if (pt.getRelateds().get(PlacedTag.superClassOf).contains(placed))
                    return pt;
271         }
272       }
273       return null;
274     }
275
276     private PlacedTag getContained(Collection<PlacedTag> listOntos, PlacedTag label
            ) {
277       for (PlacedTag tl : listOntos) {
278         if (tl != null) {
279           String term = PlacedTag.removeAccent(label.getName());
280           String[] names_ = tl.getName().split("_");
281           for (String n : names_) {
282             if (PlacedTag.isEquals(n, term)) return tl;
283           }
284
285           String[] names = tl.getName().split(" ");
286           for (String n : names) {
287             if (PlacedTag.isEquals(n, term)) return tl;
288           }
289         }
290       }
291       return null;
292     }
293 }
```

Listing A.3: Classe que implementa a montagem da nuvem de tags.

A.3 Ontologia Futebol.owl

```
1 <?xml version="1.0"?>
2
3
4 <!DOCTYPE rdf:RDF [
5     <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
6     <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
7     <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
8     <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
9     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
10    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
11    <!ENTITY protege "http://protege.stanford.edu/plugins/owl/protege#" >
12    <!ENTITY xsp "http://www.owl-ontologies.com/2005/08/07/xsp.owl#" >
13 ]>
14
15
16 <rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232
       .owl#"
17      xml:base="http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.
            owl"
```

```
18      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
19      xmlns:swrl="http://www.w3.org/2003/11/swrl#"
20      xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
21      xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
22      xmlns:owl="http://www.w3.org/2002/07/owl#"
23      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
24      xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
25      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
26    <owl:Ontology rdf:about="http://www.semanticweb.org/ontologies/2010/5/
          Ontology1277729329232.owl"/>
27
28
29
30    <!--
31    ///////////////////////////////////////////////////////////////////////////////////////
32    //
33    // Object Properties
34    //
35    ///////////////////////////////////////////////////////////////////////////////////////
36     -->
37
38
39
40
41    <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
          auxilia_tecnico -->
42
43    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
          Ontology1277729329232.owl#auxilia_tecnico">
44        <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
              Ontology1277729329232.owl#Auxiliar_Tecnico"/>
45        <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
              Ontology1277729329232.owl#Time"/>
46        <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
              /2010/5/Ontology1277729329232.owl#pertence"/>
47    </owl:ObjectProperty>
48
49
50
51    <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
          dirige -->
52
53    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
          Ontology1277729329232.owl#dirige">
54        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
55        <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
              Ontology1277729329232.owl#Dirigente"/>
56        <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
              Ontology1277729329232.owl#Time"/>
```

```
57          <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#pertence"/>
58      </owl:ObjectProperty>
59
60
61
62      <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            dirigido_por -->
63
64      <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#dirigido_por">
65          <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
66          <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Dirigente"/>
67          <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
68          <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#tem"/>
69      </owl:ObjectProperty>
70
71
72
73      <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            disputa -->
74
75      <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#disputa">
76          <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogo"/>
77          <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
78          <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#disputado"/>
79          <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
80      </owl:ObjectProperty>
81
82
83
84      <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            disputado -->
85
86      <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#disputado">
87          <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogo"/>
88          <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
89          <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
90      </owl:ObjectProperty>
91
92
93
```

```
94      <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            feito_por -->
95
96      <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#feito_por">
97          <rdf:type rdf:resource="&owl;FunctionalProperty"/>
98          <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Evento"/>
99          <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogador"/>
100         <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#fez"/>
101         <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
102     </owl:ObjectProperty>
103
104
105
106     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            fez -->
107
108     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#fez">
109         <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
110         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Evento"/>
111         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogador"/>
112         <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
113     </owl:ObjectProperty>
114
115
116
117     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            joga_no -->
118
119     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#joga_no">
120         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogador"/>
121         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
122         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#pertence"/>
123     </owl:ObjectProperty>
124
125
126
127     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            ocorre_em -->
128
129     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#ocorre_em">
```

