UNIVERSIDADE FEDERAL DE SANTA CATARINA

FERRAMENTA PARA EXTRAÇÃO E ANÁLISE DE PUBLICAÇÕES EM SITES DE ARTIGOS ACADÊMICOS

DIORGES FILIPE LOHN

Florianópolis, SC
2013/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

FERRAMENTA PARA EXTRAÇÃO E ANÁLISE DE PUBLICAÇÕES EM SITES DE ARTIGOS ACADÊMICOS

DIORGES FILIPE LOHN

Trabalho de conclusão de curso apresentado como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Florianópolis, SC
2013/2

DIORGES FILIPE LOHN

FERRAMENTA PARA EXTRAÇÃO E ANÁLISE DE PUBLICAÇÕES EM SITES DE ARTIGOS ACADÊMICOS

Trabalho de conclusão de curso apresentado como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador(a): Carina Friedrich Dorneles

Banca examinadora:

Frank Augusto Siqueira

Renato Fileto

Agradeço primeiramente aos meus pais pelo apoio até hoje,
posteriormente agradeço, aos professores que eu tive
ao longo desta jornada, aos amigos de perto
e aos de longe, aos que fizeram e fazem parte
desta minha vida acadêmica, alguns de um modo
bastante especial, pois sem o auxílio e conforto
de alguns, talvez esta etapa não teria sido concluída.
Agradeço, e muito, àqueles que tornaram
este trabalho possível. Pela paciência,
conversas, apoio, ideias, e por fim,
por terem sido peça fundamental
nessa reta final da graduação.
Saibam que, sem a ajuda de vocês,
este trabalho com certeza não estaria aqui.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

HTTP	HyperText Transfer Protocol
SQL	Structured Query Language
HTML	HyperText Markup Language
BD	DataBase
API	Application Programming Interface
IDE	Integrated Development Environment
SGBD	Sistema de Gerenciamento de Banco de Dados
URL	Uniform resource locator
CSS	Cascading Style Shield
DIV	Division (*tag* HTML)
DOM	Document Object Model
W3C	World Wide Web Consortium

Resumo

Com o crescente número de páginas e dados publicados na *Web*, cresceu também a dificuldade de se localizar determinados assuntos mais específicos. Existe um imenso trabalho por parte das empresas que oferecem o serviço de busca na internet para tornar possível a captura de todos os endereços disponíveis na *Web* dos mais variados assuntos.

Junto ao crescimento do conteúdo contido na *Web*, aumentam os esforços da comunidade de pesquisa de se estudar, tentar compreender e se aprofundar nos novos conceitos e tendências gerados a partir destes assuntos. A partir destes estudos são gerados, muitas vezes, artigos científicos, papers, resumos, livros e demais tipos de publicações acadêmicas, sempre com a orientação de algum professor especialista ou entusiasta no assunto. Estes documentos acadêmicos são geralmente submetidos a avaliações e caso atinjam os níveis cabíveis, são publicados em sites de simpósios específicos da área a qual o conteúdo está relacionado.

A ideia deste trabalho de conclusão de curso consiste na criação de um *Web Crawler* focado, batizado com o nome de PubFinder, com a função de buscar junto a três sites de simpósios ou site de publicações de artigos científicos o nome de professores e seus respectivos trabalhos publicados. É possível também, em dois dos três site escolhidos, a busca pelo título da publicação ou por uma instituição de ensino.

Com base no resultado obtido pelo *Crawler*, quando usado como parâmetro de busca o nome de um professor, pode-se ter ideia por exemplo, comparando o número de publicações encontrado, o quão atualizado está o currículo *Lattes* do professor indicado na busca.

Palavras-chave: Web Crawler; Web Crawler focado; Artigos Acadêmicos; Simpósio; Extração de dados.

Abstract

With the increasing number of pages and data published on the Web, also increased the difficulty in locating certain more specific matters. There is a lot of work for companies that provide the service of searching the internet to turn it possible to capture all addresses available on the Web all kinds of subjects.

Along the growth of the content contained on the Web increases to the efforts of research community to study, try to understand and delve into new concepts and trends generated from these subjects. From these studies are generated, often scientific articles, papers, abstracts, books and other types of academic publications, always with the guidance of a specialist teacher or enthusiast on the subject. These academic papers are usually assessed with case and reach reasonable levels, are published symposia on specific sites of the area to which the content is related.

The idea of this work for completion of course is the creation of a focused Web crawler, named as PubFinder, with the function of looking at three specific sites symposia or site of scientific articles publishing the name of teachers and their work published. It is also possible, in two of the three sites chosen, search for the title of the publication or an educational institution.

Based on the result obtained by Crawler, when used as a search parameter name of a teacher, it may have an idea for example, by comparing the number of publications found how up to date is the teacher Lattes indicated in the search.

Keywords: Web Crawler, Focused Web Crawler; Academic Articles; Symposium; Data extract.

1. Introdução

Embora a Internet tenha um número gigantesco de páginas a tarefa de buscar informações nela é, ou pelo menos parece ser, trivial ao usuário, já que esta é realizada em frações de segundos por sites buscadores. Porém tornar esta busca viável exige, e muito, dos motores de busca. Principalmente quando levado em consideração o fator de a *Web* ainda não ser semântica, assim sendo, um motor de busca faz todo o trabalho, basicamente, baseado em heurísticas no que se refere ao fator semântico dessas buscas. Dito isso, cabe então saber, pelo menos em linhas gerais, o conceito de um motor de busca e como ele consegue respostas tão rápidas em um meio tão vasto de informações, como é a *Web* hoje. Por fim, cabe ainda, relacionar as semelhanças da tarefa feita pelos motores de busca com a extração de informação realizada pelo *Crawler* implementado neste trabalho.

Um motor de busca pode ser conceituado como um software projetado para encontrar informações a partir de palavras-chave, reduzindo tanto quanto for possível o tempo necessário para encontrar as informações. A varredura na *Web* é uma parte essencial do motor de busca, sendo que para alcançar uma varredura de grande desempenho, é apropriado otimizar a eficiência, já que o número de páginas cresce exponencialmente com a profundidade da varredura.

Motores de busca da *Web* trabalham armazenando informações sobre muitas páginas da *Web*, que eles recuperam a partir do próprio HTML. Estas páginas são recuperadas por um *Web Crawler* (também conhecido como *Spider*) - um *WebBrowser* automatizado que segue cada link no site. O conteúdo de cada página é então analisado para determinar como ele deve ser indexado (por exemplo, as palavras podem ser extraídas a partir dos títulos, o conteúdo da página, cabeçalhos ou campos especiais chamados *meta tags*). Os dados sobre páginas da *Web* são armazenados em um banco de dados de índices para uso em consultas posteriores. Esta consulta pode ser uma única palavra. O índice ajuda a encontrar as informações o mais rápido possível. Exemplificando: alguns motores de busca, como o Google, guardam toda ou parte da página de origem (referido como um cache), bem como informações sobre as páginas *Web*, enquanto outros, como a AltaVista, armazenam cada palavra de cada

página que encontrar. Tudo o que é armazenado influi diretamente na velocidade de resposta e no poder de precisão do motor.

Para manter estes arquivos de indexação de páginas de um motor de busca atualizados, implementa-se um *Web Crawler*, que fica varrendo a Internet indo de uma página para outra, armazenando os endereços visitados [LIU]. Os motores de busca mais comuns como o Google, buscam manter, de uma forma geral, uma base de dados com o maior número de informações possível sobre as páginas contidas na Internet. Esta estratégia de indexação é chamada de Big Table e é amplamente utilizada hoje em dia. Além de saber qual é o endereço de uma página, um motor de busca tem interesse em saber também o conteúdo da mesma, para que em uma busca, seja possível informar quais informações são fornecidas por determinada página. Desta forma, neste trabalho é abordada a extração de informações referentes a conteúdo de sites.

Quanto ao *Crawler* desenvolvido neste trabalho, trata-se de um *Crawler* focado e como tal faz buscas em páginas de interesse definidas previamente, onde sua finalidade é relacionada a publicações científicas publicadas em sites repositórios específicos. Assim, o usuário digita o termo desejado, em qual site repositório será feita a busca e qual o termo que ele está buscando, tendo desta forma uma busca direcionada. Após a realização da busca e extração das informações, que são baseadas em tags HTML, o resultado é exibido ao usuário e também armazenado em um banco de dados. O propósito desde armazenamento é que em eventuais próximas buscas iguais o banco de dados funcione com uma espécie de cache, ou seja, ao invés de acontecer uma nova varredura é utilizada uma “memória” de acesso muito mais rápida, neste caso a base de dados.

Durante o desenvolvimento da ferramenta foram encontrados alguns problemas, uns de fácil resolução e outros de maior complexidade. Cada repositório esboçou uma dificuldade na extração das informações. No portal DBLP a dificuldade principal foi encontrar o início das informações relevantes de cada busca. No portal PubMed o desafio foi fazer a paginação entre os registros. Já no portal Capes, o mais problemático dos portais utilizados o problema maior era na inconstância da navegação pelo site, fazendo com que muitas vezes o *Crawler* tivesse acesso a páginas com erros,

sem conteúdo ou inexistentes. Identificar estes erros foi demasiadamente trabalhoso. Estes problemas serão detalhados no capítulo de desenvolvimento da ferramenta.

Desta forma, salvas as devidas proporções e o fato de que neste trabalho há um escopo restrito, assim como os nomes passíveis de busca, é possível dizer que o *Crawler* implementado é uma espécie de motor de busca, já que faz uma varredura automática sob uma determinada fonte de dados e guarda buscas já executadas a fim de otimizar desempenho.

Ainda sobre o *Crawler* implementado neste trabalho, o objetivo principal do mesmo é gerar uma base de conhecimento das publicações disponibilizadas na *Web*. No que se refere à sua utilização em casos práticos, o resultado obtido por ele, quando realizada a busca por publicações de determinado professor/pesquisador, pode ser comparado ao número de publicações que este possui em seu Currículo Lattes. Da comparação destes números é possível inferir um nível preliminar de atualização do Currículo Lattes, uma vez que se o número de publicações no Lattes for menor, há então o caso de publicações existentes e não constantes no Lattes.

Este trabalho está estruturado da seguinte forma: No capítulo 2 é tratada a fundamentação teórica utilizada para o entendimento do desenvolvimento da ferramenta; no capítulo 3 são apresentados os trabalhos relacionados com o tema ao qual a ferramenta se designa; no capítulo 4 é descrita a ferramenta, sua visão geral, arquitetura e modo de implementação; no capítulo 5 são apresentados os resultados dos experimentos sobre a ferramenta e por fim; no capítulo 6 são apresentadas as conclusões e trabalhos futuros.

2. Conceitos básicos

Este capitulo irá abordar os conceitos básicos que são necessários para o entendimento da idéia, concepção e desenvolvimento da ferramenta relatada por este trabalho.

2.1 Web Crawlers

Um *Web Crawler* é um programa ou *Script* que metodicamente analisa e percorre páginas *Web* para criar um índice dos dados de interesse. É considerado um agente de software que utiliza uma lista de URLs a serem navegadas. A partir dessa navegação o *Crawler* identifica todos os links das páginas e adicionas-os na lista de URL que serão visitadas [DHENAKARAN et. al, 2011].

O desenho típico dos motores de busca é uma "cascata", onde ela rastreia, indexa e pesquisa. A maioria dos projetos de motores de busca consideram o *Crawler* apenas como a primeira etapa em uma busca da *Web*, com pouco feedback dos algoritmos de classificação para o processo de rastreamento. Este é um modelo em que as operações são executadas nesta ordem: primeiro o rastreamento, em seguida, indexação, e depois a pesquisa como mostra a Figura 1. [CASTILHO, 2004].

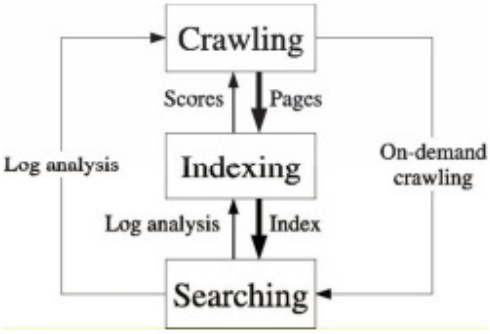

Figura 1 - Arquitetura dos Motores de Busca (CASTILHO, 2004)

Os *Crawlers* podem ser classificados segundo sua forma de rastreamento, ou seja, a regra que utilizam para rastrear páginas. Seguem algumas classificações [CASTILHO, 2004]:

a. *Crawlers* de caminho ascendente: capturam todos os recursos de um site em particular. Por exemplo, dado um link http://www.ensino.com.br/curso/graduação/, o *Crawler* analisa este caminho completo e posteriormente percorre cada nível na hierarquia do link até chegar à raiz (http://www.ensino.com.br/curso/ e http://www.ensino.com.br/) [DHENAKARAN et. al, 2011];
b. *Crawlers* de normalização de URL: referem-se ao processo de modificar e padronizar uma URL para torná-la consistente, a fim de evitar a indexação do mesmo recurso mais de uma vez. Existem vários tipos de normalização que podem ser realizadas, incluindo a conversão de URLs em letras minúsculas, remoção de segmentos "." e "..", entre outras [DHENAKARAN et. al, 2011] [BATSAKIS et. al, 2009];
c. *Crawlers* Focados [MENCZER et al., 2004]: o usuário define como entrada um tópico e um conjunto de URLs de partida (semente) para orientar a busca através de páginas de interesse. Eles incorporam critérios para a atribuição de prioridades de *download* mais altas com base na probabilidade de conduzir a páginas sobre o tema da consulta. O *Crawler* percorre recursivamente os *links* contidos nas páginas baixadas. Normalmente, as prioridades de *download* são calculadas com base na semelhança entre o tema e o texto âncora de um *link* ou entre o tema e o texto da página que contém o *link* (mais provável, eles estão relacionados a páginas sobre o tema da consulta);
d. *Crawlers* Semânticos [EHRIG & MAEDCHE, 2003]: são uma variação do clássico *Crawler* focado. Prioridades de *downloads* são atribuídas a páginas através da aplicação de critérios de similaridade semântica para a relevância: a página e o tópico podem ser relevantes se eles compartilham conceitualmente (mas não necessariamente lexicalmente) termos

semelhantes. A semelhança conceitual entre os termos é definida usando ontologias;

e. *Crawlers* de Aprendizagem [PANT & SRINIVASAN, 2005]: aplicam um processo de treinamento para a atribuição de prioridades de visitas a páginas da *Web* e para orientar o processo de rastreamento. São caracterizados pelo aprendizado das páginas *Web* ou caminhos (através de links) relevantes.

O processo de percorrer a *Web* é chamado de *crawling* ou *spidering*. Muitos motores de busca utilizam *Crawlers* para encontrar conteúdo atualizado e agilizar o processo de busca. Além disso, um *Crawler* pode ser utilizado para auxiliar e automatizar diversas tarefas, tais como: procurar e-mails, validar links, validar conteúdos HTML, extrair informações, etc. [CORRÊA, 2012]

A seguir segue um exemplo de arquitetura de *Cloud Crawler* proposta para avaliar o desempenho de aplicações em nuvem de infraestrutura. A arquitetura do *Crawler* é composta de quatro módulos: *Engine, Parser, Builder e Executer* (ver Figura 2).

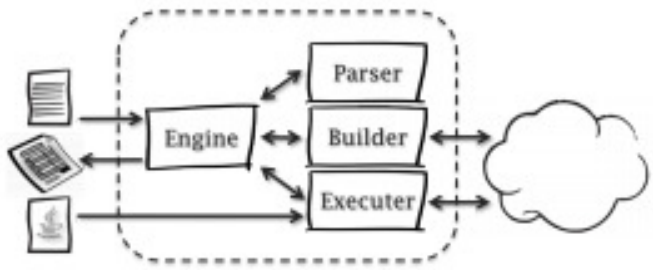

Figura 2 - Arquitetura do *Cloud Crawler.*

- *Engine* é o módulo principal da ferramenta, e tem como função responder aos comandos de invocação do usuário e coordenar as ações dos demais módulos.

- O *Parser* tem a função de interpretar e validar a especificação dos cenários descritos em Crawl, tambem sendo responsável pela resolução das variáveis declaras dinamicamente na linguagem.
- A função do *Builder* é preparar os recursos virtuais na nuvem e configurar os componentes da aplicação que serão executados nesses recursos. Para a preparação dos recursos, o *Builder* utiliza a biblioteca de código aberto *jclouds*, e para a configuração dos componentes na nuvem é utilizada a ferramenta JSch.
- O modulo *Executer* se encarrega de executar os cenários na nuvem com os recursos preparados pelo *Builder* e de coletar os resultados das avaliações. A interação do *Executer* com os componentes da aplicação durante a execução dos cenários é feita exclusivamente através da interface de comunicação com a nuvem, cuja implementação é fornecida pelo usuário.

Ao final, o módulo Engine consolida os dados coletados em cada cenário pelo *Executer* e em seguida os disponibiliza ao usuário na forma de planilhas. Uma característica importante do *Crawler* é a forma como ele gerencia o ciclo de vida das máquinas virtuais na nuvem. Em vez de solicitar a criação de novas máquinas virtuais ao provedor de nuvem, o *Crawler* reutiliza as maquinas virtuais previamente criadas pelo usuário, apenas alterando os seus perfis, caso necessário. Essa estratégia tem como principal vantagem a diminuição do custo de execução dos cenários na nuvem, uma vez que, como a maioria dos provedores IaaS atuais adota um modelo de precificação onde é cobrado um mesmo valor por hora ou fração de hora utilizada, dependendo do tempo de execução dos cenários, reutilizar uma máquina virtual existente pode sair muito mais barato do que solicitar a criação de uma nova máquina ao provedor. [CUNHA, NABOR & SAMPAIO]

2.2 Dados estruturados, semi-estruturados e não estruturados

A informação armazenada em banco de dados é conhecida como dados estruturados porque é representada em um formato rígido, já que cada registro em uma tabela de banco de dado relacional segue o mesmo formato dos outros registros daquela tabela. Porém nem todos os dados seguem esta classificação, em algumas aplicações, os dados são coletados de maneira *ad hoc* antes que se saiba como eles serão armazenados e gerenciados. Estes dados podem possuir uma estrutura, mas nem toda a informação coletada terá estrutura idêntica. Alguns atributos podem ser compartilhados entre várias entidades, mas outros podem existir apenas para algumas poucas entidades. Além disso, atributos adicionais podem ser introduzidos, e não há nenhum esquema predefinido para isso. Esse tipo de dado é conhecido como dado semi-estruturado. [ELMASRI & NAVATHE]

Uma diferença importante entre dados estruturados e semi-estruturados refere-se a como os construtores do esquema são manipulados. Em dados semi-estruturados, a informação do esquema está misturada com os valores dos dados, uma vez que cada objeto de dados pode ter atributos diferentes que não são conhecidos com antecedência. Por isso, esse tipo de dados é às vezes chamado dado autodescritivo. [ELMASRI & NAVATHE]

Além dos dados estruturados e semi-estruturados, existe uma terceira categoria, conhecida por dados não estruturados porque existe uma indicação muito limitada do tipo de dados. Um exemplo típico é um documento de texto que contém informação embutida. As páginas Web em HTML que contêm alguns dados são consideradas dados não estruturados. [ELMASRI & NAVATHE]

É importante ressaltar que a extração de informação, por meio de fontes de dados não estruturadas, pode tornar-se uma tarefa desafiadora, pois requer mecanismos para acessar o conhecimento, gerir e manipular grande um volume de dados, além de outras funcionalidades. A comunidade de Processamento de Linguagem Natural (PLN) (PLN é um tipo de extração de informação em fontes não estruturadas) está ligada a muitas outras áreas distintas, tais como: Aprendizado de

Máquina, Recuperação de Informação, Banco de Dados, *Web* e Análise de Documentos. [CORRÊA, 2012]

2.3 Extração de Informação

Cada desenvolvedor estrutura suas páginas de acordo com suas necessidades e conhecimento, aumentando bastante a complexidade da tarefa de extração de dados relevantes dessas diversas fontes. Portanto, é necessário utilizar técnicas e/ou ferramentas para Extração de Informação (EI). Estas são definidas como sistemas capazes de capturar somente os dados relevantes disponíveis em documentos diversos. EI é uma técnica clássica de mineração de texto, que tem como objetivo encontrar espécies de informações em determinados textos. [CORRÊA, 2012]

A estratégia mais utilizada é a análise de "*tags*" (marcas) nos textos, podendo indicar a presença de um dado. Por exemplo, o termo "CEP" pode indicar um número precedente ao código postal de um endereço [CORRÊA, 2012].

