UNIVERSIDADE FEDERAL DE SANTA CATARINA

qFex: um crawler para busca e extração de questionários de pesquisa em documentos HTML

Gilney Nathanael Mathias

Florianópolis – SC

2017/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

qFex: um crawler para busca e extração de questionários de pesquisa em documentos HTML

Gilney Nathanael Mathias

Trabalho de Conclusão de Curso, a ser submetido ao Curso de Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.

Florianópolis – SC

2017/2

Gilney Nathanael Mathias

qFex: um crawler para busca e extração de questionários de pesquisa em documentos HTML

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador(a): Carina Friedrich Dorneles

Banca examinadora

Richard Henrique de Souza

Ronaldo dos Santos Mello

SUMÁRIO

Lista de figuras

Lista de Reduções

HTML – Hypertext Markup Language

URL – Uniform Resource Locator

DOM – Document Object Model

DAO – Data Access Object

CSS – Cascading Style Sheets

JSON – JavaScript Object Notation

RESUMO

Questionários de pesquisa são ferramentas poderosas e importantes para coleta de opiniões e perfis de pessoas, e são construídos para os mais variados propósitos. Com a popularização da Web, tornou-se comum a disponibilização destes questionários de forma online, a fim de que a participação das pessoas seja mais efetiva. Tais questionários podem ser utilizados sob vários contextos, tais como: em empresas para avaliar seus produtos, a satisfação de serviços, o atendimento de seus funcionários, etc; em instituições de ensino e pesquisa para avaliação de seus corpos docentes e discentes, por exemplo; ou por pesquisadores para coleta de dados que são posteriormente usados em estudos e pesquisas. Alguns problemas na criação de tais questionários envolvem: decidir quais perguntas fazer, como fazê-las e como organizá-las. Portanto, pode ser bastante útil reusar questionários já criados para a realização de novas coletas de dados. Visando isso, este trabalho propõe a criação de um *Web Crawler*, que varra a Web em busca de sites que possivelmente contenham questionários, e de um Extrator, que seja capaz de extrair os questionários de uma lista de sites e salvá-los em um banco de dados relacional de forma estruturada, e criar uma base centralizada de exemplos para a elaboração de novos questionários ou ainda para o reuso de questões existentes.

Palavras chaves: Banco de Dados. Crawler. Extração de Dados.

1 Introdução

Nos últimos anos temos visto um aumento gigantesco no uso da Web e nos dados encontrados na mesma. A Internet facilita muito a comunicação entre pessoas de vários cantos do mundo e a busca por informações e conhecimento. Tais fatos abriram as portas para o uso de questionários online, uma forma mais abrangente e fácil para empresas e pesquisadores coletarem dados ou perfis de pessoas. A grande vantagem do uso de tais questionários online é a possibilidade de alcançar um grande número de pessoas, com características em comum e/ou únicas, de forma rápida e barata (WRIGHT, 2005).

“Questionários desse tipo normalmente começam com a necessidade de se obter informação aonde nenhum dado – ou uma quantidade insuficiente de dados – existe” (CANADA, 2003, p. 1, tradução nossa). Alguns problemas encontrados na fase de *design* de tais questionários incluem: decidir quais questões perguntar, qual o melhor jeito de expressá-las e como arranjar as perguntas para se obter a informação necessária (CANADA, 2003, p. 3).

Tais questionários podem ser utilizados sob vários contextos, tais como: em empresas para avaliar seus produtos, a satisfação de serviços, o atendimento de seus funcionários, etc; em instituições de ensino e pesquisa para avaliação de seus corpos docente e discente (SILVA, 2012), por exemplo; ou por pesquisadores para coleta de dados que são posteriormente usados em estudos e pesquisas. Portanto, pode ser bastante útil reusar questionários já criados para a realização de novas coletas de dados.

Em vista disso, este trabalho tem como objetivo coletar questionários de pesquisa na Web e extrair os dados para construir um banco de dados centralizado das informações coletadas. Os dados extraídos servirão como uma

base de conhecimento que pode ser usada como ponto de partida para a construção de novos questionários ou para a análise das características presentes visando extrair algum tipo de informação útil. Vale ressaltar que até a presente data não se encontrou nenhum trabalho que tenha esse foco na busca e extração dos dados de questionários de pesquisa na Web.

O grande desafio na identificação e extração dos dados de questionários online é a quantidade enorme de diferentes formas utilizadas em sua construção utilizando HTML. Cada site tem sua própria maneira de estruturar o HTML, de estilizar os elementos da página e de enviar os dados do cliente para o servidor. Além disso, as páginas podem ter conteúdo estático, aonde todas as perguntas do questionário são carregadas após o usuário responder uma ou mais perguntas anteriores ou clicar em um botão de 'próximo', redirecionando o usuário para a próxima parte ou pagina do questionário. Desta forma, é importante ressaltar que este trabalho se foca na identificação e extração de questionários contidos em páginas com conteúdo estático.

Para lidar com a falta de padronização na criação de questionários online utilizando HTML, propõe-se a criação de um conjunto de heurísticas que possa ser utilizado para fazer a identificação de questionários em páginas HTML e para fazer a extração das questões presentes nesses questionários. Além disso, é proposta a implementação de um *Web Crawler* focado, um programa que varre a Web fazendo o download de páginas que possuam certas características interessantes para o usuário (LIU, 2011, p. 311 e p. 327), bem como de um Extrator, que implementem as heurísticas mencionadas e que juntos sejam capazes de fazer a coleta dos questionários e a extração de seus dados para uma base de dados.

Este trabalho está estruturado da seguinte forma: o Capitulo 2 trata sobre a fundamentação teórica necessária para um melhor entendimento dos principais pontos abordados no projeto; o Capitulo 3 apresenta alguns trabalhos relacionados a *crawling* e extração de dados na Web; no Capitulo 4 são apresentados os detalhes do desenvolvimento e implementação das ferramentas criadas; o Capitulo 5 apresenta os experimentos feitos para validar as abordagens definidas e por fim, no Capitulo 6 são expostas as considerações finais sobre o trabalho e os possíveis trabalhos futuros.

2 Fundamentação teórica

Este capitulo visa tratar da fundamentação teórica necessária para um melhor entendimento deste trabalho. São abordados os principais conceitos, definições e características relacionados a Numeração Dewey, Extração de dados e Web Crawlers.

2.1 Numeração Dewey

"A Numeração Dewey é baseada na *Dewey Decimal Classification* desenvolvida para a classificação geral de conhecimento" (TATARINOV, 2002, p. 4, tradução nossa). Essa numeração pode ser utilizada para dar um identificador (ID) único para os elementos de uma estrutura de árvore e também serve como uma forma de saber a posição desses elementos na mesma. O nodo raiz recebe o identificador "1" e então cada um dos seus nodos filhos recebem um ID composto do ID de seu nodo pai unido com um número representando a posição que esse nodo filho aparece na árvore. Por exemplo: o primeiro nodo filho receberá o ID "1.1", o segundo "1.2", e assim por diante. A Figura 1 mostra um exemplo de como esta numeração pode ser utilizada em uma página HTML.

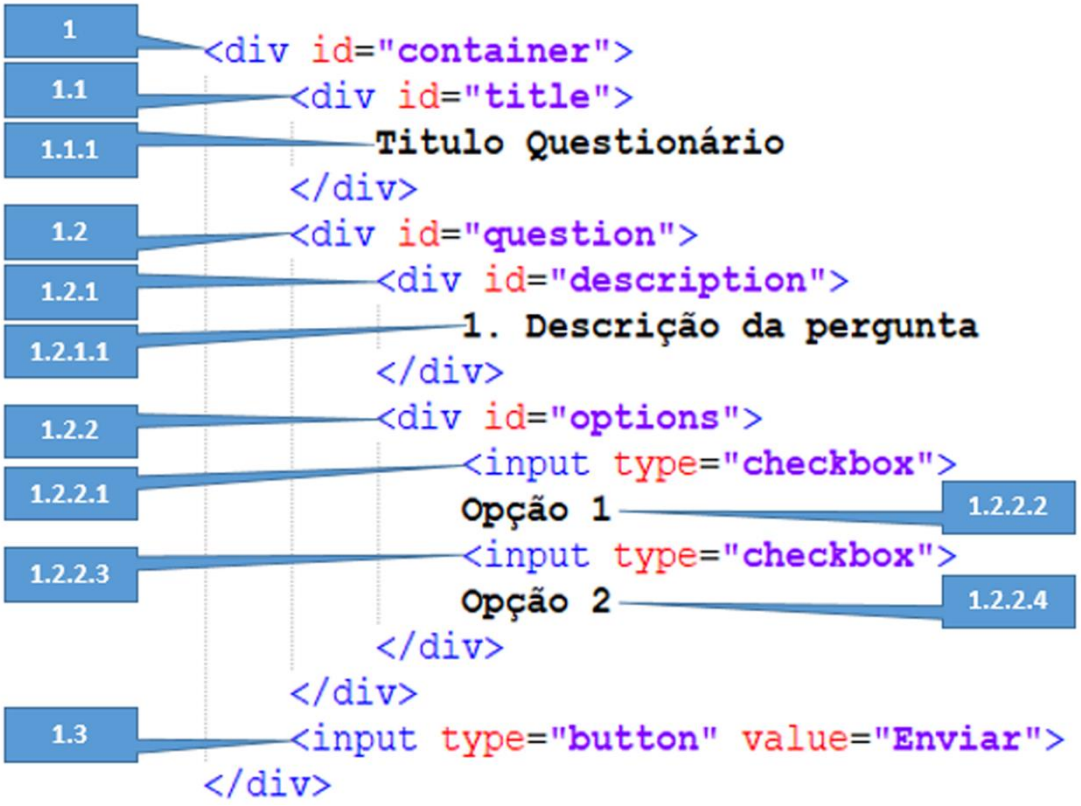

FIGURA 1. DEMONSTRAÇÃO DO USO DA NUMERAÇÃO DEWEY NO HTML.

Uma outra utilidade da Numeração Dewey é a possibilidade de calcular a distância entre os elementos, como explorado em Santos (2012, p. 5). Essa distância é calculada simplesmente fazendo-se a subtração dos IDs dos nodos, como pode ser visto na Figura 2. O cálculo é feito partindo-se do começo dos IDs e fazendo a subtração numero a número, ignorando os pontos e os primeiros valores iguais. Caso um dos IDs tenha o comprimento maior do que o outro, o cálculo é feito normalmente, levando em consideração o tamanho do menor, e em seguida o restante dos números do maior ID são copiados para o resultado da distância. Por exemplo, na Figura 2(b), o ID ‘1.1.1’ tem comprimento maior do que o ID ‘1.3’ e por isso o último número ‘1’ do primeiro ID é copiado para o resultado da distância. Isso é equivalente a preencher com zeros o menor ID, ficando, por exemplo, ‘1.3.0’.

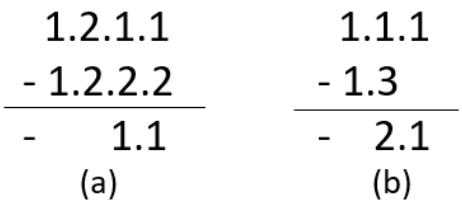

FIGURA 2. EXEMPLOS DO CÁLCULO DA DISTÂNCIA ENTRE DOIS NODOS.

2.2 Extração de dados da Web

A extração de informações da Web se baseia no problema de extrair itens alvos de informação de páginas Web. Existem dois problemas gerais nesta área: extrair informação de textos em linguagem natural e extrair dados estruturados de páginas Web (Liu, 2011, p. 363). Este trabalho enfatiza a extração de dados estruturados (questionários) de páginas Web.

"Uma abordagem tradicional para a extração de dados de páginas Web é a escrita de programas especializados, chamados de *wrappers*, que identificam os dados de interesse e mapeiam eles para um formato adequado" (LAENDER et al., 2002, tradução nossa). A seguir são descritos os seis principais grupos de ferramentas para a geração de *wrappers* apresentados em Laender et al. (2002):

- **Linguagens para desenvolvimento de *wrappers*:** Linguagens especificamente criadas para ajudar os usuários a construir *wrappers*. Estas linguagens foram propostas como alternativas a linguagens de propósito geral como Java. Alguns exemplos de ferramentas que adotam essa abordagem são: Minerva (CRESCENZI; MECCA, 1998) e TSIMMIS (HAMMER; MCHUGH; GARCIA-MOLINA, 1997).
- **Ferramentas conscientes do HTML:** Ferramentas que dependem da estrutura inerente dos documentos HTML. Antes de começar o processo de extração, estas ferramentas transformam o documento em uma árvore

de analise, uma representação que reflete a hierarquia de *tags* da página HTML. Em seguida, regras de extração são geradas semiautomaticamente ou automaticamente e são aplicadas a árvore. Algumas ferramentas representativas baseadas nessa abordagem são: W4F (SAHUGUET; AZAVANT, 2001) e XWRAP (LIU; PU; HAN, 2000).

- **Ferramentas baseadas no Processamento de Linguagem Natural:** Estas ferramentas geralmente aplicam técnicas como filtragem de dados e marcação léxica e semântica para construir relações entre elementos de frases e sentenças para derivar regras de extração. Tais regras são baseadas em restrições sintáticas e semânticas que ajudam a identificar informações relevantes dentro de documentos. Algumas ferramentas que utilizam essa abordagem são: RAPIER (CALIEF; MOONEY, 1997) e SRV (FREITAG, 2000).
- **Ferramentas para indução de *wrappers:*** Ferramentas neste grupo geram regras de extração baseadas em delimitadores derivados de um determinado conjunto de exemplos de treinamento. A principal distinção entre essas ferramentas e as baseadas em Processamento de Linguagem Natural é que elas não dependem de restrições linguísticas, mas sim em características de formatação que implicitamente delineiam a estrutura dos pedaços de dados encontrados. Ferramentas como WIEN (KUSHMERICK, 2000) e SoftMealy (HSU; DUNG, 1998) são representativos dessa abordagem.
- **Ferramentas baseadas em modelagem:** Ferramentas nesta categoria recebem como entrada certas estruturas, como listas e tabelas, e tentam localizar nas páginas Web porções de dados que implicitamente se

acomodam a essas estruturas. Algumas ferramentas que adotam essa abordagem são: NoDoSE (ADELBERG, 1998) e DEByE (LAENDER; RIBEIRO-NETO; DA SILVA, 2001).

- **Ferramentas baseadas em Ontologias:** Diferente das abordagens apresentadas acima, que dependem da estrutura de apresentação dos dados para gerar regras de extração, as ferramentas deste grupo dependem diretamente dos dados das páginas Web e utilizam ontologias para localizar constantes nessas páginas e criar objetos a partir dessas constantes. A ferramenta mais representativa dessa abordagem é a ferramenta desenvolvida pelo Grupo de Extração de Dados da Universidade Brigham Young (EMBLEY et al., 1999).

2.3 Web Crawler

"Um *web crawler*, também conhecido como *spider* ou robô, é um programa que baixa automaticamente páginas da Web" (Liu, 2011, p. 311, tradução nossa). Ele pode ser utilizado para vários fins, como por exemplo, para indexar páginas web para motores de busca, como o da Google, para fazer o arquivamento da Web, como o providenciado pelo *Internet archive*[1], e para realizar *data mining* (Olston; Najork, 2010).

O algoritmo básico de um *web crawler* é bem simples: Dada uma lista de URLs iniciais, chamadas de *seeds*, o *crawler* faz o download de todas as páginas web endereçadas por essas URLs iniciais, extrai os *hyperlinks* contidos nessas páginas, e interativamente faz o download das páginas web apontadas por esses *hyperlinks* (Olston; Najork, 2010).

[1] https://archive.org/

Muitas vezes pode-se não querer fazer o *crawling* de toda a Web, focando-se apenas em acessar páginas de certas categorias ou tópicos. Um *web crawler* focado tenta direcionar o *crawler* para páginas de certas categorias interessantes para o usuário (Liu, 2011, p. 327). Este trabalho implementa um *web crawler* focado que busca na Web páginas que possuam questionários online e faz a extração dos dados destes questionários.

3 Trabalhos Relacionados

Dado que, até a presente data, não se conseguiu encontrar trabalhos que tratem da busca e extração dos dados de questionários de pesquisa na Web, optou-se por apresentar neste capítulo apenas trabalhos relacionados a área de *crawling* e extração de dados estruturados ou semiestruturados na Web.

3.1 Ifrit

O Ifrit (Santos, 2012) é um *web crawler* focado para fazer a detecção de formulários na Web, principalmente os que são criados sem utilizar a *tag* FORM, e para fazer a associação dos componentes desses formulários (INPUTs, TEXTAREAs, afins) com os seus respectivos rótulos (*labels*). Para alcançar esses objetivos, o Ifrit utiliza como base a Numeração Dewey e a ideia de distância entre elementos, apresentados na seção 2.1, além de um conjunto de heurísticas.

Para fazer a identificação de formulários, o Ifrit primeiro cria uma árvore de elementos que mantém a mesma estrutura contida na página HTML e durante esse processo vai adicionando um identificador (ID), referente a Numeração Dewey, para cada um desses elementos. Em seguida, utiliza-se um algoritmo recursivo que percorre a árvore fazendo vários testes para tentar identificar estruturas que se pareçam com formulários. Estes testes envolvem encontrar um elemento que possa servir como botão de envio do formulário, verificar as distancias máximas entre os elementos e o *container* (DIVs, SECTIONs, afins) principal que os armazena, analisar a densidade dos elementos e entre outros. Muitos aspectos desses testes podem ser modificados pelo usuário via parâmetros de configuração.

Caso seja possível afirmar a presença de um questionário na página web, o algoritmo então entra na fase de associação de rótulos. A entrada para este processamento são duas listas, uma contendo os componentes de formulário e a outra os possíveis rótulos, que foram identificados na etapa anterior. Para fazer tal associação, o algoritmo cria uma matriz que fornece as distancias de cada componente em relação a cada rótulo e então atribui para cada componente o rótulo que estiver mais próximo dele.

3.2 WT2SQL

O objetivo do *web crawler* WT2SQL (SCHEIDT, 2013) é fazer a detecção e extração dos dados de tabelas (*WebTables*) contidas em páginas da Web. Para tal, a ferramenta varre a Web, a partir de um conjunto de URLs iniciais, encontra as tabelas, procurando pelos elementos TABLE da página, e utiliza algumas heurísticas para analisar essas tabelas, verificando se são tabelas de dados ou de formatação, e para fazer suas extrações.

A análise mencionada acima é feita para eliminar tabelas que são utilizadas apenas para a formatação visual de elementos na página. Isto é feito verificando-se a quantidade e linhas e colunas na tabela e se ela possui *tags* UL, OL, LI ou componentes de formulário.

O processo de extração das tabelas de dados começa procurando pelo cabeçalho dessas tabelas, em seguida procura-se identificar a forma que os dados das tabelas estão sendo apresentados, horizontalmente ou verticalmente, e por último se faz a extração em si desses dados. Para encontrar o cabeçalho, o algoritmo de extração primeiro verifica se a tabela HTML possui *tags* THs, caso isso ocorra, os valores dessas *tags* THs formam o cabeçalho da tabela. Caso

contrário, é verificado a quantidade de linhas e colunas da tabela. Quando houver mais linhas do que colunas, a primeira linha é eleita o cabeçalho, se não, a primeira coluna é eleita.

3.3 Extração automática de sites de notícias

A proposta abordada em (REIS et al., 2004) para fazer a extração automática de notícias contidas em sites da Web utiliza como base o algoritmo de Distância de Edição entre Árvores (*Tree Edit Distance*). A ideia deste algoritmo é encontrar um conjunto mínimo de operações para realizar a transformação de uma árvore A em outra árvore B e esta ideia é utilizada pelos autores de (REIS et al., 2004) para criar o seu próprio algoritmo, chamado de RTDM, que é então usado para identificar passagens de texto relevantes contendo notícias e seus componentes, extrair eles e descartar elementos desnecessários como *banners*, links, etc.

A parte de extração das notícias é feita em quatro passos:

1. **Agrupamento de páginas**: Este passo recebe como entrada um conjunto de páginas de treinamento e utiliza um algoritmo de agrupamento hierárquico (*hierarchical clustering*) para gerar os *clusters* de páginas que compartilham características comuns de formatação/layout. A medida de distância utilizada pelo algoritmo de *clustering* é o valor gerado pelo algoritmo RTDM.
2. **Geração de padrões de extração**: Neste passo, é feito uma generalização dos *clusters* de páginas providenciados pelo passo 1 em padrões que os autores chamam de *node extraction pattern* (*ne-pattern*), que seriam basicamente expressões regulares para árvores.

3. **Correspondência de dados**: Neste passo é feito uma comparação entre os *ne-patterns* gerados no passo anterior e as árvores das páginas sendo visitadas pelo *crawler* para tentar encontrar o *ne-pattern* mais apropriado para fazer a extração de cada página.
4. **Rotulação dos dados**: A saída do passo 3 é um conjunto de passagens de texto ordenadas. O objetivo deste passo então é tentar encontrar nesse conjunto as passagens que correspondem ao título e ao corpo da notícia. Para isso são aplicadas algumas heurísticas sobre as passagens desse conjunto, como: o corpo da notícia é a maior passagem com mais de 100 palavras e o título da notícia é a passagem que possui entre 1 ou 20 palavras e é a mais próxima do corpo da notícia.

4 qFex

Este capítulo apresenta a proposta do *Web Crawler* focado e Extrator qFex (**F**inder and **Ex**tractor of **Q**uestionnaires). Inicialmente é descrita uma visão geral do trabalho e em seguida são explicados os conceitos e as ferramentas utilizadas no desenvolvimento do mesmo. São dois os principais problemas abordados por este trabalho: fazer a detecção de questionários em uma página qualquer da Web; e conseguir fazer a extração dos dados neles contidos, de forma genérica, para um banco de dados relacional.

4.1 Visão geral

Esta seção apresenta um apanhado geral das aplicações desenvolvidas neste trabalho, mostrando seus diferentes componentes, suas entradas e saídas e as interações que ocorrem entre os mesmos.

Como pode ser visto na Figura 3, o processo de coleta e extração de questionários foi dividido em dois componentes: *Crawler* e *Extractor*. Ambos recebem como entrada um arquivo de configuração que possui as configurações do banco de dados, da biblioteca de *crawler*, de *seeds*[2] para busca/extração e de valores para os parâmetros utilizados nos mesmos. O *Crawler* deve varrer a Web, utilizando como base as *seeds* fornecidas no arquivo de configuração, em busca de questionários que possuam certas características e então deve salvar seus links em um banco de dados. O *Extractor*, por sua vez, deve apanhar os links do banco de dados do *Crawler*, e possivelmente também do arquivo de configuração, e extrair os dados dos questionários para um outro banco de dados.

[2] Lista de URLs iniciais utilizadas pelo *Crawler*/Extrator.

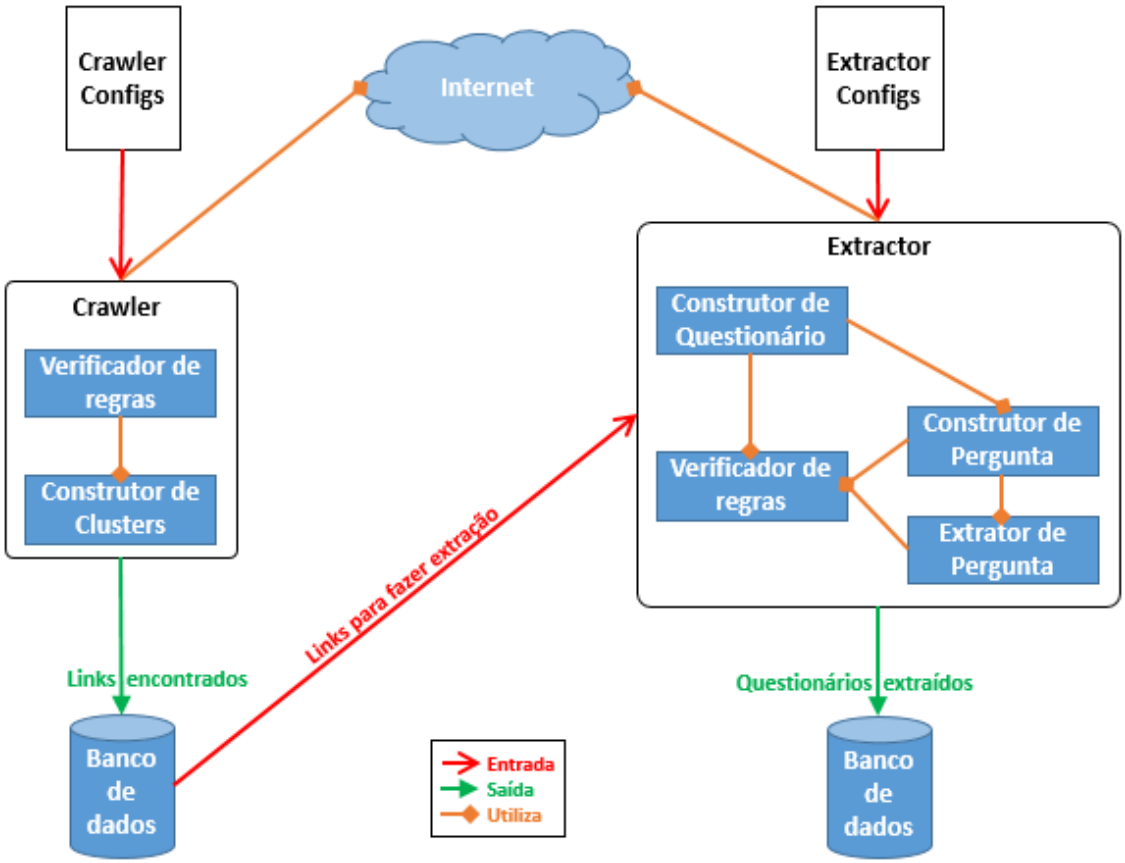

FIGURA 3. VISÃO GERAL DO QFEX.

4.2 Dewey-Ext

Neste trabalho, a numeração Dewey foi estendida e chamada de Dewey-Ext de forma a permitir o acréscimo de novos conceitos utilizados nas heurísticas empregadas. Tais conceitos servem para averiguar as relações entre os nodos e são utilizados em quase todos os parâmetros de configuração e heurísticas do *Crawler* e do *Extractor*. Os conceitos adicionados a numeração Dewey são os seguintes:

- ***Height***: A 'altura' se refere ao valor do primeiro número do ID da distância entre dois nodos. No caso da Figura 4(a), a altura é 1, ou seja, os primeiros nodos que eles têm de diferença estão em sequência na estrutura HTML;

- ***Max Height***: A ‘altura máxima’ é o valor do maior número do ID da distância. No exemplo da Figura 4(b), a altura máxima é 2;
- ***Width***: A ‘largura’ é o valor do comprimento do ID da distância. No caso da Figura 4(c), este valor é 2. Isto pode ser utilizado para verificar se os nodos não estão em profundidades muito distintas na árvore;
- ***Common Prefix***: Se refere ao maior ‘prefixo comum’ entre os IDs dos nodos. Este cálculo é demonstrado na Figura 4(d) e 4(e) e é feito varrendo-se os IDs da esquerda para a direita e copiando os seus valores iguais até que se encontre o primeiro valor diferente. Este resultado pode ser utilizado para verificar se três ou mais nodos possuem a mesma sequência de parentes, caso os prefixos sejam iguais, ou para se averiguar a quantidade de parentes em comum, olhando o comprimento dos prefixos.

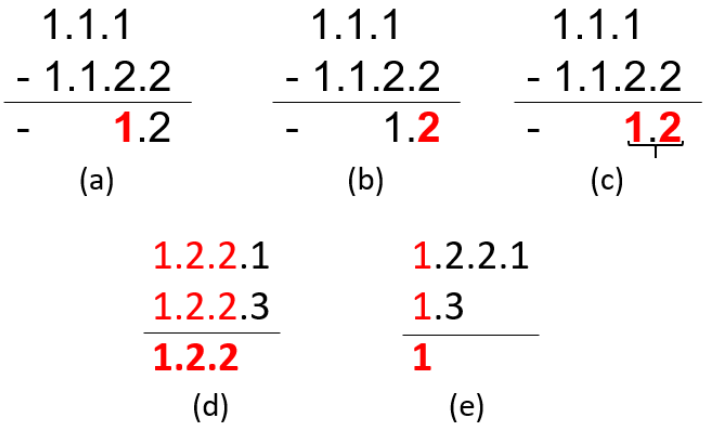

FIGURA 4. EXEMPLOS DO CÁLCULO DA DISTÂNCIA E PREFIXO COMUM ENTRE DOIS NODOS.

Além dos conceitos recém introduzido, vale ressaltar outras alterações realizadas neste trabalho em relação à Numeração Dewey:

- Nodos de comentário, *tags* BR, DIV, SPAN, P, TH, e A, vazias e sem o atributo *href*, são ignorados na hora de montar os IDs. Isto foi feito pois esses elementos não agregam em nada para o objetivo do trabalho e

ainda por cima poderiam atrapalhar nas medidas de distância entre os elementos do questionário;

- Nodos de texto separados por *tags* de quebra de linha, BR, são unidos em um único nodo. Isto ajuda em alguns aspectos da implementação, como a descoberta dos cabeçalhos de matrizes aonde o texto dos mesmos foi quebrado com o uso da *tag* BR para uma melhor representação visual;
- Os IDs são representados com preenchimento de até dois zeros, por exemplo: o ID 1.10.100 é representado no projeto como 001.010.100. Esta forma de representação permite que algumas operações sejam facilitadas.

4.3 Heurísticas

Para realizar a detecção, e em seguida a extração, de questionários em páginas Web, foi necessário primeiro encontrar características que distinguem os questionários online de outros tipos de formulários (*WebForms*) quaisquer. Após a análise de diversos sites, e de vários testes, chegou-se a um conjunto de heurísticas que são utilizadas nos algoritmos do *Crawler* e do Extrator para fazer tal detecção e extração. A seguir são apresentadas as principais heurísticas que são comuns entre o *Crawler* e o Extrator, sendo que as demais são explicadas nas próximas seções:

Heurística 1. A descrição de uma pergunta normalmente começa com um número ou uma palavra qualquer iniciando com letra maiúscula, possuindo pelo menos quatro caracteres, um espaço e terminando com o caracter ‘:’, ‘?’ ou ‘.’.

Como é possível ver na Figura 5, isso nem sempre é verdade, mas quando ocorre acaba sendo uma grande indicação de uma possível pergunta.

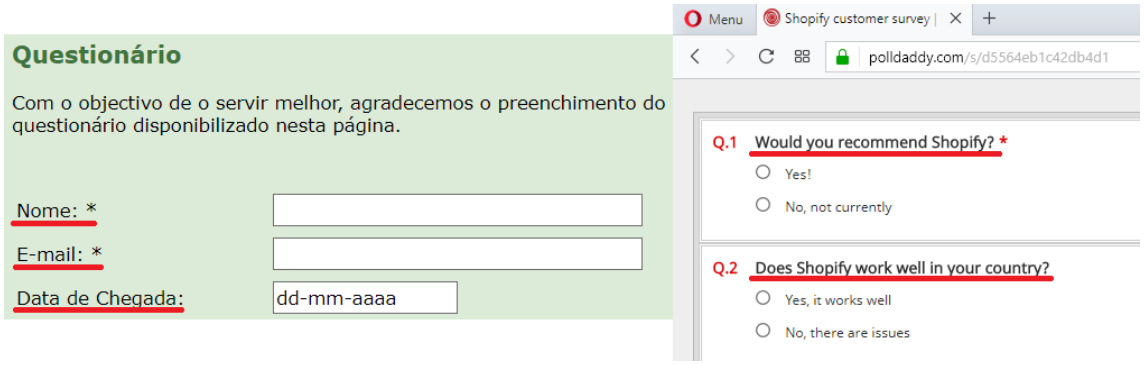

FIGURA 5. EXEMPLOS DE DESCRIÇÕES DE PERGUNTAS EM UM QUESTIONÁRIO.

Heurística 2. Perguntas normalmente possuem suas descrições e componentes de formulário próximos uns dos outros.

No caso, componentes de formulários são os elementos de INPUT, TEXTAREA, SELECT e afins. A Figura 6 apresenta um exemplo disso, aonde é possível ver que, neste caso, a altura da distância entre as descrições das perguntas e seus componentes de formulário é apenas 1 e a largura é 2.

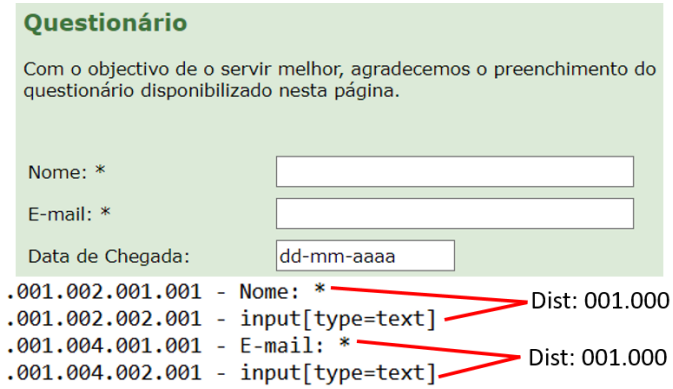

FIGURA 6. EXEMPLO DA DISTÂNCIA ENTRE AS DESCRIÇÕES DAS PERGUNTAS E SEUS COMPONENTES.

Heurística 3. É possível eliminar certos formulários/campos que não pertencem a um questionário, verificando a distância entre os componentes e certas palavras/frases que são normalmente encontradas nos mesmos.

Esta heurística é utilizada principalmente para eliminar componentes soltos pelas páginas web, como campos de busca, ou ainda para casos de formulários de *login*, registro e afins. A Figura 7 exibe algumas das palavras/frases que são normalmente encontradas nesse tipo de formulário.

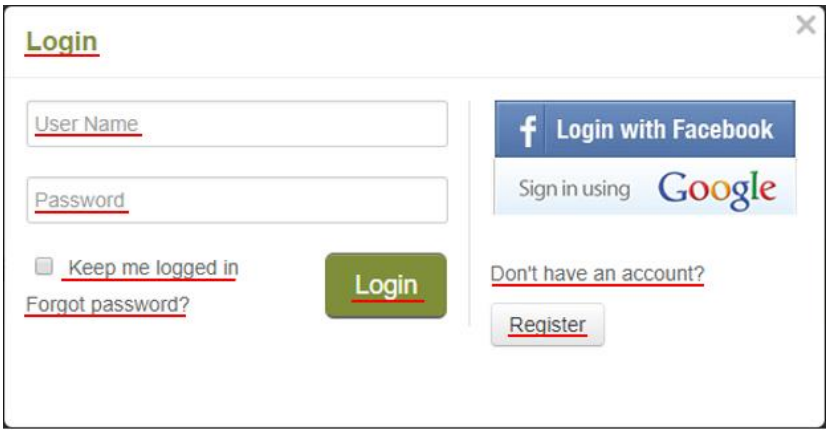

FIGURA 7. EXEMPLO DE PALAVRAS/FRASES COMUNS EM FORMULÁRIOS DE *LOGIN*.

4.4 Detecção de questionários

Além das heurísticas apresentadas na seção anterior, o algoritmo de detecção de questionários também utiliza algumas heurísticas a mais que são apresentadas a seguir:

Heurística 4. Páginas com questionários normalmente possuem certas palavras comuns em seu título e/ou em seu corpo.

Algumas palavras são bastante comuns em questionários online e podem ser utilizadas como uma filtragem inicial para verificar se uma dada página web possui ou não um questionário. A Figura 8 mostra alguns exemplos dessas palavras.

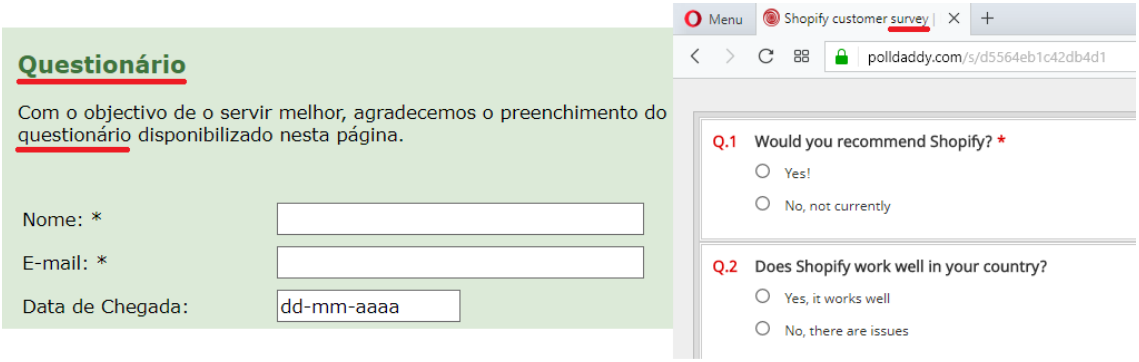

FIGURA 8. EXEMPLOS DE PALAVRAS NORMALMENTE ENCONTRADAS EM QUESTIONÁRIOS ONLINE.

Heurística 5. É possível determinar a presença de um questionário verificando a quantidade de *clusters* (agrupamento de nodos) que possuam pelo menos um componente de formulário ou que possuam uma certa quantidade de componentes de formulário.

Como é possível ver na Figura 9, um questionário normalmente vai possuir uma certa quantidade mínima de clusters em sequência que possuam componentes de formulário. Só que as vezes as perguntas estão tão próximas umas das outras que muitas ficam agrupadas no mesmo cluster e por isso que se tem que verificar também a quantidade de componentes em um mesmo cluster, além da quantidade de clusters com componentes.

```
<1>Cluster:
        001.001.002.001.004.001.001 - Q.1
        001.001.002.001.004.002.001.001 - Would you recommend Shopify?
        001.001.002.001.004.002.002.001.001.001 - input[type=radio]
        001.001.002.001.004.002.002.001.001.002.001 - Yes!
        001.001.002.001.004.002.002.001.002.001 - input[type=radio]
        001.001.002.001.004.002.002.001.002.002.001 - No, not currently

<2>Cluster:
        001.001.002.001.005.001.001 - Q.2
        001.001.002.001.005.002.001.001 - Does Shopify work well in your country?
        001.001.002.001.005.002.002.001.001.001 - input[type=radio]
        001.001.002.001.005.002.002.001.001.002.001 - Yes, it works well
        001.001.002.001.005.002.002.001.002.001 - input[type=radio]
        001.001.002.001.005.002.002.001.002.002.001 - No, there are issues

<3>Cluster:
        001.001.002.001.006.001.001 - Q.3
        001.001.002.001.006.002.001.001 - How did you hear about Shopify?
        001.001.002.001.006.002.002.001.001.001 - select
        001.001.002.001.006.002.002.001.001.001.001 - option
        001.001.002.001.006.002.002.001.001.001.002 - option
```

FIGURA 9. EXEMPLO DE CLUSTERS DE PERGUNTAS COM COMPONENTES DE FORMULÁRIO.

Os dois pseudocódigos apresentados a seguir são simplificações dos dois principais algoritmos utilizados pelo Crawler para fazer a detecção de questionários em páginas web.