```
130         <rdf:type rdf:resource="&owl;FunctionalProperty"/>
131         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Evento"/>
132         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogo"/>
133         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#pertence"/>
134     </owl:ObjectProperty>
135
136
137
138     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            ocorreu -->
139
140     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#ocorreu">
141         <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
142         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Evento"/>
143         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogo"/>
144         <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#ocorre_em"/>
145         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#tem"/>
146     </owl:ObjectProperty>
147
148
149
150     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            pertence -->
151
152     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#pertence">
153         <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
154     </owl:ObjectProperty>
155
156
157
158     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            prepara_fisicamente -->
159
160     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#prepara_fisicamente">
161         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Preparador_Fisico"/>
162         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
163         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#pertence"/>
164     </owl:ObjectProperty>
165
```

```
166
167
168     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            preparado_fisicamente_por -->
169
170     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#preparado_fisicamente_por">
171         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Preparador_Fisico"/>
172         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
173         <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#prepara_fisicamente"/>
174         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#tem"/>
175     </owl:ObjectProperty>
176
177
178
179     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            sofre -->
180
181     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#sofre">
182         <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
183         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Evento"/>
184         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogador"/>
185         <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
186     </owl:ObjectProperty>
187
188
189
190     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            sofrido_por -->
191
192     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#sofrido_por">
193         <rdf:type rdf:resource="&owl;FunctionalProperty"/>
194         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Evento"/>
195         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogador"/>
196         <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#sofre"/>
197         <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
198     </owl:ObjectProperty>
199
200
201
```

```
202     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            tecnico_auxiliado_por -->
203
204     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#tecnico_auxiliado_por">
205         <rdf:type rdf:resource="&owl;FunctionalProperty"/>
206         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Auxiliar_Tecnico"/>
207         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
208         <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#auxilia_tecnico"/>
209         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#tem"/>
210     </owl:ObjectProperty>
211
212
213
214     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            tem -->
215
216     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#tem">
217         <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#pertence"/>
218         <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
219     </owl:ObjectProperty>
220
221
222
223     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            tem_jogador -->
224
225     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#tem_jogador">
226         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogador"/>
227         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
228         <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#joga_no"/>
229         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#tem"/>
230     </owl:ObjectProperty>
231
232
233
234     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            tem_medico -->
235
236     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#tem_medico">
```

```
237         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Medico"/>
238         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
239         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#tem"/>
240         <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#trabalha"/>
241     </owl:ObjectProperty>
242
243
244
245     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            trabalha -->
246
247     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#trabalha">
248         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Medico"/>
249         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
250         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#pertence"/>
251     </owl:ObjectProperty>
252
253
254
255     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            treina -->
256
257     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#treina">
258         <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
259         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Tecnico"/>
260         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
261         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#pertence"/>
262     </owl:ObjectProperty>
263
264
265
266     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            treinado_por -->
267
268     <owl:ObjectProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#treinado_por">
269         <rdf:type rdf:resource="&owl;FunctionalProperty"/>
270         <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Tecnico"/>
```

```
271         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
272         <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#tem"/>
273         <owl:inverseOf rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#treina"/>
274     </owl:ObjectProperty>
275
276
277
278     <!-- http://www.w3.org/2002/07/owl#topObjectProperty -->
279
280     <owl:ObjectProperty rdf:about="&owl;topObjectProperty"/>
281
282
283
284     <!--
285     ////////////////////////////////////////////////////////////////////////////////////////

286     //
287     // Data properties
288     //
289     ////////////////////////////////////////////////////////////////////////////////////////

290      -->
291
292
293
294
295     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            data_nascimento -->
296
297     <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#data_nascimento">
298         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Pessoa"/>
299         <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Time"/>
300         <rdfs:range rdf:resource="&xsd;dateTime"/>
301         <rdfs:subPropertyOf rdf:resource="&owl;topDataProperty"/>
302     </owl:DatatypeProperty>
303
304
305
306     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            nome -->
307
308     <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#nome">
309         <rdfs:domain rdf:resource="&owl;Thing"/>
310         <rdfs:subPropertyOf rdf:resource="&owl;topDataProperty"/>
311     </owl:DatatypeProperty>
```