As definições acima cabem muito bem à realidade deste trabalho, pois a extração de informação realizada pelo *Crawler* em questão é através de *tags*, no caso *tags* HTML, e esta extração é bem específica, sendo válida somente para os repositórios definidos. No caso de outras páginas HTML, com outras definições de *tags*, o *Crawler* não opera.

3. Trabalhos Relacionados

Este capítulo faz um resumo de alguns trabalhos que estão relacionados ao desenvolvimento da ferramenta PubFinder. São apresentados suas principais características, semelhanças e diferenças em relação à abordagem proposta pelo trabalho e que será detalhadamente descrita ao longo dos capítulos.

3.1 I-FRIT - Ferramenta para extração de Web Forms

Este artigo [DOS SANTOS] foi motivado devido à imensa quantidade de dados digitais que encontram-se em bancos de dados escondidos, também chamados de *Deep Web*, fato que faz com que a comunidade de pesquisa em Banco de Dados desenvolva soluções que permitam o gerenciamento desses dados com vistas principalmente a atividades de integração e busca. O acesso a estes bancos de dados é possível somente através de formulários presentes em páginas *Web*, também chamados de "pontos de entrada" para o banco de dados. Estes formulários exibem alguns atributos do banco de dados (campos do formulário *Web*) sobre os quais o usuário especifica filtros e então submete consultas ao banco de dados. Pode-se citar como exemplo disto, encontrar *web*sites que ofereçam veículos para venda e que permitam definir consultas integradas a vários sites por determinadas marcas e modelos.

A dificuldade deste tipo de abordagem é a descoberta e a posterior extração de rótulos. Rótulos representam a identificação de atributos (nomenclatura de campos) presentes em formulários de acesso a um banco de dados escondido. O tratamento para esta dificuldade citada é bastante relevante, pois serve de base para o reconhecimento do esquema de bancos escondidos e para viabilizar subsequentes mecanismos de busca.

Este artigo aborda uma aplicação para detecção e extração automática de rótulos focada na análise estrutural do código HTML presente em páginas de formulários *Web*. Esta análise se baseia em um esquema de numeração para a hierarquia de tags HTML e um algoritmo que associa componentes de texto a campos

do formulário, garantindo uma descoberta eficiente de rótulos presentes nestes formulários.

Dos módulos implementados pelo autor, a API é o módulo mais importante, pois toda a lógica de reconhecimento e extração de formulários e rótulos está implementada nele. Para que o módulo API funcione, é necessário que ele receba como entrada uma URL e as configurações feitas pelo usuário no arquivo API.txt. Para realizar a navegação entre as páginas e a extração das URL's que irão abastecer a API, foi desenvolvido o módulo Crawler, que implementa a lógica de um *Web crawler*, além da interface com o usuário. Como já dito, o *Crawler* recebe como entrada as configurações feitas no arquivo Crawler.txt, que basicamente mantém a quantidade total de páginas a ser visitada e o *browser* que deve ser usado para acessar as páginas. Por fim, o módulo DAO é responsável pela conexão com o banco de dados e inserção das informações coletadas no processo.

Para resolver o problema da descoberta de um formulário em uma página qualquer da *Web*, foram definidas as características que o código HTML deve apresentar para que seja possível inferir de maneira segura a existência de um formulário. Uma característica é a forma de submissão dos dados preenchidos, que normalmente é feita através de botões. Sendo assim, é necessário detectar essas diferentes construções e classificá-las como sendo o elemento responsável pela submissão do formulário. Outra característica importante é a distância entre os elementos que compõem um formulário. Para este quesito, foram definidos dois parâmetros: 1) a distância máxima entre cada elemento do formulário e o elemento que representa o container ; e 2) a densidade máxima dos componentes de um formulário, que é calculada pela distância de um elemento do formulário e o próximo elemento do formulário, que esteja acima ou abaixo deste. Além disso, é possível escolher entre considerar apenas os elementos que estão visíveis na página ou considerar todos os componentes (visíveis e invisíveis).

O algoritmo foi desenvolvido com o uso de parâmetros configuráveis. Estes parâmetros podem ser modificados de acordo com a flexibilidade que se deseja permitir na busca.

O primeiro passo executado pelo algoritmo é a construção de uma árvore de elementos. A medida que a árvore vai sendo construída, cada novo elemento adicionado à estrutura recebe um ID, que o acompanha pelo resto do processamento. Este esquema permite não apenas identificar o elemento, mas também informar sua localização na árvore, uma vez que o ID do elemento pai é usado para compor o ID do elemento filho. O algoritmo que verifica a existência de um formulário faz, basicamente, uma varredura pela árvore de elementos e uma série de testes para saber se as configurações apresentadas anteriormente estão sendo atendidas. O processamento é feito de maneira recursiva para cada nodo contido na árvore.

O processo de extração de rótulos faz uma verificação com base nos ID's dos componentes e dos rótulos para determinar se é possível associar os elementos. A entrada para este processamento são basicamente duas listas de elementos: uma representando os componentes do formulário e a outra representando os possíveis rótulos. O algoritmo tenta encontrar o rótulo que está mais próximo de determinado elemento para então proceder a associação. Este cálculo de distância é feito utilizando a mesma ideia do algoritmo construído para reconhecimento de formulários.

Conforme a varredura se aprofunda cada elemento ganha um ID, por exemplo: 1, 1.1, 1.1.2, etc, todos de acordo com sua profundidade na árvore HTML. Estes IDs são montados em uma matriz e quanto mais próximos eles são maiores as chances deles serem considerados formulários. A partir desta aproximação são realizados os testes específicos para verificar se aqueles campos realmente contém rótulos e se são formulários de preenchimentos para banco de dados escondidos.

Comparando o resultado obtido pela extração automatizada da ferramenta com uma extração normal, a ferramenta obteve índices maiores do que 70% no cálculo da Medida-F (média das métricas de avaliação de testes revocação e precisão) caracterizando-se como um resultado positivo para o desempenho dela.

3.2 Listas Brancas e Negras, a evolução da estratégia de crawling sobre motor de Busca.

No artigo [WU ET AL] foi apresentado um estudo preliminar da evolução de um *Crawler* implementado para o sistema CiteSeerX. Este sistema utiliza um motor de busca que procura na *Web* por documentos acadêmicos e de pesquisa, principalmente em temas de ciências da computação e informação, para então retirar informações e indexá-las como metadados OAI, citações, tabelas e outros.

Para melhorar a precisão das pesquisas, foi substituída a lista negra por uma lista branca e foi comparada a eficiência da varredura de dados antes e depois dessa mudança. Vale destacar que lista branca significa que apenas certos domínios são considerados e outros não são percorridos, já a lista negra significa que a varredura é proibida para certas URL's, mas para os outros artigos ela é ilimitada.

O motor de busca CiteSeerX dá ao usuário acesso livre a milhões de artigos, pesquisas e livros. Ele foi desenvolvido para acesso de artigos públicos no formato PDF. Por causa disso, as buscas são altamente concentradas em instituições que realizam pesquisas, principalmente, página de universidades com acesso gratuito. Entretanto, além dos tradicionais artigos acadêmicos essas pesquisas podem englobar apresentações de slides, notas de classe, manuais e até propaganda. Embora tenham-se ferramentas para classificar os conteúdos, os documentos originais são obtidos e classificados numa escala de tempo geralmente maior do que as típicas páginas de internet.

Uma solução prática e intuitiva é criar uma lista negra, que contenha nomes de sites hospedeiros para se evitar. Quando o *Crawler* recebe uma informação de um título de uma página, ele confere se o título está na lista negra e então mostra ou não o título. Porém, essa solução tem várias desvantagens, sendo a mais evidente criá-la e editá-la manualmente decidindo o que deva ou não constar nela.

Pode-se pensar numa solução alternativa utilizando uma lista branca que contenha uma relação de links de alta qualidade, e então, compara-se a eficiência da varredura alternando entre usar as listas brancas e negras. Essa lista é gerada baseando-se num longo histórico de varredura e deve ter as seguintes propriedades:

Maioria de documentos úteis e páginas que são classificadas em um ranking no qual os primeiros colocados têm prioridade na busca.

Em virtude da grande quantidade de páginas da *Web*, é essencial que a listagem dos links priorizem as melhores, para isso existem técnicas que organizam URLs em níveis, por ranking, ou ainda pelo tamanho do site. O *Crawler* CiteSeerX desenvolve uma busca focando em certos tipos de documentos, o ranking tem um equilíbrio no número de documentos e citações, e ambos são representados em quantidade e qualidade de links.

Como selecionador de links temos o chamado F1 Ranking Indicator que tem sua pontuação definida com duas características: a média harmônica e a precisão. A fim de utilizador esse indicador, foram modificadas essas características substituindo-as por 'PD' e 'PC' que são definidas respectivamente como: fração de documentos varridos entre o número total de documentos na lista D (total de documentos recebidos) e a fração de citações entre o total recebido na lista C (total de citações recebidas).

Percebeu-se que o *Crawler* não precisa visitar mais do que a 5 páginas para alcançar 90% das páginas que geralmente os usuários visitam. Para restringir a varredura da lista branca, foi implementado o Parent Domain que é um filtro no qual apenas são aceitas páginas que tem o mesmo domínio que suas semelhantes. Notou-se também que o F1 trabalha melhor para documentos e citações em uma escala maior do que 10, quando são menores os valores de F1 são parecidos e assim perde-se eficiência.

Fazendo uma comparação entre as listas negras e brancas em um mesmo período de tempo, na branca o número de requisições por dia não muda significativamente, mas o número de documentos obtidos é o triplo se comparado com a da lista negra. Isso deve-se ao fato da varredura boa (branca) reduzir o numero de falhas e requisições filtradas. Já na lista negra apenas 20% dos novos documentos são úteis, em contraste com 50% da lista branca.

Finalizando, neste artigo foi construída uma lista branca contendo mais de 500.000 links de 8.000 domínios da *Web* baseados em mais de 700.000 links semelhantes acumulados no CiteSeerX. Os links desta lista branca fornecem 99,99%

de todos os documentos varridos. Esses links são organizados pelo ranking F1 que é calculado com base no número de documentos e citações de cada domínio.

Foi feita uma comparação entre as listas negras e brancas, e concluiu-se que a implementação da branca reduz significativamente o número de links não úteis e aumenta a fração de documentos uteis. Vale destacar que reduzindo o numero de links pode-se reduzir a oportunidade de descobrir novas pesquisas. Diante disto, fez-se uma combinação das lista negras e brancas.

Embora o estudo seja baseado no projeto do CiteSeerX, as conclusões podem ser generalizadas para tópicos genéricos focados em *Crawlers*. Os links são essenciais para melhorar a eficiência da varredura. Aumentar a profundidade da busca não é a melhor maneira de colher resultados uteis. A seleção de links deve ser equilibrada em quantidade e qualidade trazendo resultados baseados tanto na lista branca como na lista negra.

3.3. Análise Comparativa

A tabela a seguir mostra as semelhanças e as diferenças entre os trabalhos relacionados descritos no capítulo acima e a ferramenta desenvolvida neste trabalho (PubFinder), que será detalhada nos próximos capítulos.

Deve-se levar em consideração ao comparar as ferramentas citadas com a ferramenta desenvolvida neste trabalho de que a aplicação PubFinder possui um escopo definido e limitado. Seu embasamento foi na criação de uma ferramenta que faça a verificação de veracidade de atualização dos currículos Lattes de pesquisadores, da qual atualmente, não há nada desenvolvido que atenda esta necessidade.

Características	I-FRIT	CiteSeerX	PubFinder
Método de coleta	Varredura pelo HTML	Varredura pelo HTML	Varredura pelo HTML
Dados considerados	Formulários de envio de dados de sites	Conteúdo de documentos acadêmicos ou de pesquisa (PDFs, slides, notas de classe, manuais, propagandas)	Informações de identificação de publicações acadêmicas (Autor, título e ano)
Interface de coleta	Web (HTML)	Web (HTML)	Web (HTML)
Capacidade de aprendizado da máquina	Não	Não	Não
Navegação entre páginas	Sim	Sim	Sim
Crawler Focado	Sim	Sim	Sim
Extração de informações	Sim	Sim	Sim
Armazenagem em banco de conhecimento/dados	Sim	Sim	Sim
Indexação para melhor desempenho de pesquisa	Não	Sim	Não
Utilização do mecanismo de busca do site	Não	Sim	Sim

Tabela 1 - Comparação entre trabalhos relacionados e a ferramenta PubFinder.

A seguir, a descrição de cada característica.

Método de Coleta: tipo de ação que a ferramenta executa para atingir seu objetivo.

Dados considerados: tipo de dados que a ferramenta se objetiva a extrair.

Interface de coleta: ambiente de ação da ferramenta.

Capacidade de aprendizado da máquina: capacidade que a ferramenta tem de aprimorar os resultados com o tempo.

Navegação entre páginas: capacidade da ferramenta de interagir e navegar entre páginas *Web*.

Crawler Focado: especifica se a ferramenta tem um foco específico de navegação ou extração de informações.

Extração de informações: especifica se a ferramenta extrai aquela informação ou apenas é realizada a leitura.

Armazenagem em banco de conhecimento/dados: capacidade da ferramenta em gravar os dados obtidos em um banco de dados.

Indexação para melhor desempenho de pesquisa: capacidade da ferramenta em fazer uma indexação ou ranking de prioridade da informação para um melhor desempenho da ferramenta.

Utilização do mecanismo de busca do site: utilização do mecanismo de busca do *website* no qual a ferramenta está acessando.

Através da Tabela 1 pode-se observar que as ferramentas analisadas possuem características muito parecidas, o que favorece a troca de conhecimento da comunidade desenvolvedora. Embora as ferramentas CiteSeerX e PubFinder sejam muito semelhantes, a primeira é um projeto contínuo, bem difundido e iniciado em 1997, com ampla colaboração da comunidade acadêmica e desenvolvedora em geral.

Já a principal diferença entre a ferramenta desenvolvida e os trabalhos relacionados é o foco de extração das informações onde cada uma se propõe a extração de um tipo de informação, ressalvando o caso da ferramenta CiteSeerX, cujo objetivo é a obtenção de todo o conteúdo de uma publicação acadêmica ou de pesquisa, enquanto na ferramenta PubFinder o objetivo é listar apenas as informações de identificação de uma publicação (autor, título da publicação e ano). Com a simplicidade das informações buscadas pelo PubFinder já é possível cumprir o objetivo de se comparar as informações buscadas com os currículos Lattes dos referidos professores e pesquisadores.

Um dos fatores que levaram ao desenvolvimento da ferramenta é abrangência da ferramenta CiteSeerX, da qual não é tão difundida para pesquisas nacionais quanto a ferramenta PubFinder, uma vez que a ferramenta desenvolvida neste trabalho faz pesquisas diretamente no repositório Internacional de publicações de Ciência da Computação (DBLP), no maior portal Internacional de publicações de Medicina e Biologia (PUBMED) e no maior portal de publicações nacionais (Portal Capes) tendo melhores resultados nas buscas por nomes ou títulos nacionais.

4. Ferramenta PubFinder

Este capítulo aborda a ferramenta PubFinder, que permite ao usuário realizar buscas pré-definidas em 3 sites repositórios de publicações de artigos científicos, acadêmicos, livros ou periódicos. O nome da ferramenta pode ser traduzido pela junção das palavras *Publication* e *Finder,* onde tais termos são, na tradução literal da lingua inglesa, Publicação e Buscador. Estes termos definem basicamente o que a ferramenta faz.

O capítulo se divide nas seguintes partes: visão geral, onde é introduzido o funcionamento da ferramenta; arquitetura, onde são exibidas as diferentes partes do sistema; implementação, onde são discutidos os conceitos e as tecnologias empregadas no desenvolvimento da ferramenta.

4.1 Visão Geral

A ferramenta trabalha a partir de três etapas diferentes: (I) escolha do repositório, tipo da pesquisa e inserção do nome a ser pesquisado; (II) busca e extração dos dados no repositório escolhido e (III) exibição dos dados na tela para o usuário e gravação no banco de dados.

(I) ESCOLHA DO REPOSITÓRIO, TIPO DE PESQUISA E TERMO A SER BUSCADO

A primeira etapa consiste na interação do usuário com a interface gráfica da ferramenta, escolhendo em qual repositório ele deseja realizar a pesquisa, o tipo de informação que ele deseja e o nome a ser pesquisado. Os repositórios passíveis de busca são: DBLP - Computer Science Bibliography[1]; PUBMED - US National Library of Medicine[2]; e Portal Capes[3]. O motivo da escolha destes três repositórios foi proposital para que a ferramenta pudesse contemplar 3 diferentes áreas das publicações científicas. No primeiro se concentram as principais publicações internacionais a

[1] http://dblp.l3s.de/
[2] http://www.ncbi.nlm.nih.gov/pubmed
[3] http://www.periodicos.capes.gov.br/

respeito da Ciência da Computação, no segundo concentram-se todas as publicações a nível internacional da Medicina e Biologia, por fim, no terceiro repositório temos o portal brasileiro de publicações, o qual abrange não somente artigos científicos, como referência e visualização de livros, periódicos e recursos textuais.

(II) BUSCA E EXTRAÇÃO DOS DADOS

Na segunda etapa a ferramenta realiza a busca e extração dos dados selecionados pelo usuário junto ao portal de publicações escolhido. Esta etapa será melhor detalhada nos tópicos abaixo.

(III) EXIBIÇÃO DOS DADOS E INSERÇÃO NO BANCO DE DADOS

Por fim, na terceira etapa os dados extraídos do repositório são exibidos na interface gráfica da ferramenta e armazenados em um banco de dados. O intuito destes dados serem armazenados no banco de dados é de agilizar uma próxima pesquisa, ou seja, caso seja realizada uma nova busca por aquele mesmo termo e no mesmo repositório, o resultado será exibido quase instantaneamente, sem a necessidade de ser feita toda a varredura no site repositório.

4.2 Arquitetura

A Figura 3 mostra a visão geral da arquitetura desenvolvida, com seus principais módulos listados.