Algorithm 1 Verificador de páginas HTML

```
 1: boolean shouldSave(html: HTML)
 2:     root <- get body of html;
 3:     if ((not text of page has certain words) AND
 4:         (not title of page has certain words)) then
 5:        return false;
 6:     end if
 7:     nodes <- find nodes of root;
 8:     clusters <- group nearby nodes;
 9:     clusters <- group clusters by heuristics;
10:     if hasQuestionnaire( clusters ) then
11:        return true;
12:     else
13:        return false;
14:     end if
15: end shouldSave
```

A ideia básica do Algoritmo 1 é determinar se o link da página HTML passada a ele pelo *web crawler* deve ou não ser salvo no banco de dados. Para tal, ele faz uma verificação no título e no corpo da página, encontra e agrupa os

nodos de texto, imagem e componentes de formulário e chama o Algoritmo 2 para verificar a presença ou não de um questionário nesta página HTML.

A verificação inicial do Algoritmo 1 visa procurar por certas palavras/frases que são comuns em questionários online, como: questionário, *survey*, *questionnaire*, etc. Os agrupamentos dos nodos são feitos inicialmente por proximidade e em seguida utilizando certas heurísticas que podem ser notadas após o primeiro agrupamento, como unir *clusters* de inputs em sequência e *clusters* de texto seguidos de *clusters* de componentes.

Algorithm 2 Verificador de questionários

```
 1: boolean hasQuestionnaire(clusters: List)
 2:     cCout <- 0.0; # componentCout
 3:     qCount <- 0; # questionCount
 4:     for each cluster in clusters do
 5:        cCout <- count the number of components in cluster;
 6:        if (is starting a new questionnaire) then
 7:           qCount = 0;
 8:        end if
 9:        if (cCout >= 1.0) then
10:           if (cCout >= MIN_COMPS_IN_ONE_CLUSTER) then
11:              return true;
12:           else
13:              qCount = qCount + 1;
14:           end if
15:           if (qCout == MIN_CLUSTERS_WITH_COMP) then
16:               return true;
17:           end if
18:        end if
19:     end for
20:     return false;
21: end hasQuestionnaire
```

Para determinar se uma página possui um questionário, o Algoritmo 2 percorre os *clusters* passados a ele, verificando a quantidade de componentes de formulário que cada um deles possui e contando a quantidade de *clusters* que possuam no mínimo um componente. Caso a quantidade de componentes em um *cluster* seja maior ou igual a um certo valor ou caso seja encontrado uma

certa quantidade de *clusters* com pelo menos um componente, então o algoritmo retorna *true*, indicando a possível presença de um questionário nesta página.

A verificação da quantidade de componentes em um *cluster* é necessária pois certos questionários são feitos de forma tão simples que todo o questionário é agrupado por proximidade em um mesmo *cluster*. Já a checagem da linha 6 é utilizada para verificar se o possível questionário atual sendo analisado já chegou ao fim. Tal verificação leva em consideração a distância entre os *clusters* e a quantidade de *clusters* que possuem apenas texto e/ou imagens entre *clusters* que possuem componentes de formulário.

O cálculo da quantidade de componentes em um *cluster* da linha 5 leva em conta alguns detalhes:

- Para que um componente seja adicionado a esta contagem, ele precisa ter pelo menos um nodo de texto acima dele que possua todas as características de uma descrição de pergunta apresentadas na Heurística 1. Isto ajuda a eliminar componentes soltos pelas páginas HTML que são usados para outros propósitos.
- Componentes de RADIO INPUT e CHECKBOX que possuam todas as características mencionadas na Heurística 1 contam como 0.75. Caso eles apenas possuam um nodo de texto acima que inicie com letra maiúscula ou número eles contam como 0.25. Todos os outros tipos de componentes precisam ter todas as características da Heurística 1 para contar como 1.0.
- *Clusters* que possuam as palavras/frases mencionadas na Heurística 3 são ignorados desta contagem com o intuito de conseguir uma maior precisão na detecção de questionários.

A Figura 10 apresenta um exemplo deste cálculo. O primeiro INPUT do primeiro *cluster* conta como 0.75 pois o texto acima dele possui todas as características citadas na Heurística 1. Já o segundo INPUT desse mesmo *cluster* conta como 0.25 pois o nodo de texto acima dele apenas começa com letra maiúscula e não possui espaço e nem termina com os caracteres ‘:’, ‘?’ ou ‘.’. Logo, somando os dois valores dos INPUT’s, o primeiro *cluster* fica com o valor final de 1.0. O mesmo ocorre para o segundo *cluster*. No caso do terceiro *cluster*, o SELECT já conta como 1.0 pois ele não é um RADIO ou CHECKBOX INPUT e o texto acima dele possui as características da Heurística 1.

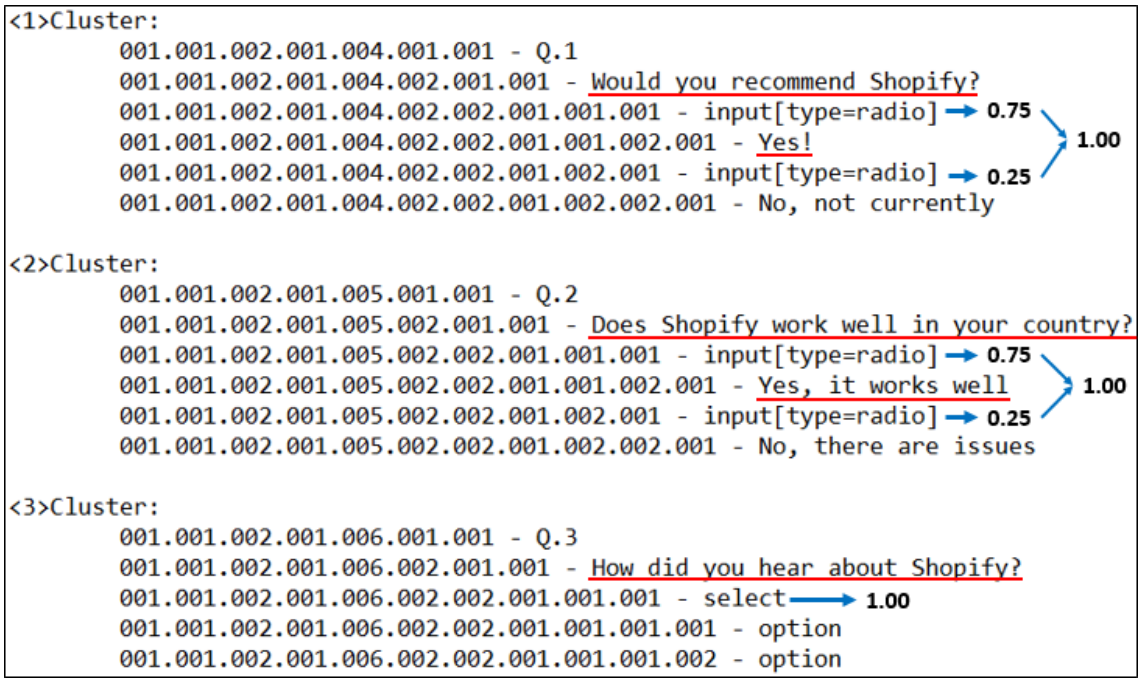

FIGURA 10. EXEMPLO DO CÁLCULO DE COMPONENTES DE FORMULÁRIO EM *CLUSTERS*.

4.5 Extração dos questionários

O algoritmo de extração de questionários utiliza as heurísticas apresentadas na seção 4.3 e adiciona uma nova heurística, que pode parecer meio obvia, mas foi de grande importância para o desenvolvimento deste trabalho.

Heurística 6. Todas as formas de perguntas seguem um ou mais padrões diferentes em suas estruturações nas páginas HTML e é possível utilizar tais padrões para fazer a extração das mesmas.

Durante a fase de análise dos sites com questionários, notou-se alguns padrões distintos para cada forma diferente de pergunta presente nesses sites. Por exemplo, perguntas de INPUT normalmente seguem o padrão: Descrição da pergunta seguida do INPUT, como é possível ver na Figura 11(a). Já perguntas de SELECT seguem o padrão: Descrição da pergunta → SELECT → OPTION → texto da opção 1 → OPTION → texto da opção 2 → etc, como apresentado na Figura 11(b).

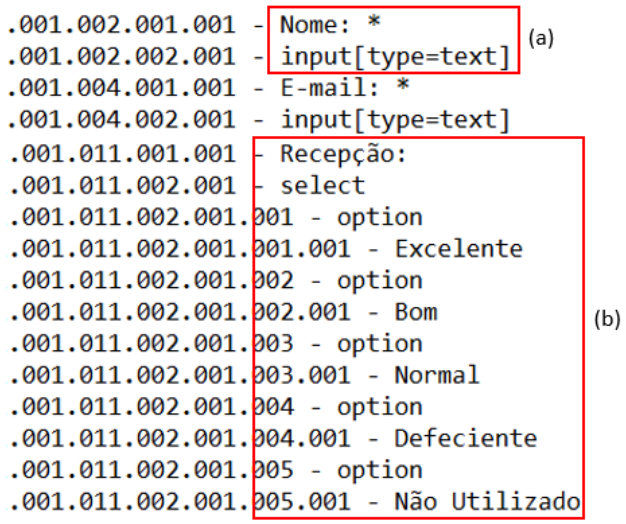

FIGURA 11. EXEMPLOS DOS PADRÕES DE ESTRUTURAÇÃO DE PERGUNTAS.

Os Algoritmos 3 e 4 demonstram simplificações dos principais algoritmos responsáveis por fazer a extração dos questionários.

Algorithm 3 Construtor de questionários

```
1: List buildQuestionnaires(html : HTML)
2:     root <- get body of html;
3:     nodes <- find nodes of root;
4:     stack <- create new Stack;
5:     cluster <- create new Cluster;
6:     questionnaire <- create new Questionnaire;
7:     questionnaires <- create new List of questionnaires;
```

```
 8:
 9:    for each node in nodes do
10:       if (node is a text OR node is an image) then
11:          if (should create a new cluster) then
12:             stack.add( cluster );
13:             cluster <- create new cluster;
14:          end if
15:          if (is starting a new questionnaire) then
16:             if (is a valid questionnaire) then
17:                questionnaires <- questionnaire;
18:             end if
19:             questionnaire <- create new questionnaire;
20:          end if
21:          cluster.add( node );
22:       else if (node is a component) then
23:          stack.add( cluster );
24:          questionnaire.addQuestion( buildQuestion( ... ) );
25:          cluster <- create new cluster;
26:       end if
27:    end for
28:    if (is a valid questionnaire) then
29:       questionnaires <- questionnaire;
30:    end if
31:    return questionnaires;
32: end buildQuestionnaires
```

A ideia básica do Algoritmo 3 é ir agrupando nodos de texto e imagem que estiverem próximos uns dos outros até que se encontre um nodo de componente aonde, neste ponto, o algoritmo então tenta fazer a extração da pergunta referente a este componente. A Figura 1 representa um bom exemplo para o melhor entendimento dos passos descritos acima. O algoritmo irá criar um cluster para o texto “Titulo Questionário”, um outro para “1. Descrição da pergunta” e então encontrará um CHECKBOX, aonde tentará fazer a extração do mesmo.

A verificação da linha 15 é parecida com a feita pelo *Crawler*, ou seja, ela leva em conta a distância entre os *clusters* e a quantidade de *clusters* de texto/imagem entre *clusters* com componentes. Já a verificação das linhas 16 e 28 é utilizada para tentar eliminar os formulários e campos soltos pela página. Ela verifica se o questionário possui uma quantidade mínima de perguntas e

analisa o assunto do questionário e a descrição de suas perguntas em busca das palavras/frases apresentadas na Heurística 3.

Algorithm 4 Construtor de perguntas

```
 1: Question buildQuestion(questionnaire : Questionnaire, nodes :
List, stack : Stack)
 2:    question <- create new Question;
 3:    if (have components in sequence) then
 4:      do the extraction depending on the form of the question;
 5:    else
 6:       do the extraction depending on the type of component;
 7:    end if
 8:    if (question was extracted) then
 9:       update the description of the question;
10:       check if it is a matrix;
11:       check if it is a question with subquestions;
12:       check if the question has an associated image;
13:       if (questionnaire doesn't have a subject) then
14:          check if last cluster is the description of a group;
15:          check if questionnaire has an associated image;
16:          find the subject of the questionnaire;
17:       else
18:          check if last cluster is the description of a group;
19:       end if
20:    end if
21:    return question;
22: end buildQuestion
```

O Algoritmo 4 é responsável por fazer a extração em si das perguntas do questionário e, além disso, ele também deve encontrar outros elementos importantes, como o assunto do questionário, imagens relacionadas ao mesmo ou a pergunta atual sendo extraída, as descrições dos grupos de perguntas e entre outras informações. A parte de extração das perguntas é feita utilizando os padrões descritos na Heurística 6.

Um último ponto que é necessário mencionar é que os *clusters* criados no Algoritmo 3 vão sendo adicionados em uma pilha (*stack*) e são retirados em ordem sempre que for preciso encontrar alguma informação acima do ponto atual de extração, por exemplo, na linha 9 do Algoritmo 4, é necessário encontrar a

descrição da pergunta e para isso o algoritmo irá remover o primeiro elemento da *stack* e fazer alguns testes nele para verificar se é possível afirmar que ele é realmente a descrição desta pergunta.

4.6 Implementação

Todo o trabalho foi desenvolvido utilizando a IDE (*Integrated Development Environment*) Eclipse em sua versão Neon (ECLIPSE, 2017) e a linguagem de programação JAVA da Oracle em conjunto com as seguintes bibliotecas externas: Crawler4j[3], que possui todas as funções que um *crawler* deve implementar; Jsoup[4], para fazer a manipulação da árvore DOM do HTML; e Json[5], para lidar com arquivos no formato *Json*.

Este trabalho foi dividido em três subprojetos (*Common Lib*, *Crawler* e *Extractor*) com o intuito de diminuir a duplicidade de código. A seguir são detalhadas as classes e pacotes utilizados em cada um destes projetos.

4.6.1 Common Lib

Este módulo foi criado pois notou-se que ambos, *Crawler* e *Extractor*, dividiriam muitas funcionalidades em comum e isso geraria código duplicado nos mesmos. Ele possui classes para lidar com a conexão com o banco de dados, manipulação do arquivo de configuração, classes de utilidade e entre outras.

[3] https://github.com/yasserg/crawler4j
[4] https://jsoup.org/
[5] https://github.com/stleary/JSON-java

4.6.1.1 Arquitetura da Common Lib

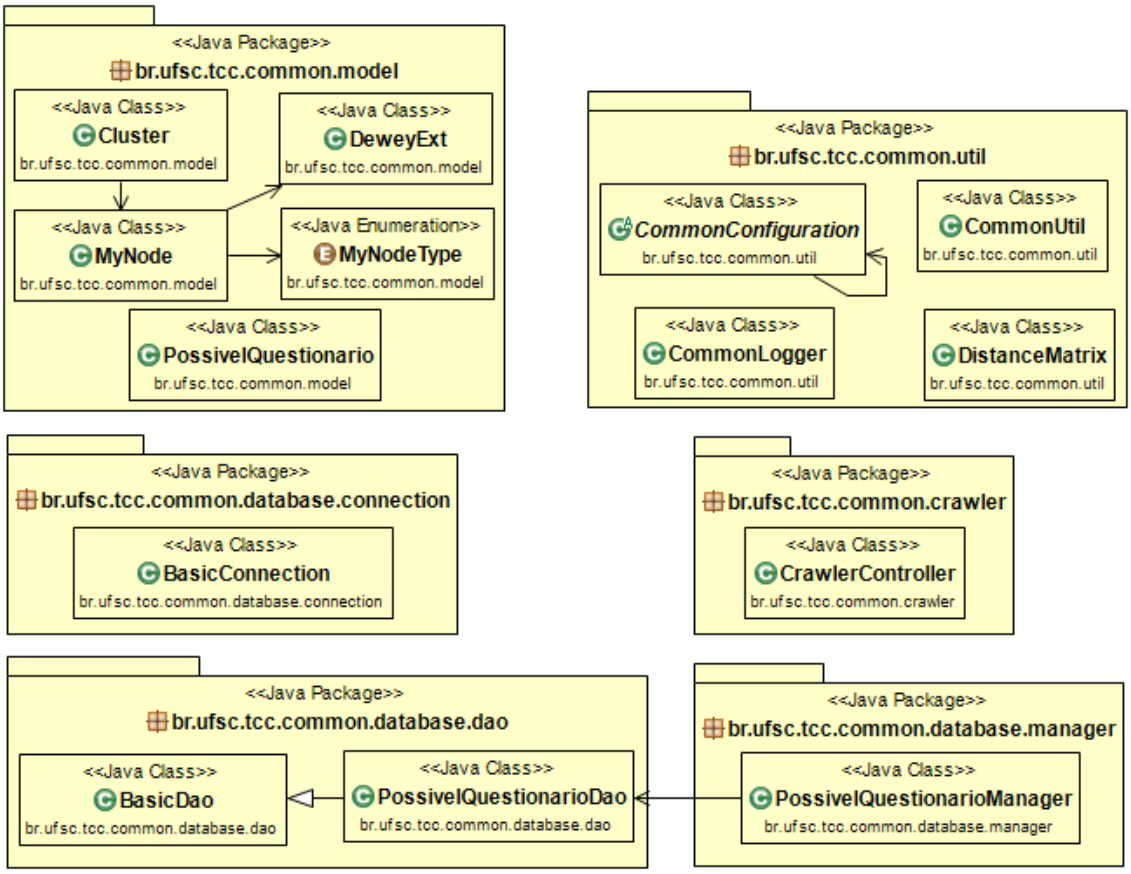

FIGURA 12. ARQUITETURA DA COMMON LIB.

A seguir é realizada uma breve descrição de cada pacote presente na *Common Lib*:

- **Pacote *Crawler:*** A classe *CrawlerController* contida neste pacote serve como um *wrapper* sobre a classe *CrawlController* fornecida pela biblioteca *Crawler4j*. Esta classe é responsável por guardar as configurações do *crawler* e iniciar as operações do mesmo.
- **Pacote *Util:*** Pacote que contém as classes: *CommonLogger*, que cuida do sistema de *log* do trabalho; *CommonUtil*, que possui vários métodos estáticos que são utilizados por todos os projetos; *DistanceMatrix*, responsável por armazenar e lidar com as distancias entre nodos; e

CommonConfiguration, que é uma classe básica para lidar com o arquivo de configuração.

- **Pacote *Model:*** Possui classes que representam alguns elementos utilizados no trabalho, como a classe *DeweyExt*, responsável por fornecer todas as operações referentes a Dewey-Ext, e a classe *MyNode*, que possui as características que representam um nodo da árvore DOM.
- **Pacote *Connection:*** A classe existente neste pacote lida com a conexão com o banco de dados.
- **Pacote *Dao:*** Engloba as classes que lidam com a comunicação com o banco de dados e oferecem métodos para salvar, atualizar, deletar e encontrar dados no mesmo.
- **Pacote *Manager:*** A classe contida neste pacote serve como um ponto intermediário entre o sistema e os DAOs. Ela possui métodos de mais alto nível para lidar com o banco de dados fazendo uso dos DAOs.

4.6.2 Crawler

Este módulo foi desenvolvido utilizando a biblioteca externa *Crawler4j* que já implementa todas as operações que um crawler deve possuir, possibilitando assim um foco maior na detecção dos questionários online. Abaixo são apresentadas algumas das configurações especificas do *Crawler* que podem ser modificadas no arquivo de configuração do mesmo:

- **excludedFilesExtensions**: Configuração que indica quais extensões de arquivos não devem ser baixadas pelo *Crawler*, como .js para arquivos Javascript e .css para arquivos CSS.
- **excludedDomains**: Domínios que devem ser ignorados pelo *Crawler*.

- **excludedLanguages**: Expressão regular com os domínios de topo de código de países (*country-code top-level domains* ou ccTLD) e abreviações de línguas que são ignoradas pelo *Crawler*. Este parâmetro foi criado pois pode não ser interessante vasculhar um mesmo site em diversas línguas diferentes. Para tentar determinar a linguagem utilizada em um site, são extraídas 3 informações da URL do mesmo: o início do seu subdomínio (**us**.xxxx.com), o final do seu domínio (xxxx.**us**) e o início do seu caminho (xxxx.com/**en**), se algum desses valores for reconhecido pela expressão regular deste parâmetro, então está URL será ignorada pelo *Crawler*.
- **surveyWordsRegex**: Palavras que um site deve possuir para indicar a possível presença de um questionário no mesmo, como: questionário, survey, questionnaire e afins.
- **distBetweenNearQuestions**: Altura máxima entre *clusters* em sequência. Caso o algoritmo do *Crawler* já tenha achado algum *cluster* com componente e ele então encontre um *cluster* muito distante do outro, ele considerara que os componentes encontrados até o momento não pertencem a um questionário e zerara o contador de questões.
- **minCompsInOneCluster**: Número mínimo de componentes que um único *cluster* deve possuir para que o algoritmo do *Crawler* considera a existência de um questionário.
- **minClustersWithComp**: Número mínimo de *clusters* com pelo menos um componente para que o *Crawler* considere a existência de um questionário.

- **maxClustersBetweenClustersWithComp**: Número máximo de *clusters* que possuam apenas texto/imagem entre *clusters* que possuem um ou mais componentes. Caso o número encontrado, em sequência, seja maior que o valor deste parâmetro o algoritmo considerara que os componentes encontrados até o momento não pertencem a um questionário e zerara o contador de questões.

4.6.2.1 Arquitetura do Crawler

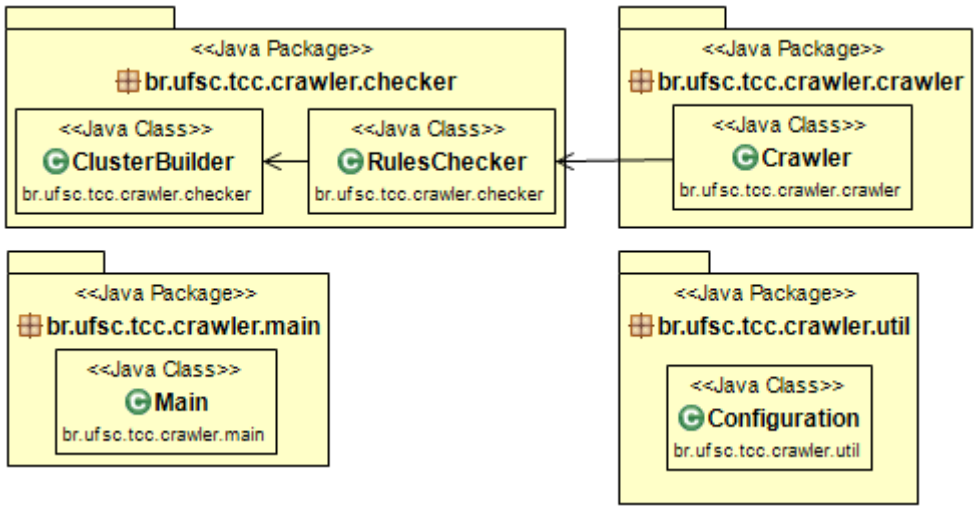

FIGURA 13. ARQUITETURA DO CRAWLER.

A seguir é realizada uma breve descrição de cada pacote presente no *Crawler*:

- **Pacote *Main:*** Possui apenas uma classe responsável por inicializar a aplicação.
- **Pacote *Checker:*** Contém a classe *RulesChecker*, encarregada de fazer todas as verificações nas páginas HTML para determinar a presença, ou não, de questionários nas mesmas, e a classe *ClusterBuilder*, responsável por construir os *clusters* utilizados pelo *RulesChecker* para fazer suas verificações.

- **Pacote *Crawler:*** A classe *Crawler* contida neste pacote faz uso da biblioteca externa *Crawler4j* para funcionar como um *crawler*, percorrendo a Web a partir das *seeds* iniciais e passando as páginas encontradas para o *RulesChecker* para verificação.
- **Pacote *Util:*** Possui apenas uma classe, *Configuration*, que estende a classe *CommonConfiguration* da *Common Lib* para lidar com o arquivo de configuração do *Crawler*.

4.6.3 Extractor

O extrator foi a parte mais desafiadora de todo o trabalho e a principal razão para tal é a falta de padronização do HTML. Uma mesma página pode ser escrita de diversas maneiras diferentes e ainda assim possuir as mesmas funcionalidades e representação visual. Isto gera uma grande dificuldade quando se quer extrair os dados de diversos sites diferentes de forma genérica. Uma outra dificuldade encontrada foi a de lidar com a quantidade enorme de formas diferentes de se fazer uma pergunta, como utilizando INPUT's, SELECT's, TEXTAREA's, matrizes, grupo de perguntas, imagens e afins. Logo, como mencionado na seção 4.3 e 4.5, foram criadas e testadas diversas heurísticas para tentar lidar com todos os aspectos dos questionários. Tais heurísticas utilizam parâmetros configuráveis que são explicados abaixo e podem ser modificadas no arquivo de configuração:

- **minQuestionsOnQuestionnaire**: Quantidade mínima de perguntas para que seja considerado um questionário. Isto é usado para se eliminar componentes isolados na página.

- **maxWordsInAGroupDescription**: Quantidade máxima de palavras que a descrição de um grupo pode ter. Isto é utilizado para tentar evitar que trechos de texto avulsos dentro de questionários sejam considerados como descrições de grupos.
- **maxTextClustersBetweenQuestions**: Número máximo de *clusters* que possuam apenas texto/imagem entre questões. Caso o número encontrado, em sequência, seja maior que o valor deste parâmetro o algoritmo considerara que o questionário atual acabou e iniciará a extração de um novo.
- **distBetweenTextsInsideQuestionnaire**: Altura máxima que os textos de um questionário podem estar. Se a extração de um questionário já tiver começado e o algoritmo encontrar um texto que esteja muito distante do último texto encontrado, isso indica que, provavelmente, o questionário que estava sendo extraído já terminou.
- **distBetweenCompAndText**: Se refere à altura e altura máxima que um texto pode estar do seu componente, por exemplo, o texto de uma alternativa em relação ao CHECKBOX que ele representa.
- **distBetweenDescAndQuestion**: Se refere à altura e altura máxima que a descrição da pergunta pode estar das suas alternativas/componentes. Este parâmetro foi criado pois existem sites que colocam INPUT's soltos pelo meio do questionário para serem utilizados por algoritmos JavaScript, mas que não fazem parte em si do questionário.
- **distBetweenGroupAndFirstQuestion**: Trata da altura e largura máxima entre o título de um grupo de perguntas para a sua primeira pergunta. No

caso da Figura 14, se refere a distância entre o texto “Caracterização da organização” e o texto “NOME DA EMPRESA”.

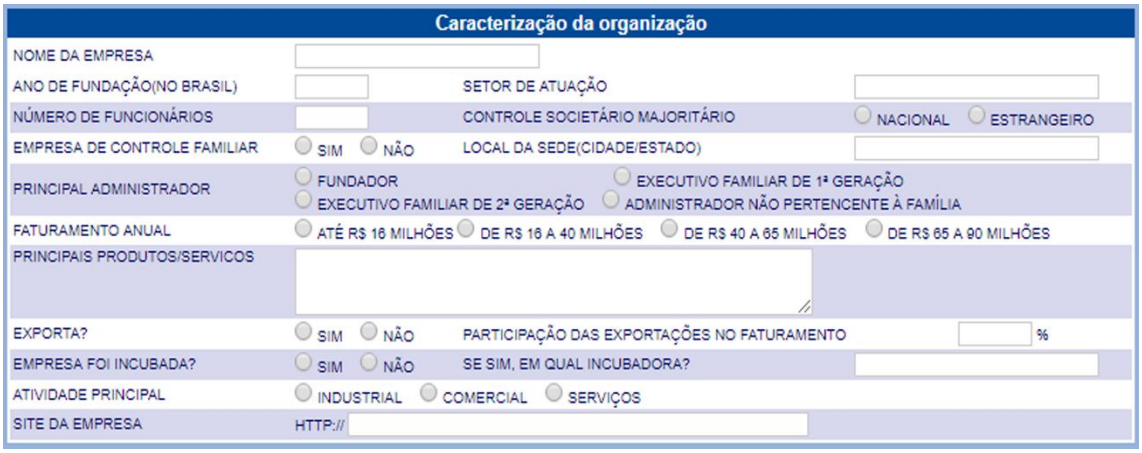

FIGURA 14. EXEMPLO DE UM GRUPO DE PERGUNTAS.[6]

- **distBetweenDescAndComplementaryText**: Altura, altura máxima e largura máxima entre um componente de formulário e um possível texto que o segue. Se refere a casos, como o da Figura 15, onde se tem a descrição da pergunta, seguida do componente de formulário e de um texto complementar.

FIGURA 15. EXEMPLO DE PERGUNTAS COM TEXTOS COMPLEMENTARES.[7]

- **distBetweenTextsInQuestionWithSubQuestions**: Trata da altura e largura máxima entre os textos de uma pergunta com subperguntas.
- **distBetweenPartsOfDescription**: Define a altura, altura máxima e largura máxima entre partes da descrição de uma pergunta. Tal configuração foi criada para lidar com as formatações visuais dos textos no HTML, exemplificado na Figura 16. Dado o uso da *tag* STRONG, a

[6] http://anpei.tempsite.ws/intranet/mediaempresa/
[7] https://www.bioinfo.mpg.de/mctq/core_work_life/core/core.jsp?language=por_b

descrição da pergunta da Figura 16 será separada em três nodos diferentes na árvore de elementos HTML e, por isso, eles também receberão três IDs diferentes. Logo, durante a extração é necessário juntar esses três pedaços de texto em um só para que a descrição da pergunta fique completa.


```
<div class="form-section" id="close-acc1">
    <div class="field-group clearfix">
        <label style="margin-bottom: 0px;">
            2. Did using <strong>Telstra.com</strong>
            save you from having to call us or visit a store?
        </label>
        <label class="help" style="color:#999">
            <em>(Select one or more options)</em>
        </label>
        <div class="radio-group clearfix">
            ...
        </div>
    </div>
</div>
```

FIGURA 16. EXEMPLO DE FORMATAÇÃO VISUAL NO HTML.[8]

- **distBetweenTextsOfSameAlternative**: Trata da altura, altura máxima e largura máxima entre os textos de uma mesma alternativa de CHECKBOX ou RADIO INPUT, por exemplo. Serve basicamente para o mesmo propósito do parâmetro do item anterior mais com valores um pouco diferentes.
- **distBetweenHeaderAndFirstAlternative:** Altura e largura entre o '*header*' de uma 'matriz' e a sua primeira alternativa. Neste caso, este parâmetro é utilizado para perguntas que seguem o quarto padrão de

[8] https://www.telstra.com.au/webforms/consumer-survey/index.cfm

perguntas da forma RADIO/CHECKBOX INPUT e o segundo padrão de Matrizes, que serão explicados mais à frente no texto.

- **distBetweenEvaluationLevelsAndDesc, maxSpacesAndNewLinesInEvaluationLevels, evaluationLevelsWordsRegex:** Se referem à altura, o número máximo de espaços e quebra de linhas e a palavras encontradas em textos de níveis de avaliação. Estes três parâmetros são utilizados em conjunto para tentar lidar com casos como os da Figura 17 onde existem níveis de avaliação em cima das opções de perguntas de RADIO INPUT ou matrizes.

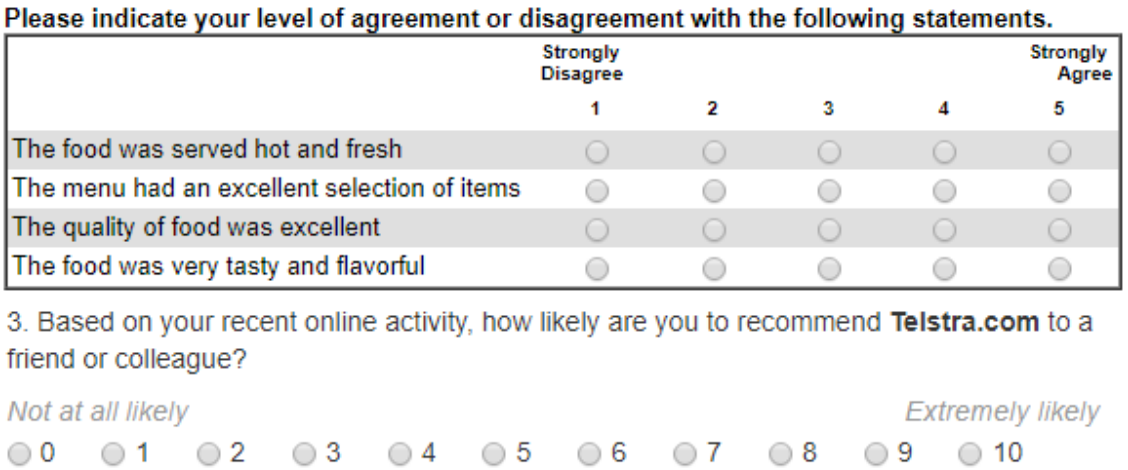

FIGURA 17. EXEMPLOS DE PERGUNTAS COM NÍVEIS DE AVALIAÇÃO.[9]

Como mencionado na seção 4.5, as perguntas de um questionário seguem certos padrões que foram observados durante a fase de análise. A seguir são apresentados todos os padrões que podem ser reconhecidos pelo *Extractor*, o que possibilita um melhor entendimento do escopo e dos limites do mesmo.

[9] https://statpac.com/online-surveys/resturaunt_customer_satisfaction_survey.htm e https://www.telstra.com.au/webforms/consumer-survey/index.cfm

- **TEXT**, **NUMBER**, **EMAIL**, **DATE**, **TEL**, **TIME** e **URL INPUTs**: Seguem o padrão: Descrição da pergunta -> INPUT.
- **TEXTAREA:** Segue o padrão: Descrição da pergunta -> TEXTAREA.
- **SELECT**: Segue o padrão: Descrição da pergunta -> SELECT -> OPTION -> texto da opção -> OPTION -> texto da opção -> etc.
- **RADIO** e **CHECKBOX INPUTs**: Foram observados 4 tipos diferentes de padrões de perguntas utilizando estes componentes. O primeiro é o padrão mais comum: Descrição da pergunta -> INPUT -> texto do input -> INPUT -> texto do input -> etc. O segundo é parecido com o primeiro, mas invertendo a ordem dos inputs e textos: Descrição da pergunta -> texto do input -> INPUT -> texto do input -> INPUT -> etc. O terceiro faz uso apenas de imagens, sem texto: Descrição da pergunta -> INPUT -> imagem -> INPUT -> imagem -> etc. E o último utiliza uma estrutura parecida com uma matriz: Descrição da pergunta -> todos os textos dos inputs em sequência -> todos os INPUT's em sequência. Além disso, nos dois primeiros tipos é possível ter imagens entre as alternativas e apenas no primeiro tipo também é possível ter um TEXT INPUT ou TEXTAREA no final que é associado a última alternativa da pergunta.
- **RATING:** São perguntas aonde você tem que dar uma nota entre 0 e X para alguma coisa. Elas seguem o padrão: Descrição da pergunta -> um número X de RADIO INPUT's em sequência.
- **Perguntas com vários componentes:** São perguntas que possuem vários componentes em sequência, como perguntas que solicitam mais de uma resposta do usuário. Seguem o padrão: Descrição da pergunta -

> um número X de componentes em sequência (não precisa ser o mesmo tipo de componente de formulário).

- **Perguntas com subperguntas**: Seguem o padrão: Descrição da pergunta -> subpergunta 1 -> subpergunta 2 -> etc. Cada subpergunta deste padrão pode ser de uma das formas apresentadas acima.
- **Matrizes**: As perguntas com matrizes seguem o padrão: Descrição da pergunta -> cabeçalho da matriz -> subperguntas da matriz. Mas como pode ser visto na Figura 18, a forma que as subperguntas são feitas no código HTML pode diferenciar. No caso da Figura 18(a) os textos do cabeçalho também aparecem nas subperguntas, enquanto no caso da Figura 18(b) as subperguntas só possuem o mesmo número de INPUTs que a quantidade de textos no cabeçalho. As matrizes não necessariamente precisam ser formadas de apenas RADIO INPUTs.

```
Se o preço do "produto" fosse maior/menor, quantas
unidades a mais/menos você compraria?

    4 ou mais a menos
    1-3 a menos
    1-3 a mais
    4 ou mais a mais

    +20%
        input[type=radio]
        4 ou mais a menos
        input[type=radio]
        1-3 a menos
        input[type=radio]
        1-3 a mais
        input[type=radio]
        4 ou mais a mais

    +10%
        input[type=radio]
        4 ou mais a menos
        input[type=radio]
        1-3 a menos
        input[type=radio]
        1-3 a mais
        input[type=radio]
        4 ou mais a mais
```

```
How does this product compare with other similar
products in the market?

    Better
    Similar
    Worse

    Quality
        input[type=radio]
        input[type=radio]
        input[type=radio]

    Price
        input[type=radio]
        input[type=radio]
        input[type=radio]
```

FIGURA 18. EXEMPLOS DE PADRÕES DE PERGUNTAS DE MATRIZ.

4.6.3.1 Modelo banco de dados

Como o objetivo deste trabalho é salvar os dados dos questionários online em um banco de dados, o banco de dados em si é uma parte essencial do mesmo. A Figura 19 apresenta o esquema do banco de dados utilizado pelo *Extractor*.