```
312
313
314
315     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            nome_completo -->
316
317     <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#nome_completo">
318         <rdf:type rdf:resource="&owl;FunctionalProperty"/>
319         <rdfs:domain rdf:resource="&owl;Thing"/>
320         <rdfs:subPropertyOf rdf:resource="&owl;topDataProperty"/>
321     </owl:DatatypeProperty>
322
323
324
325     <!-- http://www.w3.org/2002/07/owl#topDataProperty -->
326
327     <owl:DatatypeProperty rdf:about="&owl;topDataProperty"/>
328
329
330
331     <!--
332     ////////////////////////////////////////////////////////////////////////////////////////

333     //
334     // Classes
335     //
336     ////////////////////////////////////////////////////////////////////////////////////////

337      -->
338
339
340
341
342     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Arbitro -->
343
344     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Arbitro">
345         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Pessoa"/>
346         <owl:disjointWith rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Auxiliar_Tecnico"/>
347         <owl:disjointWith rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Dirigente"/>
348         <owl:disjointWith rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Jogador"/>
349         <owl:disjointWith rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Preparador_Fisico"/>
350         <owl:disjointWith rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Tecnico"/>
351     </owl:Class>
```

```
352
353
354
355     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Atacante -->
356
357     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Atacante">
358         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Jogador_Linha"/>
359     </owl:Class>
360
361
362
363     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Auxiliar -->
364
365     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Auxiliar">
366         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Arbitro"/>
367     </owl:Class>
368
369
370
371     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Auxiliar_Tecnico -->
372
373     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Auxiliar_Tecnico">
374         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Membro_Comissao_Tecnica"/>
375     </owl:Class>
376
377
378
379     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Club -->
380
381     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Club">
382         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Time"/>
383     </owl:Class>
384
385
386
387     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Defensor -->
388
389     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Defensor">
```

```
390         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Jogador_Linha"/>
391     </owl:Class>
392
393
394
395     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Dirigente -->
396
397     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Dirigente">
398         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Pessoa"/>
399     </owl:Class>
400
401
402
403     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Evento -->
404
405     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Evento"/>
406
407
408
409     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Falta -->
410
411     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Falta">
412         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Evento"/>
413     </owl:Class>
414
415
416
417     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Gol -->
418
419     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Gol">
420         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Evento"/>
421     </owl:Class>
422
423
424
425     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Goleiro -->
426
427     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Goleiro">
```

```
428         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Jogador" />
429     </owl:Class>
430
431
432
433     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Impedimento -->
434
435     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Impedimento">
436         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Evento" />
437         <rdfs:subClassOf>
438             <owl:Restriction>
439                 <owl:onProperty rdf:resource="http://www.semanticweb.org/
                        ontologies/2010/5/Ontology1277729329232.owl#sofrido_por" />
440                 <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">0</
                        owl:cardinality>
441             </owl:Restriction>
442         </rdfs:subClassOf>
443     </owl:Class>
444
445
446
447     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Jogador -->
448
449     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Jogador">
450         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Pessoa" />
451         <rdfs:subClassOf>
452             <owl:Restriction>
453                 <owl:onProperty rdf:resource="http://www.semanticweb.org/
                        ontologies/2010/5/Ontology1277729329232.owl#joga_no" />
454                 <owl:onClass rdf:resource="http://www.semanticweb.org/ontologies
                        /2010/5/Ontology1277729329232.owl#Selecao" />
455                 <owl:maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger
                        ">1</owl:maxQualifiedCardinality>
456             </owl:Restriction>
457         </rdfs:subClassOf>
458         <rdfs:subClassOf>
459             <owl:Restriction>
460                 <owl:onProperty rdf:resource="http://www.semanticweb.org/
                        ontologies/2010/5/Ontology1277729329232.owl#joga_no" />
461                 <owl:onClass rdf:resource="http://www.semanticweb.org/ontologies
                        /2010/5/Ontology1277729329232.owl#Club" />
462                 <owl:maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger
                        ">1</owl:maxQualifiedCardinality>
463             </owl:Restriction>
464         </rdfs:subClassOf>
```

```
465     </owl:Class>
466
467
468
469     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Jogador_Linha -->
470
471     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Jogador_Linha">
472         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Jogador"/>
473     </owl:Class>
474
475
476
477     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Jogo -->
478
479     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Jogo">
480         <owl:equivalentClass rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Partida"/>
481     </owl:Class>
482
483
484
485     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Juiz -->
486
487     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Juiz">
488         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Arbitro"/>
489     </owl:Class>
490
491
492
493     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Lateral -->
494
495     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Lateral">
496         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Defensor"/>
497     </owl:Class>
498
499
500
501     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Medico -->
502
```