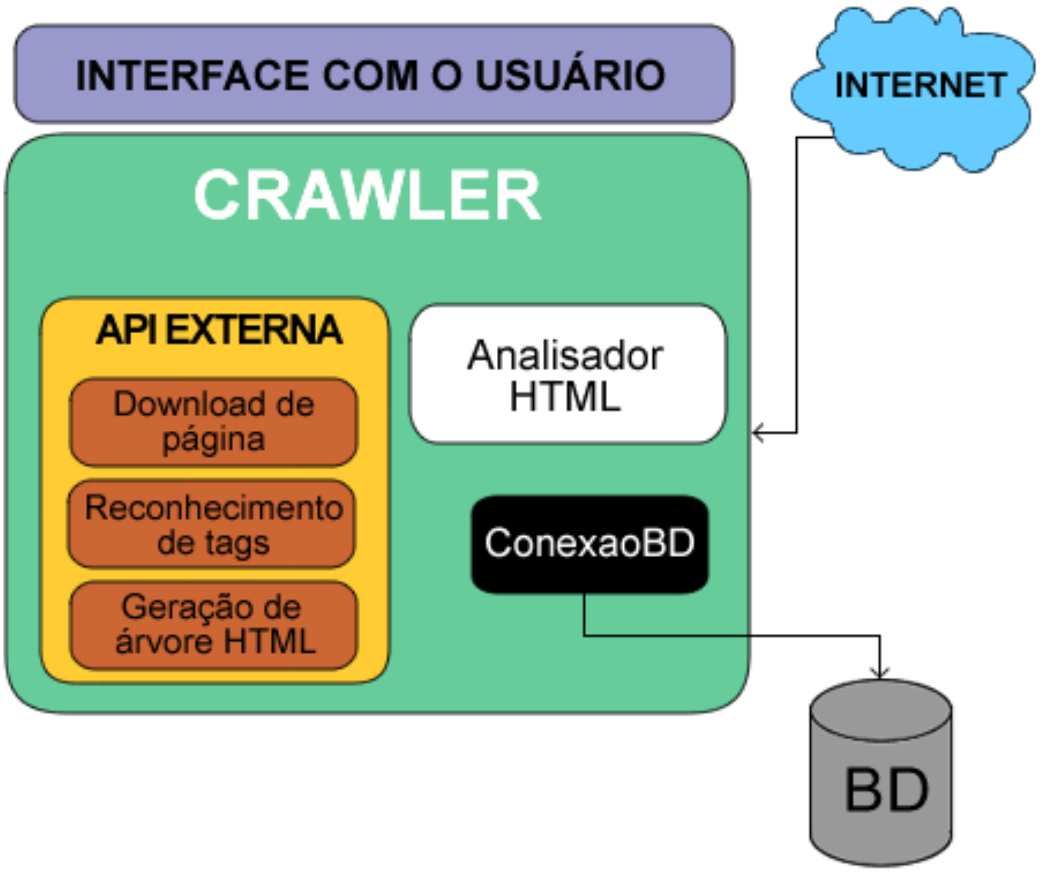

Figura 3 - Arquitetura da ferramenta PubFinder.

Detalhando um pouco mais a Figura 3, os itens abaixo mostram os principais módulos criados e qual sua interação dentro do software.

Interface: Responsável pela interação com o usuário, permitindo a especificação dos parâmetros de busca pelo usuário. Além disso, exibe o resultado da busca, ou seja, os dados extraídos durante o processo.

Etapas: comunica-se diretamente com o módulo Crawler para passar os parâmetros da busca e receber o resultado para exibição.

Crawler: Responsável por realizar o controle de toda a aplicação. É o coordenador de todas as etapas da ferramenta. Este módulo se comunica diretamente com todos os demais módulos.

Etapas: verifica junto ao módulo de Conexão no banco de dados (Módulo ConexaoBD) se aquela busca já foi realizada. Caso esta busca já tenha sido realizada, este módulo retorna para a interface os dados contidos no banco de dados para serem exibidos. Caso não existam registros desta busca no banco de dados o módulo utiliza-se da API externa para baixar a página HTML do repositório escolhido, com o termo e tipo de busca escolhidos. Esta página é então enviada ao módulo AnalisadorHTML para ser feita a extração dos dados e posteriormente sua exibição no módulo de Interface. As informações extraídas na busca, além de exibidas ao usuário, são salvas na base de dados.

AnalisadorHTML: Esta é a parte principal da aplicação. Neste módulo são realizadas todas as análises necessárias junto ao site repositório de publicações.

Etapas: as configurações passadas pelo módulo Crawler (termo buscado, repositório e tipo de busca) são usadas para extração das informações na página HTML, página esta que também é obtida através do módulo Crawler. Desta página HTML é montada, através da API Externa, uma árvore HTML onde é feita uma varredura buscando pelos dados das publicações pesquisadas. As informações encontradas são devolvidas ao módulo Crawler, que por sua vez as exibe na Interface e as armazena no banco de dados.

ConexãoBD: Como o próprio nome já sugere, este módulo é responsável pela conexão ao Banco de Dados. Nele estão as funções básicas de conexão/acesso, busca e inserção de dados junto ao banco. Nesta versão da aplicação não existe função de deleção dos dados no banco. Tal etapa deve ser realizada manualmente junto a um SGBD, caso seja necessário.

API Externa: Esta API resume-se a um *browser* sem interface gráfica, criado para poder ser incorporado em aplicações escritas em JAVA. Foi de fundamental uso

para a realização das atividades listadas na Figura 3. Sem o uso de tal recurso o projeto tornaria-se muito mais complexo e demorado para ser implementado. Conforme a Figura 3 ilustra, ela realiza as tarefas de download de páginas, a geração de uma árvore HTML e consequentemente a identificação de tags, facilitando o trabalho realizado no AnalisadorHTML na identificação do conteúdo importante para a aplicação.

4.2 Implementação

Toda a implementação do projeto foi realizada utilizando a linguagem JAVA junto com a IDE open source Eclipse versão Juno Service release 2. No projeto, conforme comentado no tópico anterior, foi necessária a utilização da API externa HtmlUnit versão 2.12 para download de páginas, geração da árvore HTML e reconhecimento/extração de tags e links. Já o banco de dados utilizado foi o SGBD PostgreSQL 8, utilizando-se da ferramenta gráfica de administração para PostgreSQL pgAdmin III. A utilização do banco de dados se justifica, a princípio por dois motivos, primeiro quando levada em consideração a possibilidade de repetição da busca, sendo neste caso buscado o resultado diretamente na base de dados, e por fim para manipular os registros de consulta da tabela.

A motivação para esta ferramenta ser desenvolvida em modo desktop (software) foi por se tratar inicialmente de um protótipo, idealizado para ser usado para um fim especifico com finalização em curto prazo e sem disponibilização de acesso público, optou-se pela utilização de tecnologias conhecidas pelo autor dar para agilidade no desenvolvimento. Por este motivo deixou-se de lado a implementação de um portal/site *Web* único de acesso e utilização.

O diagrama de classes exibido na Figura 4 demonstra as classes responsáveis pelo funcionamento da ferramenta. É importante ressaltar que a API externa acabou não sendo exibida no diagrama, mas ela foi utilizada nas classes Crawler e AnalisadorHTML. Nestas classes a API externa fez as funções de download da página, montagem da árvore e o reconhecimento dos elementos HTML.

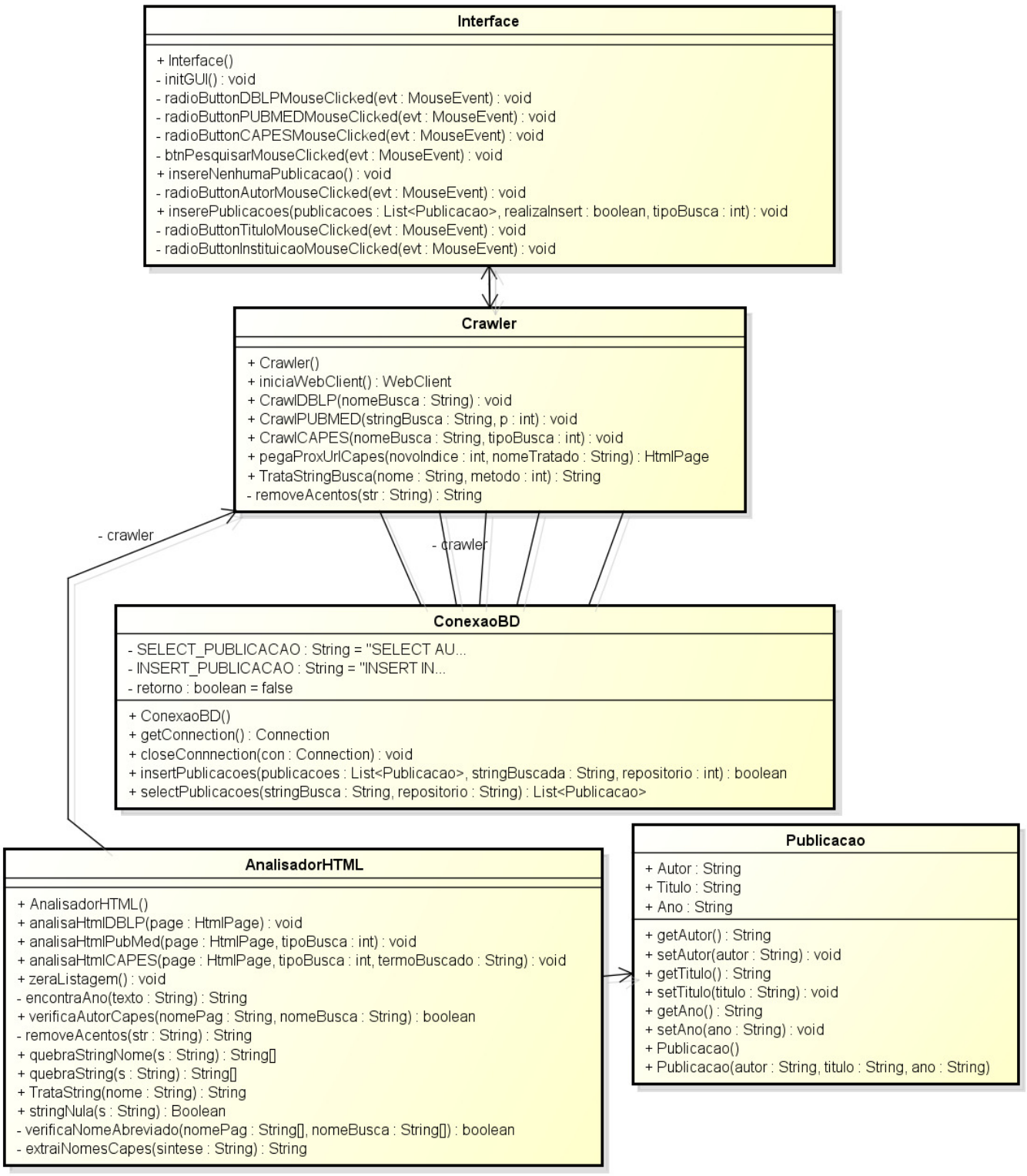

Figura 4 - Diagrama de classes da ferramenta

O funcionamento interno e a interação das classes será detalhado abaixo seguindo o fluxo do uso da aplicação.

A Interface gráfica foi implementada visando os critérios de usabilidade. Nela são escolhidos, através dos RadioButtons, o Site Repositório e o tipo de busca, assim como o termo o qual o usuário deseja pesquisar. A pesquisa é efetivamente executada no momento em que o usuário clica no botão “PESQUISAR”. A disposição das informações na interface pode ser visualizada na Figura 5.

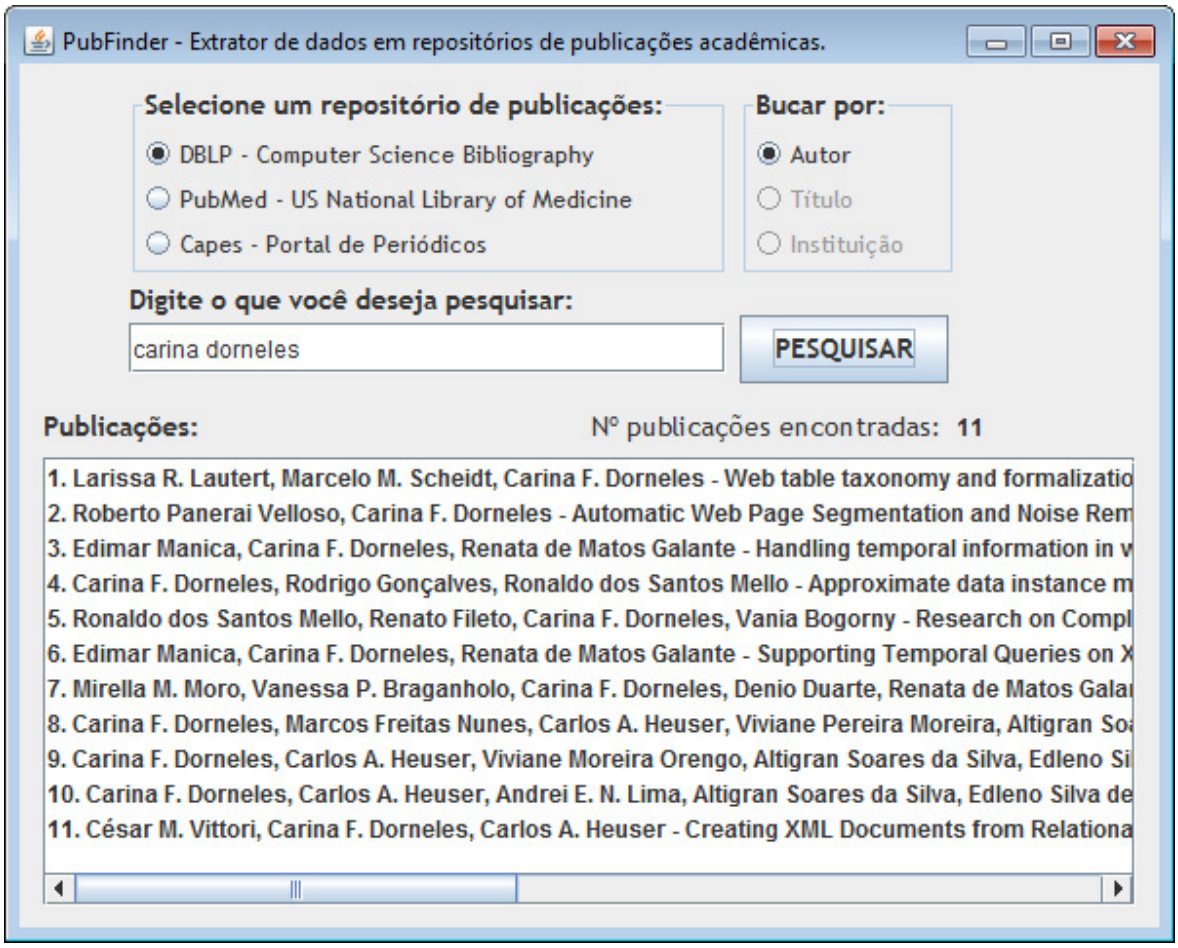

Figura 5 - Interface gráfica da aplicação exibindo o resultado de uma pesquisa

No momento em que as informações da interfaces são enviadas (ação do botão “PESQUISAR”) a classe Crawler inicializa um objeto da API HTMLUnit setando algumas configurações relevantes. O navegador que a API irá emular é o Mozilla Firefox versão

17 (atualmente tal navegador encontra-se na versão 22.0). As configurações que foram utilizadas e devem ser levadas em consideração são: a não interpretação e exibição dos erros gerados na leitura das páginas *HTML*, a interpretação de *CSS* e a habilitação do redirecionamento entre páginas. Este último parâmetro é de fundamental importância, já que caso ele não estivesse habilitado o evento de click em links não funcionaria durante a navegação virtual realizada pelo HTMLUnit.

Através do objeto API HTMLUnit já criado e configurado, a classe Crawler realiza o download da página do repositório selecionado na busca e utiliza os parâmetros informados pelo usuário. Esta página fica armazenada em memória com todos os recursos *HTML* disponíveis.

Para agilizar a busca foi utilizada a URL do sistema de busca do próprio site Repositório. Caso o sistema de busca do repositório não fosse utilizado, a ferramenta PubFinder teria que ser implementada para que fossem percorridos todos os *tags*/links de acesso de todas as páginas do site, buscando dentro de cada uma se esta teria o conteúdo de interesse. Isto tornaria a pesquisa algumas ordens de grandeza mais lenta, devido ao tamanho e tempo de processamento das páginas acessadas. Como a varredura de todas as páginas não era o foco principal da aplicação, optou-se pela utilização dos mecanismos de busca do próprio site.

A montagem da *String* correspondente à URL deste mecanismo se baseia na concatenação da URL de busca do próprio site com o termo buscado. Antes de se fazer o download da página é necessário tratar o termo buscado e realizar a adição (se possível) de auxiliadores de busca. No tratamento do termo buscado são retirados os acentos das letras ou caracteres especiais (Ç, Á, Ã, É, Õ, À) e é trocado o espaço em branco pelo seu código respectivo no site repositório (ex: %20). Já os termos auxiliadores são possíveis parâmetros inseridos na própria URL que facilitam o download. Como exemplo disto podemos citar a quantidade de registros a serem exibidos, a ordenação e qual o tipo de busca o usuário deseja realizar. Este último parâmetro só foi utilizado no portal PubMed, já que foi o único que permitiu tal configuração.

O próximo passo é enviar a página baixada para a classe AnalisadorHTML, para que dela sejam extraídas as informações requisitadas pelo usuário. Até este ponto do

fluxo as operações de *download* dos três repositórios são iguais, com exceção das *URLs* baixadas. Já na classe AnalisadorHTML cada pagina é tratada de uma maneira para que seja feita a devida extração dos dados relevantes.

Antes da implementação ser feita, nesta etapa, foi analisada a estrutura HTML de cada site repositório. Após análise da composição do HTML, foi descoberto como chegar até as informações relevantes através de suas *tags*. As informações relevantes buscadas aqui são: Autor, Título da publicação e Ano. O tipo de *tag* mais utilizado foi o *div*. A Figura 6 ilustra este processo no repositório DBLP.