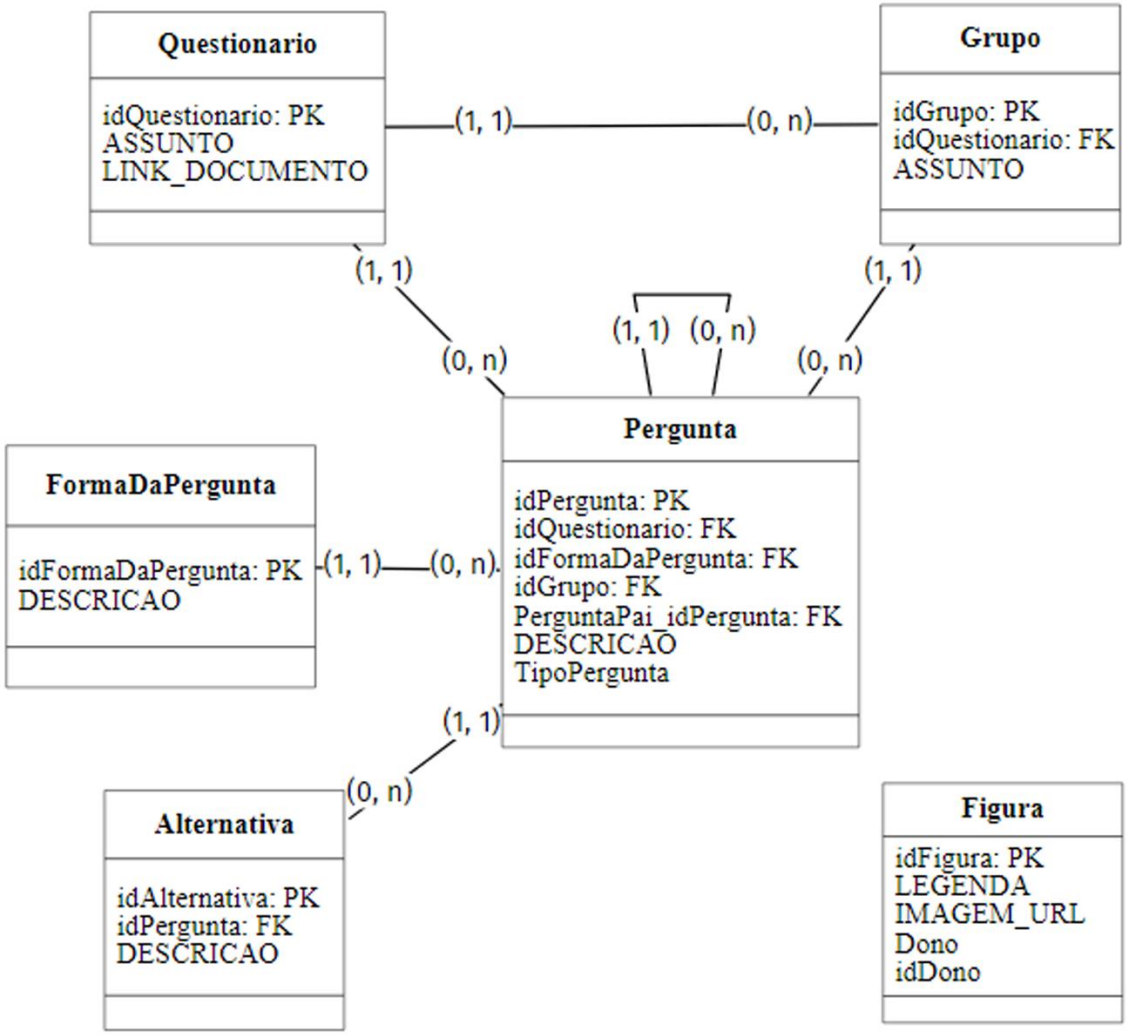

FIGURA 19. ESQUEMA DO BANCO DE DADOS.

Este modelo foi desenvolvido para ser usado em (SOUZA; DORNELES, 2017) e alguns pontos que devem ser notados sobre o mesmo são apresentados abaixo:

- A tabela *FormaDaPergunta* guarda todas as diferentes formas de se fazer uma pergunta em um questionário, como: CHECKBOX INPUT, RADIO INPUT MATRIX, TEXT INPUT GROUP e entre outros.
- A tabela *Figura* guarda as informações de imagens encontradas no questionário. Tais imagens podem pertencer ao questionário, a uma pergunta ou a uma alternativa, campos *Dono* e *idDono*.
- Uma pergunta pode ter uma ou mais **perguntas filhas**. Isto é usado para casos como matrizes e perguntas com subperguntas.
- Uma pergunta pode ser do tipo ABERTO, FECHADO ou MULTIPLA ESCOLHA, dependendo da sua forma.

4.6.3.2 Arquitetura do Extractor

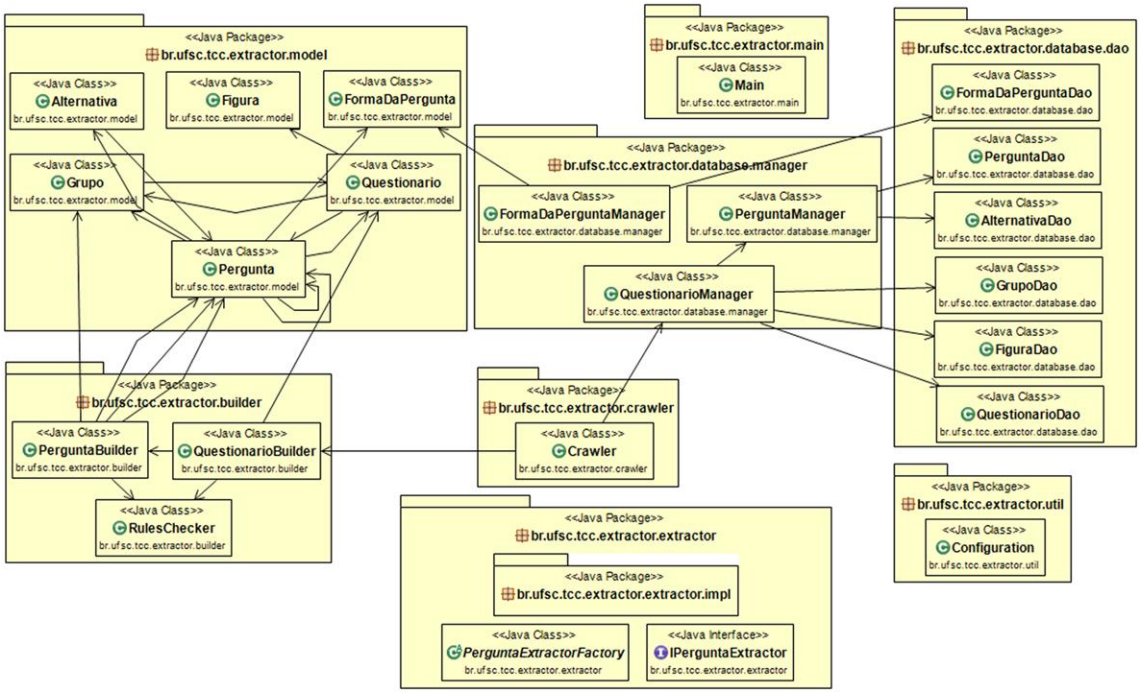

FIGURA 20. ARQUITETURA DO EXTRACTOR.

A seguir é realizada uma breve descrição de cada pacote presente no *Extractor*:

- **Pacote *Main:*** Possui apenas uma classe responsável por inicializar a aplicação.
- **Pacote *Model:*** Pacote que engloba classes que representam elementos encontrados em um questionário, como o próprio questionário, as suas perguntas e as alternativas destas perguntas.
- **Pacote *Dao:*** Contém especializações da classe *BasicDao*, encontrada no pacote *Dao* da *Common Lib*, para cada elemento do modelo da aplicação.
- **Pacote *Manager:*** As classes contidas neste pacote servem como um ponto intermediário entre o sistema e os DAOs. Elas possuem métodos de mais alto nível para lidar com o banco de dados fazendo uso dos DAOs.
- **Pacote *Builder:*** As classes presentes neste pacote são encarregadas de orquestrar toda a parte de extração, percorrendo os nodos da arvore HTML, fazendo verificações nos mesmos e os passando, quando necessário, para os extratores em si.
- **Pacote Extractor:** Possui uma interface que é implementada pelos extratores do projeto e uma classe abstrata que serve como um criador de extratores.
- **Pacote *Extractor.impl:*** Possui diversos extratores, omitidos da Figura 20 para simplificação, que são responsáveis por fazer a extração das perguntas dos questionários utilizando os padrões descritos na seção 4.6.3.
- **Pacote *Crawler:*** A classe *Crawler* contida neste pacote faz uso da biblioteca externa *Crawler4j* para funcionar como um *crawler*,

percorrendo a Web a partir das seeds iniciais e passando as páginas encontradas para o *QuestionarioBuilder* para a fase de extração.

- **Pacote Util:** Possui apenas uma classe, *Configuration*, que estende a classe *CommonConfiguration* da *Common Lib* para lidar com o arquivo de configuração do Extractor.

5 Experimentos

Neste capitulo são apresentados os experimentos conduzidos com o intuito de avaliar a qualidade das abordagens utilizadas para a solução dos problemas apresentados. Existem dois pontos principais que devem ser verificados:

- a detecção correta de questionários online pelo *Crawler*; e
- a extração correta dos dados presentes nos questionários online pelo *Extractor*.

5.1 Base de dados

Para fazer a avaliação do *Crawler* deixou-se o mesmo rodando por 11 horas e 05 minutos, o que resultou em 2262 links de possíveis questionários encontrados em 32 web sites diferentes.

Os experimentos feitos para avaliar o *Extractor* foram realizados com uma base de dados de 510 questionários, em sua maioria encontrados pelo *Crawler*, que foram coletados de um total de 37 web sites diferentes. Optou-se por utilizar no máximo 25 questionários de um mesmo web site para tentar homogeneizar a base de dados já que alguns sites possuíam centenas de questionários enquanto outros possuem apenas um ou dois.

No geral, os questionários coletados possuem 5765 perguntas no total, com uma média de 11 perguntas por questionário, sendo que o menor questionário possui 2 perguntas e o maior 53 perguntas. Deste total, 4161 são perguntas **fechadas**, ou seja, perguntas que possuem alternativas fixas onde o usuário tem que escolher exatamente uma delas, 1351 são de perguntas **abertas**, em que os usuários são livres para informar as respostas que

desejarem, e 254 são de perguntas de **múltipla escolha**, onde os usuários podem escolher entre uma ou mais das alternativas presentes.

Os questionários utilizados nos experimentos foram classificados em 8 domínios de pesquisa diferentes que são apresentados na Figura 21.

Domínio	Quantidade	% do Total
Avaliação/Satisfação de/com produtos, serviços, etc	214	41,96%
Outros assuntos diversos	88	17,25%
Pesquisa de Mercado, Negócios e Marketing	76	14,90%
Recursos Humanos e ambiente empresarial	47	9,22%
Educação e Treinamento	30	5,88%
Saude e Esportes	22	4,31%
Entreterimento e Eventos	22	4,31%
Comunidade e ONGs	11	2,16%

FIGURA 21. TABELA COM OS DADOS DOS DOMÍNIOS DE PESQUISA.

5.2 Variáveis

Para fazer a avaliação do *Crawler* e do *Extractor* foram utilizadas as seguintes variáveis:

- **paginasCrawler**: Conjunto de páginas encontradas pelo *Crawler*.
- **paginasCrawlerComQuestionario**: Conjunto de páginas encontradas pelo *Crawler* que realmente possuíam um ou mais questionários.
- **perguntasExtractor**: Conjunto de perguntas extraídas pelo *Extractor*.
- **perguntasReal**: Conjunto de perguntas que o questionário online realmente possuía.
- **alternativasExtractor**: Conjunto de alternativas extraídas pelo *Extractor*.

- **alternativasReal**: Conjunto de alternativas que o questionário online realmente possuía.
- **perguntasFilhasExtractor**: Conjunto de perguntas filhas extraídas pelo *Extractor*.
- **perguntasFilhasReal**: Conjunto de perguntas filhas que o questionário online realmente possuía.

5.3 Métricas

Foram usadas métricas de Recuperação da Informação: Precisão, Revocação e Medida-F (RIJSBERGEN, 1979).

$$Precisão PaginasCrawler = \frac{|\ paginasCrawlerComQuestionario\ |}{|\ paginasCrawler\ |}$$

EQUAÇÃO 1 – PRECISÃO DAS PÁGINAS ENCONTRADAS PELO CRAWLER.

$$PrecisãoPerguntas = \frac{|\ perguntasExtractor \cap\ perguntasReal\ |}{|\ perguntasExtractor\ |}$$

EQUAÇÃO 2 – PRECISÃO DAS PERGUNTAS EXTRAÍDAS PELO EXTRACTOR.

$$RevocaçãoPerguntas = \frac{|\ perguntasExtractor \cap\ perguntasReal\ |}{|\ perguntasReal\ |}$$

EQUAÇÃO 3 – REVOCAÇÃO DAS PERGUNTAS EXTRAÍDAS PELO EXTRACTOR.

$$MedidaFPerguntas = \frac{2\ x\ PrecisãoPerguntas\ x\ RevocaçãoPerguntas}{PrecisãoPerguntas + RevocaçãoPerguntas}$$

EQUAÇÃO 4 – MEDIDA-F DAS PERGUNTAS EXTRAÍDAS PELO EXTRACTOR.

$$Precisão Alternativas = \frac{|\ alternativasExtractor \cap\ alternativasReal\ |}{|\ alternativasExtractor\ |}$$

EQUAÇÃO 5 – PRECISÃO DAS ALTERNATIVAS EXTRAÍDAS PELO EXTRACTOR.

$$Revocação Alternativas = \frac{|\ alternativasExtractor \cap\ alternativasReal\ |}{|\ alternativasReal\ |}$$

EQUAÇÃO 6 – REVOCAÇÃO DAS ALTERNATIVAS EXTRAÍDAS PELO EXTRACTOR.

$$MedidaFAlternativas = \frac{2\ x\ Precisão Alternativas\ x\ Revocação Alternativas}{Precisão Alternativas + Revocação Alternativas}$$

EQUAÇÃO 7 – MEDIDA-F DAS ALTERNATIVAS EXTRAÍDAS PELO EXTRACTOR.

$$Precisão PerguntasFilhas = \frac{|\ perguntasFilhasExtractor \cap\ perguntasFilhasReal\ |}{|\ perguntasFilhasExtractor\ |}$$

EQUAÇÃO 8 – PRECISÃO DAS PERGUNTAS FILHAS EXTRAÍDAS PELO EXTRACTOR.

$$Revocação PerguntasFilhas = \frac{|\ perguntasFilhasExtractor \cap\ perguntasFilhasReal\ |}{|\ perguntasFilhasReal\ |}$$

EQUAÇÃO 9 – REVOCAÇÃO DAS PERGUNTAS EXTRAÍDAS PELO EXTRACTOR.

$$MedidaFPerguntasFilhas = \frac{2\ x\ Precisão PerguntasFilhas\ x\ Revocação PerguntasFilhas}{Precisão PerguntasFilhas + Revocação PerguntasFilhas}$$

EQUAÇÃO 10 – MEDIDA-F DAS PERGUNTAS FILHAS EXTRAÍDAS PELO EXTRACTOR.

5.4 Similaridade entre variáveis

Vale ressaltar que, a avaliação de similaridade utilizada nas métricas apresentas acima, levou em consideração apenas que a descrição real das perguntas, alternativas e perguntas filhas deveria estar **contida** na descrição

extraída pelo *Extractor*. Por exemplo, para que uma pergunta extraída seja considerada similar a pergunta real, a descrição da pergunta real deve estar contida na descrição da pergunta extraída.

Escolheu-se utilizar este método de avaliação pois muitos sites deixam textos de informação extra ou de controle escondidos no HTML e, como utilizou-se um mesmo arquivo de configuração para extrair todos os sites, o algoritmo acabou considerando alguns desses textos como parte das descrições dos elementos dos questionários.

5.5 Resultados

Abaixo são apresentados os principais resultados encontrados com os experimentos realizados com o *Crawler* e o *Extractor*.

A precisão do *Crawler* atingiu 94,47%, ou seja, 2137 dos links encontrados pelo *Crawler* realmente possuíam um ou mais questionários neles. Notou-se que outros 5,53% dos links encontrados não possuíam questionários, mas sim formulários muito grandes ou múltiplos formulários pequenos, porém próximos uns dos outros, o que acabou confundindo o algoritmo de detecção de questionários do *Crawler*.

A Figura 22 contém as médias gerais das métricas utilizadas para avaliar a extração de perguntas do *Extractor*. Como pode ser visto, todas as três métricas ficaram acima de 90%, o que indica que a maioria das descrições de perguntas foram extraídas corretamente.

Um fator que vale mencionar neste ponto é que alguns sites possuem um componente de formulário ou uma pergunta escondida pelo meio do questionário que servem como ‘armadilhas’ para pessoas que tentam responder estes

questionários de uma forma automatizada, utilizando *bots* por exemplo. Este tipo de ‘armadilha’ é conhecida como *honeypot captcha* (CHANDAN; THACKER; SAXENA, 2016, p. 6). Dentro da base de dados utilizada para os experimentos foram encontrados 4 sites que utilizam este método, totalizando 84 questionários. Sendo que 2 destes sites, 48 questionários, possuem apenas um componente ‘armadilha’ que não influencia muito nos resultados, já que são simplesmente ignorados pelo *Extractor* por não possuírem descrição, e os outros 2 sites, 36 questionários, possuem uma pergunta ‘armadilha’, que acaba sendo considerada pelo *Extractor* como parte do questionário.

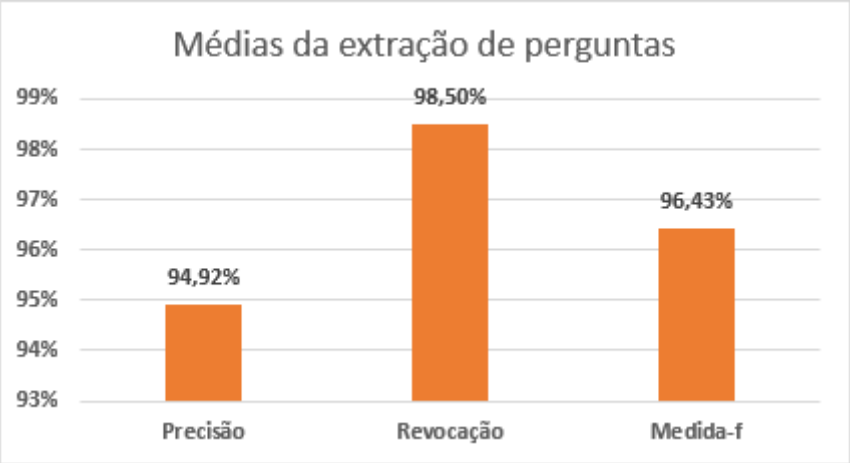

FIGURA 22. RESULTADO GERAL PARA A EXTRAÇÃO DE PERGUNTAS.

A Figura 23 apresenta as médias gerais da extração de alternativas. Neste caso percebe-se que todas as métricas ficaram bem próximas de 100%, o que indica que os padrões de perguntas apresentados no final da seção 4.6.3 foram eficazes para a extração das alternativas.

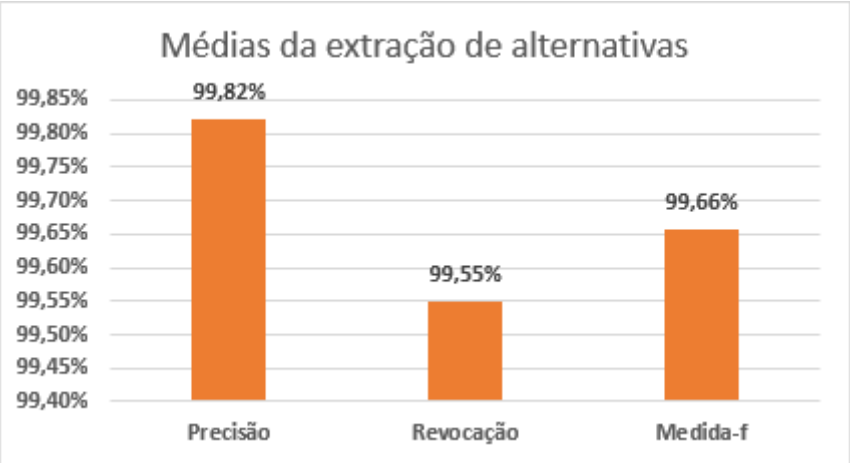

FIGURA 23. RESULTADO GERAL PARA A EXTRAÇÃO DE ALTERNATIVAS.

A Figura 24 exibe os resultados para a extração de perguntas filhas, aonde é possível ver que a precisão chegou a um pouco mais que 92%, a revocação a 90% e a medida-f a 91%. Existem dois motivos principais para estes resultados reduzidos em comparação a extração de perguntas e alternativas:

1. Alguns sites colocam as perguntas filhas no mesmo nível que uma pergunta normal, fazendo com que o *Extractor* se confunda e as considere como perguntas normais;
2. Uma deficiência no reconhecimento da descrição da pergunta pai de uma pergunta com subperguntas faz com que, em alguns casos, o assunto do questionário seja considerado como pergunta pai da(s) primeira(s) pergunta(s) do questionário. Isto ocorre porque, nestes casos específicos, a estrutura do início do questionário fica idêntica a estrutura de uma pergunta com subpergunta.

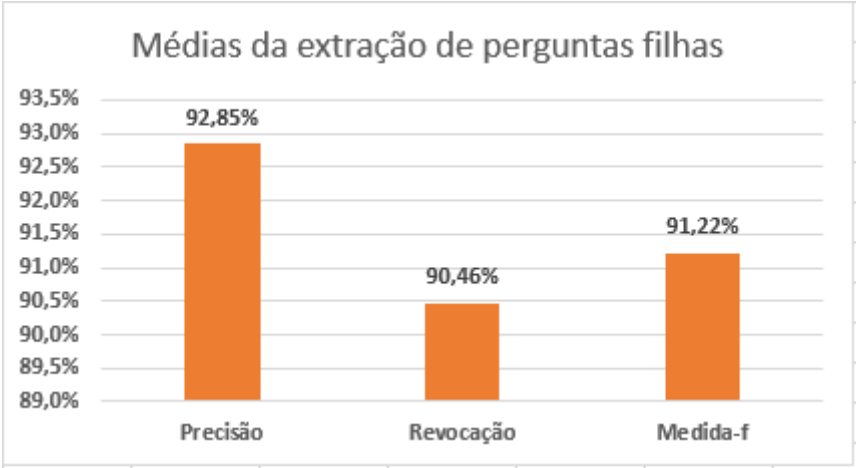

FIGURA 24. RESULTADO GERAL PARA A EXTRAÇÃO DE PERGUNTAS FILHAS.

As Figuras 25, 26 e 27 apresentam os resultados separados por domínios. Apenas olhando para os gráficos não foi possível tirar muitas conclusões, já que os resultados estão variando bastante e também deve-se lembrar que existe uma grande discrepância na quantidade de questionários de cada domínio. Por este motivo, decidiu-se analisar apenas os dados da extração de perguntas filhas, tendo em vista que os mesmos possuem a maior variação nos resultados.

Com essa mudança na perspectiva de analise notou-se que os questionários do domínio 'Outros' possuem a maior quantidade de casos onde a precisão ou a revocação resultaram em uma divisão de 0/0 (vide as equações da seção 5.3). Nesses casos, existia uma ou mais perguntas filha para serem extraídas, mas o *Extractor* não extraiu nenhuma ou não existia pergunta filha para ser extraída, mas foram extraídas uma ou mais perguntas filhas, resultando em um valor 0 para precisão, revocação e medida-f.

Percebeu-se também que o domínio com a menor quantidade destes casos foi o de 'RH', o que pode ser uma indicação dos resultados apresentados pela Figura 27.

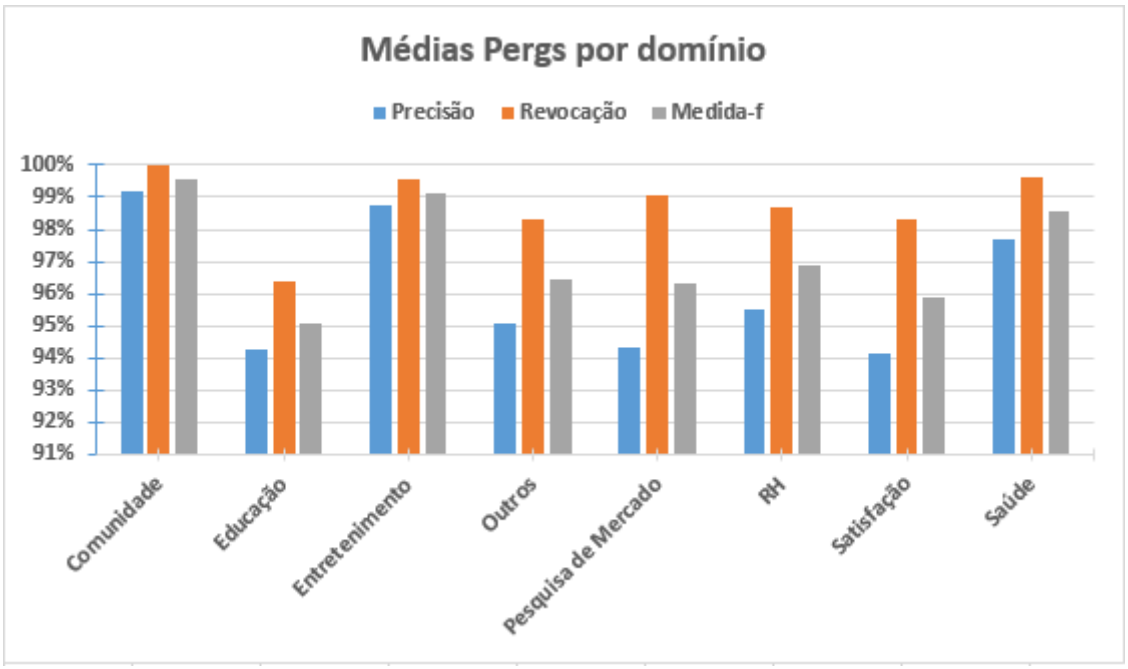

FIGURA 25. RESULTADO POR DOMÍNIO DA EXTRAÇÃO DE PERGUNTAS.

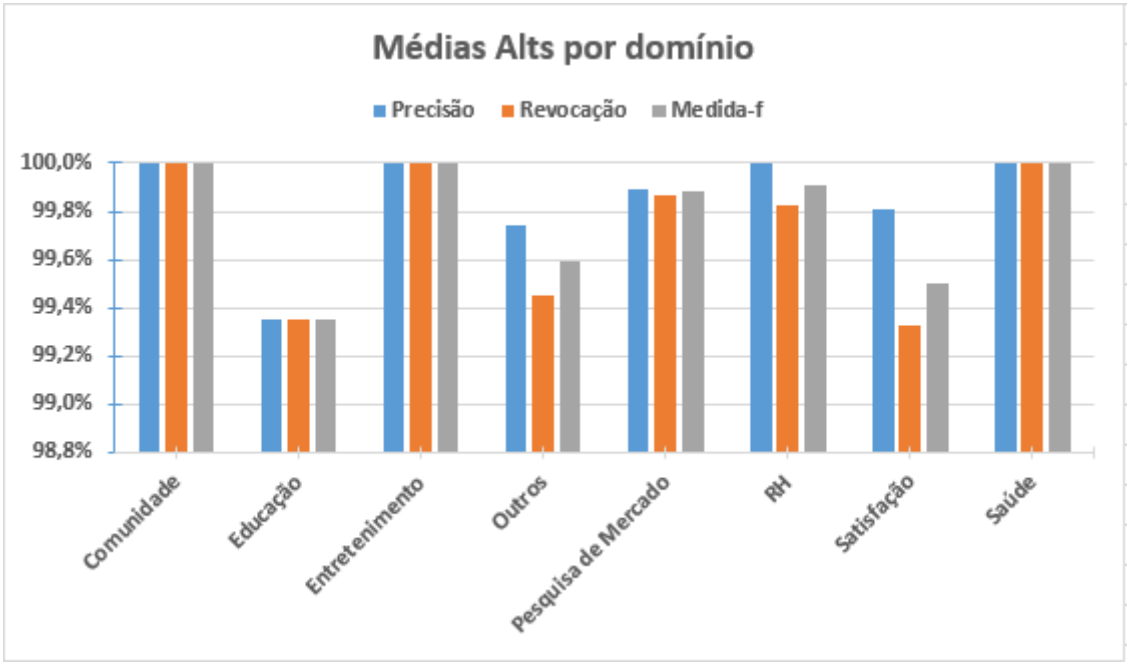

FIGURA 26. RESULTADO POR DOMÍNIO DA EXTRAÇÃO DE ALTERNATIVAS.

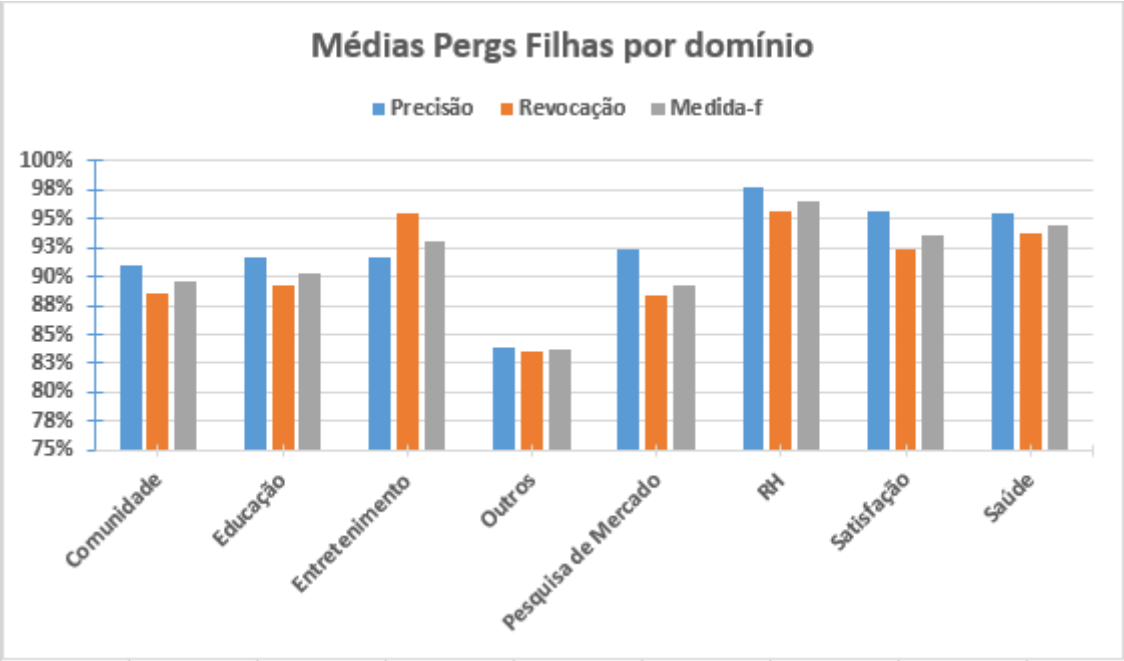

FIGURA 27. RESULTADO POR DOMÍNIO DA EXTRAÇÃO DE PERGUNTAS FILHAS.

6 Considerações finais e Trabalhos futuros

O grande número e variedade de pessoas utilizando a Internet diariamente pelo mundo faz com que a rede mundial de computadores se torne um ótimo lugar para se fazer coleta de dados de pesquisa. Um dos meios de se fazer tal coleta, é utilizando questionários online. Porém existem algumas dificuldades na criação de tais questionários, como decidir quais perguntas fazer e como fazê-las.

Os principais problemas abordados por este trabalho foram a descoberta e extração de questionários de pesquisa encontrados na Web. Para resolver tais problemas foram desenvolvidos um *Web Crawler* focado capaz de varrer a Web em busca de questionários online e um Extrator que consegue extrair os dados dos questionários para um banco de dados relacional. A seguir são descritos os principais pontos necessários para a criação e validação do *Crawler* e Extrator mencionados anteriormente.

- Definição de um conjunto de heurísticas para a descoberta de questionários em páginas Web e para a extração dos dados contidos nos mesmos.
- Desenvolvimento de um *Web Crawler* que implementasse as heurísticas previamente definidas.
- Desenvolvimento de um Extrator que implementasse as heurísticas previamente definidas.
- Realização de experimentos para verificar as abordagens definidas.

A Tabela 1 apresenta uma comparação entre os trabalhos relacionados apresentados no Capítulo 3 e a ferramenta proposta por este trabalho.

Característica	Ifrit	WT2SQL	Extração automática de sites de notícias	qFex
Item alvo	Formulários	*WebTables*	Notícias	Questionários
Principais objetivos	Identificação e rotulação dos campos	Identificação e extração dos dados	Identificação e extração dos dados	Identificação e extração dos dados
Técnica	Heurísticas e Numeração Dewey	Heurísticas	Tree Edit Distance e Heurísticas	Heurísticas e Numeração Dewey
Algoritmo utilizado	Do autor	Do autor	RTDM	Do autor

TABELA 1. COMPARAÇÃO COM TRABALHOS RELACIONADOS.

Percebe-se que todas as ferramentas possuem várias características em comum, mas cada uma se especializa na identificação e extração de um tipo diferente de dado, sendo que a ferramenta desenvolvida é a única que se foca em questionários online.

A seguir são apresentados alguns tópicos que devem ser considerados para trabalhos futuros relacionados a este trabalho:

1. Adicionar suporte a detecção e extração de questionários em páginas com conteúdo dinâmico.
2. Melhorar a extração de perguntas filhas, principalmente nos dois casos apontados na seção 5.5.
3. Implementar um algoritmo que consiga detectar a língua em que os questionários estão escritos (português, inglês, espanhol, etc). Isto pode ser útil para a análise dos dados separada por língua.

7 Referências bibliográficas

WRIGHT, K. B. (2005), Researching Internet-Based Populations: Advantages and Disadvantages of Online Survey Research, Online Questionnaire Authoring Software Packages, and Web Survey Services. Journal of Computer-Mediated Communication, 10: 00. doi:10.1111/j.1083-6101.2005.tb00259.x.

Statistics Canada. Survey Methods and Pratices. Disponível em: <http://www.statcan.gc.ca/pub/12-587-x/12-587-x2003001-eng.pdf>. Acesso em 23 de março de 2017.

SILVA, José Marcos da. Collecta : um sistema computacional de coleta de dados e avaliação institucional para apoio à tomada de decisão na Universidade Federal de Santa Catarina. Florianópolis, 2012. 191 p. Dissertação (Mestrado profissional) - Universidade Federal de Santa Catarina, Centro Sócio Econômico. Programa de Pós-Graduação em Administração Universitária.

LIU, B. Web Data Mining: Exploring Hyperlinks, Contents and Usage. (Data-Centric Systems and Applications). Chicago, 2011.

Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang, C. Storing and querying ordered xml using a relational database system. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data. SIGMOD '02. ACM, New York, NY, USA, pp. 204_215, 2002.

SANTOS, L. B. ; DORNELES, C. F. ; MELLO, R. S. . An Approach for Extracting Web Form Labels Based on Distance Analysis Of HTML Components. In: IADIS WWW/Internet Conference, 2012, Madrid. Proceeding of IADIS WWW/Internet Conference, 2012.

LAENDER, A. H. F.; RIBEIRO-NETO, B. A.; da SILVA, A. S.; TEIXEIRA, J. S. A Brief Survey of Web Data Extraction Tools. SIGMOD Record, New York, v. 31, n. 2, Junho, 2002.

Crescenzi, V., and Mecca, G. Grammars Have Exceptions. Information Systems 23, 8 (1998), 539-656.

Hammer, J., McHugh, J., and Garcia-Molina, H. Semistructured Data: The TSIMMIS Experience. In *Proceedings of the First East-European Symposium on Advances in Databases and Information Systems* (ADBIS'97)(St. Petersburg, Rusia, 199), pp. 1-8.

Sahuguet, A., and Azavant, F. Building intelligent web applications using lightweight wrappers. *Data and Knowledge Engineering* 36, 3 (2001), 283-316.

Liu, L., Pu, C., and Han, W. XWRAP: An XML-enable Wrapper Construction System for Web Information Sources. In *Proceedings of the 16th IEEE International Conference on Data Engineering* (San Diego, California, 2000), pp. 611-621.

Califf, M. E., and Mooney, R. J. Relational Learning of Pattern-Match Rules for Information Extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence* (Orlando, Florida, 1999), pp. 328-334.

Freitag, D. Machine Learning for Information Extraction in Informal Domains. *Machine Learning 39*, 2/3 (2000), 169-202.

Kushmerick, N. Wrapper Induction: Efficiency and Expressiveness. Artificial Intelligence Journal 118, 1-2 (2000), 15-68.

Hsu, C.-N., and Dung, M.-T. Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. Information Systems 23, 8 (1998), 521-538.

Adelberg, B. NoDoSE: A Tool for Semi-Automatically Extracting Structured and Semi-Structured Data from Text Documents. SIGMOD Record 27, 2 (1998), 283-294.

Laender, A. H. F., Ribeiro-Neto, B. A., and da Silva., A. S. DeByE – Data Extraction by Example. *Data and Knowledge Engineering* (2001). To appear.

Embley, D. W., Campbell, D. M., Jiang, Y. S., Liddle, S. W., kai Ng, Y., Quass, D., and Smith, R. D. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. *Data and Knoledge Engineering 31*, 3 (1999), 227-251.

Christopher Olston and Marc Najork (2010), "Web Crawling", Foundations and Trends® in Information Retrieval: Vol. 4: No. 3, pp 175-246. http://dx.doi.org/10.1561/1500000017.

SCHEIDT, M. M. Ferramenta para Extração de WebTables e Criação de Scripts SQL. 2013. 49 f. Trabalho de conclusão de curso (Monografia) – Curso de Sistemas da Informação, Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina. 2013.

Reis, D. C., Golgher, P. B., Silva, A. S., and Laender, A. F. 2004. Automatic web news extraction using tree edit distance. In Proceedings of the 13th international conference on World Wide Web (WWW '04). ACM, New York, NY, USA, 502-511. DOI=http://dx.doi.org/10.1145/988672.988740.

ECLIPSE. The Eclipse Foundation. Disponível em: <http://www.eclipse.org>. Acesso em 15 de fevereiro de 2017.

ORACLE. Java Software. Disponível em: <https://www.oracle.com/java/index.html>. Acesso em 15 de fevereiro de 2017.

SOUZA, R. H. d.; DORNELES, C. F. Analisando a eficácia do modelo vetorial de busca na ordenação de questionários. XIII Simpósio Brasileiro de Sistemas de Informação, SBSI 2017, Lavras, MG, BR, jun. 2017.

Rijsbergen, C. v. (1975). Informational Retrieval. London: ButterWorths

CHANDAN, Ms Parita; THACKER, Mr Chintan; SAXENA, Manish. A Review Paper on Analysis of Decisive and Non-Intrusive Technique to Combat Form Spam. 2016.