```
503     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Medico">
504         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Membro_Comissao_Tecnica"/>
505     </owl:Class>
506
507
508
509     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Meia -->
510
511     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Meia">
512         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Meio_Campo"/>
513     </owl:Class>
514
515
516
517     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Meio_Campo -->
518
519     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Meio_Campo">
520         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Jogador_Linha"/>
521     </owl:Class>
522
523
524
525     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Membro_Comissao_Tecnica -->
526
527     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Membro_Comissao_Tecnica">
528         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Pessoa"/>
529     </owl:Class>
530
531
532
533     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Partida -->
534
535     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Partida"/>
536
537
538
539     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Pessoa -->
540
```

```
541     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Pessoa"/>
542
543
544
545     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Preparador_Fisico -->
546
547     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Preparador_Fisico">
548         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Membro_Comissao_Tecnica"/>
549     </owl:Class>
550
551
552
553     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Reserva -->
554
555     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Reserva">
556         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Arbitro"/>
557     </owl:Class>
558
559
560
561     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Selecao -->
562
563     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Selecao">
564         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Time"/>
565     </owl:Class>
566
567
568
569     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Tecnico -->
570
571     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Tecnico">
572         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Membro_Comissao_Tecnica"/>
573     </owl:Class>
574
575
576
577     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Time -->
578
```

```
579     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Time"/>
580
581
582
583     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Volante -->
584
585     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Volante">
586         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Meio_Campo"/>
587     </owl:Class>
588
589
590
591     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Zagueiro -->
592
593     <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Zagueiro">
594         <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Defensor"/>
595     </owl:Class>
596
597
598
599     <!-- http://www.w3.org/2002/07/owl#Thing -->
600
601     <owl:Class rdf:about="&owl;Thing"/>
602
603
604
605     <!--
606     ////////////////////////////////////////////////////////////////////////////////////////
607     //
608     // Individuals
609     //
610     ////////////////////////////////////////////////////////////////////////////////////////
611      -->
612
613
614
615
616     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Adilson_Batista -->
617
618     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Adilson_Batista">
```

```
619         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Tecnico"/>
620     </owl:NamedIndividual>
621
622
623
624     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Allysson -->
625
626     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Allysson">
627         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Zagueiro"/>
628     </owl:NamedIndividual>
629
630
631
632     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Andres_Sanchez -->
633
634     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Andres_Sanchez">
635         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Dirigente"/>
636     </owl:NamedIndividual>
637
638
639
640     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Bahia -->
641
642     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Bahia">
643         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Club"/>
644     </owl:NamedIndividual>
645
646
647
648     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Bahia_vs_Corinthians -->
649
650     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Bahia_vs_Corinthians">
651         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Jogo"/>
652     </owl:NamedIndividual>
653
654
655
656     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Brasil -->
```

```
657
658     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Brasil">
659         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Selecao"/>
660     </owl:NamedIndividual>
661
662
663
664     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Corinthians -->
665
666     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Corinthians">
667         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Club"/>
668     </owl:NamedIndividual>
669
670
671
672     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Eduardo -->
673
674     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Eduardo">
675         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Preparador_Fisico"/>
676     </owl:NamedIndividual>
677
678
679
680     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Elias -->
681
682     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Elias">
683         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Volante"/>
684     </owl:NamedIndividual>
685
686
687
688     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Fabio -->
689
690     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Fabio">
691         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Auxiliar_Tecnico"/>
692     </owl:NamedIndividual>
693
694
```

```
695
696     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Fenomeno -->
697
698     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Fenomeno">
699         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Atacante"/>
700         <owl:sameAs rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Ronaldo"/>
701     </owl:NamedIndividual>
702
703
704
705     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Flamengo -->
706
707     <owl:Thing rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Flamengo">
708         <rdf:type rdf:resource="&owl;NamedIndividual"/>
709         <tecnico_auxiliado_por rdf:resource="http://www.semanticweb.org/ontologies
                /2010/5/Ontology1277729329232.owl#Romario"/>
710     </owl:Thing>
711
712
713
714     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Jael -->
715
716     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Jael">
717         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Atacante"/>
718     </owl:NamedIndividual>
719
720
721
722     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Joaquim_Grava -->
723
724     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Joaquim_Grava">
725         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Medico"/>
726     </owl:NamedIndividual>
727
728
729
730     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Julio_Cesar -->
731
```