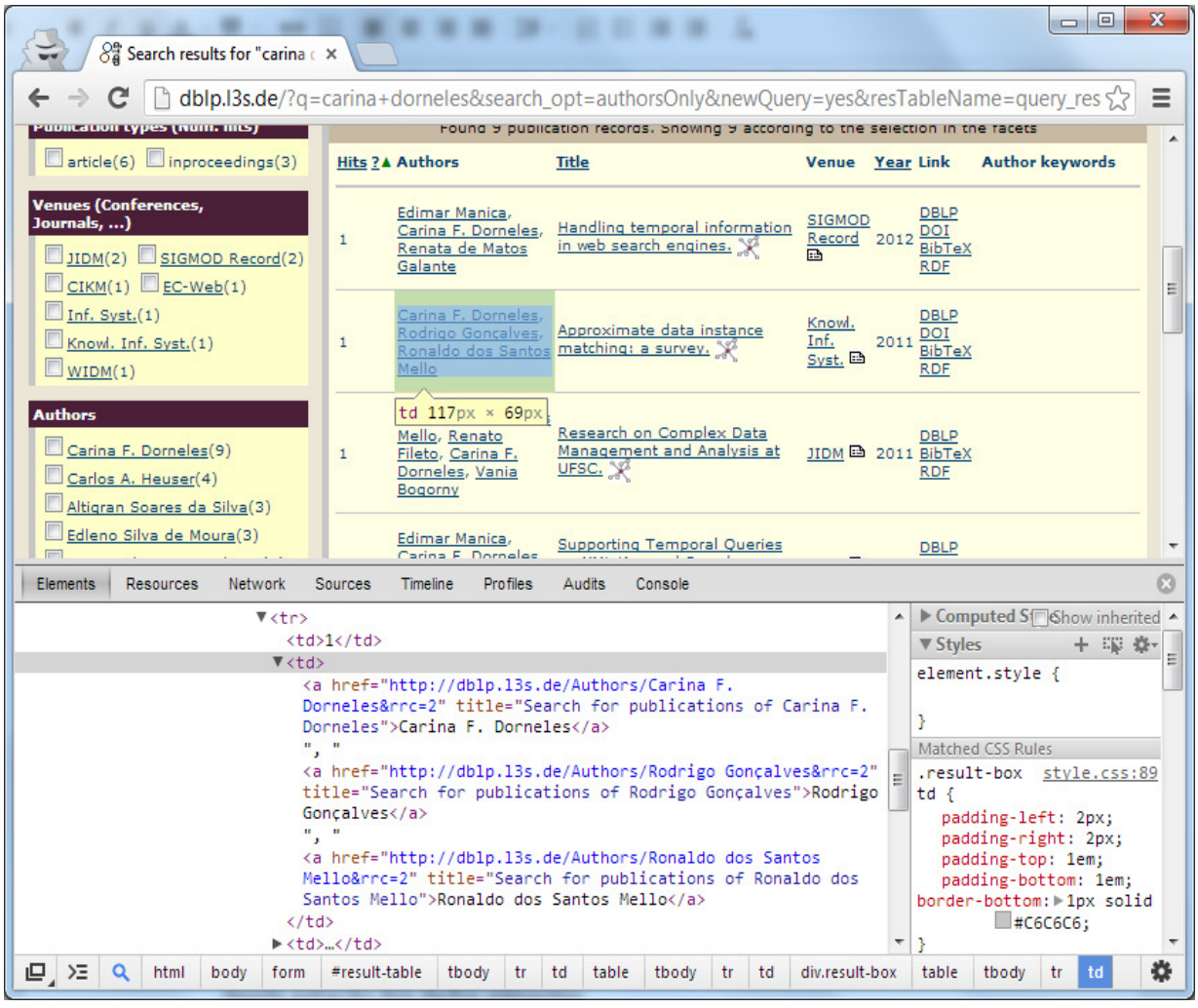

Figura 6 - Análise das tags HTML importantes para a extração das informações - DBLP

Todos os métodos de extração de dados se basearam em uma mesma premissa: a partir da árvore de elementos, percorrer todos os elementos em busca das informações relevantes. Quando achada a *tag* com o conteudo de interesse da ferramenta foram utilizados métodos da especificação DOM, fornecidos pela API externa, para a captura do texto relevante. Os métodos mais utilizados são: *ToString, AsText e GetFirstByXPath*.

Nas seções a seguir será detalhado como foi implementado cada método de extração dos Repositórios da ferramenta, uma vez que cada um dos mecanismos de extração de informações passa por processamentos diferentes tendo em vista que cada Portal possui uma estrutura diferente.

O conteúdo dos métodos de extração de cada repositório será explicado de modo genérico e ilustrativo através de pseudo-códigos. Esta codificação da qual a ilustração genérica ilustrará é encontrada dentro do módulo/classe denominado AnalisadorHTML.

4.2.1 DBLP

Neste repositório, pelo fato das informações serem dispostas em uma tabela, conforme a Figura 6 ilustra, foi necessário varrer esta tabela como uma matriz, analisando se os textos contidos nas células da tabela eram relevantes para a extração.

Conforme dito anteriormente, foi necessário encontrar *tags* que definissem a partir de onde se iniciariam os dados relevantes. Para este repositório analisou-se que a *div* “result-box” dava inicio à marcação HTML da matriz de dados, sendo que a partir desta *tag* se iniciaria a análise e extração dos dados relevantes. Para extração dos dados é implementado um loop que percorre a árvore de *tags* HTML passada pela classe Crawler. A Figura 7, contendo um pseudo-código, ilustra a estrutura de implementação do referido método.

```
#Declaração de método
BEGIN analisaHtmlDBLP( HtmlPage page )
     #Variavel contendo todos os elementos tag da pagina acessada
     DomNode arvoreDeTagsHTML =  page.getDescendants();

     #Percorre todo o array de elementos HTML
     FOR EACH( arvoreDeTagsHTML )

           #Verifica se o tag HTML é do tipo result-box
           IF (tagHTML.contains("<div class=\"result-box\">"))

                 #Aqui certifica-se de que está no array de dados, iniciando a extração
                 ContadorDeColunas++;
                 IF ( ContadorDeColunas == 1 )
                        Autor = tagHTML.asText();
                 IF ( ContadorDeColunas == 2 )
                        Titulo = tagHTML.asText();
                 IF ( ContadorDeColunas == 4 )
                        Ano = tagHTML.asText();

                 #Caso identificada a ultima coluna da matriz de dados
                 IF ( ContadorDeColunas == 6 )
                        ContadorDeColunas = -1;
                        #Junto ao Array de Publicações são adicionados os dados encontrados
                        Publicacoes.ADD( new Publicacao ( Autor, Titulo, Ano ) );
                 END IF
            END IF
     END FOREACH

     #Caso existam dados as serem exibidos, estes são enviados para que a classe Crawler mostre na interface
     IF ( ArrayDePublicacoes.Size() > 0)
           Crawler.InserePublicacoes( ArrayDePublicacoes );
     ELSE
          Crawler.InsereNenhumaPublicacaoEncontrada();

END METHOD
```

Figura 7 - Pseudo-código ilustrando a extração de dados do repositório DBLP.

Pelo fato deste mecanismo de pesquisa do site, selecionado dentre os possíveis no repositório, só permitir a busca por Autores os demais tipos de busca (por título e por instituição) foram descartados.

No caso do repositório DBLP não foi necessária a implementação de uma estrutura de paginação, visto que esta pesquisa suporta a exibição de até 800 registros por pesquisa. Já os demais repositórios necessitaram desta estrutura, que será detalhada em seus respectivos tópicos.

4.2.2 PubMed

Neste repositório as informações são exibidas através de uma listagem com informações a respeito da publicação. A Figura 8 ilustra esta abordagem. Para encontrar as *tags* que continham as informações importantes para esta busca, assim como nos demais repositórios, foi necessário fazer a análise da estrutura HTML do site. Nesta análise constatou-se que cada informação contida no site era encapsulada (ou poderia ser extraida) por *tags* do tipo *div, p* (paragraph) e *a* (anchor). Cada *tag*, por ter aplicação de um CSS de cores e estilo, acaba tendo um identificador único, o qual foi utilizado para busca e extração das informações. Para facilitar o encontro destas *tags* específicas foi utilizado o método *GetFirstByXPath,* da especificação W3C para interfaces DOM, cujo atributo passado foi o identificador ou nome da classe da *tag* em questão.

No portal PubMed, por se tratar de uma listagem e permitir o número máximo de 200 registros, foi necessária a implementação de um mecanismo para realizar a paginação destes registros. Como pode-se observar na Figura 8, ao lado do número de páginas existem os botões de navegação sobre as informações: *Next* e *Previous*. Foi através da ação de *click* sobre estes botões que foi implementada a navegação na página. Utilizando-se do método *click* da API HTMLUnit foi possível baixar e guardar em memória a página HTML associada à ação de *click*, simulando assim a navegação realizada por um *browser*.

A motivação da escolha para implementar a navegação utilizando o método de *click* no botão *next* ou *previous* foi devido ao mecanismo de paginação do próprio site repositório. O site utiliza um mecanismo de processamento de paginação via JavaScript, onde as páginas são montadas dinamicamente no lado do servidor e apenas exibidas no lado cliente (navegador), alterando as informações no mesmo HTML na qual as informações da página anterior estavam. Desta maneira a URL do website mantinha-se inalterada, não exibindo nenhum índice, contador ou informação de avanço/retrocesso da paginação que facilitasse a implementação de uma lógica de avanço da captura do conteúdo.

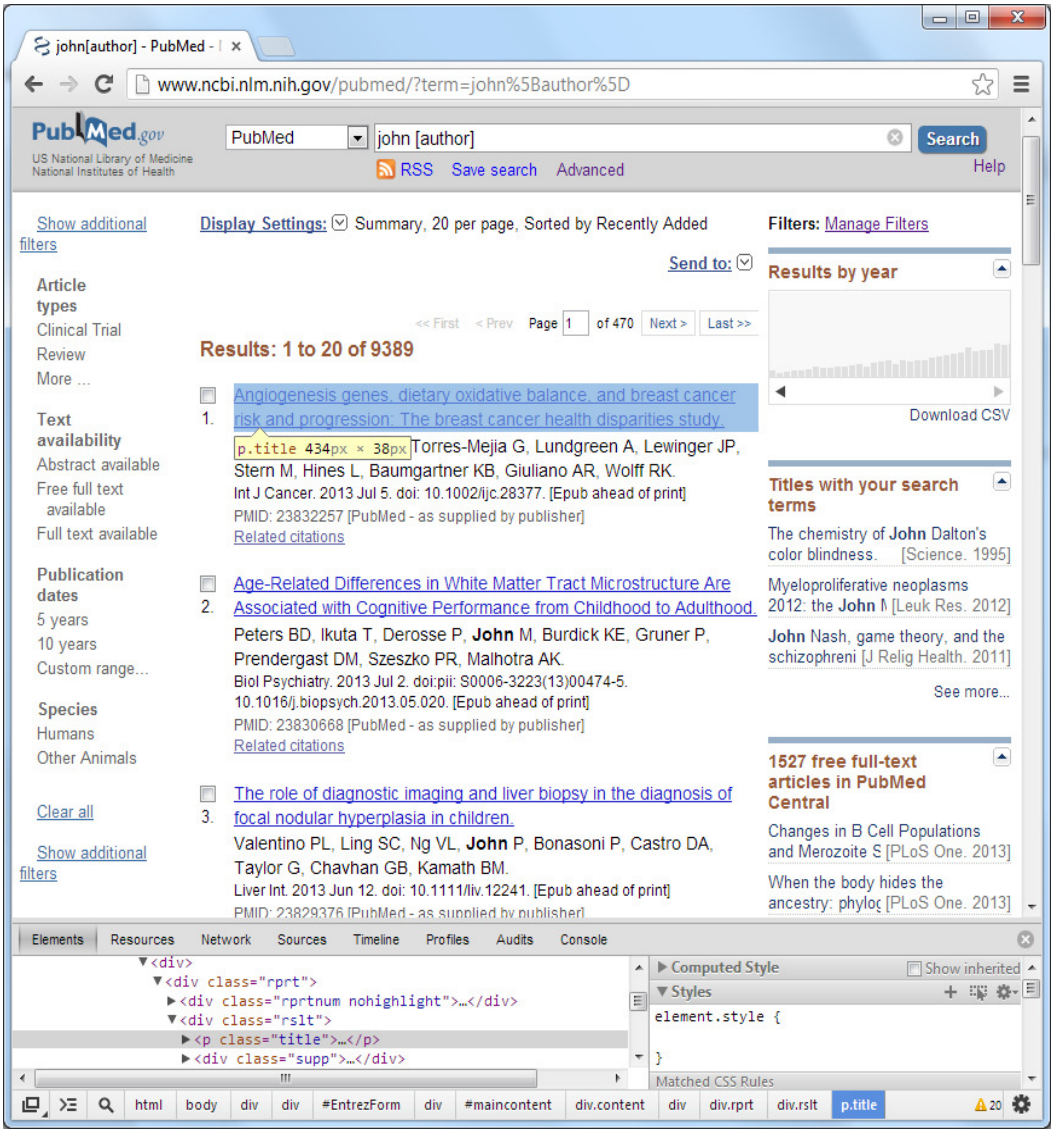

Figura 8 - Exibição das informações e identificação das tags importantes - PubMed

Diferentemente da implementação do primeiro repositório, neste método de extração utilizou-se um recurso de consulta, chamado de XPath. Tal recurso, definido pela W3C, permite que se construam expressões que percorrem e processam um documento XML ou HTML de modo semelhante a uma expressão regular. Este recurso é utilizado através do método *getByXPath* da biblioteca do HTMLUnit (API externa). O retorno do método *getByXPath* é uma lista de objetos DOM contendo o termo buscado. Esta lista é percorrida verificando a existência de dados e assim extraindo a informação necessária.

A Figura 9, através de um pseudo-código, ilustra a lógica implementada para o método de extração das informações do repositório PubMed.

```
#Declaração de método
BEGIN analisaHtmlPubMed( HtmlPage page, int tipoPesquisa, String termoBuscado )
     #Variavel contendo o número de paginas
     int numeroDePaginas = CalculaNumeroPaginas(page);

     #Percorre todo o array de elementos HTML
     int indice = 0;
     WHILE( indice < numerodePaginas)
          Array ListaDeTitulos = page.getByXPath("//p[contains(@class, 'title')]");
          Array ListaDeAutores = page.getByXPath("//p[contains(@class, 'desc')]");
          Array ListaDeAnos = page.getByXPath("//p[contains(@class, 'title')]");

          #Persebeu-se que a tag que continha ano, sempre tinha dados. Logo ela controla o loop
          int indice2 = 0;

          #Looping para a extração dos dados daquela página
          WHILE ( indice2 < ListaDeAnos.Size() )
              String titulo = (String)ListaDeTitulos.getElement(indice2);
              String autor = (String)ListaDeAutores.getElement(indice2);
              String ano = (String)ListaDeAnos.getElement(indice2);

              #A publicação encontrada é adicionada em uma lista de publicações em memória
              ListaDePublicacoes.add( new Publicacao( autor, titulo, ano ));
              indice2++;
          END WHILE

          #Caso o botão de paginação NEXT esteja ativo (seja encontrado) vai para próxima pagina
          IF ( temProximaPagina(page) )
              #Ação de click sobre o botão. Suportada pela API HTMLUnit
              page = (HtmlPage)page.ClicaNEXT();
          END IF
          indice2++;
     END WHILE

     #Caso existam dados as serem exibidos, estes são enviados para que a classe Crawler mostre na interface
     IF ( ArrayDePublicacoes.Size() > 0)
           Crawler.InserePublicacoes( ArrayDePublicacoes );
     ELSE
          Crawler.InsereNenhumaPublicacaoEncontrada();

END METHOD
```

Figura 9 - Pseudo-código ilustrando a extração de dados no repositório PubMed.

4.2.3 Portal Capes

A extração de dados deste repositório foi sem dúvidas a mais complexa de ser implementada devido a uma série de detalhes. Exemplificando-os: a quantidade de *tags* contida na estrutura de seu HTML foi um dos motivos, devido a isso a varredura e extração dos dados deste repositório é realizada de forma mais lenta.

Outro empecilho, que também colabora para o baixo desempenho na extração dos dados é o fato de a página listar somente 10 registros por vez, sem opção de mudança, exigindo muitas requisições HTTP ao servidor. Estas requisições, quando realizadas de forma repetitiva e sequencial ao servidor acabam gerando uma certa carga de processamento, embora nem sempre todas as requisições sejam respondidas. Este fato acaba gerando, muitas vezes, alguns tipos de erro como: *timeout* da requisição, página sem conteúdo ou algum tipo de erro interno do servidor. Desta maneira, ocorre a falta de informações na alimentação do *Crawler*. Tais erros foram tratados da seguinte maneira: a aplicação repete novamente a mesma requisição tentando obter resultado, caso não obtenha, dependendo do erro acima citado ela pula esta requisição (pula a estrutura de índice contida na URL da página) ou simplesmente para a extração e exibe uma informação de erro para o usuário.

Apesar dos problemas encontrados, o desenvolvimento do método de extração deste repositório é realizado da mesma forma que nos demais repositórios, utilizando o mecanismo de busca do próprio site repositório em questão, ou seja, inserindo a *String* buscada pelo usuário diretamente no link HTML da página, apenas com tratamento para acentos.

Além disso, foi necessário implementar um sistema de paginação, já que, como dito anteriormente, o Portal exibe uma listagem de dados com apenas 10 registros por página, sem opção de mudança desta configuração. A exibição da listagem dos dados pode ser observada na Figura 10.

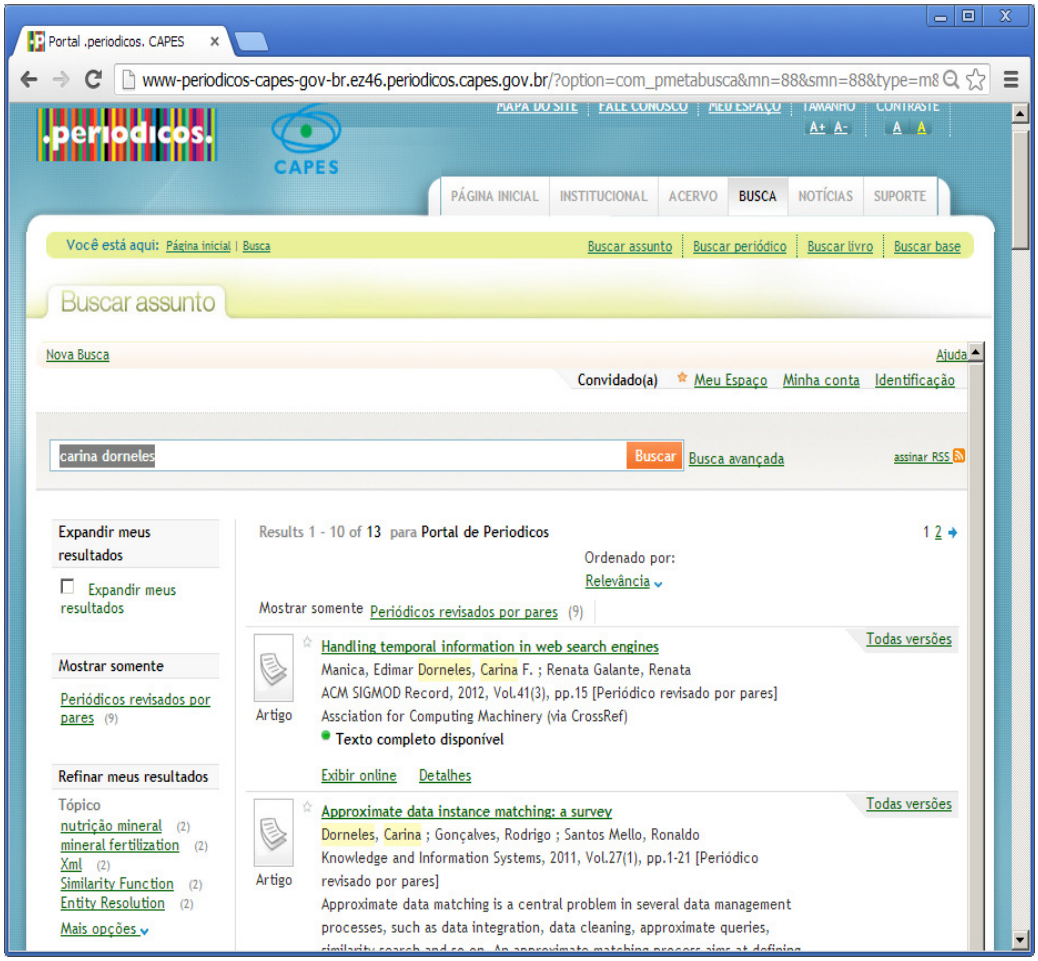

Figura 10 - Exibição das informações no Portal Capes.