APÊNDICE A – Arquivo de Configuração

Como mencionado anteriormente, tanto o *Crawler* como o *Extractor* utilizam arquivos de configuração para dar ao usuário o poder de modificar diversos aspectos dos dois projetos. A seguir são explicadas algumas dessas configurações, sendo que as demais já foram apresentadas nas seções 3.5.2 e 3.5.3:

- **logLevels**: Indica que níveis de *log* podem ser apresentados pela aplicação: ‘FATAL_ERROR’, ‘ERROR’, ‘INFO’ e ‘DEBUG’. Para habilitar mais de um nível, pode-se utilizar o símbolo ‘|’ como separador, por exemplo: “ERROR|DEBUG”. Vale ressaltar que o nível ‘FATAL_ERROR’ está sempre ativo por padrão e que a biblioteca de *crawler* utilizada possui alguns *logs* internos que não são afetados por esta configuração;
- **numberOfCrawlers**: Número de *threads* que serão utilizadas para fazer o *crawling* na Web;
- **crawlStorageFolder**: Pasta que será utilizada pelo *crawler* para guardar dados intermediários;
- **politenessDelay**: Atraso, em milissegundos, entre o envio de duas requisições para o mesmo *host*. Este atraso é utilizado para não sobrecarregar os servidores dos sites que estão sendo visitados pelo *crawler*;
- **maxDepthOfCrawling**: Refere-se a profundidade máxima que o *crawler* deve seguir os links de uma página, como por exemplo: o valor 2 indicaria que o *crawler* deve pegar todos os links da página atual e então todos os links das páginas apontadas por eles. Para uma profundidade ilimitada utiliza-se o valor -1;

- **maxPagesToFetch**: Número máximo de páginas, contando todas as *threads*, que podem ser visitadas pelo crawler. Para um número ilimitado utiliza-se o valor -1;
- **includeBinaryContentInCrawling**: Informa se o *crawler* deve baixar conteúdo binário, como imagens, áudio, etc;
- **resumableCrawling**: Indica se o *crawler* deve continuar de onde parou quando for interrompido. Caso esta opção esteja ativa, o *crawling* será levemente mais lento;
- **distBetweenNearNodes**: Este é um parâmetro utilizado no agrupamento dos nodos da estrutura HTML. Ele se refere a altura máxima e a largura da distância entre dois nodos para que eles sejam considerados 'perto' um do outro;
- **loadSeedsFromCrawler**: É um parâmetro especifico do *Extractor* e que serve para informar se deve-se carregar as *seeds* do banco de dados do *Crawler* ou não. Tal parâmetro foi criado para permitir ao usuário especificar apenas certos links para o *Extractor*, utilizando o parâmetro ***seeds*** do arquivo de configuração do mesmo, por exemplo, quando a extração de um site não for satisfatória, o usuário poderá modificar alguns dos parâmetros de distância e tentar de novo até conseguir o resultado esperado. Quando especificado como *true*, todos os questionários referentes aos links do banco de dados do *Crawler* e do parâmetro *seeds* serão apagados do banco de dados do *Extractor*, e quando for *false*, apenas os questionários referentes aos links do parâmetro *seeds* serão apagados;

- **phrasesToIgnoreRegex**: Palavras/frases que quando encontradas em um *cluster*, *Crawler*, ou em um questionário, *Extractor*, fazem com que o *cluster* seja ignorado pela contagem do *Crawler* e que o questionário não seja salvo no banco de dados do *Extractor*. Entram aqui palavras/frases que são normalmente encontradas em formulários de *login*, registro e afins, como as apresentadas na Figura 7 da seção 4.3.

APÊNDICE B – Artigo

qFex: um crawler para busca e extração de questionários de pesquisa em documentos HTML

Gilney Nathanael Mathias

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

gilney.salvo@gmail.com

Abstract. *The large increase in Internet availability around the world has opened the doors to an easier and more comprehensive way of doing research and studies using online questionnaires. Some of the problems in creating such questionnaires involve: deciding what questions to ask, how to ask them, and how to organize them. Aiming at this, this article discusses the development of a Web Crawler and an Extractor that together, are capable of generating a database of questionnaires that can be used for the analysis of this data and / or as a basis of examples for generating new questionnaires or still for the reuse of existing questions.*

Resumo. *O grande aumento na disponibilidade da Internet pelo mundo abriu as portas para uma forma mais fácil e abrangente de se fazer pesquisas e estudos utilizando questionários online. Alguns dos problemas na criação de tais questionários envolvem: decidir quais perguntas fazer, como faze-las e como organiza-las. Visando isso, este artigo aborda o desenvolvimento de um Web Crawler e de um Extrator que juntos, são capazes de gerar um banco de dados de questionários que pode servir para a análise desses dados e/ou como uma base de exemplos para geração de novos questionários ou ainda para o reuso de questões já existentes.*

1. Introdução

Nos últimos anos temos visto um aumento gigantesco no uso da Web e nos dados encontrados na mesma. A Internet facilita muito a comunicação entre pessoas de vários cantos do mundo e a busca por informações e conhecimento. Tais fatos abriram as portas para o uso de questionários online, uma forma mais abrangente e fácil para empresas e pesquisadores coletarem dados. A grande vantagem do uso de tais questionários online é a possibilidade de alcançar um grande número de pessoas, com características em comum e/ou únicas, de forma rápida e barata [Wright 2005].

Alguns problemas encontrados na fase de design de tais questionários incluem: decidir quais questões perguntar, qual o melhor jeito de expressa-las e como arranjar as perguntas para se obter a informação necessária [Canada 2003].

Tais questionários podem ser utilizados sob vários contextos, tais como: em empresas, para avaliar seus produtos e a satisfação de seus serviços, ou em instituições de ensino e pesquisa para avaliação de seus corpos docentes e discentes [Silva 2012] e

entre outros. Portanto, pode ser bastante útil reusar questionários já criados para a realização de novas coletas de dados.

Em vista disso, este trabalho tem como objetivo coletar questionários de pesquisa na Web e extrair seus dados para construir um banco de dados centralizado das informações coletadas. Os dados extraídos servirão como uma base de conhecimento que pode ser usada como ponto de partida para a construção de novos questionários ou para a análise das características presentes visando extrair algum tipo de informação útil. Vale ressaltar que até a presente data não se encontrou nenhum trabalho que tenha esse foco na busca e extração dos dados de questionários de pesquisa na Web.

Um grande desafio na identificação e extração dos dados de questionários online está na quantidade enorme de diferentes formas de se construir os mesmos utilizando HTML. Cada site tem sua própria maneira de estruturar o HTML, de estilizar os elementos da página e de enviar os dados do cliente para o servidor. Além disso as páginas podem ter conteúdo estático, todas as perguntas do questionário são carregadas junto com a página, ou dinâmico, aonde certas perguntas só são carregadas após alguma interação do usuário. Desta forma, é importante ressaltar que este trabalho se foca na identificação e extração de questionários contidos em páginas com conteúdo estático.

Para lidar com a falta de padronização na criação de questionários online utilizando HTML, propõe-se a criação de um conjunto de heurísticas que possa ser utilizado para fazer a identificação de questionários em páginas HTML e para fazer a extração das questões presentes nesses questionários. Além disso, é proposto a implementação de um *Web Crawler* focado e de um Extrator que implementem tais heurísticas para fazer a coleta dos questionários e a extração de seus dados para uma base de dados.

2. Fundamentação Teórica

2.1. Numeração Dewey

A Numeração Dewey é baseada na *Dewey Decimal Classification* desenvolvida para a classificação geral do conhecimento [Tatarinov 2002]. Essa numeração pode ser utilizada para dar um identificador (ID) único para os elementos de uma estrutura de árvore e também serve como uma forma de saber a posição desses elementos na mesma. O nodo raiz recebe o identificador “1” e então cada um dos seus nodos filhos recebem um ID composto do ID de seu nodo pai unido com um número representando a posição que esse nodo filho aparece na árvore. A Figura 1 apresenta um exemplo de como esta numeração pode ser utilizada em uma página HTML.

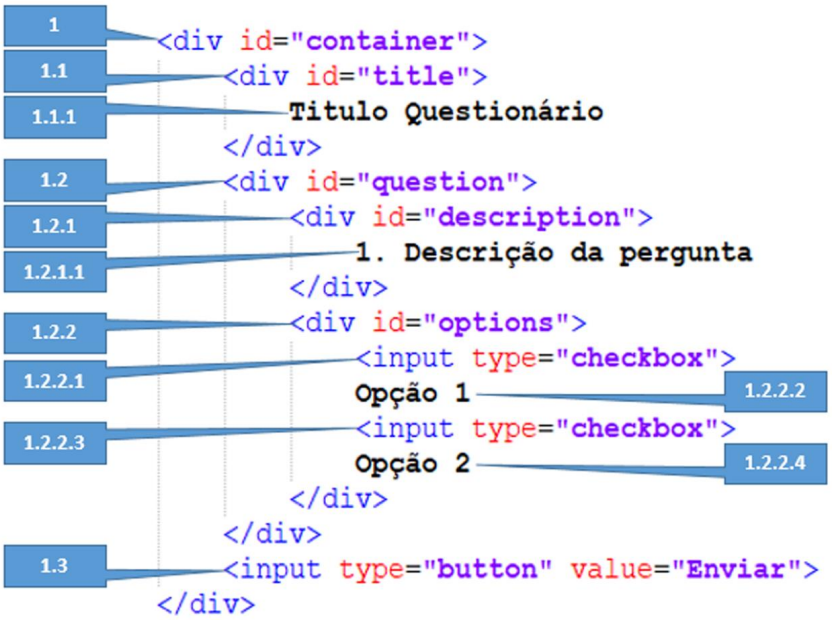

Figura 1. Demonstração do uso da Numeração Dewey no HTML.

Uma outra utilidade da Numeração Dewey é a possibilidade de calcular a distância entre os elementos, como explorado em Santos (2012). Essa distância é calculada fazendo-se a subtração número a número dos IDs, ignorando os pontos e os primeiros valores iguais. Caso um dos IDs tenha o comprimento maior do que o outro, o cálculo é feito normalmente, levando em consideração o tamanho do menor, e em seguida o restante dos números do maior ID são copiados para o resultado da distância.

2.2. Extração de dados da Web

A extração de informação da Web se baseia no problema de extrair itens alvos de informação de páginas web. Existem dois problemas gerais nesta área: extrair informação de textos em linguagem natural e extrair dados estruturados [Liu 2011].

Uma abordagem tradicional para a extração de dados de páginas web é a escrita de programas especializados, chamados de *wrappers*, que identificam os dados de interesse e mapeiam eles para um formato adequado [Laender 2002]. A seguir são apresentados os seis principais grupos de ferramentas para a geração de *wrappers* descritos em Laender et al. (2012):

- **Linguagens para desenvolvimento de *wrappers***: São linguagens especificamente criadas para ajudar os usuários a construir *wrappers*. Alguns exemplos de ferramentas que adotam essa abordagem são: Minerva [Crescenzu e Mecca 1998] e TSIMMIS [Hammer, Mchugh e Garcia-Molina 1997];
- **Ferramentas conscientes do HTML**: São agrupados aqui ferramentas que dependem da estrutura inerente dos documentos HTML. Algumas ferramentas representativas baseadas nessa abordagem são: W4F [Sahuguet e Azavant 2001] e XWRAP [Liu, Pu e Han 2000];
- **Ferramentas baseadas no Processamento de Linguagem Natural**: Estas ferramentas geralmente aplicam técnicas como filtragem de dados e marcação léxica e semântica para construir relações entre elementos de frases e sentenças

para derivar regras de extração. Algumas ferramentas que utilizam essa abordagem são: RAPIER [Calief e Mooney 1997] e SRV [Freitag 2000];

- **Ferramentas para indução de *wrappers***: Ferramentas neste grupo geram regras de extração baseadas em delimitadores derivados de um determinado conjunto de exemplos de treinamento. Ferramentas como WIEN [Kushmerick 2000] e SoftMealy [Hsu e Dung 1998] são representativos dessa abordagem;
- **Ferramentas baseadas em modelagem**: Ferramentas nesta categoria recebem como entrada certas estruturas, como listas e tabelas, e tentam localizar nas páginas web porções de dados que implicitamente se acomodam a essas estruturas. Algumas ferramentas que adotam essa abordagem são: NoDoSE [Adelberg 1998] e DEByE [Laender, Ribeiro-Neto e da Silva 2001];
- **Ferramentas baseadas em Otologias**: As ferramentas deste grupo dependem diretamente dos dados das páginas web e utilizam ontologias para localizar constantes nessas páginas e criar objetos a partir dessas constantes. A ferramenta mais representativa dessa abordagem é a ferramenta desenvolvida pelo Grupo de Extração de Dados da Universidade Brigham Young [Embley et al. 1999].

2.3. Web Crawler

Um *Web Crawler*, também conhecido como *spider* ou robô, é um programa que baixa automaticamente páginas da Web [Liu 2011]. Ele pode ser utilizado para vários fins, como por exemplo, para indexar páginas web para motores de busca, como o da Google, para fazer o arquivamento da Web, como o providenciado pelo *Internet Archive*, e para realizar data mining [Olston e Najork 2010].

O algoritmo básico de um *web crawler* é bem simples: Dada uma lista de URLs iniciais, chamadas de *seeds*, o *crawler* faz o download de todas as páginas web endereçadas por essas URLs iniciais, extrai os *hyperlinks* contidos nessas páginas, e interativamente faz o download das páginas web apontadas por esses *hyperlinks* [Olston e Najork 2010].

Muitas vezes pode-se não querer fazer o *crawling* de toda a Web, focando-se apenas em acessar páginas de certas categorias ou tópicos. Um *web crawler* focado tenta direcionar o crawler para páginas de certas categorias interessantes para o usuário [Liu 2011].

3. Trabalhos Relacionados

Dado que, até a presente data, não se conseguiu encontrar trabalhos que tratem da busca e extração dos dados de questionários de pesquisa na Web optou-se por apresentar nesta seção apenas trabalhos relacionados a área de *crawling* e extração de dados estruturados ou semiestruturados na Web.

O Ifrit [Santos 2012] é um *web crawler* focado para fazer a detecção de formulários na Web, principalmente os que são criados sem utilizar a *tag* FORM, e para fazer a associação dos componentes desses formulários (INPUTs, TEXTAREAs, afins) com os seus respectivos rótulos (*labels*). Para alcançar esses objetivos, o Ifrit utiliza como base a Numeração Dewey e a ideia de distância entre elementos, apresentados na seção 2.1, além de um conjunto de heurísticas.

O objetivo do *web crawler* WT2SQL [Scheidt 2013] é fazer a detecção e extração dos dados de tabelas (*WebTables*) contidas em páginas da Web. Para tal, a ferramenta varre a Web, a partir de um conjunto de URLs iniciais, encontra as tabelas, procurando pelos elementos TABLE da página, e utiliza algumas heurísticas para analisar essas tabelas, verificando se são tabelas de dados ou de formatação, e para fazer suas extrações.

A proposta abordada em [Reis et al. 2004] para fazer a extração automática de notícias contidas em sites da Web utiliza como base o algoritmo de Distância de Edição entre Árvores (*Tree Edit Distance*). A ideia deste algoritmo é encontrar um conjunto mínimo de operações para realizar a transformação de uma árvore A em outra árvore B e esta ideia é utilizada pelos autores de [Reis et al. 2004] para criar o seu próprio algoritmo, chamado de RTDM, que é então usado para identificar passagens de texto relevantes contendo notícias e seus componentes, extrair eles e descartar elementos desnecessários como *banners*, links, etc.

4. qFex

Esta seção apresenta a proposta do *Web Crawler* focado e Extrator qFex (**F**inder and **Ex**tractor of **Q**uestionnaires). São dois os principais problemas abordados por este trabalho: fazer a detecção de questionários em uma página qualquer da Web; e conseguir fazer a extração dos dados neles contidos, de forma genérica, para um banco de dados reacional.

4.1. Visão Geral

Como pode ser visto na Figura 2, o processo de coleta e extração de questionários foi dividido em dois componentes: *Crawler* e *Extractor*. Ambos recebem como entrada um arquivo de configuração que possui as configurações do banco de dados, da biblioteca de crawler utilizada, de *seeds* para a busca/extração e de valores para os parâmetros utilizados nos mesmos.

O *Crawler* deve varrer a Web, utilizando como base as *seeds* fornecidas no arquivo de configuração, em busca de questionários que possuam certas características e então deve salvar seus links em um banco de dados. O *Extractor*, por sua vez, deve apanhar os links do banco de dados do *Crawler*, e possivelmente também do arquivo de configuração, e extrair os dados dos questionários para uma outra base de dados.

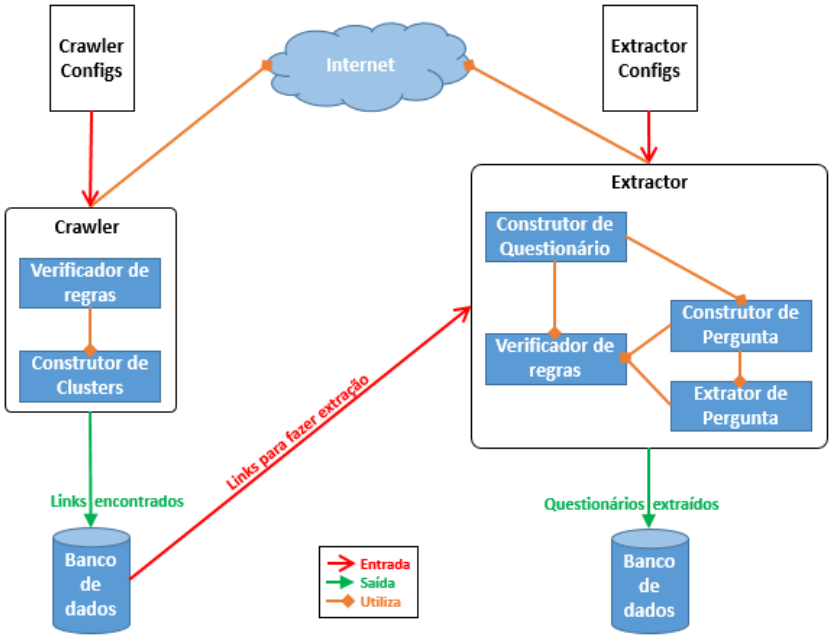

Figura 2. Visão geral do qFex.

4.2. Dewey-Ext

Neste trabalho, a Numeração Dewey foi estendida e chamada de Dewey-Ext de forma a permitir o acréscimo de novos conceitos utilizados nas heurísticas empregadas. Tais conceitos servem para averiguar as relações entre os nodos e são utilizados em quase todos os parâmetros de configuração e heurísticas do *Crawler* e do *Extractor*. Os conceitos adicionados a Numeração Dewey são os seguintes:

- **Height**: A 'altura' se refere ao valor do primeiro número do ID da distância entre dois nodos. No caso da Figura 3(a), a altura é 1;
- **Max Height**: A 'altura máxima' é o valor do maior número do ID da distância. No exemplo da Figura 3(b), a altura máxima é 2;
- **Width**: A 'largura' é o valor do comprimento do ID da distância. No caso da Figura 3(c), este valor é 2.
- **Common Prefix**: Se refere ao maior 'prefixo comum' entre os IDs dos nodos. Este cálculo é demonstrado na Figura 3(d) e 3(e) e é feito varrendo-se os IDs da esquerda para a direita e copiando os seus valores iguais até que se encontre o primeiro valor diferente. Este resultado pode ser utilizado para verificar se três ou mais nodos possuem a mesma sequência de parentes, caso os prefixos sejam iguais, ou para se averiguar a quantidade de parentes em comum, olhando o *width* dos prefixos.

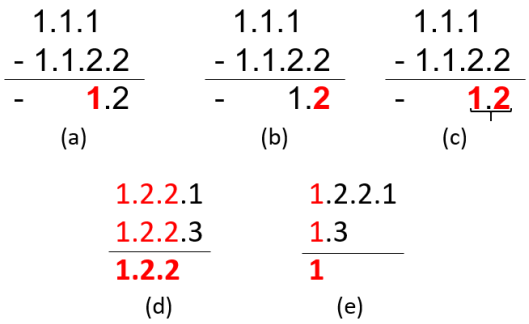

Figura 3. Exemplos de cálculos de distância e prefixo comum entre nodos.

4.3 Heurísticas

Para realizar a detecção, e em seguida a extração, de questionários em páginas Web, foi necessário primeiro encontrar características que distinguem os questionários online de outros tipos de formulários (*WebForms*) quaisquer. Após a análise de diversos sites, e de vários testes, chegou-se a um conjunto de heurísticas que são utilizadas nos algoritmos do Crawler e do Extrator para fazer tal detecção e extração. A seguir são apresentadas as principais heurísticas que são comuns entre o Crawler e o Extrator, sendo que as demais são explicadas nas próximas seções:

- **Heurística 1:** A descrição de uma pergunta normalmente começa com um número ou uma palavra qualquer iniciando com letra maiúscula, possuindo pelo menos quatro caracteres, um espaço e terminando com o caracter ‘:’, ‘?’ ou ‘.’.
- **Heurística 2:** Perguntas normalmente possuem suas descrições e componentes de formulário (INPUT’s, TEXTAREA’s, SELECT’s, entre outros) próximos uns dos outros.
- **Heurística 3:** É possível eliminar certos formulários/campos que não pertencem a um questionário, verificando a distância entre os componentes e certas palavras/frases que são normalmente encontradas nos mesmos.

4.4. Detecção de questionários

Além das heurísticas apresentadas na seção anterior, o algoritmo de detecção de questionários também utiliza algumas heurísticas a mais que são apresentadas a seguir:

- **Heurística 4:** Páginas com questionários normalmente possuem certas palavras comuns em seu título e/ou em seu corpo.
- **Heurística 5:** É possível determinar a presença de um questionário verificando a quantidade de clusters (agrupamento de nodos) que possuam pelo menos um componente de formulário ou que possuam uma certa quantidade de componentes de formulário.

A partir dessas heurísticas, criou-se o algoritmo de detecção de questionários que possui os seguintes grandes passos:

1. Faça uma verificação do título e corpo da página web passada ao algoritmo em busca de certas palavras/frases que são comuns em questionários online, como: questionário, *survey*, *questionnaire*, etc.

2. Faça o agrupamento de nodos de texto, imagem e componentes de formulário que estão próximos uns dos outros.
3. Faça a verificação da presença de um questionário a partir dos agrupamentos, *clusters*, gerados no passo anterior.

A verificação do passo 3 é feita olhando-se a quantidade de componentes de formulário que cada um dos clusters possui e também a quantidade de clusters que possuem no mínimo um componente de formulário. Caso a quantidade de componentes de um cluster seja maior ou igual a um certo valor ou caso seja encontrado uma certa quantidade de clusters com pelo menos um componente, então o algoritmo retorna *true*, indicando a possível presença de um questionário na página web passada a ele.

O cálculo da quantidade de componentes em um cluster leva em conta alguns detalhes:

- Para que um componente seja adicionado a esta contagem, ele precisa ter pelo menos um nodo de texto acima dele que possua todas as características de uma descrição de pergunta apresentadas na Heurística 1. Isto ajuda a eliminar componentes soltos pelas páginas HTML que são usados para outros propósitos;
- Componentes de RADIO INPUT e CHECKBOX que possuam todas as características mencionadas na Heurística 1 contam como 0.75. Caso eles apenas possuam um nodo de texto acima que inicie com letra maiúscula ou número eles contam como 0.25. Todos os outros tipos de componentes precisam ter todas as características da Heurística 1 para contar como 1.0;
- Clusters que possuam as palavras/frases mencionadas na Heurística 3 são ignorados desta contagem com o intuito de conseguir uma maior precisão na detecção de questionários.

4.5. Extração de questionários

Além das heurísticas apresentadas na seção 4.3, o algoritmo de extração de questionários utiliza uma outra heurística referente aos padrões observados em páginas web que possuíam questionários online:

- **Heurística 6:** Todas as formas de perguntas seguem um ou mais padrões diferentes em suas estruturações nas páginas HTML e é possível utilizar tais padrões para fazer a extração das mesmas.

A ideia básica do algoritmo de extração de questionários é ir agrupando nodos de texto e imagem que estiverem próximos uns dos outros até que se encontre um nodo de componente de formulário aonde, neste ponto, o algoritmo então tenta fazer a extração da pergunta referente a este componente. Além disso, claro, o algoritmo também deve encontrar outros aspectos importantes dos questionários, como os seus assuntos, as descrições dos grupos de perguntas que eles possuem e entre outras informações.

A parte da extração é feita utilizando padrões que são seguidos para cada forma diferente de pergunta de um questionário e que foram observados durante a fase de análise, por exemplo, perguntas de CHECKBOX seguem o padrão: Descrição da pergunta, CHECKBOX, texto da alternativa 1, CHCKBOX, texto da alternativa 2, etc. Já as outras informações são encontradas utilizando parâmetros configuráveis que utilizam como base os conceitos de distância entre nodos da Dewey-Ext.

5. Experimentos

Os experimentos conduzidos tiveram o intuito de avaliar a qualidade das abordagens utilizadas para a solução dos problemas apresentados.

Para fazer a avaliação do Crawler deixou-se o mesmo rodando por 11 horas e 05 minutos, o que resultou em 2262 links de possíveis questionários encontrados em 32 web sites diferentes. Já os experimentos feitos para avaliar o Extrator foram realizados com uma base de dados de 510 questionários que foram pegos de um total de 37 web sites diferentes.

A precisão (*precision*) do Crawler atingiu 94,47%, ou seja, 2137 dos links encontrados pelo Crawler realmente possuíam um ou mais questionários neles. Notou-se que os outros 5,53% de links encontrados não possuíam questionários, mas sim formulários muito grandes ou múltiplos formulários pequenos, porém próximos uns dos outros, o que acabou confundindo o algoritmo de detecção de questionários do Crawler.

A Figura 4 contém as médias gerais das métricas utilizadas para avaliar a extração de perguntas do Extrator. Como pode ser visto, todas as três métricas ficaram acima de 90%, o que indica que a maioria das descrições de perguntas foram extraídas corretamente.

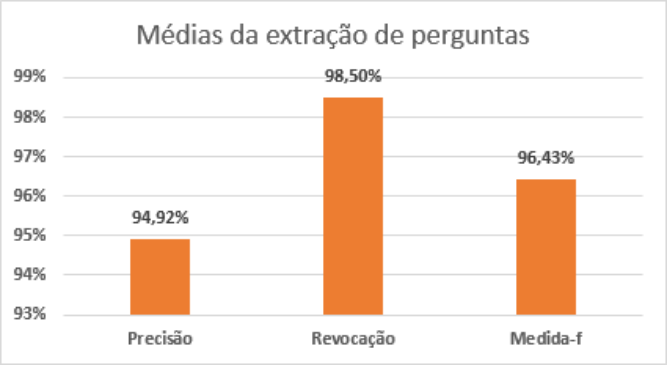

Figura 4. Resultado geral para a extração de perguntas.

A Figura 5 apresenta as médias gerais da extração de alternativas. Neste caso percebe-se que todas as métricas ficaram bem próximas de 100%, o que indica que os padrões de perguntas discutidos na seção 4.5 foram eficazes para a extração das alternativas.

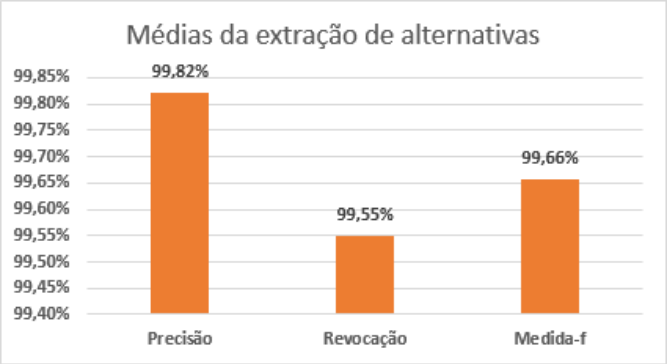

Figura 5. Resultado geral para a extração de alternativas.

A Figura 6 exibe os resultados para a extração de perguntas filhas, aonde é possível ver que a precisão chegou a um pouco mais de 92% e a revocação (*recall*) e a medida-f (*f-measure*) ficaram próximas de 91%. Existem dois motivos principais para estes resultados reduzidos em comparação a extração de perguntas e alternativas:

1. Alguns sites colocam as perguntas filhas no mesmo nível que uma pergunta normal, fazendo com que o Extrator se confunda e as considere como perguntas normais.
2. Uma deficiência no reconhecimento da descrição da perguntas pai de uma pergunta com subperguntas faz com que, em alguns casos, o assunto do questionário seja considerado como pergunta pai da(s) primeira(s) pergunta(s) do questionário. Isto ocorre porque nesses casos específicos, a estrutura do início do questionário fica idêntica a estrutura de uma pergunta com subpergunta.

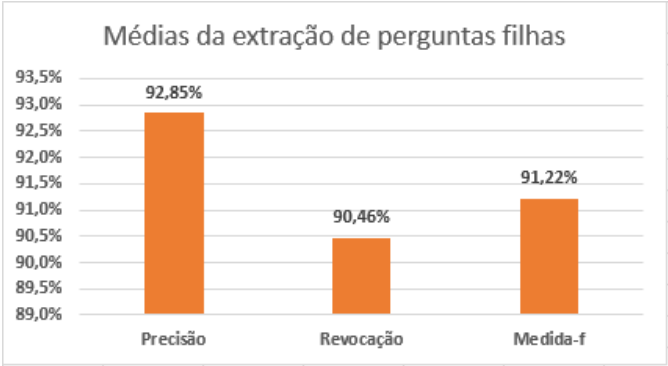

Figura 6. Resultado geral para a extração de perguntas filhas.

6. Considerações finais e Trabalhos futuros

O grande número e variedade de pessoas utilizando a Internet diariamente pelo mundo faz com que a rede mundial de computadores se torne um ótimo lugar para se fazer coleta de dados de pesquisa. Um dos meios de se fazer tal coleta, é utilizando questionários online. Porém, existem algumas dificuldades na criação de tais questionários, como decidir quais perguntas fazer e como faze-las.

Os principais problemas abordados por este trabalho foram a descoberta e extração de questionários de pesquisa encontrados na Web. Para resolver tais problemas foram desenvolvidos um *web crawler* focado capaz de varrer a Web em busca de questionários online e um Extrator que consegue extrair os dados dos questionários para um banco de dados relacional. A seguir são descritos os principais pontos necessários para a criação e validação do Crawler e Extrator mencionados anteriormente.

- Definição de um conjunto de heurísticas para a descoberta de questionários em páginas web e para a extração dos dados contidos nos mesmos.
- Desenvolvimento de um Web Crawler que implementasse as heurísticas previamente definidas.
- Desenvolvimento de um Extrator que implementasse as heurísticas previamente definidas.
- Realização de experimentos para verificar as abordagens definidas.

A seguir são apresentados alguns tópicos que devem ser considerados para trabalhos futuros relacionados a este trabalho:

1. Adicionar suporte a detecção e extração de questionários em páginas com conteúdo dinâmico.
2. Melhorar a extração de perguntas filhas, principalmente nos dois casos apontados na seção 5.
3. Implementar um algoritmo que consiga detectar a língua em que os questionários estão escritos (português, inglês, espanhol, etc). Isto pode ser útil para a análise dos dados separada por língua.

6. Referencias bibliográficas

Wright, K. B. (2005), Researching Internet-Based Populations: Advantages and Disadvantages of Online Survey Research, Online Questionnaire Authoring Software Packages, and Web Survey Services. Journal of Computer-Mediated Communication, 10: 00. doi:10.1111/j.1083-6101.2005.tb00259.x.

Statistics Canada. Survey Methods and Pratices. Disponível em: <http://www.statcan.gc.ca/pub/12-587-x/12-587-x2003001-eng.pdf >. Acesso em 23 de março de 2017.

SILVA, José Marcos da. Collecta : um sistema computacional de coleta de dados e avaliação institucional para apoio à tomada de decisão na Universidade Federal de Santa Catarina. Florianópolis, 2012. 191 p. Dissertação (Mestrado profissional) - Universidade Federal de Santa Catarina, Centro Sócio Econômico. Programa de Pós-Graduação em Administração Universitária.

LIU, B. Web Data Mining: Exploring Hyperlinks, Contents and Usage. (Data-Centric Systems and Applications). Chicago, 2011.

Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang, C. Storing and querying ordered xml using a relational database system. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data. SIGMOD '02. ACM, New York, NY, USA, pp. 204_215, 2002.

SANTOS, L. B. ; DORNELES, C. F. ; MELLO, R. S. . An Approach for Extracting Web Form Labels Based on Distance Analysis Of HTML Components. In: IADIS WWW/Internet Conference, 2012, Madrid. Proceeding of IADIS WWW/Internet Conference, 2012.

LAENDER, A. H. F.; RIBEIRO-NETO, B. A.; da SILVA, A. S.; TEIXEIRA, J. S. A Brief Survey of Web Data Extraction Tools. SIGMOD Record, New York, v. 31, n. 2, Junho, 2002.

Crescenzi, V., and Mecca, G. Grammars Have Exceptions. Information Systems 23, 8 (1998), 539-656.

Hammer, J., McHugh, J., and Garcia-Molina, H. Semistructured Data: The TSIMMIS Experience. In Proceedings of the First East-European Symposium on Advances in Databases and Information Systems (ADBIS'97)(St. Petersburg, Rusia, 199), pp. 1-8.

Sahuguet, A., and Azavant, F. Building intelligent web applications using lightweight wrappers. Data and Knowledge Engineering 36, 3 (2001), 283-316.

Liu, L., Pu, C., and Han, W. XWRAP: An XML-enable Wrapper Construction System for Web Information Sources. In Proceedings of the 16th IEEE International Conference on Data Engineering (San Diego, California, 2000), pp. 611-621.

Califf, M. E., and Mooney, R. J. Relational Learning of Pattern-Match Rules for Information Extraction. In Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (Orlando, Florida, 1999), pp. 328-334.

Freitag, D. Machine Learning for Information Extraction in Informal Domains. Machine Learning 39, 2/3 (2000), 169-202.

Kushmerick, N. Wrapper Induction: Efficiency and Expressiveness. Artificial Intelligence Journal 118, 1-2 (2000), 15-68.

Hsu, C.-N., and Dung, M.-T. Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. Information Systems 23, 8 (1998), 521-538.

Adelberg, B. NoDoSE: A Tool for Semi-Automatically Extracting Structured and Semi-Structured Data from Text Documents. SIGMOD Record 27, 2 (1998), 283-294.

Laender, A. H. F., Ribeiro-Neto, B. A., and da Silva., A. S. DeByE – Data Extraction by Example. Data and Knowledge Engineering (2001). To appear.

Embley, D. W., Campbell, D. M., Jiang, Y. S., Liddle, S. W., kai Ng, Y., Quass, D., and Smith, R. D. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. Data and Knoledge Engineering 31, 3 (1999), 227-251.

Christopher Olston and Marc Najork (2010), "Web Crawling", Foundations and Trends® in Information Retrieval: Vol. 4: No. 3, pp 175-246. http://dx.doi.org/10.1561/1500000017

SCHEIDT, M. M. Ferramenta para Extração de WebTables e Criação de Scripts SQL. 2013. 49 f. Trabalho de conclusão de curso (Monografia) – Curso de Sistemas da Informação, Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina. 2013.

Reis, D. C., Golgher, P. B., Silva, A. S., and Laender, A. F. 2004. Automatic web news extraction using tree edit distance. In Proceedings of the 13th international conference on World Wide Web (WWW '04). ACM, New York, NY, USA, 502-511. DOI=http://dx.doi.org/10.1145/988672.988740.

SOUZA, R. H. d.; DORNELES, C. F. Analisando a eficácia do modelo vetorial de busca na ordenação de questionários. XIII Simpósio Brasileiro de Sistemas de Informação, SBSI 2017, Lavras, MG, BR, jun. 2017.