```
732     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Julio_Cesar">
733         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Goleiro"/>
734     </owl:NamedIndividual>
735
736
737
738     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Morais -->
739
740     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Morais">
741         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Meia"/>
742     </owl:NamedIndividual>
743
744
745
746     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Roberto_Carlos -->
747
748     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Roberto_Carlos">
749         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Lateral"/>
750     </owl:NamedIndividual>
751
752
753
754     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Romario -->
755
756     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Romario">
757         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Pessoa"/>
758     </owl:NamedIndividual>
759
760
761
762     <!-- http://www.semanticweb.org/ontologies/2010/5/Ontology1277729329232.owl#
            Ronaldo -->
763
764     <owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/2010/5/
            Ontology1277729329232.owl#Ronaldo">
765         <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Atacante"/>
766         <joga_no rdf:resource="http://www.semanticweb.org/ontologies/2010/5/
                Ontology1277729329232.owl#Corinthians"/>
767     </owl:NamedIndividual>
768 </rdf:RDF>
```

```
769
770
771
772 <!-- Generated by the OWL API (version 3.1.0.20069) http://owlapi.sourceforge.net
    -->
```

Listing A.4: Ontologia de Futebol Desenvolvida para Avaliação do MOFI.

Referências Bibliográficas

ALMEIDA, M. *Uma Ferramenta para Mineração Visual de Dados Usuando Mapas em Árvores e suas Aplicações*. Dissertação (Mestrado Profissional em Redes de Computadores) — Universidade Salvador, Salvador, 2003.

BEPPLER, F. D. *Emprego de RBC para Recuperação inteligente de informações*. Dissertação (Mestrado em Engenharia de Produção) — Universidade Federal de Santa Catarina, Santa Catarina, 2002.

BEPPLER, F. D. et al. Uma arquitetura para recuperação de informação aplicada ao processo de cooperação universidade empresa. 2005.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. 2001.

BREITMAN, K. *We Semântica a Internet do Futuro*. Rio de Janeiro: LTC, 2005.

DAVENPORT, T. H. *Ecologia da Informação: Por que so a tecnologia nao basta para o sucesso na era da informação*. [S.l.]: Futura, 1998.

FONTES, C. A.; CAVALCANTI, M. C.; MOURA, A. M.; SINAY, M. C. Recuperação de informações em documentos anotados semanticamente na Área de gestão ambiental. ONTOBRAS - III Seminário de Pesquisa em Ontologia no Brasil, 2010.

FRIEDMAN, V. Google pagerank: What do we know about it? 2007. Disponível em: <http://www.smashingmagazine.com/2007/06/05/google-pagerank-what-do-we-really-know-about-it>.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Padrões de Projeto - Soluções Reutilizáveis de Software Orientado a Objeto*. Porto Alegre: Bookman, 2000.

GRUBER, T. A translation approach to portable ontology specifications. 1993.

GRUBER, T. Collective knowledge systems: Where the social web meets the semantic web. 2007.

HATCHER, E.; GOSPODNETIC, O. *Lucene in Action - A guide to the Java search engine*. São Paulo: Manning, 2005.

HENDLER, J.; GOLBECK, J. Metcalfe's law, web 2.0, and the semantic web. 2007.

HORRIDGE, M.; KNUBLAUCH, H.; RECTOR, A.; STEVENS, R.; WROE, C. A practical guide to building owl ontologies using the protege-owl plugin and co-ode tools edition 1.0. August 2004. Disponível em: <http://www.co-ode.org/resources/tutorials%-/ProtegeOWLTutorial.pd>.