O sistema de paginação foi implementado da mesma maneira que a paginação do repositório PubMed, que utiliza a ação de *click* sobre o botão *next*. No caso do Portal Capes, este botão *next* é representado por uma pequena seta ou flecha localizada à direita da paginação por números, que pode ser visualizada na Figura 10. A presença desta seta revela que a página visualizada no momento não é a ultima, indicando assim que existem mais páginas a percorrer.

Antes do desenvolvimento do método de extração deste site repositório, assim como os demais, foi necessário analisar a estrutura HTML para que fossem identificadas *tags* relevantes para a extração das informações.

A implementação deste método foi realizada de forma semelhante ao portal PubMed onde foram utilizadas expressões para encontrar determinadas *tags* no

método *getByXPath* contendo dados relevantes. Este conteúdo relevante foi primeiramente a quantidade de páginas para se realizar um controle na paginação. O próximo passo implementado foi percorrer a árvore de elementos HTML baixada pelo HTMLUnit verificando se foi encontrado o conteúdo (devido à quantidade de *divs* desnecessárias à esquerda do portal).

Achado o conteúdo relevante no site é iniciada a extração, de fato, dos dados importantes. Nesta, a cada iteração do *loop* é verificado se a *tag* em questão contém identificadores ou classes *CSS* já analisados anteriormente. Caso sejam encontradas, as informações contidas dentro destas *tags* são extraídas e armazenadas em variáveis locais, para posteriormente serem inseridas na lista de publicações encontradas e esta, por fim, ser enviada ao banco de dados e Interface.

Uma observação cabe a ser feita aqui: pelo fato de, muitas vezes, os nomes dos professores responsáveis pela publicação estarem abreviados, foi utilizado um método de comparação de *Strings*, nomeado como *verificaAutorCapes*, utilizado para confirmar se o que o usuário de fato está procurando é aquele que a pesquisa do site repositório retorna.

Após a extração das informações é verificada a existência do botão *next*, caso este seja encontrado é realizada a ação de *click* sobre tal botão, acessando próxima página e é reiniciado o processo de busca pelo conteúdo relevante comentado acima. Caso o botão não seja encontrado temos 2 situações: esta é a ultima página da listagem ou ocorreu algum erro, conforme comentado no início da explicação.

Neste momento é verificada também através do método *getByXPath* a existência de determinado elemento ou texto que confirme qual tipo de erro ocorreu. Caso o erro seja identificado, é realizado o tratamento do erro caso seja possível. Após estas verificações, caso não seja encontrado erro ou o botão *next*, a listagem com as publicações encontradas é enviada a classe Crawler para que então sejam armazenadas no banco de dados e exibidas na interface para o usuário.

A Figura 11 ilustra toda a explicação dada acima através de um pseudo-código.

```
#Declaração de método
BEGIN analisaHtmlCAPES( HtmlPage page, int tipoPesquisa, String termoBuscado )
     #Variavel contendo o número de paginas
     HtmlElement numPaginas = (HtmlElement)page.getFirstByXPath("//div[@id = 'resultsHeaderNoId']/div/h1/em");

     #Pega o objeto que faz referência a próxima página
     HtmlAnchor proxPagina = (HtmlAnchor) page.getFirstByXPath("//a[contains(@class, 'EXLNext')]");
     #Percorre todo o array de elementos HTML
     DO
          boolean achouConteudo = false;
          if (tagString.contains("resultsNumbersTile"))
              achouConteudo = true;
          FOREACH (HtmlElement htmlElement : descendentes)
              IF ( achouConteudo)
                  IF (tagString.contains("EXLResultTitle"))
                       Titulo = tagString;
                  IF (tagString.contains("EXLResultAuthor") & validaAutor(tagString))
                       Autor = tagString;
                  IF (tagString.contains("EXLResultDetails"))
                       Ano = encontraAno( tagString );

                  IF ( Titulo & Autor & Ano != null)
                       #A publicação encontrada é adicionada em uma lista de publicações em memória
                       ListaDePublicacoes.add( new Publicacao( autor, titulo, ano ));

            END FOREACH
          #Caso o botão de paginação NEXT esteja ativo (seja encontrado) vai para próxima pagina
          next = (HtmlAnchor) page.getFirstByXPath("//a[contains(@class, 'EXLNext')]");

          IF ( next == null )
               #O erro é tratado
              TrataErro(page);
          ELSE
                 page = next.click();
     WHILE (next != null);

     #Caso existam dados as serem exibidos, estes são enviados para que a classe Crawler mostre na interface
     IF ( ArrayDePublicacoes.Size() > 0)
           Crawler.InserePublicacoes( ArrayDePublicacoes );
     ELSE
          Crawler.InsereNenhumaPublicacaoEncontrada();

END METHOD
```

Figura 11 - Pseudo-código ilustrando a extração de informações no Portal Capes.

Conforme comentado anteriormente, durante as etapas de desenvolvimento do módulo de extração das informações, o portal Capes teve comportamentos anormais quanto à apresentação de sua estrutura. As Figuras 12, 13 e 14 ilustram os erros encontrados durante o desenvolvimento da aplicação.

Durante os testes em pesquisas de termos que retornam muitos registros, como por exemplo universidades de grande porte internacional, ocorreram erros diversos. Geralmente estes erros estavam relacionados a erros de processamento da página no lado do servidor, tanto o hospedeiro *Web* do portal como no servidor que processa as

requisições de página (*click*, pesquisa, paginação). O erro encontrado na Figura 14 em específico foi encontrado durante a paginação do termo “UFSC”, onde dois índices da paginação do termo buscado não são mostrados pelo portal. Em casos como estes a tratativa para continuar a extração dos dados foi tentar avançar o índice da página e ignorar a página não exibida na coleta de informações.

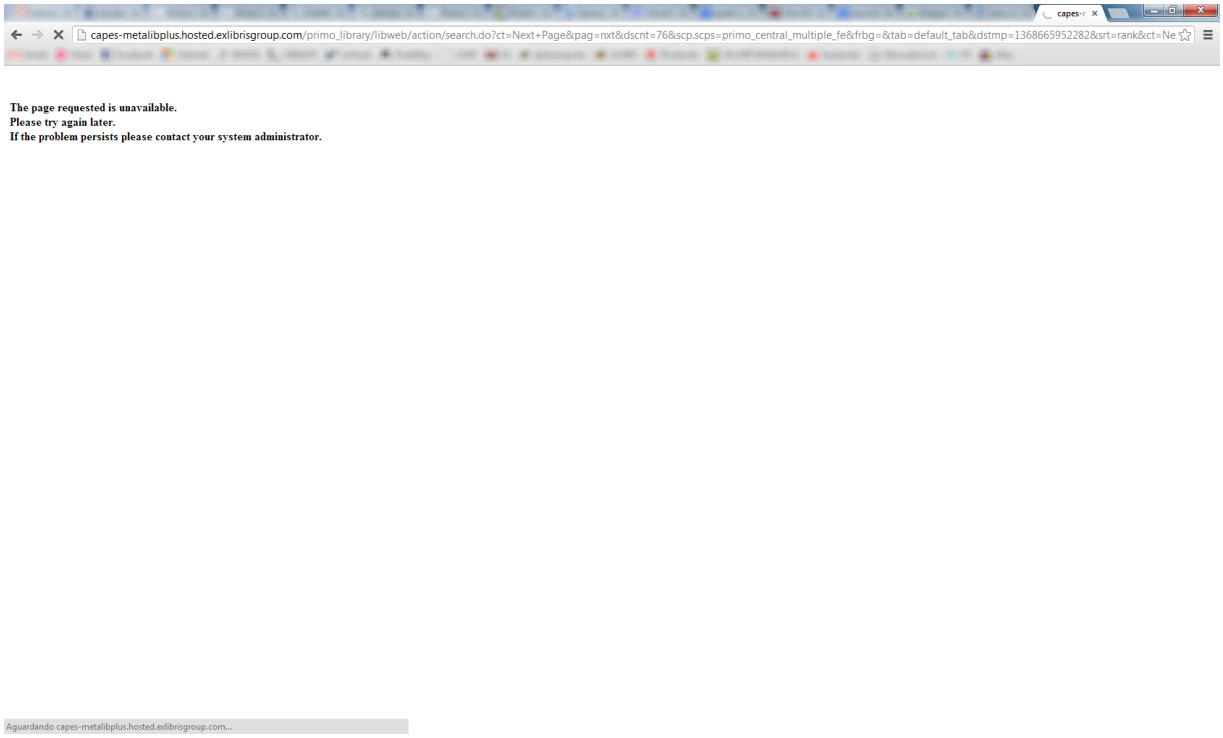

Figura 12 - Portal Capes - Erro de processamento da página.

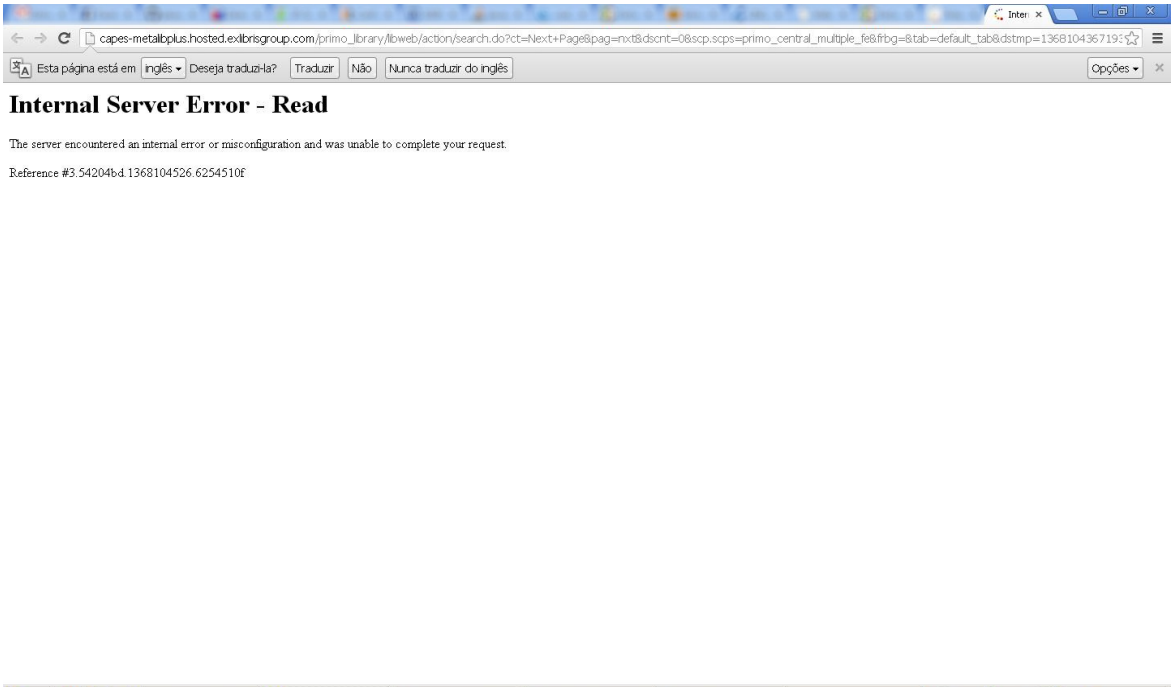

Figura 13 - Portal Capes - Erro de interno do site.

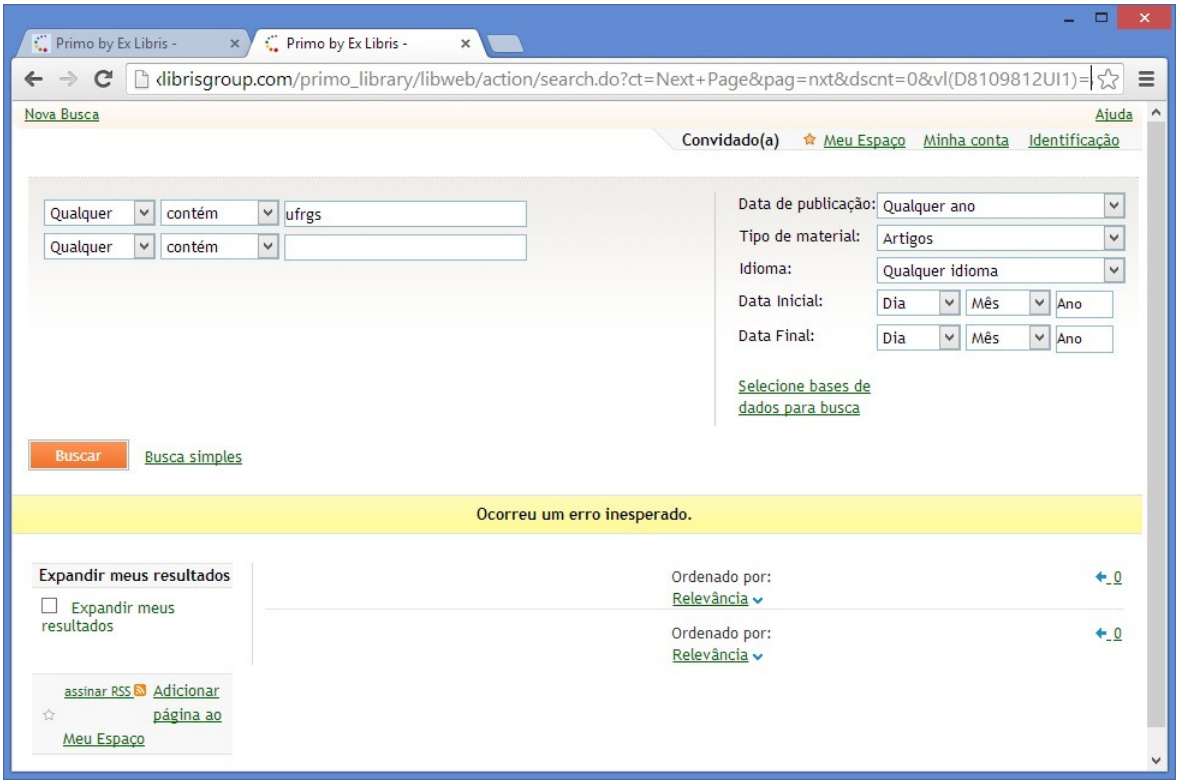

Figura 14 - Portal Capes - Erro inesperado.

5. Experimentos

Neste capítulo são expostos os experimentos realizados e seus respectivos resultados. Para isso, foram realizadas 30 pesquisas em cada opção de site repositório da ferramenta PubFinder (DBLP, PubMed e Capes) e suas devidas opções de busca (autor, título e instituição).

Para medir o desempenho da ferramenta foram utilizadas três métricas de avaliação dos testes: revocação, precisão e Medida-F. Revocação trata da quantidade de publicações encontradas comparadas com a quantidade total de publicações relevantes existentes no site. Já a precisão é a quantidade de publicações encontradas de maneira correta dentre todas as publicações encontradas. Por fim, a Medida-F é uma média harmônica entre as duas medidas obtidas, sendo obtida através da fórmula exibida na imagem 15.

$$\text{Medida-F} = \frac{2 \times \text{Precisão} \times \text{Revocação}}{\text{Precisão} + \text{Revocação}}$$

Figura 15 - Fórmula para o cálculo da Medida-F.

Lembrando que no número total de publicações serão consideradas apenas aquelas publicações que retornam ao serem buscadas diretamente no site repositório em um navegador com a possível análise se aquele dado é de fato verdadeiro ou não. Caso o site não exiba determinadas publicações estas não são consideradas.

Conforme a Tabela 1 exibe, foram feitas 30 buscas distintas para cada tipo de Pesquisa que a ferramenta contempla, e delas foram feitas as análises a seguir.

Repositório	Autor	Título	Instituição	Nº de Buscas
DBLP	X	-	-	30
PubMed	X	X	X	90
Capes	X	X	X	90
			Total:	**210**

Tabela 2 - Configurações de teste para avaliação de desempenho da Ferramenta.

O tipo de gráfico escolhido para exibição dos resultados foi o gráfico em formato de Pizza. O motivo desta escolha foi a facilidade da montagem destes gráficos e da facilidade da exibição dos resultados por faixas de amostras. Nestes gráficos, os resultados de desempenho estão divididos em faixas de acerto dentro das métricas escolhidas (0%, entre 1 e 24%, entre 25 a 49%, entre 50 e 74%, entre 75 e 99% e 100%). Nos gráficos contendo a precisão e revocação da ferramenta são exibidos os números de quantas buscas estão dentro daquela faixa de acerto. Já os gráficos com a Medida-F exibem a porcentagem de buscas que se encontram dentro da faixa de acerto de acordo com seu respectivo cálculo.

Conforme a Figura 16 ilustra, após a realização dos 30 testes de buscas por autores no repositório DBLP foi obtido 100% nas duas métricas de avaliação, mostrando que a ferramenta é eficaz na busca por nome de professores deste portal, uma vez que ela aproveita-se do fato que o próprio portal processa o nome inserido na busca e exibe as informações dos nomes contidos na base de dados.

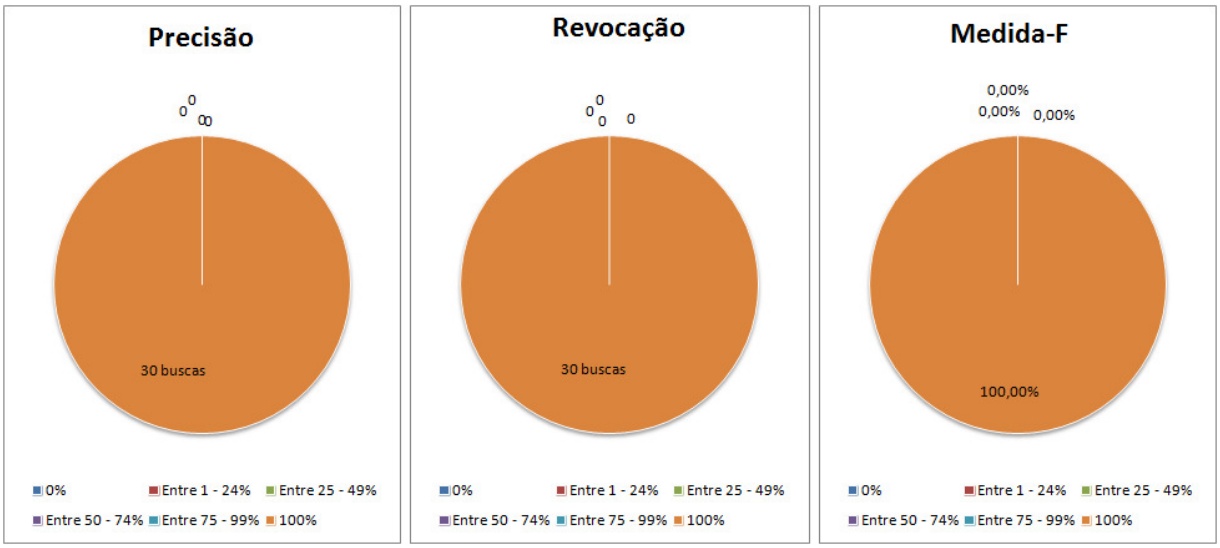

Figura 16 - Gráficos com os resultados dos testes de Autores no repositório DBLP.