APÊNDICE C – Código-fonte

questionnaires_common_lib/br/ufsc/tcc/common/crawler/CrawlerController.java

```
package br.ufsc.tcc.common.crawler;

import org.json.JSONObject;
import edu.uci.ics.crawler4j.crawler.CrawlConfig;
import edu.uci.ics.crawler4j.crawler.CrawlController;
import edu.uci.ics.crawler4j.crawler.WebCrawler;
import edu.uci.ics.crawler4j.fetcher.PageFetcher;
import edu.uci.ics.crawler4j.robotstxt.RobotstxtConfig;
import edu.uci.ics.crawler4j.robotstxt.RobotstxtServer;

public class CrawlerController {
    private JSONObject userConfigs;
    private CrawlConfig configs;
    private CrawlController controller;

    public CrawlerController() throws Exception {
        this(null);
    }

    public CrawlerController(JSONObject userConfigs)
        throws Exception {
        this.userConfigs = userConfigs==null?
                new JSONObject() : userConfigs;

        this.init();
    }

    /**
     * Inicializa o Crawler utilizando as configurações do usuário,
ou usando
     * alguns valores padrão.  </br>
     * Os valores padrão utilizados são: </br>
     * <ul>
     *  <li>CrawlerStorageFolder = ./tmp</li>
     *  <li>PolitenessDelay = 1000</li>
     *  <li>MaxDepthOfCrawling = -1 (sem limite)</li>
     *  <li>MaxPagesToFetch = 10000</li>
     *  <li>IncludeBinaryContentInCrawling = false</li>
     *  <li>ResumableCrawling = false</li>
     *  <li>NumberOfCrawlers = 5</li>
     * </ul>
     *
     * @throws Exception
     */
    private void init() throws Exception {
        configs = new CrawlConfig();
        // Usa as configurações vindas do arquivo json ou usa
        // alguns valores default
        this.configs.setCrawlStorageFolder(
            userConfigs.optString("crawlStorageFolder", "./tmp"));
        this.configs.setPolitenessDelay(
            userConfigs.optInt("politenessDelay", 500));
        this.configs.setMaxDepthOfCrawling(
            userConfigs.optInt("maxDepthOfCrawling", -1));
        this.configs.setMaxPagesToFetch(
            userConfigs.optInt("maxPagesToFetch", 10000));
        this.configs.setIncludeBinaryContentInCrawling(
```

```
            userConfigs.optBoolean("includeBinaryContentInCrawling",
false));
        this.configs.setResumableCrawling(
            userConfigs.optBoolean("resumableCrawling", false));

        PageFetcher pageFetcher = new PageFetcher(this.configs);
        RobotstxtConfig robotstxtConfig = new RobotstxtConfig();
        RobotstxtServer robotstxtServer = new RobotstxtServer(
            robotstxtConfig, pageFetcher);
        this.controller = new CrawlController(
            this.configs, pageFetcher, robotstxtServer);
    }

    public void addSeed(String url){
        this.controller.addSeed(url);
    }

    public void start(Class<? extends WebCrawler> crawlerClass){
        this.controller.start(crawlerClass,
            userConfigs.optInt("numberOfCrawlers", 5));
    }

    public void setCustomData(String[] data){
        this.controller.setCustomData(data);
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/database/connection/BasicConnection.java

```
package br.ufsc.tcc.common.database.connection;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import org.json.JSONObject;
import br.ufsc.tcc.common.util.CommonLogger;

public class BasicConnection {
    protected Connection conn;
    protected String database, dbms, host, login, password;

    public BasicConnection(JSONObject configs) {
        this(configs.getString("dbms"),
                configs.getString("name"),
                configs.getString("host"),
                configs.getString("login"),
                configs.getString("password"));
    }

    public BasicConnection(String dbms, String database,
        String host, String login, String password){
        if(dbms.equals(""))
            CommonLogger.fatalError(new SQLException(
                "DBMS nao especificado!"));
        else if(database.equals(""))
            CommonLogger.fatalError(new SQLException(
                "Nome do banco de dados nao especificado!"));

        this.dbms = dbms;
        this.database = database;
        this.host = host;
```

```
        this.login = login;
        this.password = password;
    }

    public Connection getConnection() {
        if(this.conn != null)
            return this.conn;
        return this.newConnection();
    }

    private Connection newConnection() {
        try{
            this.conn = DriverManager
                .getConnection("jdbc:"+this.dbms+"://"+
                    this.host+"/"+this.database,
                    this.login, this.password);
        }catch(Exception e){
            CommonLogger.fatalError(e);
        }
        return this.conn;
    }

    public void close(){
        try {
            this.conn.close();
        } catch (SQLException e) {
            CommonLogger.error(e);
        }
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/database/dao/BasicDao.java

```
package br.ufsc.tcc.common.database.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Set;
import br.ufsc.tcc.common.database.connection.BasicConnection;

/**
 * Classe básica para manipulação do banco de dados.
 *
 * @author Gilney N. Mathias
 */
public class BasicDao {
    private ResultSet resultSet = null;
    protected PreparedStatement psmt = null;
    protected Connection conn = null;
    protected String table;

    public BasicDao(BasicConnection c, String table) {
        this.conn = c.getConnection();
        this.table = table;
    }

    /**
```

```
     * Método responsável por preparar a query antes de ser
executada.
     *
     * @param sql               Sql que se quer preparar.
     * @throws Exception
     */
    protected void query(String sql) throws Exception {
        // Para arrumar bug: 'ERROR: syntax error at
        // or near "RETURNING"'
        if(sql.matches("(?i)^select.*"))
            this.psmt = this.conn.prepareStatement(sql);
        else
            this.psmt = this.conn.prepareStatement(sql,
                Statement.RETURN_GENERATED_KEYS);
    }

    /**
     * Retorna o ultimo UID gerado pela ultima query executada.
     *
     * @return              Ultimo UID gerado.
     * @throws Exception
     */
    protected long getLastUID() throws Exception {
        if(psmt != null) {
            ResultSet rs = psmt.getGeneratedKeys();
            if(rs.next())
                return rs.getLong(1);
        }
        return -1L;
    }

    /**
     * Executa a ultima query que foi preparada pelo método {@link
#query(String) query}.
     *
     * @throws Exception
     */
    final protected void exec() throws Exception {
        if(psmt != null)
            this.resultSet = this.psmt.executeQuery();
    }

    /**
     * Retorna o resultado da ultima query executada pelo método
{@link #exec() exec}.
     *
     * @return  Resultado da ultima query executada.
     */
    final protected ResultSet getResultSet() {
        return this.resultSet;
    }

    /**
     * Executa um sql Insert utilizando os dados passados pelo
parâmetro {@code data}.
     *
     * @param data          HashMap com os dados a serem inseridos
utilizando o formato: (campo, valor).
     * @throws Exception
     */
```

```
    protected void insert(HashMap<String, Object> data) throws
Exception {
        if(data == null || data.isEmpty())
            throw new SQLException(
                "Parametro data nao especificado.");

        StringBuilder SQL = new StringBuilder(
            "INSERT INTO "+this.table+" (");
        Set<String> fields = data.keySet();

        for(String field : fields){
            SQL.append(field + ",");
        }
        SQL.setLength(SQL.length()-1);
        SQL.append(") VALUES (");

        for(int i = 0; i < data.size(); i++){
            SQL.append("?,");
        }
        SQL.setLength(SQL.length()-1);
        SQL.append(")");

        this.query(SQL.toString());

        int count = 1;
        for(String field : fields){
            this.psmt.setObject(count, data.get(field));
            count++;
        }
        this.psmt.execute();
    }

    /**
     * Executa um sql Update utilizando os dados passados pelo
parâmetro {@code data} e as restrições
     * passadas pelo parâmetro {@code where}.<br>
     * Todas as restrições neste caso são da forma 'igual', ou seja,
'where campo1 = valor1'.
     *
     * @param data          HashMap com os dados a serem atualizados
utilizando o formato: (campo, valor).
     * @param where         HashMap com as restrições ao Update
utilizando o formato: (campo, valor).
     * @throws Exception
     */
    protected void update(HashMap<String, Object> data,
        HashMap<String, Object> where) throws Exception {
        if(data == null || data.isEmpty() || where == null ||
           where.isEmpty())
            throw new SQLException("Parametro data ou where nao
especificado.");

        StringBuilder SQL = new StringBuilder(
            "UPDATE "+this.table+" SET ");
        Set<String> fields = data.keySet();

        for(String field : fields){
            SQL.append(field + "=?,");
        }
        SQL.setLength(SQL.length()-1);
```

```
        SQL.append(" WHERE ");
        Set<String> conditions = where.keySet();

        for(String condition : conditions){
            SQL.append(condition + "=? AND ");
        }
        SQL.setLength(SQL.length()-5);

        this.query(SQL.toString());

        int count = 1;
        for(String field : fields){
            this.psmt.setObject(count, data.get(field));
            count++;
        }

        for(String condition : conditions){
            this.psmt.setObject(count, where.get(condition));
            count++;
        }
        this.psmt.execute();
    }

    /**
     * Executa um sql Select utilizando os campos passados pelo
parâmetro {@code fields} e as restrições
     * passadas pelo parâmetro {@code where}.<br>
     * Todas as restrições neste caso são da forma 'igual', ou seja,
'where campo1 = valor1'.
     *
     * @param fields        String com os campos a serem retornados
utilizando o formato: campo1, campo2, etc. ou
     *                      apenas com o caracter '*' para retornar
todos os campos.
     * @param where         HashMap com as restrições ao Select
utilizando o formato: (campo, valor).
     * @throws Exception
     */
    protected void select(String fields,
        HashMap<String, Object> where) throws Exception {
        if(where == null || where.isEmpty()){
            this.select(fields);
            return;
        }

        String SQL = "SELECT "+fields+" FROM "+this.table+" WHERE ";

        Set<String> conditions = where.keySet();
        for (String condition : conditions){
            SQL += condition + "=? AND ";
        }
        SQL = SQL.substring(0, SQL.length()-5);

        this.query(SQL);

        int count = 1;
        for (String condition : conditions){
            this.psmt.setObject(count, where.get(condition));
            count++;
        }
        this.exec();
```

```
    }

    /**
     * Executa um sql Select utilizando os campos passados pelo
parâmetro {@code fields}.
     *
     * @param fields        String com os campos a serem retornados
utilizando o formato: campo1, campo2, etc. ou
     *                      apenas com o caracter '*' para retornar
todos os campos.
     * @throws Exception
     */
    protected void select(String fields) throws Exception {
        fields = fields.isEmpty() ? "*" : fields;
        String SQL = "SELECT "+fields+" FROM "+this.table+";";

        this.query(SQL);
        this.exec();
    }

    /**
     * Executa um sql Delete utilizando as restrições passadas pelo
parâmetro {@code where}.<br>
     * Todas as restrições neste caso são da forma 'igual', ou seja,
'where campo1 = valor1'.
     *
     * @param where         HashMap com as restrições ao Delete
utilizando o formato: (campo, valor).
     * @throws Exception
     */
    protected void delete(HashMap<String, Object> where)
        throws Exception {
        if(where == null || where.isEmpty()){
            this.delete();
            return;
        }

        String SQL = "DELETE FROM "+this.table+" WHERE ";
        Set<String> conditions = where.keySet();

        for(String condition : conditions){
            SQL += condition += "=? AND ";
        }
        SQL = SQL.substring(0, SQL.length()-5);

        this.query(SQL);

        int count = 1;
        for(String condition : conditions){
            this.psmt.setObject(count, where.get(condition));
            count++;
        }
        this.psmt.execute();
    }

    /**
     * Executa um sql Delete utilizando nenhuma restrição.
     *
     * @throws Exception
     */
    protected void delete() throws Exception {
```

```
        String SQL = "DELETE FROM "+this.table;

        this.query(SQL);
        this.psmt.execute();
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/database/dao/PossivelQuestionarioDao.java

```
package br.ufsc.tcc.common.database.dao;

import java.nio.charset.Charset;
import java.sql.ResultSet;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;
import com.google.common.hash.BloomFilter;
import com.google.common.hash.Funnels;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.model.PossivelQuestionario;

public class PossivelQuestionarioDao extends BasicDao {
    private static final int BloomFilter_BASE_SIZE = 1000;
    private static final double BloomFilter_BASE_FPP = 0.01;

    public PossivelQuestionarioDao(BasicConnection c) {
        super(c, "PossivelQuestionario");
    }

    public void save(PossivelQuestionario q) throws Exception{
        HashMap<String, Object> data = new HashMap<>();

        data.put("LINK_DOCUMENTO", q.getLink_doc());
        data.put("TITULO_DOCUMENTO", q.getTitulo_doc());
        data.put("ENCONTRADO_EM", q.getEncontradoEm());

        this.insert(data);
        q.setId(getLastUID());
    }

    public BloomFilter<String> getAllLinksAsABloomFilter()
        throws Exception {
        this.select("COUNT(*) AS rowcount");

        ResultSet result = this.getResultSet();
        int size = BloomFilter_BASE_SIZE;
        if(result.next())
            size += result.getInt("rowcount");

        BloomFilter<String> resp = BloomFilter.create(
                Funnels.stringFunnel(Charset.forName("utf-8")),
                size, BloomFilter_BASE_FPP);

        this.select("LINK_DOCUMENTO");

        result = this.getResultSet();
        while(result != null && result.next()){
            resp.put(result.getString("LINK_DOCUMENTO"));
        }

        result.close();
```

```
        return resp;
    }

    public Set<String> getAllLinksAsASet() throws Exception {
        this.select("LINK_DOCUMENTO");

        Set<String> resp = new HashSet<>();
        ResultSet result = this.getResultSet();
        while(result != null && result.next()){
            resp.add(result.getString("LINK_DOCUMENTO"));
        }

        result.close();
        return resp;
    }

    public Set<PossivelQuestionario> getAll() throws Exception {
        Set<PossivelQuestionario> resp = new HashSet<>();
        PossivelQuestionario pq = null;

        this.select("*");
        ResultSet result = this.getResultSet();
        while(result != null && result.next()){
            pq = new PossivelQuestionario();
            pq.setId(result.getLong("idPossivelQuestionario"));
            pq.setLink_doc(result.getString("LINK_DOCUMENTO"));
            pq.setTitulo_doc(result.getString("TITULO_DOCUMENTO"));
            pq.setEncontradoEm(result
              .getTimestamp("ENCONTRADO_EM"));
            resp.add(pq);
        }

        result.close();
        return resp;
    }

    public boolean containsLink(String link) throws Exception {
        HashMap<String, Object> where = new HashMap<>();
        where.put("LINK_DOCUMENTO", link);

        this.select("COUNT(*) AS rowcount", where);

        ResultSet result = this.getResultSet();
        int size = 0;
        if(result.next())
            size = result.getInt("rowcount");

        result.close();
        return size > 0;
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/database/manager/PossivelQuestionarioManager.java

```
package br.ufsc.tcc.common.database.manager;

import java.util.Set;
import com.google.common.hash.BloomFilter;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.database.dao.PossivelQuestionarioDao;
import br.ufsc.tcc.common.model.PossivelQuestionario;
```

```
import br.ufsc.tcc.common.util.CommonConfiguration;
import br.ufsc.tcc.common.util.CommonLogger;

/**
 * Classe de mais alto nível responsável por lidar com operações do
banco de dados
 * relacionadas a classe/tabela PossivelQuestionario.
 *
 * @author Gilney N. Mathias
 */
public class PossivelQuestionarioManager {
    private PossivelQuestionarioDao pqDao;

    /**
     * BloomFilter é uma estrutura de dados rápida e eficiente mas
que
     * possui um problema, ela pode gerar falso possitivos quando se
     * verifica a existência de um elemento nela.
     */
    private static BloomFilter<String> bfSavedLinks = null;
    private static Set<String> setSavedLinks = null;

    // Construtores
    /**
     * Construtor da classe.
     *
     * @param loadLinksAsASet       Deve-se carregar os links do
banco de dados
     *                              em uma estrutura de <b>Set</b>?
<br>
     *                              Caso seja passado o valor
<b>False</b>, os links
     *                              serão carregados em uma
estrutura de <b>BloomFilter</b>.
     */
    public PossivelQuestionarioManager(boolean loadLinksAsASet) {
        this(new BasicConnection(CommonConfiguration
            .getInstance().getCrawlerDatabaseConfigs()),
            loadLinksAsASet);
    }

    /**
     * Construtor da classe.
     *
     * @param connection            Uma instancia de
BasicConnection.
     * @param loadLinksAsASet       Deve-se carregar os links do
banco de dados
     *                              em uma estrutura de <b>Set</b>?
<br>
     *                              Caso seja passado o valor
<b>False</b>, os links
     *                              serão carregados em uma
estrutura de <b>BloomFilter</b>.
     */
    public PossivelQuestionarioManager(BasicConnection connection,
        boolean loadLinksAsASet) {
        this.pqDao = new PossivelQuestionarioDao(connection);

        if(loadLinksAsASet)
            PossivelQuestionarioManager.loadLinksAsASet(this.pqDao);
```

```
        else
            PossivelQuestionarioManager
                .loadLinksAsABloomFilter(this.pqDao);
    }

    // Demais métodos
    public boolean containsLink(String link) throws Exception {
        return this.pqDao.containsLink(link);
    }

    public Set<PossivelQuestionario> getAll() throws Exception {
        return pqDao.getAll();
    }

    public void save(PossivelQuestionario q) throws Exception {
        pqDao.save(q);
        if(bfSavedLinks != null)
            bfSavedLinks.put(q.getLink_doc());
        else
            setSavedLinks.add(q.getLink_doc());
    }

    // Métodos estáticos
    public static boolean linkWasSaved(String link){
        if(bfSavedLinks != null)
            return bfSavedLinks.mightContain(link);
        else if(setSavedLinks != null)
            return setSavedLinks.contains(link);
        else
            return false;
    }

    public static Set<String> getLinksAsASet(){
        return setSavedLinks;
    }

    public static BloomFilter<String> getLinksAsABloomFilter(){
        return bfSavedLinks;
    }

    public static synchronized void loadLinksAsASet() {
        BasicConnection conn = new
BasicConnection(CommonConfiguration.getInstance().getCrawlerDatabase
Configs());
        loadLinksAsASet(new PossivelQuestionarioDao(conn));
        conn.close();
    }

    private static synchronized void
loadLinksAsASet(PossivelQuestionarioDao dao) {
        if(setSavedLinks != null) return;

        try {
            setSavedLinks = dao.getAllLinksAsASet();

            CommonLogger.debug("{} carregou os links dos possiveis
questionarios do banco de dados.",
                    Thread.currentThread().getName());
        } catch (Exception e) {
            // Database não deve esta funcionando, então mata a
aplicação
```

```
            CommonLogger.fatalError(e);
        }
    }

    public static synchronized void loadLinksAsABloomFilter() {
        BasicConnection conn = new
BasicConnection(CommonConfiguration.getInstance().getCrawlerDatabase
Configs());
        loadLinksAsABloomFilter(new PossivelQuestionarioDao(conn));
        conn.close();
    }

    private static synchronized void
loadLinksAsABloomFilter(PossivelQuestionarioDao dao) {
        if(bfSavedLinks != null) return;

        try {
            bfSavedLinks = dao.getAllLinksAsABloomFilter();

            CommonLogger.debug("{} carregou os links dos possiveis
questionarios do banco de dados.",
                    Thread.currentThread().getName());
        } catch (Exception e) {
            // Database não deve esta funcionando, então mata a
aplicação
            CommonLogger.fatalError(e);
        }
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/model/Cluster.java

```
package br.ufsc.tcc.common.model;

import java.util.ArrayList;

public class Cluster {
    private ArrayList<MyNode> group;

    // Construtores
    public Cluster(){
        this.group = new ArrayList<>();
    }

    // Getters e Setters
    public MyNode get(int index){
        if(index < 0 || index >= this.group.size())
            return null;
        return this.group.get(index);
    }

    public ArrayList<MyNode> getGroup(){
        return this.group;
    }

    public int size(){
        return this.group.size();
    }

    public boolean isEmpty(){
        return this.group.isEmpty();
    }
```

```
    /**
     * Retorna uma String contendo o texto de todos os nodos deste
Cluster
     * concatenadas com '\n', independente se o nodo é de texto ou
não.
     *
     * @return      String contendo o texto de todos os nodos deste
Cluster.
     */
    public String getAllNodesText(){
        StringBuilder builder = new StringBuilder();
        for(MyNode node : this.group){
            builder.append(node.getText() +"\n");
        }
        if(builder.length() == 0) return "";
        builder.setLength(builder.length()-1);
        return builder.toString();
    }

    /**
     * Retorna uma String contendo o texto de todos os nodos de
texto
     * deste Cluster concatenadas com '\n'.
     *
     * @return      String contendo o texto de todos os nodos deste
Cluster.
     */
    public String getText(){
        return getText(false);
    }

    /**
     * Retorna uma String contendo o texto de todos os nodos de
texto e,
     * caso especificado, os atributos 'value' de input_text
desabilitados
     * deste Cluster concatenadas com '\n'.
     *
     * @param useValueOfInputDisabled   Usar os atributos 'value' de
input_text desabilitados?
     * @return                          String contendo o texto de
todos os nodos deste Cluster.
     */
    public String getText(boolean useValueOfInputDisabled) {
        StringBuilder builder = new StringBuilder();
        for(MyNode node : this.group){
            if(node.isText())
                builder.append(node.getText() +"\n");
            else if(useValueOfInputDisabled &&
node.isATextInputDisabledWithValue())
                builder.append(node.getAttr("value") +"\n");
        }
        if(builder.length() == 0) return "";
        builder.setLength(builder.length()-1);
        return builder.toString();
    }

    // Demais métodos
    public Cluster add(MyNode node){
        if(node != null && !this.group.contains(node))
```

```
            this.group.add(node);
        return this;
    }

    /**
     * Adiciona todos os nodos do parâmetro {@code other} a este
Cluster.
     *
     * @param other     Cluster que se quer pegar os nodos.
     * @return          Este Cluster.
     */
    public Cluster join(Cluster other){
        if(other != null){
            for(MyNode node : other.group)
                this.add(node);
        }
        return this;
    }

    /**
     * Retorna o primeiro nodo deste Cluster.
     *
     * @return      O primeiro nodo deste Cluster.
     */
    public MyNode first(){
        if(!this.group.isEmpty())
            return this.group.get(0);
        return null;
    }

    /**
     * Retorna o nodo do meio deste Cluster.
     *
     * @return      O nodo do meio deste Cluster.
     */
    public MyNode middle(){
        if(!this.group.isEmpty())
            return this.group.get((group.size()-1)/2);
        return null;
    }

    /**
     * Retorna o ultimo nodo deste Cluster.
     *
     * @return      O ultimo nodo deste Cluster.
     */
    public MyNode last(){
        if(!this.group.isEmpty())
            return this.group.get(this.group.size()-1);
        return null;
    }

    /**
     * Verifica se todos os nodos deste Cluster são nodos de texto.
     *
     * @return      <b>True</b> caso todos os nodos sejam de texto
ou<br>
     *              <b>False</b> caso contrario.
     */
    public boolean isAllText(){
        return this.group.parallelStream()
```

```
                .reduce(true,
                        (a,b) -> a && b.isText(),
                        Boolean::logicalAnd);
    }

    /**
     * Verifica se todos os nodos deste Cluster são nodos de texto
ou imagem.
     *
     * @return      <b>True</b> caso todos os nodos sejam de texto
ou imagem ou<br>
     *              <b>False</b> caso contrario.
     */
    public boolean isAllTextOrImg(){
        return this.group.parallelStream()
                .reduce(true,
                        (a,b) -> a && b.isImgOrText(),
                        Boolean::logicalAnd);
    }

    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        builder.append("Cluster:\n");
        for(MyNode node : this.group){
            builder.append("\t"+ node.toString() +"\n");
        }
        return builder.toString();
    }

    @Override
    public Cluster clone() {
        Cluster result = new Cluster();
        for(MyNode n : this.group)
            result.add(n);
        return result;
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/model/DeweyExt.java

```
package br.ufsc.tcc.common.model;

import java.util.ArrayList;
import br.ufsc.tcc.common.util.CommonUtil;

/**
 * Classe responsável por fornecer todas as operações referentes a
Dewey-Ext.
 *
 * @author Gilney N. Mathias
 */
public class DeweyExt {

    private String value;
    private ArrayList<Integer> numbers;

    // Construtores
    public DeweyExt(){
        this("");
    }
```

```
    public DeweyExt(String value){
        this.value = value;
        this.numbers = this.parseValueToNumbers(value);
    }

    // Getters e Setters
    public String getValue(){
        //Atualiza a 'cache'
        if(this.value.isEmpty())
            this.value = this.parseNumbersToValue(this.numbers);
        return this.value;
    }

    public ArrayList<Integer> getNumbers(){
        return this.numbers;
    }

    public String getCommonPrefix(DeweyExt other){
        if(other == null) return "";

        String str1 = this.getValue(),
               str2 = other.getValue();

        String result = "";
        int n1 = str1.length(), n2 = str2.length();
        for (int i=0, j=0; i<n1 && j<n2; i++, j++){
            if (str1.charAt(i) != str2.charAt(j))
                break;
            result += str1.charAt(i);
        }
        if(!result.isEmpty()){
            int i = result.lastIndexOf('.');
            if(i == -1)
                result = "";
            else if(result.length()-i <= 3
                result = result.substring(0,
result.lastIndexOf('.'));
        }
        return result;
    }

    public int getWidth(){
        return this.numbers.size();
    }

    public int getHeight(){
        if(!this.numbers.isEmpty())
            return Math.abs(this.numbers.get(0));
        return Integer.MAX_VALUE;
    }

    public int getMaxHeight(){
        if(!this.numbers.isEmpty()) {
            int max = Math.abs(this.numbers.get(0)), tmp = 0;
            for(int i = 1; i<this.numbers.size(); i++){
                tmp = Math.abs(this.numbers.get(i));
                if(tmp > max)
                    max = tmp;
            }
            return max;
        }
```

```
        return Integer.MAX_VALUE;
    }

    // Demais métodos
    private DeweyExt add(int n){
        // Se o 'numbers' estiver vazio não se deve adicionar
        // o valor zero, para evitar coisas como: 000.001 ...
        if(!this.numbers.isEmpty() || n != 0){
            this.numbers.add(n);
            //Limpa a 'cache'
            this.value = "";
        }
        return this;
    }

    public DeweyExt distanceOf(DeweyExt other){
        if(other == null) return null;

        DeweyExt dist = new DeweyExt();
        ArrayList<Integer> n1 = this.numbers,
                n2 = other.numbers,
                n3 = null;
        int min = Math.min(n1.size(), n2.size());

        for(int i = 0; i<min; i++)
            dist.add(n1.get(i) - n2.get(i));

        // Caso um deles seja maior que o outro, é preciso
        // salvar o restante dos numeros do maior
        if(n1.size() < n2.size()) n3 = n2;
        else if(n1.size() > n2.size()) n3 = n1;

        if(n3 != null){
            // Se a diferença esta vazia, e caso o maior seja o
            // de baixo, então o 1* numero deve ser negativo
            if(dist.numbers.isEmpty() && n3 == n2){
                dist.add(-n3.get(min++));
            }
            for(int i = min; i<n3.size(); i++){
                dist.add(n3.get(i));
            }
        }
        return dist;
    }

    public ArrayList<Integer> parseValueToNumbers(String value) {
        ArrayList<Integer> numbers = new ArrayList<>();
        if(!value.isEmpty()){
            value = value.trim();
            String[] tmp = value.split("\\.");
            for(String s : tmp){
                numbers.add(Integer.valueOf(s));
            }
        }
        return numbers;
    }

    public String parseNumbersToValue(ArrayList<Integer> numbers){
        String value = "";
        if(numbers != null && !numbers.isEmpty()){
            for(int n : numbers){
```

```
                value += CommonUtil.padNumber(n) + ".";
            }
            value = value.substring(0, value.length()-1);
        }
        return value;
    }

    /**
     * Verifica se o primeiro valor deste DeweyExt é negativo.<br>
     *
     * @return  <b>True</b> caso o primeiro valor seja negativo
ou<br>
     *          <b>False</b> caso contrario.
     */
    public boolean isNegative(){
        if(!this.numbers.isEmpty())
            return this.numbers.get(0) < 0;
        return false;
    }

    @Override
    public String toString() {
        return this.getValue();
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;
        DeweyExt other = (DeweyExt) obj;
        return this.getValue().equals(other.getValue());
    }

    @Override
    public int hashCode() {
        return this.getValue().hashCode();
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/model/MyNode.java

```
package br.ufsc.tcc.common.model;

import org.jsoup.nodes.Attributes;
import org.jsoup.nodes.Node;
import br.ufsc.tcc.common.util.CommonUtil;

/**
 * Classe que possui os atribuitos necessários para representar
 * um nodo da árvore HTML.
 *
 * @author Gilney N. Mathias
 */
public class MyNode implements Comparable<MyNode> {
    private String text;
    private MyNodeType type;
    private Attributes attrs;
    private DeweyExt dewey;

    // Construtores
```

```
    public MyNode() {
        this(null, null);
    }

    public MyNode(Node node, DeweyExt dewey) {
        if(node != null){
            this.text = CommonUtil.getNodeRepresentation(node);
            this.attrs = node.attributes();
            this.type = MyNodeType.get(this.text, node.nodeName());
        }else{
            this.text = "";
            this.attrs = null;
            this.type = MyNodeType.UNKNOWN;
        }
        this.dewey = dewey;
    }

    // Getters e Setters
    public String getText() {
        return this.text;
    }

    public void setText(String text){
        this.text = text;
    }

    public MyNodeType getType(){
        return this.type;
    }

    public Attributes getAttributes(){
        return this.attrs;
    }

    public String getAttr(String key){
        if(this.attrs == null) return "";
        String attr = this.attrs.get(key.toLowerCase());
        if(attr.isEmpty())
            attr = this.attrs.get(key.toUpperCase());
        return attr;
    }

    public DeweyExt getDewey() {
        return dewey;
    }

    public void setDewey(DeweyExt dewey){
        this.dewey = dewey;
    }

    // Demais métodos
    /**
     * Verifica se este Nodo é do tipo passado pelo parâmetro {@code
type}.<br>
     * Os tipos de um Nodo podem ser vistos na classe {@link
MyNodeType}.
     *
     * @param type      Tipo para verificação.
     * @return          <b>True</b> caso este Nodo seja do tipo
{@code type} ou<br>
     *                  <b>False</b> caso contrario.
```

```
     */
    public boolean isA(String type){
        return this.getType().toString()
                .equals(type.toUpperCase());
    }

    public boolean isImage(){
        return this.type == MyNodeType.IMG;
    }

    public boolean isText(){
        return this.type == MyNodeType.TEXT;
    }

    public boolean isImgOrText(){
        return this.isImage() || this.isText();
    }

    public boolean isComponent(){
        return !this.isImgOrText() &&
                this.type != MyNodeType.UNKNOWN;
    }

    public boolean isATextInputDisabledWithValue(){
        return this.isA("TEXT_INPUT")
&& !this.getAttr("disabled").isEmpty() &&
                !this.getAttr("value").isEmpty();
    }

    @Override
    public String toString() {
        return this.dewey +" - "+ this.text;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || this.getClass() != obj.getClass())
            return false;
        MyNode other = (MyNode) obj;
        return this.dewey.equals(other.dewey);
    }

    @Override
    public int compareTo(MyNode o) {
        if(this.dewey == null){
            if(o.dewey == null) return 0;
            return 1;
        }
        if(o.dewey == null) return -1;
        return this.dewey.toString()
                .compareTo(o.dewey.toString());
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/model/MyNodeType.java

```
package br.ufsc.tcc.common.model;

/**
```

```
 * Enum que possui os tipos de Nodos interessantes para esta
aplicação.
 *
 * @author Gilney N. Mathias
 */
public enum MyNodeType {
    UNKNOWN("unknown"),
    TEXT("#text"),
    IMG("img"),
    TEXT_INPUT("input[type=text]"),
    NUMBER_INPUT("input[type=number]"),
    DATE_INPUT("input[type=date]"),
    EMAIL_INPUT("input[type=email]"),
    TEL_INPUT("input[type=tel]"),
    TIME_INPUT("input[type=time]"),
    URL_INPUT("input[type=url]"),
    CHECKBOX_INPUT("input[type=checkbox]"),
    RADIO_INPUT("input[type=radio]"),
    RANGE_INPUT("input[type=range]"),
    TEXTAREA("textarea"),
    SELECT("select"),
    OPTION("option");

    private final String text;

    MyNodeType(String text){
        this.text = text;
    }

    public static MyNodeType get(String text, String name){
        text = text.toLowerCase();
        for(MyNodeType t : MyNodeType.values()){
            if(t.text.equals(text) || t.text.equals(name))
                return t;
        }
        return MyNodeType.UNKNOWN;
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/model/PossivelQuestionario.java

```
package br.ufsc.tcc.common.model;

import java.sql.Timestamp;
import br.ufsc.tcc.common.util.CommonUtil;

public class PossivelQuestionario {

    private long id;
    private String link_doc;
    private String titulo_doc;
    private Timestamp encontrado_em;

    // Construtor
    public PossivelQuestionario(){
        this.updateEncontradoEm();
    }

    public PossivelQuestionario(String link_doc, String titulo_doc){
        this.link_doc = link_doc;
        this.titulo_doc = titulo_doc;
```

```
        this.updateEncontradoEm();
    }

    // Getters e Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getLink_doc() {
        return link_doc;
    }

    public void setLink_doc(String link_doc) {
        this.link_doc = link_doc;
    }

    public String getTitulo_doc() {
        return titulo_doc;
    }

    public void setTitulo_doc(String titulo_doc) {
        this.titulo_doc = titulo_doc;
    }

    public Timestamp getEncontradoEm() {
        return encontrado_em;
    }

    public void setEncontradoEm(Timestamp encontradoEm) {
        this.encontrado_em = encontradoEm;
    }

    public void updateEncontradoEm(){
        this.encontrado_em = CommonUtil.getCurrentTime();
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/util/ CommonConfiguration.java

```
package br.ufsc.tcc.common.util;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

/**
 * Classe responsável por ler e validar o arquivo de configuração e
providenciar
 * uma forma simples para que outras classes possam acessar as
 * propriedades do mesmo.
 *
 * @author Gilney N. Mathias
 */
public abstract class CommonConfiguration {
    private static CommonConfiguration instance = null;
    protected String configsPath;//DEVE ser sobrescrito pelas
subclasses
```

```
    protected JSONObject configs;
    protected String validatingPath;//usado apenas na parte de
validação

    // Getters e Setters
    public static CommonConfiguration getInstance() {
        return instance;
    }

    public static void setInstance(CommonConfiguration instance) {
        CommonConfiguration.instance = instance;
    }

    public String getLogLevels() {
        return configs.optString("logLevels");
    }

    public JSONObject getDatabaseConfigs(){
        return configs.getJSONObject("database");
    }

    public boolean loadSeedsFromCrawler(){
        JSONObject tmp = getDatabaseConfigs();
        return tmp.optBoolean("loadSeedsFromCrawler", true);
    }

    public JSONObject getCrawlerDatabaseConfigs(){
        try{
            return getDatabaseConfigs().getJSONObject("crawler");
        }catch(JSONException exp){
            CommonLogger.fatalError(
                    new JSONException("Objeto 'database.crawler' não
encontrado no arquivo de configuração!"));
            return null;
        }
    }

    public JSONObject getExtractorDatabaseConfigs(){
        try{
            return getDatabaseConfigs().getJSONObject("extractor");
        }catch(JSONException exp){
            CommonLogger.fatalError(
                    new JSONException("Objeto 'database.extractor'
não encontrado no arquivo de configuração!"));
            return null;
        }
    }

    public JSONObject getCrawlerConfigs(){
        return configs.optJSONObject("crawler");
    }

    public JSONArray getSeeds(){
        JSONArray arr = configs.optJSONArray("seeds");
        return arr != null ? arr : new JSONArray();
    }

    public JSONObject getParameters(){
        return configs.getJSONObject("parameters");
    }
```

```
    // Demais métodos
    protected void loadConfigs() {
        try{
            String configContent = CommonUtil.readFile(configsPath);
            configs = CommonUtil.parseJson(configContent);
        }catch(Exception e){
            // É obrigatório fornecer o arquivo de configuração!
            CommonLogger.fatalError(e);
        }
    }

    /**
     * Tenta validar os dados obrigatórios do arquivo de
configuração.
     */
    protected void validateConfigs() {
        try {
            validatingPath = "";
            configs.getString("logLevels");
            this.validateDatabaseConfig();

            validatingPath = "";
            this.validateCrawlerConfig();

            validatingPath = "";
            this.validateParameters();
        }catch(JSONException exp){
            String msg = exp.getMessage();
            String value = msg.substring(msg.indexOf('[')+2,
msg.lastIndexOf(']')-1);
            validatingPath += !validatingPath.equals("") ? "." : "";
            msg = msg.replace(value, validatingPath+value);
            CommonLogger.fatalError(new JSONException(msg));
        }
    }

    protected void validateDatabaseConfig() {
        // Não faz nada por padrão
        // Subclasses DEVEM sobrescrever este método!
    }

    protected void validateCrawlerConfig() {
        // Não faz nada por padrão [objeto opcional]
        // Subclasses PODEM sobrescrever este método para adicionar
        // algumas verificações
    }

    protected void validateParameters() {
        // Não faz nada por padrão
        // Subclasses DEVEM sobrescrever este método!
    }

    protected void validateIntParameter(JSONObject paramsObj, String
param, String ...keys) throws JSONException {
        boolean checkingKeys = false;
        try {
            JSONObject tmp = paramsObj.getJSONObject(param);
            checkingKeys = true;
            for(String key : keys) {
                tmp.getInt(key);
            }
```

```
        } catch(JSONException exp) {
            if(checkingKeys)
                validatingPath += "."+param;
            throw exp;
        }
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/util/ CommonLogger.java

```
package br.ufsc.tcc.common.util;

import java.util.List;
import javax.swing.JOptionPane;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.helpers.MessageFormatter;

/**
 * Classe responsável por lidar com os logs do sistema.
 *
 * @author Gilney N. Mathias
 */
public class CommonLogger {
    protected static final String path = "./log.txt";
    protected static final Logger logger =
LoggerFactory.getLogger(CommonLogger.class);
    //FATAL_ERROR esta sempre ativado
    private static String enabledLevels = "FATAL_ERROR|";

    // Debug level
    public static boolean isDebugEnabled(){
        return enabledLevels.contains("|DEBUG");
    }

    public static void setDebugEnabled(boolean enabled){
        if(enabled && !isDebugEnabled())
            enabledLevels += "|DEBUG";
        else if(!enabled && isDebugEnabled())
            enabledLevels = enabledLevels.replace("|DEBUG", "");
    }

    public static void debug(String format, Object ...args){
        if(enabledLevels.contains("DEBUG")){
            System.out.println(MessageFormatter.arrayFormat(format,
args).getMessage());
        }
    }

    public static void debug(List<? extends Object> arr){
        if(enabledLevels.contains("DEBUG")){
            for(int i = 0; i<arr.size(); i++){
                System.out.println(MessageFormatter.
                        format("<{}>{}", i+1,
arr.get(i)).getMessage());
            }
            System.out.println();
        }
    }

    // INFO level
    public static boolean isInfoEnabled(){
```

```
        return enabledLevels.contains("|INFO");
    }

    public static void setInfoEnabled(boolean enabled){
        if(enabled && !isInfoEnabled())
            enabledLevels += "|INFO";
        else if(!enabled && isInfoEnabled())
            enabledLevels = enabledLevels.replace("|INFO", "");
    }

    public static void info(String format, Object ...args){
        if(enabledLevels.contains("INFO")){
            if(logger.isInfoEnabled())
                logger.info(format, args);
            else
               System.err.println(
                   MessageFormatter.arrayFormat(format,
args).getMessage());
        }
    }

    public static void info(List<? extends Object> arr){
        if(enabledLevels.contains("INFO")){
            if(logger.isInfoEnabled()){
                final StringBuilder builder = new StringBuilder();
                builder.append("\n");
                arr.forEach(e -> builder.append(e.toString()+"\n"));
                logger.info(builder.toString());
            }else{
                arr.forEach(System.err::println);
                System.out.println();
            }
        }
    }

    // ERROR level
    public static boolean isErrorEnabled(){
        return enabledLevels.contains("|ERROR");
    }

    public static void setErrorEnabled(boolean enabled){
        if(enabled && !isErrorEnabled())
            enabledLevels += "|ERROR";
        else if(!enabled && isErrorEnabled())
            enabledLevels = enabledLevels.replace("|ERROR", "");
    }

    public static void error(Throwable e){
        if(enabledLevels.contains("ERROR")){
            if(logger.isErrorEnabled())
                logger.error("Stacktrace: ", e);
            else
                e.printStackTrace();
            logToFile(e);
        }
    }

    // FATAL_ERROR level
    public static boolean isFatalErrorEnabled(){
        return enabledLevels.contains("FATAL_ERROR");
    }
```

```
    public static void fatalError(Throwable e){
        if(enabledLevels.contains("FATAL_ERROR")){
            if(!logger.isErrorEnabled())
                logger.error("Stacktrace: ", e);
            else
                e.printStackTrace();

            logToFile(e);
            JOptionPane.showMessageDialog(null, "Erro:
"+e.getMessage()+"\n\n"
                    + "Verifique o arquivo de log para mais
detalhes.", "Ocorreu um erro!",
                    JOptionPane.ERROR_MESSAGE);

            System.exit(-1);
        }
    }

    /**
     * Salva os dados de um Erro no arquivo de log.
     *
     * @param e          Erro que se quer salvar.
     * @return           <b>True</b> caso consiga salvar o arquivo
ou<br>
     *                   <b>False</b> caso contrario.
     */
    private static boolean logToFile(Throwable e) {
        String content = CommonUtil.getCurrentTime() +" |
Stacktrace: ";
        content +=
"\n"+CommonUtil.exceptionStacktraceToString(e)+"\n";
        return CommonUtil.appendToFile(path, content);
    }

    // Bloco estático
    static {
        if(CommonConfiguration.getInstance() != null)
            enabledLevels +=
CommonConfiguration.getInstance().getLogLevels().toUpperCase();
        // Pequena gambiarra para setar o nivel de log do Crawler4J
        ch.qos.logback.classic.Level level =
enabledLevels.contains("|INFO") ?
ch.qos.logback.classic.Level.INFO :
            ch.qos.logback.classic.Level.WARN;
        ch.qos.logback.classic.Logger root =
(ch.qos.logback.classic.Logger)