JONASSEN, D.; BEISSNER, K.; YACCI, M. *Structural Knowledge: Techniques for Representing, Conveying, and Acquiring Structural Knowledge*. [S.l.]: LEA - Lawrence Erlbaum Associates, 1993.

JORGE, E. *Método de Modelagem de Ontologia Baseado em Instâncias*. Tese (Qualificação de Doutorado em Difusão do Conhecimento) — UFBA, LNCC, UNEB, UEFS, UFABC, IFET, SENAI-CIMATEC, Salvador, 2010.

JORGE, E.; REIS, U.; EVANGELISTA, R.; CAJAHYBA, T.; PEREIRA, H. Web semântica: O futuro das aplicações. *Java Magazine*, 2010.

KASER, O.; LEMIERE, D. Tag-cloud drawing: Algorithms for cloud visualization. 2007.

LIMA, J. C.; CARVALHO, C. L. Ontologias - owl (web ontology language). 2005.

MANHãES, A. L. P.; SANTOS, N.; FARIAS, O. L. M. d. Ontologias aplicadas ao desenvolvimento de sigs: Estudo de caso sobre zoneamento municipal. 2006.

MANOLA, F.; MILLER, E. *RDF Primer*. Fevereiro 2004. W3C - World Wide Web Consortium. Acessado em 22 de Setembro de 2010. Disponível em: <http://www.w3-.org/TR/rdf-primer>.

MAíRA, G.; SILVA, L. C. Utilização de ontologias para marcação e recuperação de segmentos de vídeo. 2007. Monografia. Universidade Catolica do Salvador.

ONTOBRAS. *II Seminário de Pesquisa em Ontologia no Brasil*. Setembro 2009. Disponível em: <http://ontobra.comp.ime.eb.br>.

O'REILLY, T. What is web 2.0: Design patterns and business models for the next generation of software. 2005. Disponível em: <http://oreilly.com/web2/archive/what-is-web-20.htm>.

PEIRCE, C. S. *Semiótica e filosofia*. [S.l.]: Editora Univeridade de São Paulo, 1975.

PRIMO, A. O aspecto relacional das interações na web 2.0. *XXIX INTERCOM: Congresso Brasileiro de Ciências da Comunicação*, 2006.

RA, A. Morei GA, L. Alvaren RA, A. OliveiȮ nível do conhecimento e os instrumentos de representação: Tesauros e ontologias. 2004.

REIS, U.; JORGE, E.; EVANGELISTA, R.; CAJAHYBA, T. Ontology tagging - um componente para integração de ontologia e folksonomia. *ERBASE - Escola Regional Bahia Sergipe*, 2009.

REIS, U.; JORGE, E.; PEREIRA, H. Mofi: Um modelo computacional para recuperação de informação baseado em ontologias, folksonomia e indexação automática de conteúdo. *III Seminário de Pesquisa sobre Ontologia no Brasil*, 2010.

SARULADHA, K.; AGHILA, G.; PENCHALA, S. K. Design of new indexing techniques based on ontology for information retrieval systems. *International Conference, Information and Communication Techinologies*, Kochi, Kerala, India, 2010.

SILVEIRA, M. d. L. d. *Recuperação Vertical de Informação Um Estudo de Caso na Área Jurídica*. Tese (Doutorado em Ciência da Computação) — Universidade Federal de Minas Gerais, Belo Horizonte, 2003.

SILVESTRE, F. R. *XQOM: Um framework de consulta a dados semi-estruturados*. Dissertação (Mestrado em Modelagem Computacional) — Centro de Pós Graduação e Pesquisa Visconde de Cairú, Salvador, 2005.

SMITH, M. K.; WELTY, C.; MCGUINNESS, D. L. *OWL Web Ontology Language Guide*. Fevereiro 2004. W3C - World Wide Web Consortium. Acessado em 20 de Setembro de 2010. Disponível em: <http://www.w3.org/TR/2004/REC-owl-guide-20040210>.

SOMMERVILLE, I. *Engenharia de Software*. São Paulo: Addison Wesley, 2003.

SOUZA, R. R. Sistemas de recuperação de informações e mecanismos de busca na web: Panorama atual e tendências. *Perspectiva Ciência Informação*, Belo Horizonte, 2006.

WAL, T. V. Folksonomy coinage and definition. 2007.

MOFI: Modelo Computacional para Recuperação de Informação baseado em Ontologias, Folksonomia e Indexação Automática de Conteúdo

Uedson Santos Reis

Salvador, Fevereiro de 2011.