Assim como o portal DBLP, o portal PubMed também busca por um nome inserido e retorna somente os dados referentes aquele nome constante no seu banco de dados. O portal PubMed obteve resultados semelhantes ao portal DBLP, obtendo

resultado de 100% de precisão e revocação nas buscas inseridas. Isto pode ser observado nas Figuras 17, 18 e 19.

Durantes os testes do portal PubMed foi constatado que a ferramenta consome muito dos recursos de máquina: processador e memória. Quando a máquina processa muitos registros, a ferramenta às vezes para a busca e exibe erro dos próprios componentes Java. Analisando o Gerenciador de Dispositivos do Windows foi constatado o uso de 100% do processador e de quase 100% da memória disponível no computador. Este comportamento pode ser creditado ao mau gerenciamento de memória da linguagem Java nas repetitivas iterações dos *loops* contidos nos métodos de extração das informações, na criação e uso de objetos da API externa e nos *downloads* repetitivos de páginas HTML.

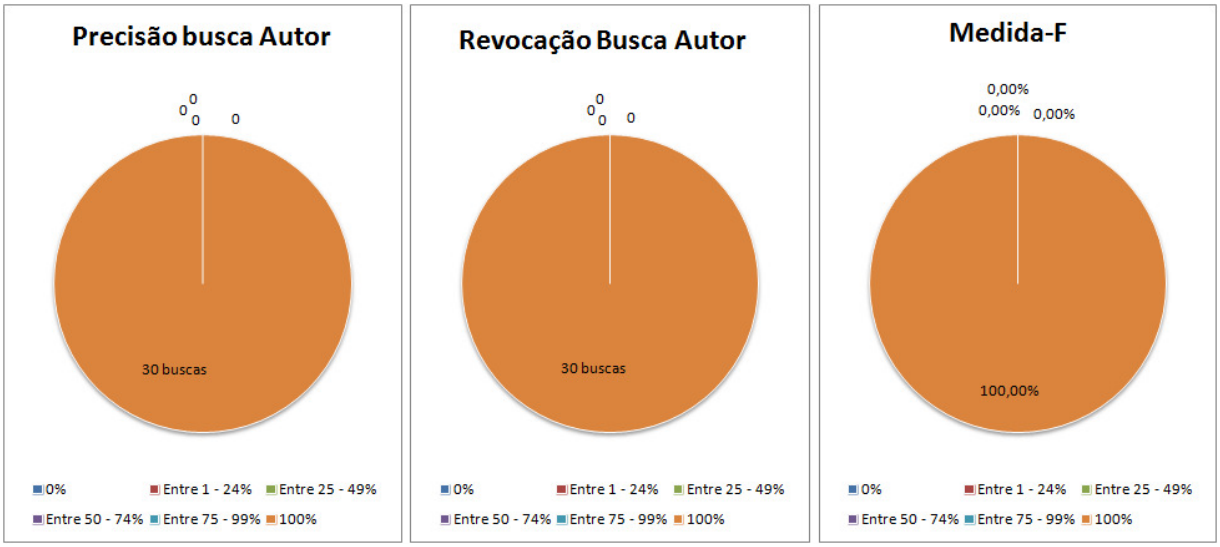

Figura 17 - Gráficos com os resultados dos testes de Autores no portal PubMed.

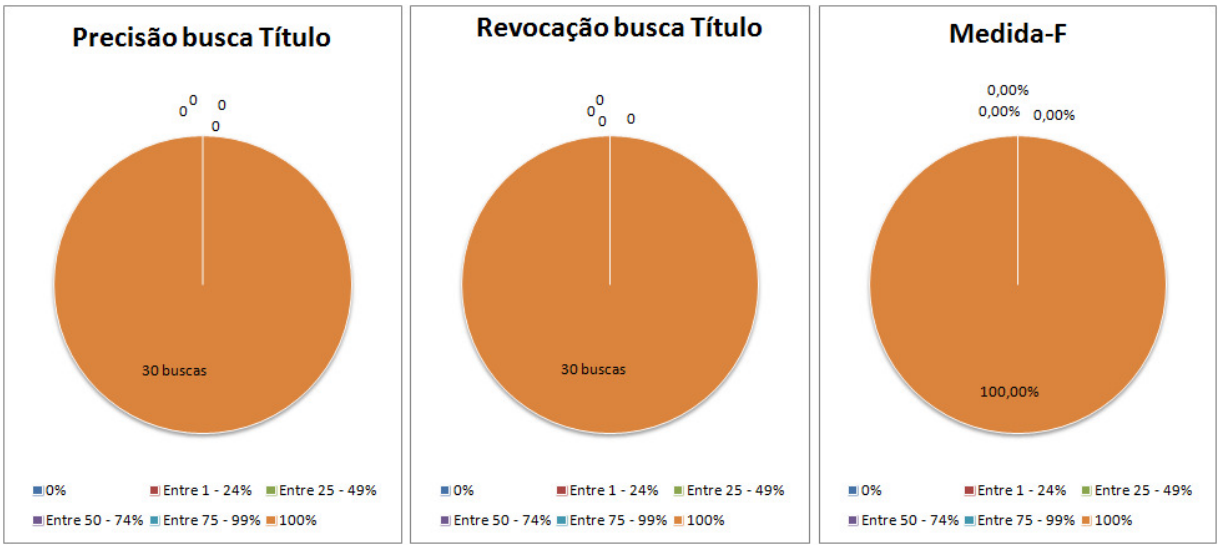

Figura 18 - Gráficos com os resultados dos testes de Títulos no portal PubMed.

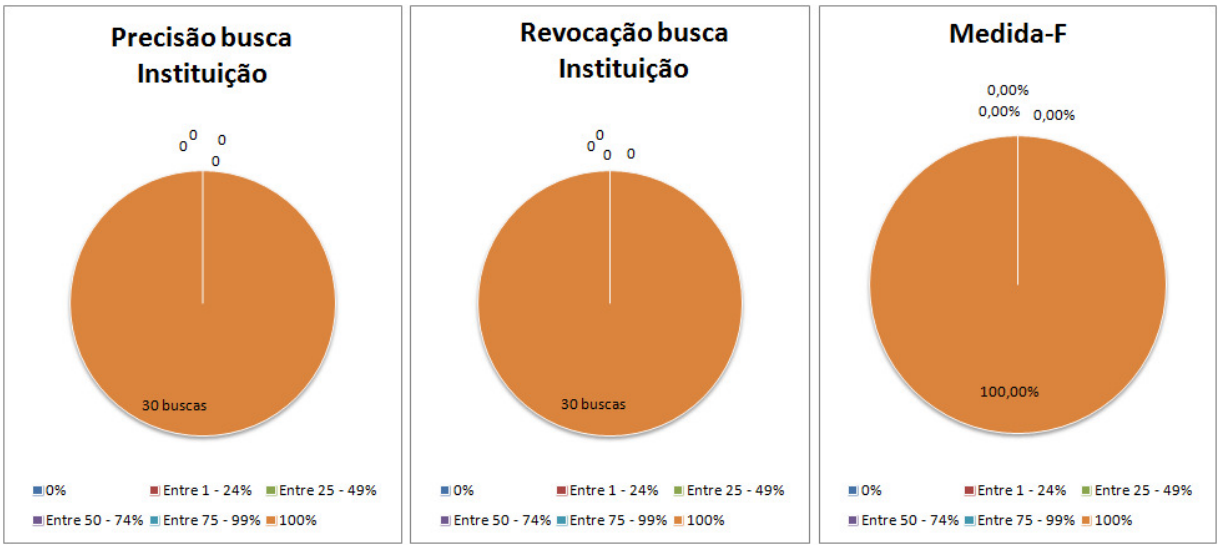

Figura 19 - Gráficos com os resultados dos testes de Instituições no portal PubMed.

Conforme os resultados das Figuras 20, 21 e 22 mostram, o portal Capes, comparado aos demais portais foi o que teve o pior desempenho, porém se manteve satisfatório com a extração das informações necessárias já que a média de seus resultados manteve-se sempre acima de 75% de precisão e revocação. O pior dos resultados se mostrou na pesquisa referente aos títulos, já que muitas vezes ao

comparar os termos contidos na página ou durante o processo de extração a publicação não continha algum dos elementos essenciais (nome, ano) ou até mesmo páginas que não carregam o conteúdo necessário, das quais conforme o tratamento de exceções realiza, pulam esta página, indo para a página seguinte, fazendo com que a busca não seja interrompida.

O portal Capes se mostrou muito ineficiente na questão de padronização da nomenclatura dos usuários, peça chave da extração das informações. Por ele ser um portal integrador, que exibe ou indica publicações de todo o mundo, não foi estabelecido nenhum padrão de nomenclatura, sendo exibidos assim os nomes dos autores do jeito que foram publicados. Muitas vezes a página fugia da regra de separação dos nomes dos autores padrão do site, que é feito por ponto e vírgula entre cada nome de autor.

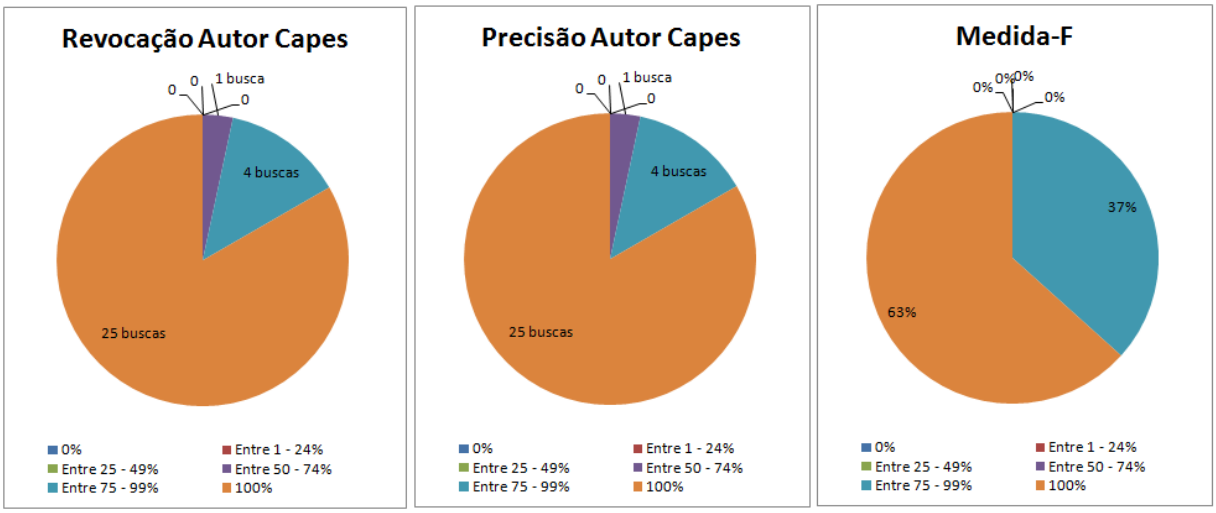

Figura 20 - Gráficos com os resultados dos testes de Autores no portal Capes.

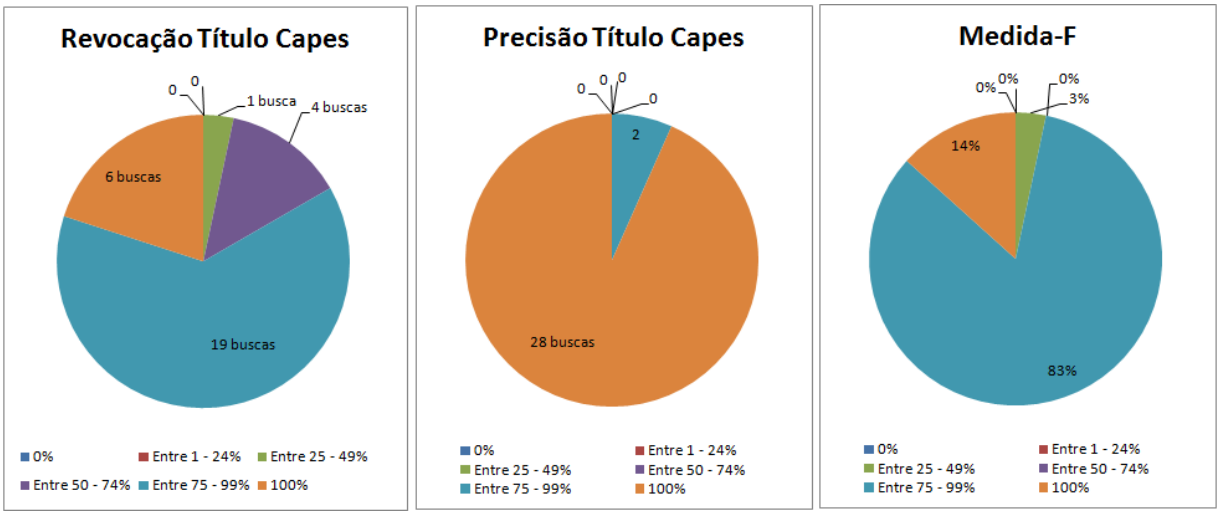

Figura 21 - Gráficos com os resultados dos testes de Títulos no portal Capes.

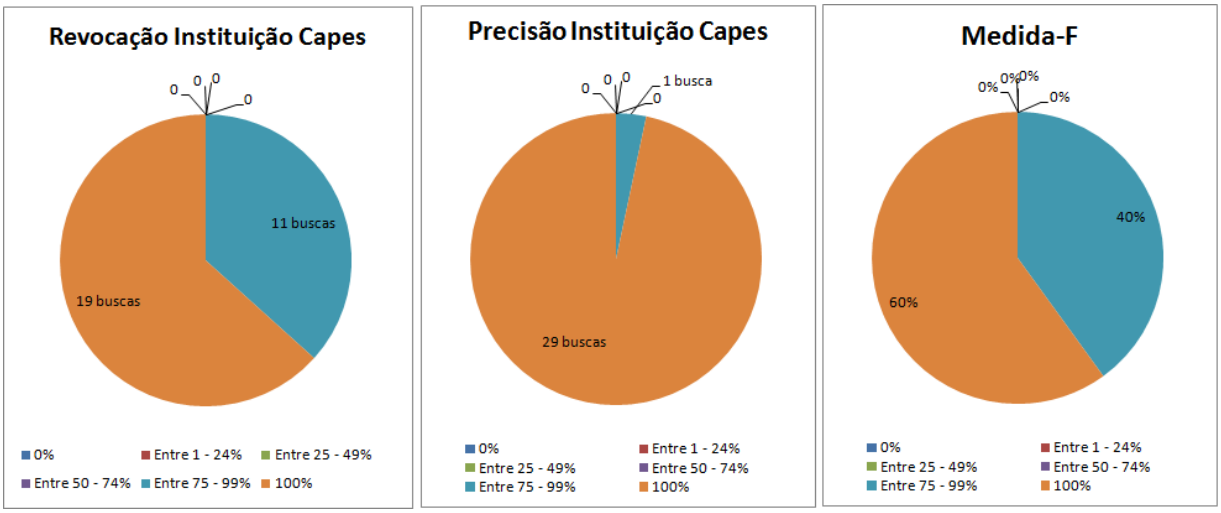

Figura 22 - Gráficos com os resultados dos testes de Instituição no portal Capes.

Como o portal Capes não teve uma medida de revocação máxima, ou perto dela, abaixo seguem exemplos de pesquisas realizadas em cada tipo de busca, exibindo o resultado que a ferramenta trouxe e o resultado esperado.

Pesquisa por Autor: Ao tentar buscar o termo "Carina Dorneles", filtrando somente por autores no portal Capes, temos o cenário exibido na Tabela 3. Nela é possível visualizar os nomes que o site traz para visualização do usuário. A ferramenta identificou 10 publicações com o termo buscado, porém se analisarmos a tabela é possível notar que existem 4 publicações com o nome Carina Seixas M. Dorneles. Os nomes iniciais e finais (nome e sobrenome final) do autor são iguais, fazendo com que o *crawler* identifique-os como informações válidas. Por este motivo, esta busca resultou em uma revocação de 67%, precisão de 60% e F-Measure de 88%.

	RESULTADO DA PESQUISA POR AUTOR NO PORTAL CAPES		
	Autor(es)	**Título**	**Ano**
1	Manica, Edimar **Dorneles, Carina F.** ; Renata Galante, Renata	Handling temporal information in web search engines	2012
2	**Dorneles, Carina** ; Gonçalves, Rodrigo ; Santos Mello, Ronaldo	Approximate data instance matching: a survey	2011
3	Bocassanta, Fabio; **Dorneles, Carina F.**	Similarity Join of XML Documents Stored in File Systems	2010
4	Cristiano Roberto Cervi ; Edimar Manica ; **Carina Friedrich Dorneles** ; Renata Galante	BDTC - Uma Biblioteca Digital de Trabalhos Científicos com Serviços Integrados	2009
5	**Dorneles, Carina F.** ; Nunes, Marcos Freitas ; Heuser, Carlos A. ; Moreira, Viviane P. ; da Silva, Altigran S. ; de Moura, Edleno S.	A strategy for allowing meaningful and comparable scores in approximate matching	2009
6	Moro, Mirella, M. ; Braganholo, Vanessa ; **Dorneles, Carina, F.** ; Duarte, Denio ; Galante, Renata ; Mello, Ronaldo, S.	XML: some papers in a haystack	2009
7	Carlos Henrique De Brito ; Ademar P De Oliveira ; Adriana Ursulino Alves ; **Carina S M Dorneles** ; João F Dos Santos ; José P R Nóbrega	Produtividade da batata-doce em função de doses de K2O em solo arenoso Sweet potato yield as a function of K2O levels in a sandy soil	2006
8	Ademar Pereira De Oliveira ; Jandiê Araújo Silva ; Adriana Ursulina Alves ; **CarinaSeixas M Dorneles** ; Anarlete Ursulino Alves ; Arnaldo Nonato P. De Oliveira ; Edson A. Cardoso ; Iordam Da Silva Cruz	Rendimento de feijão-vagem em função de doses de K2O Snap bean yield in function of K2O levels	2007
9	Oliveira, Ademar Pereira de ; Silva, Jandiê Araújo ; Alves, Adriana Ursulina ;**Dorneles, Carina Seixas M** ; Alves, Anarlete Ursulino ; Oliveira, Arnaldo Nonato P. de ; Cardoso, Edson A ; Cruz, Iordam da Silva	Rendimento de feijão-vagem em função de doses de K2O Snap bean yield in function of K2O levels	2007
10	Brito, Carlos Henrique de ; Oliveira, Ademar P de ; Alves, Adriana Ursulino ;**Dorneles, Carina S M** ; Santos, João F dos ; Nóbrega, José P R	Produtividade da batata-doce em função de doses de K2O em solo arenoso Sweet potato yield as a function of K2O levels in a sandy soil	2006
11	Körbes, Ana Paula ; Machado, Ronei **Dorneles** ; Guzman, Frank ; Almerão, Mauricio Pereira ; de Oliveira, Luiz Felipe Valter ; Loss-Morais, Guilherme ; Turchetto-Zolet, Andreia **Carina** ; Cagliari, Alexandro ; dos Santos Maraschin, Felipe ; Margis-Pinheiro, Marcia ; Margis, Rogerio Schubert, Michael (Editor)	Identifying Conserved and Novel MicroRNAs in Developing Seeds of Brassica napus Using Deep Sequencing (miRNAs in Developing Seeds of Brassica napus)	2012

Tabela 3 - Resultados de uma pesquisa por autor no Portal Capes

Pesquisa por Título: Ao tentar buscar o termo "AIDS cocktail", filtrando somente por títulos no portal Capes temos o cenário exibido na tabela 4. Nela é possível observar que temos 19 registros encontrados, tendo a ferramenta extraindo apenas 14 registros. Isto fez com que a revocação ficasse com 100% porém a precisão em 77,78% e a F-Measure em 87,5%. Para comparação de palavras contidas nos títulos, foi utilizada a função de comparação de *Strings Contains* nativa do do Java. Observou-se que este método nativo do Java é falho quando as palavras "AIDS" e "cocktail" estão contidas por aspas e distantes uma da outra, ocasionando a não identificação do termo na ferramenta.