LoggerFactory.getLogger(ch.qos.logback.classic.Logger.ROOT_LOGGER_NA
ME);
        root.setLevel(level);
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/util/CommonUtil.java

```
package br.ufsc.tcc.common.util;

import java.awt.Desktop;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.PrintStream;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.OpenOption;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.List;
import java.util.regex.Pattern;
import org.json.JSONException;
import org.json.JSONObject;
import org.jsoup.nodes.Element;
import org.jsoup.nodes.Node;
import com.google.common.base.CharMatcher;
import br.ufsc.tcc.common.model.DeweyExt;
import br.ufsc.tcc.common.model.MyNode;

public class CommonUtil {
    private static final List<String> singleComps =
            Arrays.asList("textarea",
                    "select",
                    "input[type=text]",
                    "input[type=number]",
                    "input[type=date]",
                    "input[type=email]",
                    "input[type=tel]",
                    "input[type=time]",
                    "input[type=url]",
                    "input[type=range]");
    private static final List<String> multiComps =
            Arrays.asList("input[type=radio]",
                    "input[type=checkbox]");
    private static final List<String> allComps =
            Arrays.asList("input[type=text]",
                    "input[type=number]",
                    "input[type=date]",
                    "input[type=email]",
                    "input[type=tel]",
                    "input[type=time]",
                    "input[type=url]",
                    "input[type=range]",
                    "input[type=radio]",
                    "input[type=checkbox]",
                    "textarea",
                    "select",
                    "option");
    public final static String REQUIRED_REGEX =
            "\\(?(required|resposta
exigida|requerido|\\*obrigat(ó|o)rio|\\*)\\)?";

    // Getters e Setters
    public static List<String> getSingleComps(){
        return singleComps;
    }
```

```
    public static List<String> getMultiComps(){
        return multiComps;
    }

    public static List<String> getAllComps(){
        return allComps;
    }

    // Demais métodos
    /**
     * Le todo o conteúdo de um arquivo e o retorna em
     * forma de uma String.
     *
     * @param file              Arquivo que se quer o conteúdo.
     * @return                  Uma String que contém todo o
conteúdo do arquivo
     *                          passado.
     */
    public static String readFile(String file){
        String content = "";
        try {
            Path path = Paths.get(file);

            content = new String(Files.readAllBytes(path),
                    Charset.forName("UTF-8"));
        } catch (IOException e) {
            CommonLogger.fatalError(e);
        }
        return content;
    }

    /**
     * Le todo o conteúdo de um recurso e o retorna em
     * forma de uma String.
     *
     * @param resource              Recurso que se quer o conteúdo.
     * @return                      Uma String que contém todo o
conteúdo do recurso
     *                              passado.
     */
    public static String readResource(String resource){
        String content = "";
        try {
            ClassLoader cl =
Thread.currentThread().getContextClassLoader();
            if(cl == null)
                cl = CommonUtil.class.getClassLoader();

            // Só para ter certeza...
            InputStream input = cl.getResourceAsStream(resource);
            if(input == null){
                input = cl.getResourceAsStream("/" +resource);
                if(input == null){
                    input = cl.getResourceAsStream("resources/"
+resource);
                    if(input == null){
                        input = cl.getResourceAsStream("/resources/"
+resource);
                        if(input == null)
                            throw new IOException("Resource not
found ("+resource+").");
```

```
                    }
                }
            }

            ByteArrayOutputStream result = new
ByteArrayOutputStream();
            byte[] buffer = new byte[1024];
            int length;
            while ((length = input.read(buffer)) != -1) {
                result.write(buffer, 0, length);
            }

            content = result.toString("UTF-8");
        } catch (IOException e) {
            CommonLogger.fatalError(e);
        }
        return content;
    }

    /**
     * Escreve o conteúdo passado em um arquivo.
     *
     * @param path          Caminho para o arquivo que se quer
salvar.
     * @param content       Conteúdo que se quer salvar.
     * @param options       Opções especificando como o arquivo será
aberto, utilize a classe StandardOpenOption.
     * @return              <b>TRUE</b> caso seja possivel escrever
o arquivo, ou</br>
     *                      <b>FALSE</b> caso contrario.
     */
    public static boolean writeFile(String path, String content,
OpenOption ... options){
        try {
            Files.write(Paths.get(path),
content.getBytes(StandardCharsets.UTF_8), options);
            return true;
        } catch (IOException e) {
            CommonLogger.error(e);
            return false;
        }
    }

    /**
     * Adiciona o counteúdo passado a um arquivo.
     *
     * @param path              Caminho para o arquivo que se quer
adicionar conteúdo.
     * @param content           Conteúdo que se quer adicionar.
     * @return                  <b>TRUE</b> caso seja possivel
escrever o arquivo, ou</br>
     *                          <b>FALSE</b> caso contrario.
     */
    public static boolean appendToFile(String path, String content){
        return writeFile(path, content, StandardOpenOption.CREATE,
                StandardOpenOption.APPEND);
    }

    /**
     * Abre um arquivo utilizando o programa padrão do sistema
operacional.
```

```
     *
     * @param path      Caminho para o arquivo que se quer abrir.
     * @return          <b>TRUE</b> caso seja possivel abrir o
arquivo, ou</br>
     *                  <b>FALSE</b> caso contrario.
     */
    public static boolean openFile(String path) {
        return openFile(new File(path));
    }

    /**
     * Abre um arquivo utilizando o programa padrão do sistema
operacional.
     *
     * @param file      Arquivo que se quer abrir.
     * @return          <b>TRUE</b> caso seja possivel abrir o
arquivo, ou</br>
     *                  <b>FALSE</b> caso contrario.
     */
    public static boolean openFile(final File file) {
        if (!Desktop.isDesktopSupported())
            return false;

        Desktop desktop = Desktop.getDesktop();
        if(!desktop.isSupported(Desktop.Action.OPEN))
            return false;

        try {
            desktop.open(file);
        } catch (IOException e) {
            CommonLogger.error(e);
            return false;
        }
        return true;
    }

    /**
     * Retorna um JSONObject da String {@code content} passada.
     *
     * @param content   Conteúdo que se quer gerar um JSONObject.
     * @return          Um JSONObject do conteúdo passado.
     */
    public static JSONObject parseJson(String content){
        JSONObject obj = null;
        try{
            obj = new JSONObject(content);
        }catch(JSONException e){
            CommonLogger.fatalError(e);
        }
        return obj;
    }

    /**
     * Remove espaços antes e depois da String {@code str},
incluindo os
     * caracteres '\u00a0' e '\ufeff'.
     *
     * @param str   String que se quer remover os espaços em volta.
     * @return      A String str sem espaços em volta.
     */
    public static String trim(String str){
```

```
        return str.replaceAll("\u00a0", " ")
                .replaceAll(" ", " ")
                .replaceAll("&#65279", "")
                .replaceAll("\ufeff", "")
                .trim();
    }

    /**
     * Verifica se a String {@code text} inicia com letra maiúscula.
     *
     * @param text      String que se quer verificar.
     * @return          <b>TRUE</b> caso a string comece com letra
maiúscula, <br>
     *                  <b>FALSE</b> caso contrario.
     */
    public static boolean startsWithUpperCase(String text){
        char c = text.charAt(0);
        c = c == '¿' ? text.charAt(1) : c;
        return Character.isUpperCase(c);
    }

    /**
     * Verifica se a String {@code text} inicia com um numero.
     *
     * @param text      String que se quer verificar.
     * @return          <b>TRUE</b> caso a string comece com um
numero, <br>
     *                  <b>FALSE</b> caso contrario.
     */
    public static boolean startsWithDigit(String text){
        return Character.isDigit(text.charAt(0));
    }

    /**
     * Retorna um Timestamp do tempo atual.
     *
     * @return      Timestamp do tempo atual.
     */
    public static Timestamp getCurrentTime() {
        Calendar calendar = Calendar.getInstance();
        return new Timestamp(calendar.getTime().getTime());
    }

    /**
     * Converte um Stacktrace para uma String.
     *
     * @param e         Stacktrace que se quer converter.
     * @return          Uma string contendo o Stacktrace passado.
     */
    public static String exceptionStacktraceToString(Throwable e){
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        PrintStream ps = new PrintStream(baos);
        e.printStackTrace(ps);
        ps.close();
        return baos.toString();
    }

    /**
     * Conta o tamanho do prefixo comum entre {@code d1} e {@code
d2}
     * desconsiderando sinais negativos, '-'.
```

```
     *
     * @param d1
     * @param d2
     * @return          Valor do tamanho do prefixo comum.
     */
    public static int getPrefixLenght(DeweyExt d1, DeweyExt d2){
        return getPrefixLength(d1.getCommonPrefix(d2));
    }

    /**
     * Conta o tamanho do prefixo comum {@code prefix}
     * desconsiderando sinais negativos, '-'.
     *
     * @param prefix
     * @return          Valor do tamanho do prefixo comum.
     */
    public static int getPrefixLength(String prefix){
        int n = prefix.length();
        n -= CharMatcher.is('-').countIn(prefix);
        return n;
    }

    /**
     * Verifica se a String {@code str} passada é apenas uma
palavra, ou seja,
     * se ela não contém espaços e quebra de linhas.
     *
     * @param str       String que se quer verificar.
     * @return          <b>TRUE</b> caso a string seja apenas uma
palavra, <br>
     *                  <b>FALSE</b> caso contrario.
     */
    public static boolean isOnlyOneWord(String str) {
        return CharMatcher.is(' ').countIn(str) == 0 &&
                CharMatcher.is('\n').countIn(str) == 0;
    }

    /**
     * Adiciona zeros a frente de um numero para deixa-los com
     * exatamente 3 caracteres. Por exemplo, 1 vira 001, 10 vira 010
e
     * 100 continua como 100.
     *
     * @param i         Numero que precisa de padding.
     * @return          String com exatamente 3 caracteres do numero
{@code i}.
     */
    public static String padNumber(int i){
        return ((i < 0) ? "-" : "") + String.format("%03d",
Math.abs(i));
    }

    /**
     * Retorna uma String que represente o Nodo {@code node}.
     *
     * @param node      Nodo que se quer uma representação.
     * @return          String que represente o nodo passado.
     */
    public static String getNodeRepresentation(Node node){
        if(node == null) return "";
```

```
        String name = node.nodeName();
        switch(name){
        case "#text":
            name = CommonUtil.trim(node.toString());
            break;
        case "input":
            // O tipo padrão de um input é 'text', caso não seja
especificado
            String type = node.attr("type");
            name += "[type="+ (type.isEmpty() ? "text" :
type.toLowerCase()) +"]";
            break;
        case "img":
            name += "[alt="+ node.attr("alt") +"]";
            break;
        default:
            break;
        }
        return name;
    }

    public static String removeRequiredAndTrim(String text){
        if(text.isEmpty()) return text;

        text = CommonUtil.trim(text);
        return CommonUtil.trim(text.replaceAll(
                String.format("(?ism)(^%s|%s$)",
                        REQUIRED_REGEX, REQUIRED_REGEX), ""));
    }

    /**
     * Verifica se o nodo passado é um componente de formulário,
     * uma imagem ou um texto.
     *
     * @param node      Nodo que se quer verificar.
     * @return          <b>True</b> caso este nodo seja um
componente de formulário,
     *                  uma imagem ou um texto ou<br>
     *                  <b>False</b> caso contrário.
     */
    public static boolean isCompImgOrTextNode(Node node){
        if(node == null) return false;

        String name =
removeRequiredAndTrim(getNodeRepresentation(node));
        return (!name.equals("") && node.nodeName().equals("#text"))
||
                ((name.startsWith("img") || allComps.contains(name))
&&
                        !node.attr("type").equals("hidden"));
    }

    /**
     * Compara o {@code regex} passado com cada linha da String
{@code txt}.
     *
     * @param txt
     * @param regex
     * @return              <b>True</b> caso o {@code regex} passado
seja encontrado
```

```
     *                       em alguma das linhas de {@code txt}
ou<br>
     *                       <b>False</b> caso contrario.
     */
    public static boolean matchesWithLineBreak(String txt, String
regex){
        String [] lines = txt.split("\n");
        Pattern p = Pattern.compile(regex,
Pattern.CASE_INSENSITIVE);
        for(String line : lines){
            if(p.matcher(line).matches())
                return true;
        }
        return false;
    }

    /**
     * Verifica se o {@code text} passado esta contido em alguma
linha do {@code txtToCheck}.
     *
     * @param text
     * @param txtToCheck
     * @return                  <b>True</b> caso o {@code text}
passado esteja contido
     *                          em alguma das linhas de {@code
txtToCheck} ou<br>
     *                          <b>False</b> caso contrario.
     */
    public static boolean containsWithLineBreak(String text, String
txtToCheck){
        text = text.toLowerCase();
        String [] lines = txtToCheck.split("\n");
        for(String line : lines){
            if(text.contains(line.toLowerCase()))
                return true;
        }
        return false;
    }

    /**
     * Encontra e retorna todos os nodos de componente de
formulário, imagem ou
     * texto a partir de um nodo {@code root}.
     *
     * @param root          Nodo inicial da busca.
     * @return              Lista com todos os nodos encontrados.
     */
    public static List<MyNode> findCompsImgsAndTexts(Node root) {
        List<MyNode> ret = new ArrayList<>();
        if(root != null)
            findCompsImgsAndTexts(root, "001", ret);
        return ret;
    }

    private static void findCompsImgsAndTexts(Node root, String
dewey, List<MyNode> ret) {
        if(isCompImgOrTextNode(root)){
            MyNode node = new MyNode(root, new DeweyExt(dewey));
            //Agrupa textos que foram separados por <br>
            if(!ret.isEmpty()){
                MyNode last = ret.get(ret.size()-1);
```

```
                if(node.getDewey().equals(last.getDewey()))
                    last.setText(last.getText() +"\n"+
node.getText());
                else
                    ret.add(node);
            }else
                ret.add(node);
        }

        int n = 1;
        List<Node> children = root.childNodes();
        for(int i = 0; i < children.size(); i++){
            Node child = children.get(i);

            // Ignora comentarios, tags 'br' e 'button', blocos e
textos vazios
            if(child.nodeName().equals("br") &&
isBetweenTexts(children, i))
                n--;
            if(!child.nodeName().matches("#comment|br|button") &&
                !trim(child.toString()).isEmpty()
&& !CommonUtil.checkEmptinessOfSomeBlockElems(child)) {
                    findCompsImgsAndTexts(child,
                            dewey +"."+ padNumber(n++),
                            ret);
            }
        }
    }

    /**
     * Verifica se o nodo na posição {@code i} esta entre dois nodos
de texto.
     *
     * @param children
     * @param i
     * @return              <b>True</b> caso o nodo na posição
{@code i} esteja entre
     *                      dois nodos de texto ou<br>
     *                      <b>False</b> caso contrario.
     */
    private static boolean isBetweenTexts(List<Node> children, int
i) {
        if(i-1 < 0 || i+1 >= children.size())
            return false;
        Node c1 = children.get(i-1), c2 = children.get(i+1);
        if(!c1.nodeName().equals("#text") ||
                !c2.nodeName().equals("#text"))
            return false;
        return !trim(c1.toString()).isEmpty()
&& !trim(c2.toString()).isEmpty();
    }

    /**
     * Verifica se alguns tipos de elemento de bloco estão vazios,
ou seja,
     * que não possuem nenhum elemento filho ou possuam texto
vazio.<br>
     * Os elementos verificados são: < p >, < span >, < th >, < div
> e < a >.
     *
     * @param el
```

```
     * @return               <b>True</b> caso o nodo {@code el}
passado esteja vazio ou<br>
     *                       <b>False</b> caso contrario.
     */
    private static boolean checkEmptinessOfSomeBlockElems(Node el){
        String name = el.nodeName();
        int nodeSize = el.childNodeSize();

        if(name.equals("span") && nodeSize == 1 &&
el.childNode(0).nodeName().equals("span")) {
            el = el.childNode(0);
            name = el.nodeName();
            nodeSize = el.childNodeSize();
        }

        boolean notHasElem = (nodeSize == 0 || (nodeSize == 1 &&
el.childNode(0).nodeName().equals("#text")));
        if(notHasElem && (name.matches("(p|span|th|div)") ||
                (name.equals("a") && !el.hasAttr("href")))){
            String content = el.toString();
            if(el instanceof Element) {
                content = ((Element) el).text();
            }
            return trim(content).isEmpty();
        }
        return false;
    }
}
```

questionnaires_common_lib/br/ufsc/tcc/common/util/DistanceMatrix.java

```
package br.ufsc.tcc.common.util;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Set;
import org.json.JSONObject;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.DeweyExt;
import br.ufsc.tcc.common.model.MyNode;

/**
 * Classe responsável por guardar as distâncias entre os Nodos,
 * serve como uma 'cache' para os valores.
 *
 * @author Gilney N. Mathias
 */
public class DistanceMatrix {
    private static int MAX_WIDTH = 0;
    private static int MAX_HEIGHT = 0;
    private static int MAX_MAXHEIGHT = 0;
    private HashMap<String, HashMap<String, DeweyExt>> distances;

    // Construtores
    public DistanceMatrix(){
        this.distances = new HashMap<>();
    }

    // Getters e Setters
    public DeweyExt getDist(MyNode n1, MyNode n2){
        if(n1 == null || n2 == null) return null;
```

```
        String v1 = n1.getDewey().getValue(),
                v2 = n2.getDewey().getValue();

        if(!this.distances.containsKey(v1))
            this.distances.put(v1, new HashMap<>());
        if(!this.distances.get(v1).containsKey(v2))
            this.calculateDist(n1.getDewey(), n2.getDewey());

        return this.distances.get(v1).get(v2);
    }

    public Set<String> keySet(){
        return this.distances.keySet();
    }

    public ArrayList<String> keyList(){
        return new ArrayList<String>(distances.keySet());
    }

    // Demais métodos
    /**
     * Verifica se os Clusters estão perto um do outro olhando
     * a distância do ultimo Nodo do Cluster {@code c1} e do
     * primeiro Nodo do Cluster {@code c2}.
     *
     * @param c1        Primeiro Cluster.
     * @param c2        Segundo Cluster.
     * @return          <b>True</b> caso {@code c1} seja considerado
'perto' de {@code c2} ou<br>
     *                  <False> caso contrario.
     */
    public boolean areNear(Cluster c1, Cluster c2){
        if(c1 != null && c2 != null)
            return this.areNear(c1.last(), c2.first());
        return false;
    }

    /**
     * Verifica se os Nodos {@code n1} e {@code n2} estão
     * perto um do outro.
     *
     * @param c1        Primeiro Nodo.
     * @param c2        Segundo Nodo.
     * @return          <b>True</b> caso {@code n1} seja considerado
'perto' de {@code n2} ou<br>
     *                  <False> caso contrario.
     */
    public boolean areNear(MyNode n1, MyNode n2){
        if(n1 != null && n2 != null){
            DeweyExt dist = this.getDist(n1, n2);
            return dist.getHeight() <= MAX_HEIGHT &&
                    dist.getMaxHeight() <= MAX_MAXHEIGHT &&
                    dist.getWidth() <= MAX_WIDTH;
        }
        return false;
    }

    public void clear(){
        this.distances.clear();
    }
```

```
    private void calculateDist(DeweyExt d1, DeweyExt d2) {
        DeweyExt dist = d1.distanceOf(d2);
        this.distances.get(d1.getValue()).put(d2.getValue(), dist);
    }

    // Métodos/Blocos estáticos
    static {
        //Load parameters
        JSONObject p =
CommonConfiguration.getInstance().getParameters(), tmp = null;

        tmp = p.getJSONObject("distBetweenNearNodes");
        MAX_WIDTH = tmp.getInt("width");
        MAX_HEIGHT = tmp.getInt("height");
        MAX_MAXHEIGHT = tmp.getInt("maxHeight");

        CommonLogger.debug("DistMatrix:> Static block executed!");
    }
}
```

questionnaires_crawler/br/ufsc/tcc/crawler/checker/ClusterBuilder.java

```
package br.ufsc.tcc.crawler.checker;

import java.util.ArrayList;
import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.util.DistanceMatrix;

/**
 * Classe responsável por agrupar Nodos e/ou Clusters para formar
 * outros Clusters.
 *
 * @author Gilney N. Mathias
 */
public class ClusterBuilder {
    // Construtores
    public ClusterBuilder(){}

    // Demais métodos
    /**
     * Método responsável por 'construir' uma lista de Clusters a
partir da
     * lista de nodos, {@code nodes}, e da matriz de distâncias,
{@code distMatrix},
     * passados. <br>
     * Os nodos são primeiro agrupados por distância e em seguida
são utilizadas algumas
     * heurísticas para tentar melhorar os agrupamentros.
     *
     * @param nodes
     * @param distMatrix
     * @return                  Lista de Clusters criada a partir
dos nodos e da
     *                          matriz de distância passados.
     */
    public List<Cluster> build(List<MyNode> nodes, DistanceMatrix
distMatrix){
        List<Cluster> clusters = groupNearNodes(nodes, distMatrix);
        groupClustersByHeuristics(clusters);
        return clusters;
```

```
    }

    /**
     * Agrupa os nodos que estão próximos uns dos outros.
     *
     * @param nodes
     * @param distMatrix
     * @return                  Lista de Clusters de nodos próximos
uns dos outros.
     */
    private List<Cluster> groupNearNodes(List<MyNode> nodes,
DistanceMatrix distMatrix) {
        Cluster cTmp = new Cluster();
        ArrayList<Cluster> ret = new ArrayList<>();

        for(MyNode nTmp : nodes){
            if(!cTmp.isEmpty() && !distMatrix.areNear(cTmp.last(),
nTmp)){
                ret.add(cTmp);
                cTmp = new Cluster();
            }
            cTmp.add(nTmp);
        }
        if(!cTmp.isEmpty())
            ret.add(cTmp);
        return ret;
    }

    /**
     * Agrupa os Clusters passados utilizando algumas heurísticas
que podem
     * ser encontradas no método {@link #shouldGroup(Cluster,
Cluster, Cluster) shouldGroup}.
     *
     * @param clusters
     */
    private void groupClustersByHeuristics(List<Cluster> clusters) {
        Cluster tmp1, tmp2, tmp3;
        int i, size;

        do{
            size = clusters.size();
            i = 0;

            while(i < clusters.size()-1){
                int tmpSize = clusters.size();

                tmp1 = clusters.get(i);
                tmp2 = clusters.get(i+1);
                tmp3 = i+2 < tmpSize ? clusters.get(i+2) : null;

                if(shouldGroup(tmp1, tmp2, tmp3)){
                    tmp1.join(tmp2);
                    clusters.remove(i+1);
                    i--;
                }
                i++;
            }
        }while(size != clusters.size());
    }
```

```
    /**
     * Faz uso de algumas heurísticas para verifica se
     * os Clusters {@code c1} e {@code c2} devem ser unidos.<br>
     * As heurísticas utilizadas são:
     * <ul>
     *  <li>Deve-se juntar grupos de input</li>
     *  <li>Deve-se juntar grupos de options seguidos</li>
     *  <li>Deve-se juntar padrões de clusters de texto seguidos
     *      de clusters de elementos ou clusters com apenas 1
elemento</li>
     * </ul>
     * O Cluster {@code c3} passado é utilizado na verificação da
ultima heurística.
     *
     * @param c1
     * @param c2
     * @param c3
     * @return          <b>TRUE</b> caso os clusters {@code c1} e
{@code c2} devam
     *                  ser unidos ou<br>
     *                  <b>FALSE</b> caso contrario.
     */
    private boolean shouldGroup(Cluster c1, Cluster c2, Cluster c3)
{
        //Deve-se juntar grupos de input
        //PS: pode ter uma img na frente
        final String starterInputRegex =
"(?s)^(img\\[alt=.*\\]\\n)?"
                + "(input\\[type=.+\\]).*";
        String c1Text = c1.getAllNodesText(),
                c2Text = c2.getAllNodesText();
        if(c1Text.matches(starterInputRegex) &&
c2Text.matches(starterInputRegex))
            return true;

        //Deve-se juntar grupos de options seguidos
        if(c1.size() >= 2){
            MyNode opt1 = c1.get(c1.size()-2), opt2 = c2.first();
            if(opt1.isA("OPTION") && opt2.isA("OPTION"))
                return true;
        }

        //Deve-se juntar padrões de cluster texto seguidos de
cluster elementos OU
        //cluster com 1 elemento apenas (casos raros?)
        //PS: deve-se priorizar a união de elementos a frente desta
regra
        //    -por isso o c3-
        if((c3 == null || c3.first().isText()) &&
                ((c1.isAllText() && !c2.first().isText()) ||
                        (!c2.first().isText() && c2.size() == 1)))
            return true;
        return false;
    }
}
```

questionnaires_crawler/br/ufsc/tcc/crawler/checker/RulesChecker.java

```
package br.ufsc.tcc.crawler.checker;

import java.util.ArrayList;
import java.util.List;
```

```java
import java.util.regex.Pattern;
import org.json.JSONObject;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.DeweyExt;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.model.MyNodeType;
import br.ufsc.tcc.common.util.CommonConfiguration;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.common.util.CommonUtil;
import br.ufsc.tcc.common.util.DistanceMatrix;
import edu.uci.ics.crawler4j.parser.HtmlParseData;

public class RulesChecker {
    private static Pattern SURVEY_WORDS_REGEX = null;
    private static Pattern PHRASES_TO_IGNORE_REGEX = null;
    private static int MIN_COMPS_IN_ONE_CLUSTER = 0;
    private static int MIN_CLUSTERS_WITH_COMP = 0;
    private static int MAX_CLUSTERS_BETWEEN_CLUSTERS_WITH_COMP = 0;
    private static int HEIGHT_BETWEEN_QUESTIONS = 0;
    private DistanceMatrix distMatrix;
    private ClusterBuilder builder;
    private boolean lastWasAMatrix;

    // Construtores
    public RulesChecker(){
        this.distMatrix = new DistanceMatrix();
        this.builder = new ClusterBuilder();
        this.lastWasAMatrix = false;
    }

    // Demais métodos
    /**
     * Método intermediário que converte o html passado para um
     * Document, utilizando a biblioteca {@code JSoup}, e o passa
     * para o método {@link #shouldSave(Document) shouldSave}.
     *
     * @param htmlParseData
     * @return
     */
    public boolean shouldSave(HtmlParseData htmlParseData) {
        String html = htmlParseData.getHtml();
        Document doc = Jsoup.parse(html);
        return this.shouldSave(doc);
    }

    /**
     * Método responsável por verificar se o link da pagina
representada pelo
     * Document {@code doc} passado deve ser salvo no banco de
dados.<br>
     * O método inicialmente constrói uma lista de Clusters a partir
do Document passado
     * e então chama o método {@link #hasQuestionnaire(List)
hasQuestionnaire} para
     * verificar se é possivel afirmar a presença de um questionário
nesta pagina.
     *
```

```
     * @param doc
     * @return              <b>TRUE</b> caso deva-se salvar o link
da pagina representada
     *                      pelo Document passado ou,<br>
     *                      <b>FALSE</b> caso contrario.
     */
    public boolean shouldSave(Document doc){
        Elements tmp = doc.select("body");
        if(tmp.isEmpty()) return false;

        Element root = tmp.get(0);

        if(!SURVEY_WORDS_REGEX.matcher(root.text()).matches() &&
                !SURVEY_WORDS_REGEX.matcher(doc.title()).matches()){
            this.distMatrix.clear();
            return false;
        }

        List<MyNode> nodes = CommonUtil.findCompsImgsAndTexts(root);
        List<Cluster> clusters = this.builder.build(nodes,
this.distMatrix);

        CommonLogger.debug(clusters);

        boolean ret = hasQuestionnaire(clusters);
        this.distMatrix.clear();

        return ret;
    }

    /**
     * Faz uso de algumas heurísticas para verifica se é possível
afirmar a
     * presença de um questionário a partir da lista de Clusters
passada.<br>
     * As heurísticas utilizadas são:
     * <ul>
     *  <li>Um questionario deve ter pelo menos 1 cluster/questão
com X componentes ou
     *      Y clusters/questões com pelo menos 1 componente.</li>
     *  <li>Um questionário não deve ter mais do que Z clusters em
sequencia sem
     *      componentes.</li>
     *  <li>Todo cluster/questão deve ter pelo menos 1 componente e
não deve possuir
     *      frases contidas no: PHRASES_TO_IGNORE_REGEX.</li>
     * </ul>
     *
     * @param clusters
     * @return              <b>TRUE</b> caso seja possivel afirmar a
presença
     *                      de um questionário a partir desta lista
de Clusters ou,<br>
     *                      <b>FALSE</b> caso contrario.
     */
    private boolean hasQuestionnaire(List<Cluster> clusters) {
        //Contador de componentes, Contador de 'questões', Contador
de clusters sem componentes
        double cCount = 0.0;
        int qCount = 0, ncCount = 0;
        Cluster lastCluster = null;
```

```
        //Um questionario deve ter pelo menos 1 cluster com X
componentes ou
        //X clusters com pelo menos 1 componente
        for(int i = 0; i<clusters.size(); i++){
            Cluster c = clusters.get(i);
            cCount = getCountOfComps(c);

            CommonLogger.debug("<{}>cCount: {}; qCount: {};", i+1,
cCount, qCount);

            //Atualiza o contador de clusters sem componentes e
verifica se ja
            //chego no limite
            if(cCount == 0){
                ncCount++;
                if(qCount > 0 && ncCount ==
MAX_CLUSTERS_BETWEEN_CLUSTERS_WITH_COMP)
                    qCount = 0;
            }

            //Toda questão deve ter pelo menos 1 componente e não
deve possuir
            //frases contidas no: PHRASES_TO_IGNORE_REGEX
            if(cCount >= 1){
                if(cCount < MIN_COMPS_IN_ONE_CLUSTER){
                    if(lastCluster != null){
                        if(!isStartingANewQuestionnaire(lastCluster,
c))
                            qCount++;
                        else
                            qCount = 1;
                    }else
                        qCount++;
                }else{
                    CommonLogger.debug("\n\t\t===> Minimo {}
componentes em um cluster!", MIN_COMPS_IN_ONE_CLUSTER);
                    return true;
                }

                if(qCount == MIN_CLUSTERS_WITH_COMP){
                    CommonLogger.debug("\n\t\t===> Minimo {}
clusters com componentes!", MIN_CLUSTERS_WITH_COMP);
                    return true;
                }
            }
            if(cCount >= 0.5){
                ncCount = 0;
                lastCluster = c;
            }
        }
        return false;
    }

    /**
     * Faz uso de algumas heurísticas para tentar determinar a
quantidade de
     * componentes de formulário no Cluster {@code c} passado.
     *
     * @param c
     * @return
```

```
     */
    private double getCountOfComps(Cluster c){
        // Os clusters de perguntas sempre começam com um texto ou
imagem !?
        if(c.first().isComponent())
            return 0.0;

        if(c.isAllTextOrImg())
            this.lastWasAMatrix = false;

        double count = 0.0;
        boolean hasDescriptionAbove = false, hasQuestionAbove =
false, isMultiComp = false;
        String text = "";
        ArrayList<MyNode> nodes = c.getGroup();

        for(int i = 0; i < nodes.size(); i++){
            MyNode node = nodes.get(i);
            if(node.isText()){

if(PHRASES_TO_IGNORE_REGEX.matcher(node.getText()).matches()){
                    count = -1;
                    break;
                }
                text = fixText(node.getText());
                hasDescriptionAbove = isLikelyADescription(text,
isMultiComp);
                hasQuestionAbove = hasDescriptionAbove &&
isLikelyAQuestion(text);
                isMultiComp = false;
            }else if(node.isComponent() && !node.isA("OPTION")){
                if(hasDescriptionAbove){
                    isMultiComp =
CommonUtil.getMultiComps().contains(node.getText());
                    if(isMultiComp){
                        if(this.isRatingOrMatrix(nodes, i)){
                            if(this.lastWasAMatrix)
                                return 0.5;
                            this.lastWasAMatrix = true;
                            return 1.0;
                        }else if(hasQuestionAbove)
                            count += 0.75;
                        else
                            count += 0.25;
                    }else if(hasQuestionAbove)
                        count += 1.0;
                }
                hasDescriptionAbove = false;
                hasQuestionAbove = false;
                this.lastWasAMatrix = false;
            }
        }
        return count;
    }

    private boolean isRatingOrMatrix(ArrayList<MyNode> nodes, int i)
{
        MyNodeType type = nodes.get(i).getType();

        int count = 1;
        for(int j = i+1; j<nodes.size(); j++){
```

```
            if(nodes.get(j).getType() == type)
                count++;
            else
                break;
        }
        return count >= 3;
    }

    private boolean isLikelyADescription(String text, boolean
lastCompIsMultiComp){
        return ((!lastCompIsMultiComp && text.length() >= 4 &&
text.contains(" ")) || lastCompIsMultiComp) &&
                (CommonUtil.startsWithUpperCase(text) ||
CommonUtil.startsWithDigit(text)))
                ||
text.matches("(\\d{1,3}(\\s{1,2})?(\\.|\\:|\\)|\\-)?)");
    }

    private boolean isLikelyAQuestion(String text){
        return text.contains("?") || text.contains(":") ||
                text.matches("(?m)^(\\d{1,4}).*(\\.)$");//começa com
numero e termina com '.'
    }

    private String fixText(String text){
        text = text.replaceAll("( )?\\*", "");
        text = text.endsWith(" :") ? text.substring(0,
text.length()-2) + ":" : text;
        return text;
    }

    private boolean isStartingANewQuestionnaire(Cluster lastCluster,
Cluster newCluster) {
        DeweyExt dist = this.distMatrix.getDist(lastCluster.last(),
newCluster.first());
        if(dist.getHeight() > HEIGHT_BETWEEN_QUESTIONS)
            return true;

        //Verifica se o 1* container, depois do BODY, é diferente
        DeweyExt d1 = lastCluster.last().getDewey(), d2 =
newCluster.first().getDewey();
        return d1.getNumbers().get(1) != d2.getNumbers().get(1);
    }

    // Métodos/Blocos estáticos
    static {
        //Load parameters
        JSONObject p =
CommonConfiguration.getInstance().getParameters(), tmp = null;

        String txtTmp = p.getString("surveyWordsRegex");
        SURVEY_WORDS_REGEX = Pattern.compile(txtTmp,
                Pattern.CASE_INSENSITIVE|Pattern.MULTILINE);

        txtTmp = p.getString("phrasesToIgnoreRegex");
        PHRASES_TO_IGNORE_REGEX = Pattern.compile(txtTmp,
                Pattern.CASE_INSENSITIVE);

        MIN_COMPS_IN_ONE_CLUSTER = p.getInt("minCompsInOneCluster");
        MIN_CLUSTERS_WITH_COMP = p.getInt("minClustersWithComp");
```

```
        MAX_CLUSTERS_BETWEEN_CLUSTERS_WITH_COMP =
p.getInt("maxClustersBetweenClustersWithComp");

        tmp = p.getJSONObject("distBetweenNearQuestions");
        HEIGHT_BETWEEN_QUESTIONS = tmp.optInt("height");

        CommonLogger.debug("RulesChecker:> Static block executed!");
    }

}
```

questionnaires_crawler/br/ufsc/tcc/crawler/crawler/Crawler.java

```
package br.ufsc.tcc.crawler.crawler;

import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
import java.util.regex.Pattern;
import org.json.JSONObject;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import
br.ufsc.tcc.common.database.manager.PossivelQuestionarioManager;
import br.ufsc.tcc.common.model.PossivelQuestionario;
import br.ufsc.tcc.common.util.CommonConfiguration;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.crawler.checker.RulesChecker;
import edu.uci.ics.crawler4j.crawler.Page;
import edu.uci.ics.crawler4j.crawler.WebCrawler;
import edu.uci.ics.crawler4j.parser.HtmlParseData;
import edu.uci.ics.crawler4j.url.WebURL;

public class Crawler extends WebCrawler {
    private static Pattern EXCLUDED_EXTENSIONS_REGEX;
    private static Pattern EXCLUDED_DOMAINS_REGEX;
    private static Pattern EXCLUDED_LANGUAGES_REGEX;
    private BasicConnection conn;
    private PossivelQuestionarioManager pqManager;
    private RulesChecker checker;

    @Override
    public void onStart() {
        this.conn = new
BasicConnection(CommonConfiguration.getInstance().getCrawlerDatabase
Configs());
        this.pqManager = new PossivelQuestionarioManager(this.conn,
false);
        this.checker = new RulesChecker();
    }

    @Override
    public void onBeforeExit() {
        if(this.conn != null)
            this.conn.close();
    }

    @Override
    protected WebURL handleUrlBeforeProcess(WebURL curURL) {
        String href = curURL.getURL();
        //Pequena gambiarra para esse site especifico...
        if(href.matches(".*search\\.lycos\\.com/b(njs)?\\.php.+")) {
            href = href.substring(href.indexOf("&as=")+4,
href.length()) +"/";
```

```
            try {
                href = URLDecoder.decode(href, "utf-8");
                curURL.setURL(href);
            } catch (UnsupportedEncodingException e) {}
        }
        return curURL;
    }

    @Override
    public boolean shouldVisit(Page referringPage, WebURL url) {
        String href = url.getURL();

        if(!EXCLUDED_LANGUAGES_REGEX.toString().isEmpty() &&
                this.isOfExcludedLanguage(url))
            return false;

        // Retira o http e https
        href = href.replaceAll("^((http|https)://)", "");

        // Retira opções de requisições GET
        int index = href.indexOf("?");
        if(index > 0)
            href = href.substring(0, index);

        // Verifica o filtro e os dominios
        return !EXCLUDED_EXTENSIONS_REGEX.matcher(href).matches() &&
                !EXCLUDED_DOMAINS_REGEX.matcher(href).matches();
    }

    /**
     * Verifica se a {@code url} passada é de uma língua contida no
EXCLUDED_LANGUAGES_REGEX.
     *
     * @param url
     * @return              <b>TRUE</b> caso a url passada seja de
uma língua não permitida ou,<br>
     *                      <b>FALSE</b> caso contrario.
     */
    private boolean isOfExcludedLanguage(WebURL url) {
        //Link útil:
        //  https://en.wikipedia.org/wiki/List_of_Internet_top-
level_domains

        //Ex: http://www.surveymonkey.de
        String tmp = url.getDomain();
        int idx = tmp.lastIndexOf('.');
        String language = tmp.substring(idx+1, tmp.length());

        if(EXCLUDED_LANGUAGES_REGEX.matcher(language).matches())
            return true;

        //Ex: http://sv.surveymonkey.com
        tmp = url.getSubDomain();
        idx = tmp.indexOf('.');
        idx = idx < 1 ? tmp.length() : idx;
        language = tmp.substring(0, idx);

        if(EXCLUDED_LANGUAGES_REGEX.matcher(language).matches())
            return true;

        //Ex: https://surveynuts.com/en
```

```
        tmp = url.getPath();
        tmp = !tmp.isEmpty() ? tmp.substring(1) : tmp;//retira o 1*
'/'
        idx = tmp.indexOf('/');
        idx = idx < 1 ? tmp.length() : idx;
        language = tmp.substring(0, idx);

        if(EXCLUDED_LANGUAGES_REGEX.matcher(language).matches())
            return true;
        else
            return false;
    }

    @Override
    public void visit(Page page) {
        WebURL url = page.getWebURL();

        // Verifica se o link ja foi extraido
        // PS: BloomFilter pode gerar falso positivos, por isso
deve-se
        //     checar o BD caso ele retorne true
        if(PossivelQuestionarioManager.linkWasSaved(url.getURL())) {
            try {
                if(this.pqManager.containsLink(url.getURL()))
                    return;
            } catch (Exception e) {
                CommonLogger.error(e);
            }
        }

        String contentType = page.getContentType();
        if(contentType != null && contentType.contains("html") &&
                page.getParseData() instanceof HtmlParseData){

            HtmlParseData htmlParseData = (HtmlParseData)
page.getParseData();
            if(checker.shouldSave(htmlParseData)){

System.out.println("\t<"+Thread.currentThread().getName()+">Possivel
Questionario: " +url.getURL());

                PossivelQuestionario pq = new PossivelQuestionario(
                        url.getURL(), htmlParseData.getTitle());
                try {
                    pqManager.save(pq);
                } catch (Exception e) {
                    CommonLogger.error(e);
                }
            }else{

System.out.println("<"+Thread.currentThread().getName()+">URL: "
+url.getURL());
            }
        }
    }

    @Override
    protected void onUnhandledException(WebURL webUrl, Throwable e){
        String urlStr = (webUrl == null ? "NULL" : webUrl.getURL());
        CommonLogger.info("Unhandled exception while fetching {}:
{}", urlStr, e.getMessage());
```

```
        CommonLogger.error(e);
    }

    @Override
    protected void onPageBiggerThanMaxSize(String urlStr, long
pageSize) {
        CommonLogger.info("Skipping a URL: {} which was bigger
( {} ) than max allowed size", urlStr, pageSize);
    }

    @Override
    protected void onUnexpectedStatusCode(String urlStr, int
statusCode, String contentType, String description) {
        CommonLogger.info("Skipping URL: {}, StatusCode: {}, {},
{}", urlStr, statusCode, contentType, description);
    }

    @Override
    protected void onContentFetchError(WebURL webUrl) {
        CommonLogger.info("Can't fetch content of: {}",
webUrl.getURL());
    }

    @Override
    protected void onParseError(WebURL webUrl) {
        CommonLogger.info("Parsing error of: {}", webUrl.getURL());
    }

    @Override
    protected void onRedirectedStatusCode(Page page) {
        //Subclasses can override this to add their custom
functionality
        //CommonLogger.info("Redirected to: " +page.getWebURL());
    }

    // Métodos/Blocos estáticos
    static {
        JSONObject tmp =
CommonConfiguration.getInstance().getCrawlerConfigs();
        EXCLUDED_EXTENSIONS_REGEX =
Pattern.compile(tmp.optString("excludedFilesExtensions"));
        EXCLUDED_DOMAINS_REGEX =
Pattern.compile(tmp.optString("excludedDomains"));
        EXCLUDED_LANGUAGES_REGEX =
Pattern.compile(tmp.optString("excludedLanguages"));
        CommonLogger.debug("Crawler:> Static block executed!");
    }
}
```

questionnaires_crawler/br/ufsc/tcc/crawler/main/Main.java

```
package br.ufsc.tcc.crawler.main;

import java.net.URL;
import java.util.Set;
import javax.swing.JOptionPane;
import org.json.JSONArray;
import org.json.JSONObject;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import br.ufsc.tcc.common.crawler.CrawlerController;
import br.ufsc.tcc.common.database.connection.BasicConnection;
```

```
import
br.ufsc.tcc.common.database.manager.PossivelQuestionarioManager;
import br.ufsc.tcc.common.model.PossivelQuestionario;
import br.ufsc.tcc.common.util.CommonConfiguration;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.common.util.CommonUtil;
import br.ufsc.tcc.crawler.checker.RulesChecker;
import br.ufsc.tcc.crawler.crawler.Crawler;
import br.ufsc.tcc.crawler.util.Configuration;

public class Main {
    private static String outputJsonPath =
"./possiveis_questionarios.json";

    public static void main(String[] args) {
        // Carrega as configurações do projeto
        System.out.println("Carregando arquivo de configuracao...");
        CommonConfiguration.setInstance(new Configuration());

        // Inicializa a aplicação
        if(args.length >= 1){
            if(args[0].matches("--show-links|-sl")){
                showLinks();
            }else if(args[0].matches("--help|-h")){
                System.out.println("Esta aplicacao pode ser
inicializada com os seguintes parametros:\n"
                        + "\t--run-clawler|-rc\t\t Inicia o
Crawler.\n"
                        + "\t--show-links|-sl\t\t Mostra os links
encontrados pelo Crawler ate agora.\n"
                        + "\t--help|-h\t\t Mostra esta mensagem de
ajuda.\n"
                        + "Soh eh possivel executar esta aplicacao
com um parametro por vez.\n"
                        + "Quando passado mais de um parametro por
vez, apenas o primeiro sera executado.\n"
                        + "Quando inicializado sem nenhum parametro,
a aplicacao iniciara o Crawler.");
            }else if(args[0].matches("--run-crawler|-rc")){
                runCrawler();
            }else{
                System.out.println("Parametro desconhecido: "
+args[0]+
                        "\nUse o parametro --help para obter
informacoes sobre esta aplicacao.");
            }
        }else{
//          showLinks();
            runCrawler();
//          simpleTest();
        }
    }

    private static void runCrawler() {
        // Cria o Controller do Crawler, adiciona as Seeds e inicia
o Crawler
        try {
            final CrawlerController controller = new
CrawlerController(CommonConfiguration.getInstance().getCrawlerConfig
s());
```

```
            // Adiciona as seeds do arquivo de configuração
            System.out.println("Carregando seeds do arquivo de
configuracao...");
            JSONArray seeds =
CommonConfiguration.getInstance().getSeeds();
            if(seeds != null){
                seeds.forEach((seed) ->
controller.addSeed((String)seed));
            }

            // Inicia o crawling
            System.out.println("Inicializando a aplicacao...");
            controller.start(Crawler.class);
            System.out.println("Encerrando a aplicacao...");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void showLinks() {
        JSONObject output = new JSONObject();

        BasicConnection c = new
BasicConnection(CommonConfiguration.getInstance().getCrawlerDatabase
Configs());
        PossivelQuestionarioManager pqManager = new
PossivelQuestionarioManager(c, true);

        try {
            // Carrega os dados do banco de dados
            System.out.println("Carregando dados do banco de
dados...");
            Set<PossivelQuestionario> pqs = pqManager.getAll();
            String key = "", tmpTxt = "";
            URL tmpUrl = null;
            JSONObject tmpObj = null;

            // Transforma os dados em um JSONObject
            System.out.println("Transformando os dados do banco de
dados para o formato JSON...");
            for(PossivelQuestionario pq : pqs){
                key =
pq.getEncontradoEm().toLocalDateTime().toLocalDate().toString();
                if(!output.has(key))
                    output.put(key, new JSONObject());

                tmpObj = output.getJSONObject(key);
                tmpTxt = pq.getLink_doc();
                tmpUrl = new URL(tmpTxt);
                key = tmpUrl.getHost();

                if(!tmpObj.has(key))
                    tmpObj.put(key, new JSONArray());

                tmpObj.getJSONArray(key).put(pq.getLink_doc());
            }
        } catch (Exception e) {
            //Fecha conexão com banco de dados
            c.close();
```

```
            CommonLogger.fatalError(e);
        }

        //Fecha conexão com banco de dados
        c.close();

        // Salva o arquivo em disco e abre ele usando o editor
default do sistema
        System.out.println("Escrevendo dados no arquivo de
saida...");
        if(CommonUtil.writeFile(outputJsonPath,
output.toString(4))){
            if(!CommonUtil.openFile(outputJsonPath)){
                JOptionPane.showMessageDialog(null,
                        "Não foi possivel abrir o arquivo dos links,
por favor tente abri-lo manualmente.\n"
                        + "Ele se encontra em: " +outputJsonPath+
".",
                        "Ocorreu um problema!",
                        JOptionPane.ERROR_MESSAGE);
            }
        }else{
            JOptionPane.showMessageDialog(null,
                    "Não foi possivel salvar o arquivo dos links, "
                    + "por favor verifique os logs para descobrir o
problema.",
                    "Ocorreu um problema!",
                    JOptionPane.ERROR_MESSAGE);
        }
    }

    @SuppressWarnings("unused")
    private static void simpleTest() {
        String url = "";

        RulesChecker checker = new RulesChecker();
        Document doc = null;
        try{
            if(url.startsWith("cache/")){
                String html = CommonUtil.readResource(url);
                doc = Jsoup.parse(html);
            }else{
//              System.setProperty("javax.net.debug", "all");
                System.setProperty("https.protocols",
"TLSv1,TLSv1.1,TLSv1.2");

                doc = Jsoup.connect(url)
                    .validateTLSCertificates(false)
                    .get();
            }
            checker.shouldSave(doc);
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

questionnaires_crawler/br/ufsc/tcc/crawler/util/Configuration.java

```
package br.ufsc.tcc.crawler.util;

import org.json.JSONException;
```

```java
import org.json.JSONObject;
import br.ufsc.tcc.common.util.CommonConfiguration;

public class Configuration extends CommonConfiguration {

    //Construtores
    public Configuration() {
        configsPath = "./crawler_configs.json";
        this.loadConfigs();
        this.validateConfigs();
    }

    // Demais métodos
    @Override
    protected void validateParameters() throws JSONException {
        JSONObject p = configs.getJSONObject("parameters");
        validatingPath = "parameters";

        validateIntParameter(p, "distBetweenNearNodes",
                "height", "maxHeight", "width");

        validateIntParameter(p, "distBetweenNearQuestions",
                "height");

        p.getString("surveyWordsRegex");
        p.getString("phrasesToIgnoreRegex");
        p.getInt("minCompsInOneCluster");
        p.getInt("minClustersWithComp");
        p.getInt("maxClustersBetweenClustersWithComp");
    }

    @Override
    protected void validateCrawlerConfig() {
        JSONObject crawlerObj = configs.getJSONObject("crawler");
        validatingPath = "crawler";

        crawlerObj.getString("excludedFilesExtensions");
        crawlerObj.getString("excludedDomains");
        crawlerObj.getString("excludedLanguages");
    }

    @Override
    protected void validateDatabaseConfig() {
        JSONObject dbObj = configs.getJSONObject("database");
        validatingPath = "database";

        JSONObject tmp = dbObj.getJSONObject("crawler");
        validatingPath += ".crawler";

        tmp.getString("dbms");
        tmp.getString("name");
        tmp.getString("host");
        tmp.getString("login");
        tmp.getString("password");
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/builder/PerguntaBuilder.java

```java
package br.ufsc.tcc.extractor.builder;

import java.util.ArrayList;
```

```
import java.util.List;
import java.util.Stack;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.model.MyNodeType;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.common.util.DistanceMatrix;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.extractor.IPerguntaExtractor;
import br.ufsc.tcc.extractor.extractor.PerguntaExtractorFactory;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Figura;
import br.ufsc.tcc.extractor.model.FormaDaPergunta;
import br.ufsc.tcc.extractor.model.Grupo;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

/**
 * Classe responsável por 'montar'/'construir' as perguntas de um
questionário
 * utilizando certos padrões.<br>
 * Ela também é responsável por achar outras características
importantes de
 * um questionário, como o seu assunto e imagens relacionadas ao
mesmo.
 *
 * @author Gilney N. Mathias
 */
public class PerguntaBuilder {
    // Current
    private Pergunta currentP;
    private Grupo currentG;
    private int currentI;
    // Helpers
    private DistanceMatrix distMatrix;
    private RulesChecker checker;
    // Group
    private Cluster firstGroupOfQuestionnaire;
    // Matrix
    private Cluster lastMatrixDesc;
    private Cluster lastMatrixHead;
    private Cluster lastMatrixEvaluationLevels;
    private Pergunta lastMatrix;
    private String lastMatrixCommonPrefix;
    // Question with subquestions
    private Cluster lastQWithSubQsDesc;
    private Pergunta lastQWithSubQs;
    private String lastQWithSubQsCommonPrefix;

    public PerguntaBuilder(RulesChecker checker) {
        this.currentP = null;
        this.currentG = null;
        this.currentI = 0;
        this.checker = checker;
        this.distMatrix = this.checker.getDistMatrix();
        this.firstGroupOfQuestionnaire = null;
        this.lastMatrixDesc = null;
        this.lastMatrixHead = null;
        this.lastMatrixEvaluationLevels = null;
        this.lastMatrix = null;
```

```
        //Criado por causa deste link [2* perg tem as mesmas
alternativas que a head da matriz]:
        //  Ex: https://survey.com.br/preview/hotel-satisfaction-
feedback-survey?template=true
        this.lastMatrixCommonPrefix = "";
        this.lastQWithSubQs = null;
        this.lastQWithSubQsDesc = null;
        this.lastQWithSubQsCommonPrefix = "";
    }

    public boolean hasBuildBegun() {
        return this.lastQWithSubQs != null ||
                this.lastMatrix != null;
    }

    public int build(Questionario currentQ, List<MyNode> nodes, int
i, Stack<Cluster> cStack) {
        if(cStack.isEmpty())
            return i;

        this.currentP = new Pergunta();
        this.currentI = i;

        IPerguntaExtractor extractor = null;
        MyNode firstNode = nodes.get(this.currentI),
questionLastNode = null,
                firstImg = null;
        MyNode nTmp1 = (this.currentI+1) < nodes.size() ?
nodes.get(this.currentI+1) : null,
                nTmp2 = null;
        Cluster desc = cStack.pop();
        Cluster cTmp1 = null, cTmp2 = null;
        boolean isSimpleMatrix = false;

        //Se a desc for apenas uma imagem então ela é provavelmente
a imagem
        //da 1* alternativa da pergunta; e se a desc for igual a
'('e o próximo nodo
        //for igual a ')' então provavelmente deve ser uma pergunta
de telefone: ( [ ] ) [ ]
        //      Ex: https://www.survio.com/modelo-de-
pesquisa/avaliacao-de-um-e-shop
        //      Ex:
http://www.almaderma.com.br/formulario/florais/infantil/contato.php
        if(!cStack.isEmpty() && this.checker.isOnlyOneImg(desc)) {
            firstImg = desc.last();
            desc = cStack.pop();
        }else if(!cStack.isEmpty() && desc.getText().equals("(") &&
                nTmp1 != null && nTmp1.getText().equals(")")){
            desc = cStack.pop();
        }

        //Verifica se tem componentes em sequência
        if(nTmp1 != null && firstNode.getType() != MyNodeType.SELECT
&& nTmp1.isComponent() &&
                !nTmp1.isATextInputDisabledWithValue() &&
                this.distMatrix.areNear(firstNode, nTmp1)){

            if(this.lastMatrixHead != null && !cStack.isEmpty() &&
                    this.checker.isAbove(this.lastMatrixHead.last(),
cStack.peek().first())){
```

```
                this.saveLastMatrix(currentQ);
            }
            if(this.checker.isSimpleMatrix(this.lastMatrixHead,
nodes, this.currentI, cStack)){
                if(this.lastMatrixHead == null)
                    this.lastMatrixHead = cStack.pop();

                //Ex:
http://anpei.tempsite.ws/intranet/mediaempresa/
                //Ex:
https://www.surveycrest.com/template_preview/pyof1IFwp9Xa1_x430JdUeV
suHVRKuw
                extractor =
PerguntaExtractorFactory.getExtractor("SIMPLE_MATRIX", currentQ,
                        this.currentP, this.checker);
                this.currentI =
extractor.extract(this.lastMatrixHead, nodes, this.currentI);
                isSimpleMatrix = true;
            }else
if(this.checker.isRadioInputOrCheckboxWithHeader(nodes, currentI,
desc)){

                //Ex: http://infopoll.net/live/surveys/s32805.htm
                extractor =
PerguntaExtractorFactory.getExtractor("CHOICE_INPUT_WITH_HEADER",
currentQ,
                        this.currentP, this.checker);
                this.currentI = extractor.extract(desc, nodes,
this.currentI);
                desc = !cStack.isEmpty() ? cStack.pop() : null;
            }else if(firstNode.getType() == MyNodeType.RADIO_INPUT
&&
                    nTmp1.getType() == MyNodeType.RADIO_INPUT){

                //Ex: https://www.survio.com/modelo-de-
pesquisa/avaliacao-de-um-e-shop [questão 9]
                extractor =
PerguntaExtractorFactory.getExtractor("RATING", currentQ,
                        this.currentP, this.checker);
                this.currentI = extractor.extract(null, nodes,
this.currentI);
            }else if(firstNode.getType() == nTmp1.getType()){

                //Ex:
http://lap.umd.edu/surveys/census/files/surveya1pagesbytopic/page1.h
tml [questão 2a]
                extractor =
PerguntaExtractorFactory.getExtractor("MULTI_COMP", currentQ,
                        this.currentP, this.checker);
                this.currentI = extractor.extract(null, nodes,
this.currentI);
            }
        }else{
            //Checagem duplicada mas, infelizmente, necessária
            if((firstNode.isA("CHECKBOX_INPUT") ||
firstNode.isA("RADIO_INPUT")) &&
                    checker.checkIfTextIsAbove(nodes, currentI))
                desc = cStack.pop();
            extractor =
PerguntaExtractorFactory.getExtractor(firstNode.getType().toString()
, currentQ,
```

```
                    this.currentP, this.checker);
            this.currentI = extractor.extract(desc, nodes,
this.currentI);
        }
        extractor = null;

        //Verifica se foi possível extrair a pergunta
        if(this.currentP.getForma() != null){
            this.currentP.convertFormaToTipo();

            ArrayList<Alternativa> tmpAlts =
this.currentP.getAlternativas();

            //Atualiza a desc da pergunta
            if(!cStack.isEmpty() && checker.isEvaluationLevels(desc,
cStack, false)){
                this.setEvaluationLevels(this.currentP, desc);
                desc = cStack.pop();
            }
            desc = this.checker.getCorrectDescription(desc, tmpAlts,
firstNode, cStack);

            //Verifica se a desc e a pergunta estão perto uma da
outra
            if(!checker.areDescAndPergNear(desc, firstNode))
                return this.currentI;

            // Atualiza a firstImg
            if(firstImg == null || currentQ.hasFigura(firstImg))
                firstImg = desc.last();

            questionLastNode = nodes.get(this.currentI);

            //Verifica se o texto abaixo, se tiver, não faz parte
desta pergunta (Ex: Peso: [ ] kg)
            boolean foundComplementaryText = false;
            while(this.checker.checkComplementaryText(nodes,
this.currentI)){
                foundComplementaryText = true;
                nTmp1 = nodes.get(++this.currentI);
                //Se for um CHECKBOX ou RADIO INPUT, então o texto
complementar deve
                //pertencer a uma opção 'Outro' que usa um TEXT
INPUT
                //  Ex: https://www.survio.com/modelo-de-
pesquisa/pesquisa-de-preco-do-produto [questão 2]
                if(this.currentP.isA("CHECKBOX_INPUT") ||
this.currentP.isA("RADIO_INPUT")){
                    if(this.currentP.getFilhas().size() > 0){
                        ArrayList<Pergunta> filhas =
this.currentP.getFilhas();
                        Pergunta p = filhas.get(filhas.size()-1);
                        if(nTmp1.isATextInputDisabledWithValue())
                            p.setDescricao(p.getDescricao() +"\n"+
nTmp1.getAttr("value"));
                        else
                            p.setDescricao(p.getDescricao() +"\n"+
nTmp1.getText());
                    }else
                        desc.add(nTmp1);
                }else
```

```
                    desc.add(nTmp1);
                }
                desc = this.checker.checkIfDescIsComplete(desc, cStack,
nodes, this.currentI);
                String descTxt = desc.getText(foundComplementaryText);

                //Ex: http://www.createsurvey.com/cgi-
bin/pollfrm?s=36960&m=FWIOpt&presurvey_view=1
                if(descTxt.matches("\\d\n\\d"))
                    return this.currentI;

                //Seta a descrição da pergunta
                this.currentP.setDescricao(descTxt);
                CommonLogger.debug("Descricao: {}\n\n",
this.currentP.getDescricao());

                //Verifica se é uma matriz
                //  Ex: https://www.survio.com/modelo-de-
pesquisa/pesquisa-de-preco-do-produto [questão 3]
                boolean matrixFlag = false;
                nTmp1 = desc.first();
                cTmp2 = this.lastMatrixHead;

                if(!cStack.isEmpty()){
                    if(cTmp2 == null){
                        if(!currentQ.getAssunto().isEmpty() ||
cStack.size() >= 2)
                            cTmp2 = cStack.peek();
                    }else {
                        //Ex: https://survs.com/survey-
templates/teacher-evaluation-survey/
                        if(lastQWithSubQsDesc != null &&
this.checker.isAbove(lastQWithSubQsDesc.last(), cTmp2.first()))
                            this.saveLastQWithSubQs(currentQ);
                        if(this.checker.isAbove(cTmp2.last(),
cStack.peek().first())) {
                            //Se tiver um texto no meio quer dizer que
ja termino a matriz
                            this.saveLastMatrix(currentQ);
                            cTmp2 = cStack.peek();
                        }
                    }
                }

                if(this.lastMatrix == null || isSimpleMatrix ||
                        this.checker.checkCommonPrefix(cTmp2.last(),
nTmp1, this.lastMatrixCommonPrefix)) {
                    matrixFlag =
this.checker.hasSameTexts(this.currentP, cTmp2);
                    if(matrixFlag){
                        this.updateLastMatrix(nodes, cStack, currentQ,
nTmp1, cTmp2);
                    }else if(this.lastMatrix != null){
                        this.saveLastMatrix(currentQ);
                    }
                }else {
                    this.saveLastMatrix(currentQ);
                }

                //Verifica se é uma pergunta com subperguntas
```

```
            //  Ex: https://www.surveymonkey.com/r/online-social-
networking-template [questão 4]
            boolean qWithSubQsFlag = false;

            nTmp1 = questionLastNode;//XXX qual será o melhor?
//          nTmp1 = firstNode;
//          nTmp1 = desc.first();

            cTmp1 = this.lastQWithSubQsDesc != null ?
                    this.lastQWithSubQsDesc : null;
            nTmp2 = cTmp1==null ? null : cTmp1.first();

            if(!cStack.isEmpty()){
                if(cTmp1 == null){
                    if(!currentQ.getAssunto().isEmpty() ||
cStack.size() >= 2)
                        cTmp1 = cStack.peek();
                }else if( (lastMatrixDesc != null &&
this.checker.isAbove(nTmp2, lastMatrixDesc.first())) ||
                        this.checker.isAbove(nTmp2,
cStack.peek().first())){
                    //Se tiver um texto no meio quer dizer que ja
termino as subperguntas
                    this.saveLastQWithSubQs(currentQ);
                    cTmp1 = cStack.peek();
                }
            }

            if(!matrixFlag) {
                if(cTmp1 != null && this.lastQWithSubQsDesc == null)
                    cTmp1 =
this.checker.checkIfDescIsCompleteWithClone(cTmp1, cStack, nodes,
this.currentI);
                nTmp2 = cTmp1==null ? null : cTmp1.first();

                if(nTmp2 != null && nTmp2.isText()){
                    if(checker.checkDistForQWithSubQs(nTmp2,
nTmp1)){
                        if(this.lastQWithSubQs == null &&
this.checker.
                                isQWithSubQs(nTmp1, nTmp2, nodes,
this.currentI)){
                            this.updateLastQWithSubQs(currentQ,
nodes, cStack, nTmp1);
                            qWithSubQsFlag = true;
                        }else if(this.lastQWithSubQs != null &&

this.checker.checkCommonPrefix(nTmp1, nTmp2,
this.lastQWithSubQsCommonPrefix)){

if(!this.lastQWithSubQs.getForma().toString().

startsWith(this.currentP.getForma().toString())){

this.lastQWithSubQs.setForma(FormaDaPerguntaManager.getForma("MIX_CO
MP_GROUP"));
                            }

this.lastQWithSubQs.addFilha(this.currentP);
                            qWithSubQsFlag = true;
                        }else{
```

```
                            this.saveLastQWithSubQs(currentQ);
                        }
                    }else if(this.lastQWithSubQs != null){
                        this.saveLastQWithSubQs(currentQ);
                    }
                }
            }

            //Verifica se não é a imagem da pergunta
            if(!firstImg.isImage() || currentQ.hasFigura(firstImg))
{
                if(!cStack.isEmpty()) {
                    cTmp1 = cStack.peek();
                    if(this.checker.isOnlyOneImg(cTmp1) &&
this.distMatrix.areNear(cTmp1, desc))
                        firstImg = cStack.pop().last();
                    else if(desc.first().isImage())
                        firstImg = desc.first();
                }
            }
            if(firstImg.isImage() && !currentQ.hasFigura(firstImg))
{
                Figura fig = new Figura(firstImg.getAttr("src"),
firstImg.getAttr("alt"));
                fig.setDono(this.currentP);
                currentQ.addFigura(fig);
                CommonLogger.debug("Figura da pergunta: {}\n", fig);
            }

            if(!cStack.isEmpty()){
                if(currentQ.getAssunto().isEmpty()){
                    //Encontra o assunto do questionario
                    cTmp1 = cStack.pop();
                    this.firstGroupOfQuestionnaire = null;

                    //Verifica se não é a imagem do questionário
                    MyNode imgTmp = null;
                    if(this.checker.isOnlyOneImg(cTmp1))
                        imgTmp = cTmp1.last();
                    else if(cTmp1.first().isImage())
                        imgTmp = cTmp1.first();

                    if(imgTmp != null){
                        Figura fig = new
Figura(imgTmp.getAttr("src"), imgTmp.getAttr("alt"));
                        fig.setDono(currentQ);
                        currentQ.addFigura(fig);
                        CommonLogger.debug("Figura do questionario:
{}\n", fig);
                        if(this.checker.isOnlyOneImg(cTmp1)){
                            if(!cStack.isEmpty())
                                cTmp1 = cStack.pop();
                            else
                                cTmp1 = null;
                        }
                    }

                    //Verifica se não é o texto de um grupo
                    cTmp2 = this.lastMatrixDesc != null ?
this.lastMatrixDesc :
```

```
                        this.lastQWithSubQsDesc != null ?
this.lastQWithSubQsDesc :
                            desc;
                    if(cTmp1 != null &&
this.checker.isAGroupText(cTmp1, cTmp2,
this.firstGroupOfQuestionnaire) &&
                            !cStack.isEmpty()){
                        currentG = new Grupo(cTmp1.getText());
                        currentQ.addGrupo(currentG);
                        this.firstGroupOfQuestionnaire = cTmp1;

                        CommonLogger.debug("\nGroup1: {}\n\n",
cTmp1.getText());
                        if(!cStack.isEmpty())
                            cTmp1 = cStack.pop();
                        else
                            cTmp1 = null;
                    }

                    //Seta o assunto do questionário atual
                    if(cTmp1 != null) {
                        cTmp1 =
this.checker.checkIfDescIsComplete(cTmp1, cStack, nodes,
this.currentI);
                        currentQ.setAssunto(cTmp1.getText());
                        CommonLogger.debug("Assunto1: {}\n\n",
currentQ.getAssunto());
                    }
                }else{
                    //Verifica se não é o texto de um grupo
                    cTmp1 = cStack.peek();
                    cTmp2 = this.lastMatrixDesc != null ?
this.lastMatrixDesc :
                        this.lastQWithSubQsDesc != null ?
this.lastQWithSubQsDesc :
                            desc;
                    if(this.checker.isAGroupText(cTmp1, cTmp2,
this.firstGroupOfQuestionnaire)){
                        cTmp1 = cStack.pop();
                        this.currentG = new Grupo(cTmp1.getText());
                        currentQ.addGrupo(currentG);

                        CommonLogger.debug("\nGroup2: {}\n\n",
cTmp1.getText());
                    }
                }
            }

            if(this.currentG != null){
                this.currentP.setGrupo(this.currentG);
                if(this.lastMatrix != null &&
this.lastMatrix.getGrupo() == null)
                    this.lastMatrix.setGrupo(this.currentG);
                if(this.lastQWithSubQs != null &&
this.lastQWithSubQs.getGrupo() == null)
                    this.lastQWithSubQs.setGrupo(this.currentG);
            }

            if(!matrixFlag && !qWithSubQsFlag
&& !this.currentP.getDescricao().isEmpty())
                currentQ.addPergunta(this.currentP);
```

```
        }

        return this.currentI;
    }

    /**
     * Atualiza os dados da ultima pergunta com subperguntas
encontrada.
     *
     * @param currentQ
     * @param nodes
     * @param cStack
     * @param nTmp1
     */
    private void updateLastQWithSubQs(Questionario currentQ,
List<MyNode> nodes,
            Stack<Cluster> cStack, MyNode nTmp1) {
        if(cStack.isEmpty()) return;

        this.lastQWithSubQsDesc = cStack.pop();
        this.lastQWithSubQsDesc = this.checker.
                checkIfDescIsComplete(this.lastQWithSubQsDesc,
cStack, nodes, this.currentI);
        this.lastQWithSubQsCommonPrefix = nTmp1.getDewey().

getCommonPrefix(this.lastQWithSubQsDesc.first().getDewey());

        this.lastQWithSubQs = new Pergunta();
        String forma = this.currentP.getForma().toString();

this.lastQWithSubQs.setForma(FormaDaPerguntaManager.getForma(forma+"
_GROUP"));

this.lastQWithSubQs.setDescricao(this.lastQWithSubQsDesc.getText());
        this.lastQWithSubQs.addFilha(this.currentP);

        if(this.lastQWithSubQsDesc.last().isImage()) {
            Figura fig = new
Figura(this.lastQWithSubQsDesc.last().getAttr("src"),
                    this.lastQWithSubQsDesc.last().getAttr("alt"));
            fig.setDono(this.lastMatrix);
            currentQ.addFigura(fig);
            CommonLogger.debug("Figura da qWithSubQ: {}\n", fig);
        }
    }

    /**
     * Adiciona a ultima pergunta com subperguntas encontrada ao
questionário atual.
     *
     * @param currentQ      Questionário atual.
     */
    private void saveLastQWithSubQs(Questionario currentQ) {
        if(this.lastQWithSubQs != null){
            this.lastQWithSubQs.convertFormaToTipo();
            currentQ.addPergunta(this.lastQWithSubQs);
            CommonLogger.debug("Q with SubQs descricao: {}\n\n",
this.lastQWithSubQs.getDescricao());
        }
        this.lastQWithSubQs = null;
        this.lastQWithSubQsDesc = null;
```

```
        this.lastQWithSubQsCommonPrefix = "";
    }


    /**
     * Atualiza os dados da ultima matriz encontrada.
     *
     * @param nodes
     * @param cStack
     * @param descFirstNode
     * @param cTmp2
     */
    private void updateLastMatrix(List<MyNode> nodes, Stack<Cluster>
cStack,
            Questionario currentQ, MyNode descFirstNode, Cluster
cTmp2) {
        if(!cStack.isEmpty() && cTmp2 == cStack.peek())
            this.lastMatrixHead = cStack.pop();

        if(lastMatrix == null) {
            this.lastMatrix = new Pergunta();
            FormaDaPergunta forma = this.currentP.getForma();
            if(this.currentP.isA("MIX_COMP_GROUP")){

this.lastMatrix.setForma(FormaDaPerguntaManager.getForma("MIX_COMP_M
ATRIX"));
            }else{

this.lastMatrix.setForma(FormaDaPerguntaManager.getForma(forma.toStr
ing()+"_MATRIX"));
            }

            this.lastMatrixCommonPrefix =
this.lastMatrixHead.last().getDewey()
                    .getCommonPrefix(descFirstNode.getDewey());

            Cluster desc = !cStack.isEmpty() ? cStack.peek() : null;
            if(desc != null) {
                if(!cStack.isEmpty() &&
checker.isEvaluationLevels(desc, cStack, true)){
                    desc = cStack.pop();
                    lastMatrixEvaluationLevels = desc;
                    desc = !cStack.isEmpty() ? cStack.peek() : null;
                }

                //Verifica se a desc e o cabeçalho da matriz estão
perto um do outro
                if(desc != null && checker.areDescAndPergNear(desc,
this.lastMatrixHead.first())) {
                    desc = cStack.pop();
                    desc = this.checker.checkIfDescIsComplete(desc,
cStack, nodes, this.currentI);

                    if(desc.last().isImage()) {
                        Figura fig = new
Figura(desc.last().getAttr("src"), desc.last().getAttr("alt"));
                        fig.setDono(this.lastMatrix);
                        currentQ.addFigura(fig);
                        CommonLogger.debug("Figura da matriz: {}\n",
fig);
                    }
```

```
                    this.lastMatrixDesc = desc;
                    this.lastMatrix.setDescricao(desc.getText());
                }else {
                    //Matriz sem descrição
                    //  Ex: http://www.questionpro.com/survey-
templates/employee-benefits-survey/
                    this.lastMatrixDesc = null;
                    this.lastMatrix.setDescricao("");
                }
            }else {
                //Caso meio raro...
                //  Ex: http://www.123contactform.com/js-form--
1941084.html
                this.lastMatrixDesc = null;
                this.lastMatrix.setDescricao("");
            }
        }

        if(this.lastMatrix != null)
            this.lastMatrix.addFilha(this.currentP);
    }


    /**
     * Adiciona a ultima matriz encontrada ao questionário atual.
     *
     * @param currentQ      Questionário atual.
     */
    private void saveLastMatrix(Questionario currentQ) {
        if(this.lastMatrix != null){
            if(this.lastMatrixEvaluationLevels != null)
                this.setEvaluationLevels(this.lastMatrix,
this.lastMatrixEvaluationLevels);
            this.removePergDescFromAltDesc(this.lastMatrix);
            this.lastMatrix.convertFormaToTipo();

            currentQ.addPergunta(this.lastMatrix);
            CommonLogger.debug("Matrix descricao: {}\n\n",
this.lastMatrix.getDescricao());
        }
        this.lastMatrixDesc = null;
        this.lastMatrixHead = null;
        this.lastMatrixEvaluationLevels = null;
        this.lastMatrix = null;
        this.lastMatrixCommonPrefix = "";
    }


    /**
     * Remove a descrição das perguntas filhas da Matriz da
     * descrição de suas alternativas.
     *
     * @param matrix
     */
    private void removePergDescFromAltDesc(Pergunta matrix) {
        //Ex: http://www.surveymoz.com/s/evaluation-of-company-and-
supervisor-example
        String pDesc = "", aDesc = "";

        for(Pergunta perg : matrix.getFilhas()) {
```

```
            pDesc = perg.getDescricao();
            if(!perg.getAlternativas().isEmpty()) {
                for(Alternativa alt : perg.getAlternativas()) {
                    aDesc = alt.getDescricao();
                    alt.setDescricao(aDesc.replace(pDesc,
"").trim());
                }
            }else {
                for(Pergunta p : perg.getFilhas()) {
                    aDesc = p.getDescricao();
                    p.setDescricao(aDesc.replace(pDesc, "").trim());
                }
            }
        }
    }


    private void setEvaluationLevels(Pergunta perg, Cluster
evaluationLevels) {
        if(perg.isA("RADIO_INPUT")){
            ArrayList<Alternativa> alts = perg.getAlternativas();


alts.get(0).setDescricao(evaluationLevels.first().getText() +
                    "\n" +alts.get(0).getDescricao());
            alts.get(alts.size()-
1).setDescricao(evaluationLevels.last().getText() +
                    "\n" +alts.get(alts.size()-1).getDescricao());
        }else if(perg.isA("RADIO_INPUT_MATRIX")){
            for(Pergunta p : perg.getFilhas())
                setEvaluationLevels(p, evaluationLevels);
        }
    }


    public void clearData(Questionario currentQ) {
        this.saveLastMatrix(currentQ);
        this.saveLastQWithSubQs(currentQ);

        this.currentG = null;
        this.currentP = null;
        this.distMatrix.clear();
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/builder/QuestionarioBuilder.java

```
package br.ufsc.tcc.extractor.builder;

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;
import org.json.JSONObject;
import org.jsoup.nodes.Node;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.util.CommonConfiguration;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.common.util.CommonUtil;
import br.ufsc.tcc.common.util.DistanceMatrix;
import br.ufsc.tcc.extractor.model.Questionario;
```

```
/**
 * Classe responsável por 'montar'/'construir' os questionários de
 * uma pagina Web.
 *
 * @author Gilney N. Mathias
 *
 */
public class QuestionarioBuilder {
    private static int MAX_TEXT_CLUSTERS_BETWEEN_QUESTIONS = 0;
    private Questionario currentQ;
    private String currentLink;
    private DistanceMatrix distMatrix;
    private PerguntaBuilder pBuilder;
    private RulesChecker checker;

    // Construtores
    public QuestionarioBuilder(){
        this.currentQ = null;
        this.currentLink = "";
        this.distMatrix = new DistanceMatrix();

        this.checker = new RulesChecker(this.distMatrix);
        this.pBuilder = new PerguntaBuilder(this.checker);
    }

    // Getters e Setters
    public String getCurrentLink(){
        return this.currentLink;
    }

    public void setCurrentLink(String link){
        this.currentLink = link;
    }

    // Demais métodos
    public ArrayList<Questionario> build(Node root, String docTitle)
{
        ArrayList<Questionario> ret = new ArrayList<>();
        List<MyNode> nodes = CommonUtil.findCompsImgsAndTexts(root);

        CommonLogger.debug(nodes);

        this.currentQ = new Questionario(this.currentLink);
        Stack<Cluster> cStack = new Stack<>();
        Cluster cTmp = new Cluster(), lastDesc = null;
        int clustersWithTextSinceLastQuestion = 0;

        for(int i = 0; i<nodes.size(); i++){
            MyNode nTmp = nodes.get(i);

            if(nTmp.isImgOrText()){
                //Verifica se tem que criar um novo cluster
                if(this.checker.shouldCreateNewCluster(cTmp, nTmp,
nodes, i)){
//                    CommonLogger.debug("\n{}\n\t{}\n", cTmp, nTmp);
                    cStack.add(cTmp);
                    cTmp = new Cluster();
                    clustersWithTextSinceLastQuestion++;
                }

                //Verifica se esta começando um novo questinario
```

```
                if(!this.currentQ.getPerguntas().isEmpty() ||
pBuilder.hasBuildBegun()){
                    if(clustersWithTextSinceLastQuestion >
MAX_TEXT_CLUSTERS_BETWEEN_QUESTIONS ||

checker.shouldStartNewQuestionario(lastDesc, nTmp)){
                        this.pBuilder.clearData(this.currentQ);

if(this.checker.isValidQuestionnaire(this.currentQ)){
                            //Ex:
https://polldaddy.com/s/d5564eb1c42db4d1
                            if(this.currentQ.getAssunto().isEmpty()
|| CommonUtil.isOnlyOneWord(this.currentQ.getAssunto())){
                                this.currentQ.setAssunto(docTitle);
                                CommonLogger.debug("Assunto2:
{}\n\n", currentQ.getAssunto());
                            }
                            ret.add(this.currentQ);
                        }else
                            CommonLogger.debug("==================
Questionario invalido! ==================");
                        CommonLogger.debug("==================
shouldStartNewQuestionario() ==================\n");
                        this.currentQ = new
Questionario(this.currentLink);
                    }
                }

                cTmp.add(nTmp);
            }else{
                if(!cTmp.isEmpty()){
                    cStack.add(cTmp);
                    lastDesc = cTmp;
                    clustersWithTextSinceLastQuestion = 0;
                }
                i = pBuilder.build(this.currentQ, nodes, i, cStack);
                cTmp = new Cluster();
            }
        }
        this.pBuilder.clearData(this.currentQ);
        if(this.checker.isValidQuestionnaire(this.currentQ)){
            //Ex: https://polldaddy.com/s/d5564eb1c42db4d1
            if(this.currentQ.getAssunto().isEmpty() ||
CommonUtil.isOnlyOneWord(this.currentQ.getAssunto())){
                this.currentQ.setAssunto(docTitle);
                CommonLogger.debug("Assunto2: {}\n\n",
currentQ.getAssunto());
            }
            ret.add(this.currentQ);
        }else if(!this.currentQ.getPerguntas().isEmpty())
            CommonLogger.debug("================== Questionario
invalido! ==================");