	RESULTADO DA PESQUISA POR TÍTULO NO PORTAL CAPES		
	Autor(es)	**Título**	**Ano**
1	Wolfson, Jay	The dangerous financial politics of **AIDS** "**cocktail**" therapies	1997
2	Nichols, Mark ; Wood, Chris ; Goulding, Warren	Beating **AIDS**: a **'cocktail'** treatment of several drugs offers hope.(includes related articles on the conference in Vancouver, British Columbia, Canadian research on **AIDS**, and native Canadians with **AIDS**)(Cover Story)	1996
3	Kuhr, Fred	**Cocktail** all in one: ten years after the introduction of the **AIDS** cocktail, many lives have been saved, but some problems still persist. Will a new one-a-day**AIDS** pill help?(HEALTH)	2006
4	Henkel, John	Attacking **AIDS** With a `**Cocktail**' Therapy.(includes **AIDS** glossary)	1999
5	Marrone, Nicole ; Mason, Christine R. ; Kidd, Gerald, Jr.	Evaluating the benefit of hearing **aids** in solving the **cocktail** party problem.(Special Issue: Auditory Scene Analysis)(Clinical report)	2008
6	Henkel, J	Attacking **AIDS** with a '**cocktail**' therapy?	1999
7	Haddad, Amy	Ethics in action. (ethical considerations in the use of triple drug **cocktail AIDS** treatment)	1998
8	Sadownick, Douglas	Would you like steroids with that **cocktail**? (steroid **AIDS** therapy)(Brief Article)	1998
9	Conlan, Michael F.	Costly **cocktail**: **AIDS** Rx coverage and rising tab worrying HCFA and states. (prescription drug costs; Health Care Financing Administration)	1996
10	Johnsrude, Ingrid S ; Mackey, Allison ; Hakyemez, Hélène ; Alexander, Elizabeth ; Trang, Heather P ; Carlyon, Robert P	Swinging at a **cocktail** party: voice familiarity **AIDS** speech perception in the presence of a competing voice	2013
11	Nehete, Pramod N ; Chitta, Sriram ; Hossain, Mohammad M ; Hill, Lori ; Bernacky, Bruce J ; Baze, Wallace ; Arlinghaus, Ralph B ; Sastry, K.Jagannadha	Protection against chronic infection and **AIDS** by an HIV envelope peptide-**cocktail** vaccine in a pathogenic SHIV-rhesus model	2001

12	Cowley, Geoffrey	**AIDS** drugs fail a test: new studies show they don't eliminate all HIV.('**cocktail**' drug combinations found to leave inactive but potent HIV cells behind)(Brief Article)	1997
13	Seppa, Nathan	HIV not eradicated by drug **cocktail**. (researchers found that even people who received the anti-**AIDS** drugs early, still had evidence of the virus in their bodies)(Brief Article)	1997
14	Sem autor definido	Drug "**cocktail**" benefits women with advanced HIV/**AIDS**	2004
15	Shechtman, Diana L. ; Espejo, Alexandra M.	The influence of HAART on CMV retinitis: this "triple-**cocktail**" therapy for patients with HIV/**AIDS** has had a profound effect on treatment for CMV retinitis.(**AIDS**)(highly active anti-retroviral therapy)(Cytomegallo virus)	2006
16	Sem autor definido	ACTG 315 drug **cocktail** restores immune function	1997
17	Sem autor definido	Industry to sponsor auction, **cocktail** party to fight **AIDS**. (clothing, perfume and cosmetics industries)	1986
18	Lagranderie, M ; Winter, N ; Balazuc, A M ; Gicquel, B ; Gheorghiu, M	A **cocktail** of Mycobacterium bovis BCG recombinants expressing the SIV Nef, Env, and Gag antigens induces antibody and cytotoxic responses in mice vaccinated by different mucosal routes	1998
19	Sem autor definido	Sam Crawford Architects and FOCHTA (Friends of Claude Ho in Thyolo Association) held a **cocktail** party to raise awareness concerning orphaned,**AIDS**-affected children in Malawi.(NEW SOUTH WALES)	2006

Tabela 4 - Resultados de uma pesquisa por um título no portal Capes

Devido à enorme quantidade de registros encontrados, esta pesquisa não terá tabela comparativa.

Pesquisa por Instituição: Ao tentar buscar o termo "UNICAMP" na data de 07/10/2013, sem se utilizar de filtros, já que não existe um filtro específico para instituições no Portal Capes, temos o seguinte cenário.

Foram constatados a existência de 11525 registros exibidos pelo portal *Web* no dia em que foi realizada a pesquisa. A ferramenta por sua vez trouxe 11445 registros. Isto deve-se ao acontecimento dos erros relatados no capítulo de desenvolvimento da ferramenta, como as páginas inexistentes ou que não respondiam às requisições HTTP. Isto fez com que a revocação desta busca ficasse com 99,30%, a precisão 100% e a F-Measure 99,65%.

Vale destacar que os exemplos utilizados na pesquisa por Autor e por Título não foram os de pior rendimento, mas sim os de menor expressão de registros, para que pudessem ser exibidos em tabelas comparativas e argumentados os motivos para seu desempenho na precisão das buscas. Algumas pesquisas de determinados termos foram totalmente fora do padrão de resultados, tendo níveis de registros extraídos muito baixos e por isso acarretaram em rendimentos piores do que os demais Portais, porém mantiveram-se em níveis aceitáveis de resultado para uma ferramenta do tipo *Crawler*.

6. Conclusão e Trabalhos Futuros

Através da proposta de implementação de um *Crawler* focado, na extração de informações referentes a publicações de artigos científicos de professores, foi desenvolvida a ferramenta PubFinder. Esta aplicação foi desenvolvida na linguagem Java com o objetivo de utilizar a biblioteca HTMLUnit para facilitar algumas atividades essenciais no funcionamento da ferramenta, como por exemplo o download de páginas ou a iteração/navegação entre as *tags* HTML deste site.

Ao final da implementação, a ferramenta atingiu seus objetivos, fazendo com que o usuário pudesse interagir com a ferramenta e obter resultados através de sua interface intuitiva. Por ela é possível realizar a busca selecionando o portal que deseja pesquisar, sendo eles DBLP, PubMed e Capes, o modo de pesquisa (por nome de autor, título de publicação ou instituição) e inserir o termo desejado. Após isto a ferramenta irá extrair informações referentes aos portais até então contemplados pela solução.

Através dos resultados obtidos nos testes foi possível observar que a ferramenta atende ao objetivo que lhe é solicitado nos portais DBLP e PubMed, com níveis consideráveis de aceitação, atingindo um índice muito alto de eficiência na extração. Já no portal Capes, conforme comentado no tópico anterior, devido a diversos pontos de falha do próprio portal, a implementação se tornou complicada e muitas vezes não conseguiu atingir um grau alto de satisfação. Um exemplo forte disto foram as páginas inválidas que apareceram durante a paginação da pesquisa, porém a ferramenta atinge os objetivos desejados.

Através do uso da ferramenta foi possível montar um amplo banco de conhecimento com os resultados das pesquisar realizadas nos testes. Com estas pesquisas torna-se mais fácil a reunião de informações referente a determinados assuntos ou de determinados professores que sejam pertinentes ao usuário da ferramenta. Referindo-se à utilidade prática da aplicação, através do uso da ferramenta é possível comparar manualmente, por exemplo, quanto um professor de determinada instituição atualiza de fato seu currículo Lattes com suas publicações ou de seus

orientandos. O Currículo Lattes, concebido e mantido pelo CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), se tornou um padrão nacional no registro da vida pregressa e atual dos estudantes e pesquisadores do país, e é hoje adotado pela maioria das instituições de fomento, universidades e institutos do País. [4]

Ao longo do desenvolvimento da ferramenta foram observados alguns pontos, que não foram desenvolvidas por diversos fatores. Desta forma existem alguns casos que a ferramenta não abrange e outros que poderiam ser explorados a fim de melhorar sua usabilidade.

Abaixo serão listados alguns pontos que poderiam ser implementados ou modificados na ferramenta para otimizá-la:

- Estudo para possibilidade de troca do motor de busca desenvolvido para o uso do Lucene como *software* de busca e API única de indexação dos resultados.
- Criação de um módulo de acesso e pesquisa aos dados já pesquisados (salvos em banco de dados).
- Refinar a pesquisa por nomes do portal Capes, fazendo comparações por similaridade no nome completo do autor. Seja o nome encontrado na pesquisa do portal escrito por extenso ou abreviado.
- Melhoria no tratamento dos erros encontrados no portal Capes, fazendo com que a ferramenta não pare bruscamente a busca.
- Melhoria na velocidade de extração dos dados, visando um desempenho mais eficiente da ferramenta.
- Incorporação de novos portais para busca, sendo um deles o CiteSeerX.

[4] http://www.cnpq.br/web/portal-lattes/sobre-a-plataforma

7. Referências Bibliográficas

[BATSAKIS] BATSAKIS, S., PETRAKIS, E. G. M., MILIOS E. Improving the Performance of Focused Web Crawlers. Journal Data & Knowledge Engineering archive Volume 68 Issue 10. Elsevier Science Publishers B. V. Amsterdam, The Netherlands, 2009.

[DOS SANTOS] dos Santos, L.B., Dorneles, C.F. e Mello, R.S. An Approach for Extracting Web Form Labels Based on Distance Analysis Of HTML Components. IADIS WWW/Internet Conference, Madrid, 2012.

[CASTILHO] CASTILHO, C. Effective Web Crawling. Tese de Doutorado. Departamento de Ciência da Computação - Universidade do Chile. Chile, 2004.

[CORRÊA] CORRÊA, Débora A. UMA ABORDAGEM PARA EXTRAÇÃO DE CONTEÚDOS BASEADA EM CARACTERÍSTICAS ESTRUTURAIS E NAVEGACIONAIS DE PORTAIS WEB. 2012. 109 f. Dissertação (Pós Graduação) - Departamento de Militar De Engenharia, Exército Brasileiro, Rio de Janeiro, 2012. http://www2.comp.ime.eb.br/dissertacoes/2012-Debora_Alvernaz.pdf

[CUNHA, NABOR & SAMPAIO] Cunha, M., Nabor M., Sampaio A. Cloud Crawler: Um Ambiente Programavel para Avaliar o Desempenho de Aplicacões em Nuvens de Infraestrutura.
31º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – SBRC 2013. http://sbrc2013.unb.br/files/anais/trilha-principal/artigos/artigo-52.pdf

[DHENAKARAN] DHENAKARAN, S.S., SAMBANTHAN, K. Thirugnana. Web Crawler - An Overview. (Vol.2 No.1 2011). Pages: 265-267.

[ELMASRI & NAVATHE] ELMASRI, Ramez; NAVATHE, Shamkant B. Sistemas de banco de dados. 4. ed. São Paulo.

[EHRIG & MAEDCHE] EHRIG, M. E MAEDCHE, A. Ontology-Focused Crawling of Web Documents. Proc. of the Symposium on Applied Computing (SAC 2003). March 9-12, 2003. 99

[WU ET AL] Jian, Wu, Teregowda , Pradeep, Ramírez, Juan Pablo Fernández, Zheng, Prasenjit Mitra, Shuyi, Giles, C. Lee. In proceedings of the 3rd Annual ACM Web Science Conference Pages 340-343, Evanston, IL, USA, June 2012.

[LIU] LIU, B. Web Data Mining: Exploring Hyperlinks, Contents and Usage. (Data-Centric Systems and Applications). Chicago, 2001, p. 311.

[MADHAVAN] MADHAVAN, Jayant; AFANASIEV, Loredana; ANTOVA, Lyublena; HALEVY, Alon. Harnessing the Deep Web: Present and Future. Biennial Conference on Innovative Data Systems Research - CIDR, 2009. 6 p.

[MENCZER] MENCZER, F., PANT, G. E SRINIVASAN, P. Topical Web Crawlers: Evaluating Adaptive Algorithms. ACM Transactions on Internet Technology (TOIT). 4(4):378–419, Nov. 2004.

[MOENS] MOENS, M.F. Information Extraction: Algorithms and Prospects in a Retrieval Context (The Information Retrieval Series 21). New York: Springer, 2006.

[PANT & SRINIVASAN] PANT, G. E SRINIVASAN, P. Learning to Crawl: Comparing Classification Schemes. ACM Transactions on Information Systems (TOIS). 23(4), 430-462. 2005.

Anexo A

FERRAMENTA PARA EXTRAÇÃO E ANÁLISE DE PUBLICAÇÕES EM SITES DE ARTIGOS ACADÊMICOS

Diorges Filipe Lohn
Universidade Federal de Santa Catarina (UFSC)
filipelohnn@gmail.com

Abstract. There is a lot of work on the part of the companies offering the service of searching the internet to make the capture of all available addresses on the web and the most varied subjects as possible. Along the growth of the content contained on the web, increase the efforts of the research community to study and delve into new concepts and trends generated from these subjects. These studies are generated, often scientific articles, always with the guidance of an expert teacher or enthusiast on the subject. These academic papers, if they achieve the appropriate levels, are generally published symposia on specific publications of the area to which the content is related websites. This artivo aims to create a Web Crawler focused with the task of looking at three sites symposia and publications of scientific articles about teachers or researchers and their published works site. Based on the results obtained may have an idea for example, how current is the Lattes curriculum of the person sought.

Resumo. Existe um imenso trabalho por parte das empresas que oferecem o serviço de busca na internet para tornar possível a captura de todos os endereços disponíveis na *Web* e os mais variados assuntos. Junto ao crescimento do conteúdo contido na *Web*, aumentam os esforços da comunidade de pesquisa de se estudar e se aprofundar nos novos conceitos e tendências gerados a partir destes assuntos. Nestes estudos são gerados, muitas vezes artigos científicos, sempre com a orientação de algum professor especialista ou entusiasta no assunto. Estes documentos acadêmicos, caso atinjam os níveis cabíveis, são geralmente publicados em sites de simpósios de publicações específicas da área a qual o conteúdo está relacionado. Este artigo visa a criação de um *Web Crawler* focado com a função de buscar junto a três sites de simpósios ou site de publicações de artigos científicos informações sobre professores ou pesquisadores e seus respectivos trabalhos publicados. Com base no resultado obtido pode-se ter ideia por exemplo, o quão atualizado está o currículo *Lattes* da pessoa buscada.

Keywords: Web Crawler, Focused Web Crawler; Academic Articles; Symposium; Data extract.

1. INTRODUÇÃO

Embora a Internet tenha um número gigantesco de páginas a tarefa de buscar informações nela é, ou pelo menos parece ser, trivial ao usuário, já que esta é realizada em frações de segundos por sites buscadores. Porém tornar esta busca viável exige, e muito, dos motores de busca. Um motor de busca pode ser conceituado como um software projetado para encontrar informações a partir de palavras-chave e estes trabalham armazenando informações sobre muitas páginas da *Web*, que eles recuperam a partir do próprio HTML. Estas páginas são recuperadas por um *Web Crawler* (também conhecido como *Spider*) - um *WebBrowser* automatizado que segue cada link no site. O conteúdo de cada página é então analisado para determinar como ele deve ser indexado e posteriormente armazenados em um banco de dados de índices para uso em consultas posteriores. Para manter estes arquivos de indexação de páginas de um motor de busca atualizados, implementa-se um *Web Crawler*, que fica varrendo a Internet indo de uma página para outra, armazenando os endereços visitados [LIU].

Quanto ao *Crawler* relatado neste artigo, trata-se de um *Crawler* focado e sua finalidade é relacionada a publicações científicas publicadas em sites repositórios específicos, tendo como objetivo gerar uma base de conhecimento das publicações disponibilizadas na *Web*. Assim, o usuário digita o termo desejado, em qual site repositório será feita a busca e qual o termo que ele está buscando, tendo desta forma uma busca direcionada. Após a realização da busca e extração

das informações, o resultado é exibido ao usuário e também armazenado em um banco de dados. O propósito desde armazenamento é que em eventuais próximas buscas iguais o banco de dados funcione com uma espécie de cache, ou seja, ao invés de acontecer uma nova varredura é utilizada uma "memória" de acesso muito mais rápida, neste caso a base de dados.

Este artigo está estruturado da seguinte forma: Na sessão 2 são apresentados os trabalhos relacionados com o tema ao qual a ferramenta se designa; na sessão 3 é descrita a ferramenta, sua visão geral, arquitetura e modo de implementação; na sessão 4 são apresentados os resultados dos experimentos sobre a ferramenta e por fim; na sessão 5 são apresentadas as considerações finais.

2. TRABALHOS RELACIONADOS

Motivado devido à imensa quantidade de dados digitais que encontram-se em bancos de dados escondidos, também chamados de *Deep Web*, I-FRIT [DOS SANTOS] apresenta uma abordagem simples, porém efetiva, para detecção e extração automática de rótulos centrada na análise estrutural do código HTML presente em páginas de formulários Web. Esta análise se baseia em um esquema de numeração para a hierarquia de *tags* HTML e um algoritmo que associa componentes de texto a campos do formulário, garantindo uma descoberta eficiente de rótulos presentes nestes formulários.

A abordagem descrita em [WU ET AL] mostra um estudo preliminar da evolução de um *Crawler* implementado para o sistema CiteSeerX. Este sistema utiliza um motor de busca que procura na *Web* por documentos acadêmicos e de pesquisa, principalmente em temas de ciências da computação e informação, para então retirar informações e indexá-las como metadados OAI, citações, tabelas e outros. Para melhorar a precisão das pesquisas, foi substituída a lista negra por uma lista branca e foi comparada a eficiência da varredura de dados antes e depois dessa mudança. Vale destacar que lista branca significa que apenas certos domínios são considerados e outros não são percorridos, já a lista negra significa que a varredura é proibida para certas URL's, mas para os outros artigos ela é ilimitada.

3. PUBFINDER

Esta sessão aborda a ferramenta PubFinder, que permite ao usuário realizar buscas pré-definidas em 3 sites repositórios de publicações de artigos científicos. A sessão se divide nas seguintes partes: visão geral, onde é introduzido o funcionamento da ferramenta; arquitetura, onde são exibidas as diferentes partes do sistema; implementação, onde são discutidos os conceitos e as tecnologias empregadas no desenvolvimento da ferramenta.

3.1 VISÃO GERAL

A ferramenta trabalha a partir de três etapas diferentes: (I) escolha do repositório, tipo da pesquisa e inserção do nome a ser pesquisado; (II) busca e extração dos dados no repositório escolhido e (III) exibição dos dados na tela para o usuário e gravação no banco de dados.

(I) ESCOLHA DO REPOSITÓRIO, TIPO DE PESQUISA E TERMO A SER BUSCADO
Esta primeira etapa consiste na interação do usuário com a interface gráfica da ferramenta, escolhendo em qual repositório ele deseja realizar a pesquisa, o tipo de informação que ele deseja e o nome a ser pesquisado. Os repositórios passíveis de busca são: DBLP - Computer Science

Bibliography[5]; PUBMED - US National Library of Medicine[6]; e Portal Capes[7]. O motivo da escolha destes três repositórios foi proposital para que a ferramenta pudesse contemplar 3 diferentes áreas das publicações científicas.

(II) BUSCA E EXTRAÇÃO DOS DADOS

Na segunda etapa a ferramenta realiza a busca e extração dos dados selecionados pelo usuário junto ao portal de publicações escolhido.

(III) EXIBIÇÃO DOS DADOS E INSERÇÃO NO BANCO DE DADOS

Na terceira etapa os dados extraídos do repositório são exibidos na interface gráfica da ferramenta e armazenados em um banco de dados. O intuito destes dados serem armazenados no banco de dados é de agilizar uma próxima pesquisa, ou seja, caso seja realizada uma nova busca por aquele mesmo termo e no mesmo repositório, o resultado será exibido quase instantaneamente, sem a necessidade de ser feita toda a varredura no site repositório.

3.2 ARQUITETURA

A figura 1 ilustra arquitetura interna da ferramenta.

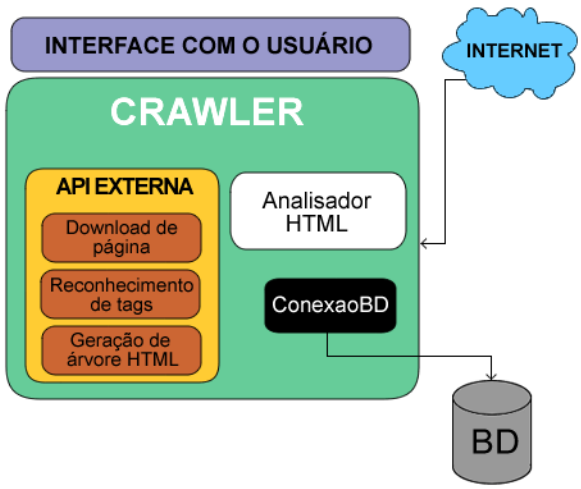

Figura 1 - Arquitetura da ferramenta PubFinder.

Detalhando um pouco mais a Figura 1, os itens abaixo mostram os principais módulos criados e qual sua interação dentro do software.

Interface: Responsável pela interação com o usuário, permitindo a especificação dos parâmetros de busca pelo usuário. Além disso, exibe o resultado da busca, ou seja, os dados extraídos durante o processo.

Crawler: Responsável por realizar o controle de toda a aplicação. É o coordenador de todas as etapas da ferramenta. Este módulo se comunica diretamente com todos os demais

[5] http://dblp.l3s.de/
[6] http://www.ncbi.nlm.nih.gov/pubmed
[7] http://www.periodicos.capes.gov.br/

módulos.

AnalisadorHTML: Esta é a parte principal da aplicação. Neste módulo são realizadas todas as análises necessárias junto ao site repositório de publicações.

ConexãoBD: Como o próprio nome já sugere, este módulo é responsável pela conexão ao Banco de Dados. Nele estão as funções básicas de conexão/acesso, busca e inserção de dados junto ao banco.

API Externa**:** Esta API resume-se a um *browser* sem interface gráfica, criado para poder ser incorporado em aplicações escritas em JAVA. Ela realiza as tarefas de download de páginas, a geração de uma árvore HTML e consequentemente a identificação de tags, facilitando o trabalho realizado no AnalisadorHTML na identificação do conteúdo importante para a aplicação.

3.3 IMPLEMENTAÇÃO

O projeto foi implementado na linguagem Java, utilizando-se da API externa HTMLUnit para realização das tarefas essenciais, como download das páginas HTML e reconhecimento das *tags*. A Interface gráfica foi implementada visando os critérios de usabilidade. Nela são escolhidos, através dos RadioButtons, o Site Repositório e o tipo de busca, assim como o termo o qual o usuário deseja pesquisar. A pesquisa é efetivamente executada no momento em que o usuário clica no botão "PESQUISAR". A disposição das informações na interface pode ser visualizada na Figura 2.

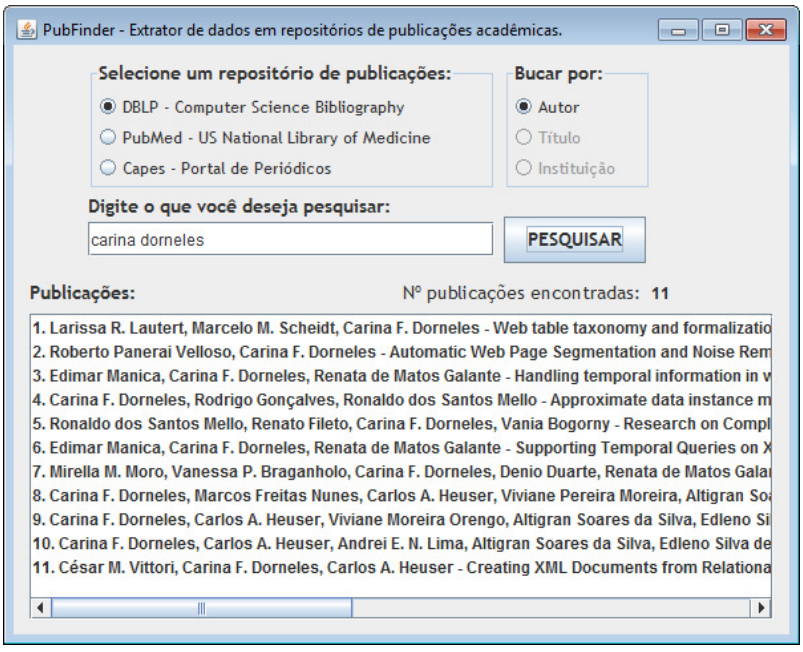

Figura 2 - Interface gráfica da ferramenta PubFinder.

No momento em que as informações da interfaces são enviadas (ação do botão "PESQUISAR") a classe Crawler inicializa um objeto da API HTMLUnit setando algumas configurações relevantes, que são: a não interpretação e exibição dos erros gerados na leitura das páginas *HTML* e a habilitação do redirecionamento entre páginas. Este último parâmetro é de

fundamental importância, já que caso ele não estivesse habilitado o evento de click em links não funcionaria durante a navegação virtual realizada pelo HTMLUnit.

Através do objeto API HTMLUnit já criado e configurado, a classe Crawler realiza o download da página do repositório selecionado na busca e utiliza os parâmetros informados pelo usuário. Esta página fica armazenada em memória com todos os recursos *HTML* disponíveis.

Para agilizar a busca foi utilizada a URL do sistema de busca do próprio site Repositório. A montagem da *String* da URL deste mecanismo se baseia na concatenação da URL de busca do próprio site com o termo buscado. Antes de se fazer o download da página é necessário tratar o termo buscado e realizar a adição (se possível) de auxiliadores de busca. No tratamento do termo buscado são retirados os acentos das letras ou caracteres especiais (Ç, Á, Ã, É, Õ, À) e é trocado o espaço em branco pelo seu código respectivo no site repositório (ex: %20).

O próximo passo é enviar a página baixada para a classe AnalisadorHTML, para que dela sejam extraídas as informações requisitadas pelo usuário. Até este ponto do fluxo as operações de *download* dos três repositórios são iguais, com exceção das *URLs* baixadas. Já na classe AnalisadorHTML cada pagina é tratada de uma maneira para que seja feita a devida extração dos dados relevantes. Antes da implementação ser feita, nesta etapa, foi analisada a estrutura HTML de cada site repositório. Após análise da composição do HTML, foi descoberto como chegar até as informações relevantes através de suas *tags*. As informações relevantes buscadas aqui são: Autor, Título da publicação e Ano. O tipo de *tag* mais utilizado foi o *div*.

Todos os métodos de extração de dados se basearam em uma mesma premissa: a partir da árvore de elementos, percorrer todos os elementos em busca das informações relevantes. Quando achada a *tag* com o conteúdo de interesse da ferramenta foram utilizados métodos da especificação DOM, fornecidos pela API externa, para a captura do texto relevante. Os métodos mais utilizados são: *ToString, AsText e GetFirstByXPath*.

3.3.1 DBLP

Neste repositório, pelo fato das informações serem dispostas em uma tabela, foi necessário varrer esta tabela como uma matriz, analisando se os textos contidos nas células da tabela eram relevantes para a extração. Para este repositório analisou-se que a *div* "result-box" dava inicio à marcação HTML da matriz de dados, sendo que a partir desta *tag* se iniciaria a análise e extração dos dados relevantes. Para extração dos dados é implementado um loop que percorre a árvore de *tags* HTML passada pela classe Crawler.

3.3.2 PUBMED

Neste repositório as informações são exibidas através de uma listagem com informações a respeito da publicação. Constatou-se que cada informação contida no site era encapsulada (ou poderia ser extraida) por *tags* do tipo *div, p* (paragraph) e *a* (anchor). Cada *tag*, por ter aplicação de um CSS de cores e estilo, acaba tendo um identificador único, o qual foi utilizado para busca e extração das informações. Para facilitar o encontro destas *tags* específicas foi utilizado o método *GetFirstByXPath,* da especificação W3C para interfaces DOM, cujo atributo passado foi o identificador ou nome da classe da *tag* em questão. Por se tratar de uma listagem e permitir o número máximo de 200 registros, foi necessária a implementação de um mecanismo para realizar a paginação destes registros. Foi através da ação de *click* sobre o botão de next que foi implementada a navegação na página. Utilizando-se do método *click* da API HTMLUnit foi

possível baixar e guardar em memória a página HTML associada à ação de *click*, simulando assim a navegação realizada por um *browser*.

3.3.3 Portal CAPES

Sendo a extração de dados deste repositório sem dúvidas a mais complexa de ser implementada, esta foi a que teve o desempenho pior, comparado as demais devido a uma série de detalhes. Os principais foram a exibição de apenas 10 registros por páginas, sem parâmetros para aumentar este valor e a inconstância do servidor do portal Capes em responder as requisições HTTP enviadas a ele.

Semelhante ao portal PUBMED, a extração dos dados do portal Capes foi utilizando-se da mesma estrutura para paginação. A implementação deste método foi realizada de forma semelhante ao portal PubMed onde foram utilizadas expressões para encontrar determinadas *tags* no método *getByXPath* contendo dados relevantes. Achado o conteúdo relevante no site é iniciada a extração, de fato, dos dados importantes. Nesta, a cada iteração de um *loop* por todos os elementos HTML da página é verificado se a *tag* em questão contém identificadores ou classes *CSS* já analisados anteriormente. Caso sejam encontradas, as informações contidas dentro destas *tags* são extraídas e armazenadas em variáveis locais, para posteriormente serem inseridas na lista de publicações encontradas e esta, por fim, ser enviada ao banco de dados e Interface.

Uma observação cabe a ser feita aqui: pelo fato de, muitas vezes, os nomes dos professores responsáveis pela publicação estarem abreviados, foi utilizado um método de comparação de *Strings*, nomeado como *verificaAutorCapes*, utilizado para confirmar se o que o usuário de fato está procurando é aquele que a pesquisa do site repositório retorna. Após a extração das informações é verificada a existência do botão *next*, caso este seja encontrado é realizada a ação de *click* sobre tal botão, acessando próxima página e é reiniciado o processo de busca pelo conteúdo relevante.

Conforme comentado anteriormente, o portal Capes teve comportamentos anormais quanto à apresentação de sua estrutura de dados. Durante os testes em pesquisas de termos que retornam muitos registros, como por exemplo universidades de grande porte internacional, ocorreram erros diversos. Geralmente estes erros estavam relacionados a erros de processamento da página no lado do servidor, tanto o hospedeiro *Web* do portal como no servidor que processa as requisições de página (*click*, pesquisa, paginação). Em casos como a exibição da página sem dados relevantes, a tratativa para continuar a extração dos dados foi tentar avançar o índice da página e ignorar a página não exibida na coleta de informações. Caso tentativa de *download* da página falhasse, a busca é encerrada e é exibido o conjunto de informações obtidos até então.

4. Experimentos

Para medir o desempenho da ferramenta foram utilizadas três métricas de avaliação dos testes: revocação, precisão e Medida-F. Revocação trata da quantidade de publicações encontradas comparadas com a quantidade total de publicações relevantes existentes no site. Já a precisão é a quantidade de publicações encontradas de maneira correta dentre todas as publicações encontradas. Por fim, a Medida-F é uma média harmônica entre as duas medidas obtidas, sendo obtida através da fórmula exibida na figura 2.

$$\text{Medida-F} = \frac{2 \times \text{Precisão} \times \text{Revocação}}{\text{Precisão} + \text{Revocação}}$$

Figura 2 - Fórmula para o cálculo da Medida-F.

Foram realizadas 30 pesquisas em cada opção de site repositório da ferramenta PubFinder (DBLP, PubMed e Capes) e suas devidas opções de busca (autor, título e instituição). Lembrando que no número total de publicações serão consideradas apenas aquelas publicações que retornam ao serem buscadas diretamente no site repositório em um navegador com a possível análise se aquele dado é de fato verdadeiro ou não. Caso o site não exiba determinadas publicações estas não são consideradas.

Conforme a Tabela 1 exibe, foram feitas 30 buscas distintas para cada tipo de Pesquisa que a ferramenta contempla, e delas foram feitas as análises a seguir.

Repositório	Autor	Título	Instituição	Nº de Buscas
DBLP	X	-	-	30
PubMed	X	X	X	90
Capes	X	X	X	90
			Total:	**210**

Tabela 1 - Configurações de teste para avaliação de desempenho da Ferramenta.

Na figura 3 encontram-se os resultados obtidos nas métricas. Todos os tipos de busca dos portais obtiveram 100% de Precisão, Revocação e Medida-F, com exceção do portal Capes, cujas métricas atingiram valores menores, porém satisfatórios, ficando com a grande maioria das faixas de acerto acima de 75% de acerto. O motivo destes valores não atingirem os 100% de desempenho deve-se ao fator comentado nas sessões anteriores da falha da disponibilidade dos dados por parte do Servidor de Dados do portal Capes ou por devidas falhas na extração precisadas informações relevantes.

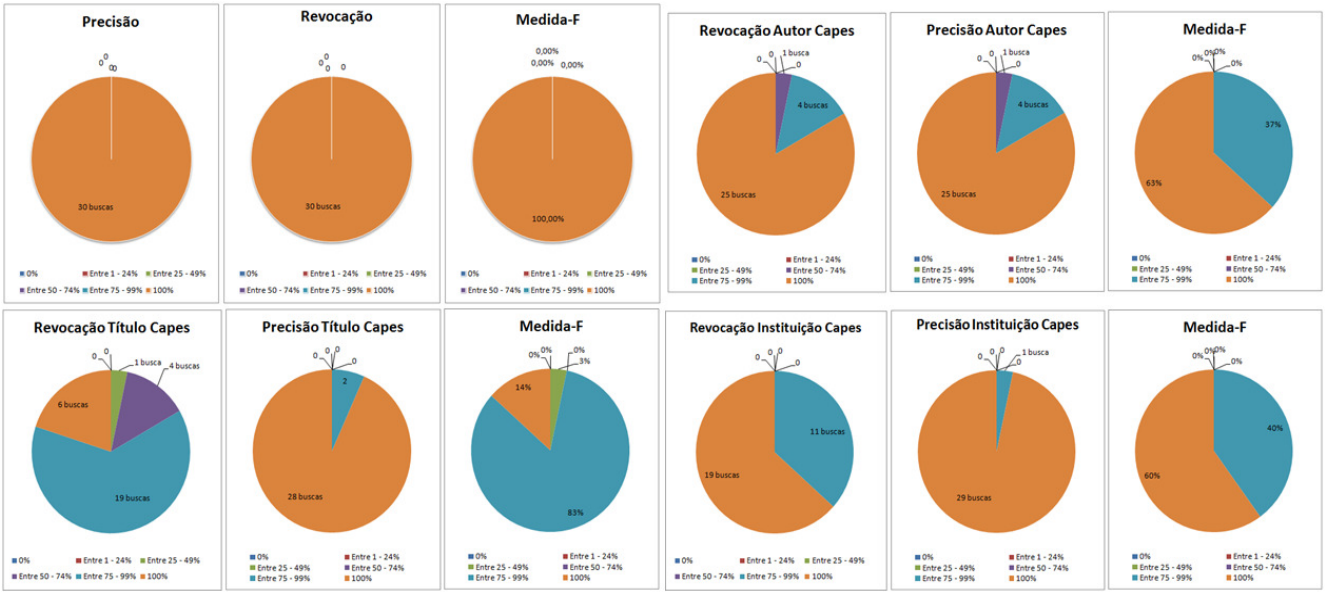

Figura 3 - Exibição dos gráficos contendo os resultados das métricas de desempenho.

5. Considerações finais

Ao final da implementação, a ferramenta atingiu seus objetivos, fazendo com que o usuário pudesse interagir com a ferramenta e obter resultados através de sua interface. Através dos resultados obtidos nos testes foi possível observar que a ferramenta atende ao objetivo que lhe é solicitado com níveis consideráveis de aceitação, atingindo um índice muito alto de eficiência na extração. Com o uso da ferramenta foi possível montar um amplo banco de conhecimento com os resultados das pesquisar realizadas nos testes. Usando-se destas pesquisas torna-se mais fácil a reunião de informações referente a determinados assuntos ou de determinados professores que sejam pertinentes ao usuário da ferramenta.

Referindo-se à utilidade prática da aplicação, através do uso da ferramenta é possível comparar manualmente, por exemplo, quanto um professor de determinada instituição atualiza de fato seu currículo Lattes com suas publicações ou de seus orientandos.

Ao longo do desenvolvimento da ferramenta foram observados alguns pontos, que não foram desenvolvidas por diversos fatores, sendo eles a melhoria contínua do processo de extração dos dados, principalmente no Portal Capes, incorporação de outros sites e a necessidade de um estudo para tentar migrar a ferramenta para um portal *Web* de acesso único pelos usuários.

REFERENCES

[LIU] LIU, B. Web Data Mining: Exploring Hyperlinks, Contents and Usage. (Data-Centric Systems and Applications). Chicago, 2001, p. 311.

[DOS SANTOS] dos Santos, L.B., Dorneles, C.F. e Mello, R.S. An Approach for Extracting Web Form Labels Based on Distance Analysis Of HTML Components. IADIS WWW/Internet Conference, Madrid, 2012.

[WU ET AL] Jian, Wu, Teregowda , Pradeep, Ramírez, Juan Pablo Fernández, Zheng, Prasenjit Mitra, Shuyi, Giles, C. Lee. In proceedings of the 3rd Annual ACM Web Science Conference Pages 340-343, Evanston, IL, USA, June 2012.