        CommonLogger.debug("\t\t\t================> Questionarios
<================");
        CommonLogger.debug(ret);
        return ret;
    }

    // Métodos/Blocos estáticos
    static {
```

```
        JSONObject p =
CommonConfiguration.getInstance().getParameters();

        //Ex: http://www.sciencebuddies.org/science-fair-
projects/project_ideas/Soc_survey_sample1.shtml
        MAX_TEXT_CLUSTERS_BETWEEN_QUESTIONS =
p.getInt("maxTextClustersBetweenQuestions");

        CommonLogger.debug("QuestionarioBuilder:> Static block
executed!");
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/builder/RulesChecker.java

```
package br.ufsc.tcc.extractor.builder;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Stack;
import java.util.regex.Pattern;
import org.apache.commons.lang.StringUtils;
import org.json.JSONObject;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.DeweyExt;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.model.MyNodeType;
import br.ufsc.tcc.common.util.CommonConfiguration;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.common.util.CommonUtil;
import br.ufsc.tcc.common.util.DistanceMatrix;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class RulesChecker {

    private static JSONObject CONFIGS = null;
    private DistanceMatrix distMatrix;

    private static final String NUMBER_REGEX =
"(\\d{1,3}(\\s{1,2})?(\\.|\\:|\\)|\\-)?)";

    // Regex usados para extrair coisas como:
    //      [ ] / month [ ] / day [ ] year
    public static final String DATE_REGEX1 = "(/|\\-)",
            DATE_REGEX2 =
"(month|day|year|m(ê|e)s|d(i|í)a|a(n|ñ)o)";
    public static final String MONEY_REGEX1 = "(\\.)",
            MONEY_REGEX2 =
"(dollars?|cents?|dólares?|reais?|centavos)";

    public RulesChecker(DistanceMatrix distMatrix) {
        this.distMatrix = distMatrix;
    }

    // Getters e Setters
    public DistanceMatrix getDistMatrix(){
        return this.distMatrix;
    }
```

```
    public static JSONObject getConfigs(){
        return CONFIGS;
    }

    // Demais métodos

    // Métodos usados pela classe QuestionarioBuilder
    public boolean shouldStartNewQuestionario(Cluster lastDesc,
MyNode newNode) {
        if(lastDesc == null || lastDesc.isEmpty() || newNode ==
null)
            return false;
        JSONObject obj =
CONFIGS.getJSONObject("distBetweenTextsInsideQuestionnaire");
        DeweyExt dist = this.distMatrix.getDist(lastDesc.last(),
newNode);

        if(dist.getHeight() > obj.getInt("height"))
            return true;

        //Verifica se o 1* container, depois do Body, é diferente
        DeweyExt d1 = lastDesc.last().getDewey(), d2 =
newNode.getDewey();
        return d1.getNumbers().get(1) != d2.getNumbers().get(1);
    }


    public boolean shouldCreateNewCluster(Cluster lastCluster,
MyNode newNode, List<MyNode> nodes, int i) {
        if(lastCluster == null || lastCluster.isEmpty() || newNode
== null ||
                i+1 >= nodes.size())
            return false;

        if(!this.distMatrix.areNear(lastCluster.last(), newNode))
            return true;

        //Todos os elementos do cluster devem ter o mesmo prefixo
        //comum ao proximo nodo encontrado
        String tTmp1 =
lastCluster.last().getDewey().getCommonPrefix(nodes.get(i+1).getDewe
y()),
                tTmp2 =
newNode.getDewey().getCommonPrefix(nodes.get(i+1).getDewey());
        return !tTmp1.equals(tTmp2);
    }

    public boolean isValidQuestionnaire(Questionario q){
        if(q.getPerguntas().size() <
CONFIGS.getInt("minQuestionsOnQuestionnaire"))
            return false;
        if(CommonUtil.matchesWithLineBreak(q.getAssunto(),
                CONFIGS.getString("phrasesToIgnoreRegex")))
            return false;

        int count = 0;
        for(Pergunta p : q.getPerguntas()){
            if(CommonUtil.matchesWithLineBreak(p.getDescricao(),
                    CONFIGS.getString("phrasesToIgnoreRegex"))){
                count++;
                if(count == 2)
```

```
                    return false;
            }

            for(Pergunta f : p.getFilhas()){
                if(CommonUtil.matchesWithLineBreak(f.getDescricao(),
                        CONFIGS.getString("phrasesToIgnoreRegex"))){
                    count++;
                    if(count == 2)
                        return false;
                }
            }
        }
        return true;
    }

    // Métodos usados pela classe PerguntaBuilder e
PerguntaExtractor
    public boolean isOnlyOneImg(Cluster c) {
        return c.size() == 1 && c.first().isImage();
    }

    public boolean areCompAndTextNear(MyNode comp, MyNode text) {
        if(comp == null || text == null)
            return false;
        JSONObject obj =
CONFIGS.getJSONObject("distBetweenCompAndText");
        DeweyExt dist = this.distMatrix.getDist(comp, text);
        return dist.getHeight() <= obj.getInt("height") &&
                dist.getMaxHeight() <= obj.getInt("maxHeight");
    }

    public Cluster getCorrectDescription(Cluster desc,
ArrayList<Alternativa> tmpAlts, MyNode firstNode,
            Stack<Cluster> cStack) {
        if(desc == null || desc.isEmpty()) return desc;

        //Ex: https://www.surveymonkey.com/r/CAHPS-Health-Plan-
Survey-40-Template
        if(desc.getText().equals(".") && !cStack.isEmpty())
            desc = cStack.pop();

        if(desc.size() != tmpAlts.size()) return desc;

        boolean flag = true;
        String altsTxt = "";

        //Pega o texto de todas as alternativas da pergunta
        for(Alternativa alt : tmpAlts){
            altsTxt += alt.getDescricao().toLowerCase()+"\n";
        }

        //Ex: https://www.surveymonkey.com/r/General-Event-Feedback-
Template [questão 1]
        while(true){
            for(MyNode node : desc.getGroup()){
                String txt = node.getText().toLowerCase();
                flag = flag && !txt.isEmpty() &&
altsTxt.contains(txt);
            }
            if(flag && !cStack.isEmpty()){
                desc = cStack.pop();
```

```
            }else
                break;
        }
        return desc;
    }

    public Cluster checkIfDescIsCompleteWithClone(Cluster desc,
Stack<Cluster> cStack, List<MyNode> nodes, int i) {
        if(desc == null || desc.isEmpty() || cStack.size() < 2 ||
i+1 >= nodes.size())
            return desc;

        Cluster tmp = cStack.get(1);
        String txt = tmp.getText();

        //Ex: https://www.survio.com/modelo-de-pesquisa/feedback-
sobre-servico
        boolean has4orMoreChars = txt.length() >= 4;
        if(!txt.isEmpty() && this.isDescriptionsNear(tmp, desc) &&
                ((has4orMoreChars
&& !this.distMatrix.areNear(tmp.last(), nodes.get(i+1)))
|| !has4orMoreChars)) {
            tmp = tmp.clone();
            tmp = tmp.join(desc);
            return tmp;
        }

        return desc;
    }

    public Cluster checkIfDescIsComplete(Cluster desc,
Stack<Cluster> cStack, List<MyNode> nodes, int i){
        if(desc == null || desc.isEmpty() || cStack.isEmpty() || i+1
>= nodes.size())
            return desc;

        Cluster tmp = cStack.peek();
        String txt = tmp.getText();

        //Ex: https://polldaddy.com/s/d5564eb1c42db4d1
        boolean has4orMoreChars = txt.length() >= 4;
        if(!txt.isEmpty() && this.isDescriptionsNear(tmp, desc) &&
                ((has4orMoreChars
&& !this.distMatrix.areNear(tmp.last(), nodes.get(i+1)))
|| !has4orMoreChars)) {
            tmp = cStack.pop();
            tmp = tmp.join(desc);
            return tmp;
        }

        return desc;
    }

    public boolean isDescriptionsNear(Cluster firstDesc, Cluster
secondDesc){
        if(firstDesc == null || firstDesc.isEmpty() ||
                secondDesc == null || secondDesc.isEmpty())
            return false;

        JSONObject obj =
CONFIGS.getJSONObject("distBetweenPartsOfDescription");
```

```
        DeweyExt dist = this.distMatrix.getDist(firstDesc.last(),
secondDesc.first());
        return dist.getHeight() <= obj.getInt("height") &&
                dist.getMaxHeight() <= obj.getInt("maxHeight") &&
                dist.getWidth() <= obj.getInt("width");
    }

    public boolean areDescAndPergNear(Cluster desc, MyNode perg) {
        if(desc == null || desc.isEmpty())
            return false;
        return areDescAndPergNear(desc.last(), perg);
    }

    public boolean areDescAndPergNear(MyNode desc, MyNode perg){
        if(desc == null || perg == null)
            return false;
        JSONObject obj =
CONFIGS.getJSONObject("distBetweenDescAndQuestion");
        DeweyExt dist = this.distMatrix.getDist(desc, perg);
        return dist.getHeight() <= obj.getInt("height") &&
                dist.getMaxHeight() <= obj.getInt("maxHeight");
    }

    public boolean isAGroupText(Cluster cTmp, Cluster desc, Cluster
firstGroupOfQuestionnaire) {
        if(cTmp == null || cTmp.isEmpty() || desc == null ||
desc.isEmpty())
            return false;

        String txt = cTmp.getText();
        txt = CommonUtil.trim(txt);
        //Cluster deve ter um e APENAS um texto e não deve ser
apenas um numero
        //  Ex:
https://www.proprofs.com/survey/t/?title=okgaw&type=template
[number]
        if(txt.isEmpty() || txt.contains("\n") ||
                txt.matches(NUMBER_REGEX))
            return false;

        JSONObject obj =
CONFIGS.getJSONObject("distBetweenGroupAndFirstQuestion");
        DeweyExt dist = distMatrix.getDist(cTmp.last(),
desc.first());

        if(dist.getHeight() <= obj.getInt("height") &&
dist.getWidth() <= obj.getInt("width")){
            //O texto do grupo deve ter no maximo X palavras (com
tamanho > 1)
            int count = Arrays.stream(txt.split(" ")).
                    reduce(0, (a,b) -> a + (b.length() > 1 ? 1 : 0),
                            (a,b) -> a+b);
            if(count <=
CONFIGS.getInt("maxWordsInAGroupDescription")){
                if(firstGroupOfQuestionnaire != null){
                    //O tamanho do Dewey dos grupos de um
questionario, geralmente,
                    //é o mesmo [utilizam o mesmo padrão de nodos
pais]
```

```
if(firstGroupOfQuestionnaire.last().getDewey().toString().length()
==

cTmp.last().getDewey().toString().length())
                        return true;
                }else{
                    return true;
                }
            }
        }
        return false;
    }

    public boolean checkComplementaryText(List<MyNode> nodes, int i)
{
        if(i+2 < nodes.size()){
            MyNode nTmp1 = nodes.get(i),
                nTmp2 = nodes.get(i+1),
                nTmp3 = nodes.get(i+2);
            DeweyExt dist = distMatrix.getDist(nTmp2, nTmp3);

            //Lida com casos aonde se tem 2 textos complementares
            //  Ex: https://www.survio.com/modelo-de-
pesquisa/feedback-sobre-servico
            if(i+3 < nodes.size() && (nTmp3.isImgOrText() ||
nTmp3.isATextInputDisabledWithValue()) &&
                    dist.getMaxHeight() == 1 && dist.getWidth() <=
2){
                nTmp3 = nodes.get(i+3);
            }

            JSONObject obj =
CONFIGS.getJSONObject("distBetweenDescAndComplementaryText");

            if((nTmp2.isText() ||
nTmp2.isATextInputDisabledWithValue()) && nTmp3.isImgOrText()){
                dist = distMatrix.getDist(nTmp1, nTmp2);
                if(dist.getHeight() <= obj.getInt("height") &&
dist.getMaxHeight() <= obj.getInt("maxHeight") &&
                        dist.getWidth() <= obj.getInt("width")){

                    String prefix1 =
nTmp1.getDewey().getCommonPrefix(nTmp3.getDewey()),
                            prefix2 =
nTmp2.getDewey().getCommonPrefix(nTmp3.getDewey()),
                            prefix3 =
nTmp1.getDewey().getCommonPrefix(nTmp2.getDewey());

                    //O prefixo3 deve ser maior que o prefix1, pois
isso indica que nTmp1 e nTmp2 estão,
                    //pelo menos, um elemento a mais juntos
                    if(prefix1.equals(prefix2) &&
                            CommonUtil.getPrefixLength(prefix3) >
CommonUtil.getPrefixLength(prefix1))
                        return true;
                }
            }
        }
        return false;
    }
```

```
    public boolean isAbove(MyNode n1, MyNode n2) {
        if(n1 == null || n2 == null) return false;
        //Verifica se n1 esta acima de n2
        DeweyExt dist = this.distMatrix.getDist(n1, n2);
        return dist.isNegative();
    }

    /**
     * Verifica se as alternativas/filhas da {@code currentP} contêm
todos os textos
     * do {@code cTmp2}, ou se a descrição da {@code currentP}
contém o
     * texto de {@code cTmp2}.
     *
     * @param currentP
     * @param cTmp2
     * @return
     */
    public boolean hasSameTexts(Pergunta currentP, Cluster cTmp2) {
        if(cTmp2 == null || cTmp2.getText().isEmpty()) return false;

        ArrayList<Alternativa> alts = currentP.getAlternativas();
        ArrayList<Pergunta> filhas = currentP.getFilhas();
        String txt = cTmp2.getText(), txtTmp = "";
        boolean flag = false;
        int count = filhas.size()+alts.size();

        if(count == 0 && cTmp2.size() == 1
&& !currentP.getDescricao().matches("\\d(\\.)?")){
            //Ex: https://www.survio.com/modelo-de-
pesquisa/pesquisa-de-preco-do-produto
            //Ex: https://www.surveyrock.com/template/sample-
student-choice-professors-classes-survey-template-1897
            txtTmp = Pattern.quote(txt);
            flag = CommonUtil.containsWithLineBreak(txtTmp,
currentP.getDescricao());
        }else if(count == cTmp2.size()){
            flag = true;
            for(int j = 0; j < alts.size(); j++){
                txtTmp = alts.get(j).getDescricao();
                txtTmp = Pattern.quote(txtTmp);
                flag = flag &&
CommonUtil.containsWithLineBreak(txtTmp, txt);
            }
            for(int j = 0; j < filhas.size(); j++){
                txtTmp = filhas.get(j).getDescricao();
                txtTmp = Pattern.quote(txtTmp);
                flag = flag &&
CommonUtil.containsWithLineBreak(txtTmp, txt);
            }
        }

        return flag;
    }

    /**
     * Verifica se esta lidando com uma 'simple matrix'. </br>
     * Uma 'simple matrix' é uma matriz que possui simplesmente
componentes em sequencia. </br>
```

```
     * A quantidade de componentes em sequencia deve bater com a
quantidade de textos no header. </br>
     * Exemplo: </br>
     * <pre>Descrição da pergunta
     *   header
     *
     *   descrição da primeira subpergunta
     *      sequencia de componentes (alternativas) desta
subpergunta
     *   descrição da segunda subpergunta
     *      sequencia de componentes (alternativas) desta
subpergunta
     *   etc...</pre>
     *
     * @param header
     * @param nodes
     * @param i
     * @param cStack
     * @return
     */
    public boolean isSimpleMatrix(Cluster header, List<MyNode>
nodes, int i, Stack<Cluster> cStack) {
        int count = 0;
        Cluster head = header != null ? header :
            !cStack.isEmpty() ? cStack.peek() : null;
        MyNode nTmp = nodes.get(i);

        if(head == null || !head.isAllText())
            return false;

        String prefix =
nTmp.getDewey().getCommonPrefix(head.last().getDewey()), pTmp = "";

        if(header == null) {// 1* pergunta da matriz
            JSONObject obj =
CONFIGS.getJSONObject("distBetweenHeaderAndFirstAlternative");
            DeweyExt dist = this.distMatrix.getDist(head.last(),
nTmp);
            if(dist.getHeight() > obj.getInt("height") ||
                    dist.getWidth() > obj.getInt("width"))
                return false;
        }

        //Conta a quantidade de componentes em sequência
        do{
            count++;
            if(i+count < nodes.size()) {
                nTmp = nodes.get(i+count);
                pTmp =
nTmp.getDewey().getCommonPrefix(head.last().getDewey());//Garantia
para não pegar componentes a +
            }else
                nTmp = null;
        }while(nTmp != null && nTmp.isComponent() &&
prefix.equals(pTmp));

        //A quantidade encontrada acima deve ser a mesma de textos
no head da matriz
        return head.size() == count;
    }
```

```
    public boolean isRadioInputOrCheckboxWithHeader(List<MyNode>
nodes, int i, Cluster head){
        int count = 0;
        MyNode nTmp = nodes.get(i);
        String type = nTmp.getType().toString();

        if(head == null || !head.isAllText()
|| !type.matches("RADIO_INPUT|CHECKBOX_INPUT"))
            return false;

        JSONObject obj =
CONFIGS.getJSONObject("distBetweenHeaderAndFirstAlternative");
        DeweyExt dist = this.distMatrix.getDist(head.last(), nTmp);
        if(dist.getHeight() > obj.getInt("height") ||
                dist.getWidth() > obj.getInt("width"))
            return false;

        //Conta a quantidade de componentes em sequência
        do{
            count++;
            if(i+count < nodes.size())
                nTmp = nodes.get(i+count);
            else
                nTmp = null;
        }while(nTmp != null && nTmp.isA(type));

        //A quantidade encontrada acima deve ser a mesma de textos
no head da matriz
        return head.size() == count;
    }

    public boolean checkCommonPrefix(MyNode n1, MyNode n2, String
prefix) {
        if(n1 == null || n2 == null || prefix.isEmpty()) return
false;

        String tmp = n1.getDewey().getCommonPrefix(n2.getDewey());
        return tmp.equals(prefix);
    }

    public boolean checkDistForQWithSubQs(MyNode n1, MyNode n2) {
        JSONObject obj =
CONFIGS.getJSONObject("distBetweenTextsInQuestionWithSubQuestions");
        DeweyExt dist = this.distMatrix.getDist(n1, n2);
        // Checa o 1* elemento depois do BODY
        if(n1.getDewey().getNumbers().get(1) !=
n2.getDewey().getNumbers().get(1))
            return false;
        return dist.getHeight() <= obj.getInt("height") &&
dist.getWidth() <= obj.getInt("width");
    }

    /**
     * Utiliza uma heurística para verificar se é uma pergunta com
subperguntas.<br>
     * <ul>
     * <li>Middle e bottom devem esta perto um do outro.</li>
     * <li>O prefixo entre middle e qWithSubQsDesc e entre bottom =
nodes.get(currentI+1)
     *     e qWithSubQsDesc deve ser o mesmo.</li>
```

```
     * <li>O prefixo acima deve ser menor que o prefixo entre middle
e bottom.</li>
     * </ul>
     *
     * @param middle
     * @param qWithSubQsDesc
     * @param nodes
     * @param currentI
     * @return
     */
    public boolean isQWithSubQs(MyNode middle, MyNode
qWithSubQsDesc, List<MyNode> nodes, int currentI) {
        if(currentI+1 >= nodes.size()) return false;

        MyNode bottom = nodes.get(currentI+1);
        String prefix1 =
middle.getDewey().getCommonPrefix(qWithSubQsDesc.getDewey());
        String prefix2 =
middle.getDewey().getCommonPrefix(bottom.getDewey());

        if(checkDistForQWithSubQs(middle, bottom) &&
                checkCommonPrefix(bottom, qWithSubQsDesc, prefix1)){
            return CommonUtil.getPrefixLength(prefix1) <
CommonUtil.getPrefixLength(prefix2);
        }
        return false;
    }

    public boolean checkDistForTextsOfAlternative(MyNode n1, MyNode
n2){
        if(n1 == null || n2 == null) return false;

        JSONObject obj =
CONFIGS.getJSONObject("distBetweenTextsOfSameAlternative");
        DeweyExt dist = this.distMatrix.getDist(n1, n2);
        return dist.getHeight() <= obj.getInt("height") &&
                dist.getMaxHeight() <= obj.getInt("maxHeight") &&
                dist.getWidth() <= obj.getInt("width");
    }

    //Checagem para coisas do tipo: Hora: [ ] : [ ] ou Data: [ ] /
[ ] / [ ] ou Telefone: ( [ ] ) [ ]
    public int checkCompositeInput(List<MyNode> nodes, String type,
int currentI) {
        if(type.matches("(TEXT|NUMBER|TEL|DATE|TIME)_INPUT")){
            Cluster c = new Cluster();
            //Cria um cluster com no max os próximos 7 nodes
            for(int i = 1; i<=7; i++){
                int j = currentI+i;
                if(j < nodes.size())
                    c.add(nodes.get(j));
            }

            String tmpType = type.replace("_INPUT",
"").toLowerCase(),
                    inputTxt = "input\\[type="+tmpType+"\\]",
                    txt = c.getAllNodesText();
            String r1 = DATE_REGEX1,
                    r2 = DATE_REGEX2,
                    r3 = MONEY_REGEX1,
                    r4 = MONEY_REGEX2;
```

```
            //Check [ ] : [ ] (: [ ])?
            //  Ex:
https://www.bioinfo.mpg.de/mctq/core_work_life/core/core.jsp?languag
e=por_b
            String regex = ":\n"+inputTxt+"(\n:"+inputTxt+")?.*";
            if(txt.matches("(?ism)"+regex))
                return 0;

            //Check ( [ ] ) [ ]
            //  Ex:
http://www.almaderma.com.br/formulario/florais/infantil/contato.php
            regex = "\\)\n"+inputTxt+".*";
            if(txt.matches("(?ism)"+regex))
                return 1;

            //Check [ ] (/|-) [ ] (/|-) [ ]
            regex = r1+"\n"+inputTxt+"\n"+r1+"\n"+inputTxt+".*";
            if(txt.matches("(?ism)"+regex))
                return 2;

            //Check [ ] (/|-) Month [ ] (/|-) Day [ ] Year
            //  Ex: https://www.jotform.com/form-
templates/preview/21014328614342?preview=true
            regex =
r1+"\n"+r2+"\n"+inputTxt+"\n"+r1+"\n"+r2+"\n"+inputTxt+"\n"+r2+".*";
            if(txt.matches("(?ism)"+regex))
                return 3;

            //Check [ ] Dollars . [ ] Cents
            //  Ex:
https://gallery.wufoo.com/embed/w1qtrb451rja978/def/embedKey=w1qtrb4
51rja9786468&entsource=&referrer=https%3Awuslashwuslashwww.wufoo.com
wuslashgallerywuslashtemplateswuslashsurveyswuslash
            regex = r4+"\n"+r3+"\n"+inputTxt+"\n"+r4+".*";
            if(txt.matches("(?ism)"+regex))
                return 4;
        }

        return -1;
    }

    public boolean isSelectGroup(List<MyNode> nodes, int currentI) {
        MyNode opt = null, text = null;
        int i = currentI;

        if(i+2 >= nodes.size()) return false;

        opt = nodes.get(++i);
        text = nodes.get(++i);
        while(opt != null && opt.getType() == MyNodeType.OPTION){
            if(text.isA("OPTION")){
                if(i+1 < nodes.size()){
                    opt = text;
                    text = nodes.get(++i);
                    continue;
                }else{
                    opt = null;
                    break;
                }
            }
```

```
            if(i+2 < nodes.size()){
                opt = nodes.get(++i);
                text = nodes.get(++i);
            }else
                opt = null;
        }

        if(opt != null)
            i -= 1;

        text = nodes.get(i);
        return text.isA("SELECT") || (text.isText() &&
                (text.getText().matches(DATE_REGEX1) ||
text.getText().matches("(?i)"+DATE_REGEX2)));
    }

    /**
     * Verifica se é um RADIO/CHECKBOX INPUT com o padrão:
     *      text -> input -> text -> input.
     *
     * @param nodes
     * @param currentI
     * @return
     */
    public boolean checkIfTextIsAbove(List<MyNode> nodes, int
currentI) {
        MyNode input = null, text = null, img = null;
        int i = currentI;
        boolean hasImgAbove = false;

        input = nodes.get(i);

        // Verifica se os elementos acima podem ser considerados
        // a descrição da alternativa e da pergunta
        text = i-1 <= nodes.size()-1 ? nodes.get(i-1) : null;
        if((text == null || !text.isA("text"))
|| !areCompAndTextNear(input, text))
            return false;
        text = i-2 <= nodes.size()-1 ? nodes.get(i-2) : null;
        if(text != null && text.isA("img")){
            text = (i-3 >= 0 && i-3 <= nodes.size()-1) ?
nodes.get(i-3) : null;
            hasImgAbove = true;
        }
        if((text == null || !text.isA("text"))
|| !areDescAndPergNear(text, input))
            return false;

        // Verifica se segue o padrão: img -> text -> input -> img,
        // se sim, então deve ser seguro retornar true
        img = i+1 < nodes.size() ? nodes.get(i+1) : null;
        if(hasImgAbove && img != null && img.isA("img"))
            return true;

        // Só para garantir, verifica se seguindo o padrão:
        //      input -> text -> input -> text
        // da erro no final
        text = nodes.get(++i);
        while(input != null &&
```

```
                (input.getType() == MyNodeType.CHECKBOX_INPUT ||
input.getType() == MyNodeType.RADIO_INPUT) &&
                text.getType() == MyNodeType.TEXT){
            if(!areCompAndTextNear(input, text))
                return true;

            if(i+2 < nodes.size()){
                input = nodes.get(++i);
                text = nodes.get(++i);
            }else
                input = null;
        }

        return false;
    }

    //Ex:
http://lap.umd.edu/surveys/census/files/surveya1pagesbytopic/page2.h
tml
    //Ex: https://statpac.com/online-
surveys/resturaunt_customer_satisfaction_survey.htm
    public boolean isEvaluationLevels(Cluster desc, Stack<Cluster>
cStack, boolean checkingTheMatrix){
        MyNode above = null;
        if(!checkingTheMatrix && cStack.size() >= 1)
            above = cStack.peek().last();
        else if(checkingTheMatrix && cStack.size() >= 2)
            //Gambiarra porque estou usando peek() para pegar a
descrição da matriz
            above = cStack.get(cStack.size()-2).last();

        if(above != null && desc.size() == 2){
            JSONObject obj =
CONFIGS.getJSONObject("distBetweenEvaluationLevelsAndDesc");
            DeweyExt dist = this.distMatrix.getDist(desc.first(),
above);
            if(dist.getHeight() <= obj.getInt("height")){
                MyNode first = desc.first(), last = desc.last();
                String regex =
CONFIGS.getString("evaluationLevelsWordsRegex");
                if(first.getText().matches(regex) &&
last.getText().matches(regex)){
                    int fspaces =
StringUtils.countMatches(first.getText(), " "), fnewlines =
StringUtils.countMatches(first.getText(), "\n"),
                            lspaces =
StringUtils.countMatches(last.getText(), " "), lnewlines =
StringUtils.countMatches(last.getText(), "\n");
                    int fsum = fspaces+fnewlines, lsum =
lspaces+lnewlines;
                    int max =
CONFIGS.getInt("maxSpacesAndNewLinesInEvaluationLevels");

                    if((fsum > 0 && fsum <= max) && (lsum > 0 &&
lsum <= max))
                        return true;
                }
            }
        }
        return false;
    }
```

```
    /**
     * Verifica se é um RADIO/CHECKBOX INPUT com apenas imagens.
     *
     * @param nodes
     * @param currentI
     * @return
     */
    public boolean isImageCheckboxOrRadioInput(List<MyNode> nodes,
int currentI){
        // Perguntas de RADIO/CHECKBOX_INPUT seguem o padrão:
        //      input -> img -> input -> img ...
        boolean ret = nodes.get(currentI).isA("RADIO_INPUT") ||
                nodes.get(currentI).isA("CHECKBOX_INPUT");
        ret = ret && nodes.get(currentI+1).isImage();
        ret = ret && nodes.get(currentI+2).isA("RADIO_INPUT") ||
                nodes.get(currentI+2).isA("CHECKBOX_INPUT");
        ret = ret && nodes.get(currentI+3).isImage();

        return ret;
    }

    // Métodos/Blocos estáticos
    static {
        //Load parameters
        CONFIGS = CommonConfiguration.getInstance().getParameters();

        CommonLogger.debug("RulesChecker:> Static block executed!");
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/crawler/Crawler.java

```
package br.ufsc.tcc.extractor.crawler;

import java.util.ArrayList;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.util.CommonConfiguration;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.builder.QuestionarioBuilder;
import br.ufsc.tcc.extractor.database.manager.QuestionarioManager;
import br.ufsc.tcc.extractor.model.Questionario;
import edu.uci.ics.crawler4j.crawler.Page;
import edu.uci.ics.crawler4j.crawler.WebCrawler;
import edu.uci.ics.crawler4j.parser.HtmlParseData;
import edu.uci.ics.crawler4j.url.WebURL;

public class Crawler extends WebCrawler {
    private BasicConnection qConn;
    private QuestionarioManager qManager;
    private QuestionarioBuilder qBuilder;

    @Override
    public void onStart() {
        this.qConn = new
BasicConnection(CommonConfiguration.getInstance().getExtractorDataba
seConfigs());
        this.qManager = new QuestionarioManager(this.qConn);
        this.qBuilder = new QuestionarioBuilder();
```

```
    }

    @Override
    public void onBeforeExit() {
        if(this.qConn != null)
            this.qConn.close();
    }

    @Override
    public boolean shouldVisit(Page referringPage, WebURL url) {
        return true;
    }

    @Override
    public void visit(Page page) {
        if(page.getParseData() instanceof HtmlParseData){
            HtmlParseData htmlParseData = (HtmlParseData)
page.getParseData();
            String link = page.getWebURL().getURL();
            CommonLogger.debug("Link: {}", link);

            Document doc = null;
            try{
                doc = Jsoup.parse(htmlParseData.getHtml());
                Element root = doc.select("body").get(0);

                qBuilder.setCurrentLink(link);
                ArrayList<Questionario> questionarios =
qBuilder.build(root, doc.title());
                for(Questionario q : questionarios){
                    this.qManager.save(q);


System.out.println("<"+Thread.currentThread().getName()+">URL: "
+link+ " - SAVE DONE!");
                }
                questionarios.clear();
            }catch (Exception e) {
                CommonLogger.error(e);
            }
        }
    }

    @Override
    protected void onUnhandledException(WebURL webUrl, Throwable e){
        String urlStr = (webUrl == null ? "NULL" : webUrl.getURL());
        CommonLogger.info("Unhandled exception while fetching {}:
{}", urlStr, e.getMessage());
        CommonLogger.error(e);
    }

    @Override
    protected void onPageBiggerThanMaxSize(String urlStr, long
pageSize) {
        CommonLogger.info("Skipping a URL: {} which was bigger
( {} ) than max allowed size", urlStr, pageSize);
    }

    @Override
    protected void onUnexpectedStatusCode(String urlStr, int
statusCode, String contentType, String description) {
```

```
        CommonLogger.info("Skipping URL: {}, StatusCode: {}, {},
{}", urlStr, statusCode, contentType, description);
    }

    @Override
    protected void onContentFetchError(WebURL webUrl) {
        CommonLogger.info("Can't fetch content of: {}",
webUrl.getURL());
    }

    @Override
    protected void onParseError(WebURL webUrl) {
        CommonLogger.info("Parsing error of: {}", webUrl.getURL());
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/database/dao/AlternativaDao.java

```
package br.ufsc.tcc.extractor.database.dao;

import java.util.HashMap;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.database.dao.BasicDao;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Pergunta;

public class AlternativaDao extends BasicDao {
    public AlternativaDao(BasicConnection c) {
        super(c, "Alternativa");
    }

    public void save(Alternativa a) throws Exception {
        HashMap<String, Object> data = new HashMap<>();

        Pergunta p = a.getPergunta();
        if(p != null){
            data.put("Pergunta_idPergunta", p.getId());
        }

        data.put("DESCRICAO", a.getDescricao());

        this.insert(data);
        a.setId(getLastUID());
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/database/dao/FiguraDao.java

```
package br.ufsc.tcc.extractor.database.dao;

import java.util.HashMap;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.database.dao.BasicDao;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Figura;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class FiguraDao extends BasicDao {
    public FiguraDao(BasicConnection c){
        super(c, "Figura");
    }
```

```
    public void save(Figura f) throws Exception{
        HashMap<String, Object> data = new HashMap<>();

        data.put("Legenda", f.getLegenda());
        data.put("imagem_url", f.getImage_url());

        Object dono = f.getDono();
        if(dono instanceof Questionario){
            data.put("dono", "Q");
            data.put("idDono", ((Questionario)dono).getId());
        }else if(dono instanceof Pergunta){
            data.put("dono", "P");
            data.put("idDono", ((Pergunta)dono).getId());
        }else if(dono instanceof Alternativa){
            data.put("dono", "A");
            data.put("idDono", ((Alternativa)dono).getId());
        }else{
            CommonLogger.info("FiguraDao:save()> Dono da Figura nao
eh um Questionario, "
                    + "uma Alternativa e nem uma Pergunta!\n\t<{}>",
f.getImage_url());
            return;
        }

        this.insert(data);
        f.setId(getLastUID());
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/database/dao/FormaDaPerguntaDao.java

```
package br.ufsc.tcc.extractor.database.dao;

import java.sql.ResultSet;
import java.util.HashMap;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.database.dao.BasicDao;
import br.ufsc.tcc.extractor.model.FormaDaPergunta;

public class FormaDaPerguntaDao extends BasicDao {
    public FormaDaPerguntaDao(BasicConnection c) {
        super(c, "FormaDaPergunta");
    }

    public void save(FormaDaPergunta fp) throws Exception{
        HashMap<String, Object> data = new HashMap<>();

        data.put("DESCRICAO", fp.getDescricao());

        this.insert(data);
        fp.setId(getLastUID());
    }

    public HashMap<String, FormaDaPergunta> getAll() throws
Exception {
        HashMap<String, FormaDaPergunta> resp = new HashMap<>();

        this.select("*");// select all

        ResultSet result = this.getResultSet();
```

```
        FormaDaPergunta tmp = null;
        while(result != null && result.next()){
            tmp = new
FormaDaPergunta(result.getLong("idFormaDaPergunta"),
                    result.getString("DESCRICAO"));

            resp.put(tmp.getDescricao(), tmp);
        }

        return resp;
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/database/dao/GrupoDao.java

```
package br.ufsc.tcc.extractor.database.dao;

import java.util.HashMap;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.database.dao.BasicDao;
import br.ufsc.tcc.extractor.model.Grupo;

public class GrupoDao extends BasicDao {
    public GrupoDao(BasicConnection c) {
        super(c, "Grupo");
    }

    public void save(Grupo g) throws Exception{
        HashMap<String, Object> data = new HashMap<>();

        data.put("Questionario_idQuestionario",
g.getQuestionario().getId());
        data.put("ASSUNTO", g.getAssunto());

        this.insert(data);
        g.setId(getLastUID());
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/database/dao/PerguntaDao.java

```
package br.ufsc.tcc.extractor.database.dao;

import java.util.HashMap;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.database.dao.BasicDao;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class PerguntaDao extends BasicDao {
    public PerguntaDao(BasicConnection c) {
        super(c, "Pergunta");
    }

    public void save(Pergunta p) throws Exception{
        HashMap<String, Object> data = new HashMap<>();

        Questionario q = p.getQuestionario();
        Pergunta pai = p.getPai();

        if(pai != null)
            data.put("PerguntaPai_idPergunta", pai.getId());
```

```
        data.put("Questionario_idQuestionario", q.getId());

        if(p.getForma() != null)
            data.put("FormaDaPergunta_idFormaDaPergunta",
p.getForma().getId());
        else
            CommonLogger.info("PerguntaDao::save()> Pergunta sem
FormaDaPergunta!\n\t [{} / {}]",
                    q.getLink_doc(), p.getDescricao());

        if(p.getGrupo() != null)
            data.put("Grupo_idGrupo", p.getGrupo().getId());

        data.put("TipoPergunta", p.getTipo());
        data.put("DESCRICAO", p.getDescricao());

        this.insert(data);
        p.setId(getLastUID());
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/database/dao/QuestionarioDao.java

```
package br.ufsc.tcc.extractor.database.dao;

import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.HashMap;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.database.dao.BasicDao;
import br.ufsc.tcc.extractor.model.Questionario;

public class QuestionarioDao extends BasicDao {
    public QuestionarioDao(BasicConnection c) {
        super(c, "Questionario");
    }

    public void save(Questionario q) throws Exception{
        HashMap<String, Object> data = new HashMap<>();

        data.put("ASSUNTO", q.getAssunto());
        data.put("LINK_DOCUMENTO", q.getLink_doc());

        this.insert(data);
        q.setId(getLastUID());
    }

    public ArrayList<String> getAllLinks() throws Exception {
        ArrayList<String> resp = new ArrayList<>();

        this.select("LINK_DOCUMENTO");

        ResultSet result = this.getResultSet();
        while(result != null && result.next()){
            resp.add(result.getString("LINK_DOCUMENTO"));
        }
        return resp;
    }

    public void clean() throws Exception {
```

```
        this.delete();
    }

    public void remove(String link) throws Exception {
        HashMap<String, Object> where = new HashMap<>();
        where.put("LINK_DOCUMENTO", link);

        this.delete(where);
    }

}
```

questionnaires_extractor/br/ufsc/tcc/extractor/database/manager/FormaDaPerguntaManager.java

```
package br.ufsc.tcc.extractor.database.manager;

import java.util.HashMap;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.database.dao.FormaDaPerguntaDao;
import br.ufsc.tcc.extractor.model.FormaDaPergunta;

/**
 * Classe de mais alto nível responsável por lidar com operações do
banco de dados
 * relacionadas a classe/tabela FormaDaPergunta.
 *
 * @author Gilney N. Mathias
 */
public class FormaDaPerguntaManager {
    private static HashMap<String, FormaDaPergunta> formas;
    private static FormaDaPerguntaDao dao;

    public static FormaDaPergunta getForma(String forma){
        if(formas == null){
            CommonLogger.info("FormaDaPerguntaManager:getForma()>
Chame a funcao 'loadFormas' "
                    +"para inicializar as informacoes!");
            return null;
        }
        forma = forma.toUpperCase();
        if(!formas.containsKey(forma)){
            CommonLogger.info("FormaDaPerguntaManager:getForma()>
forma não encontrada: " +forma);
            return null;
        }
        return formas.get(forma);
    }

    public static synchronized void loadFormas(BasicConnection c){
        if(formas != null) return;
        dao = new FormaDaPerguntaDao(c);

        try {
            // Cache as formas de pergunta para serem
            // usadas na extração dos dados
            formas = dao.getAll();

            CommonLogger.debug("{} carregou as formas das perguntas
do banco de dados.",
                    Thread.currentThread().getName());
```

```
        } catch (Exception e) {
            // Database não deve ta funcionando, então mata a
aplicação
            CommonLogger.fatalError(e);
        }
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/database/manager/PerguntaManager.java

```
package br.ufsc.tcc.extractor.database.manager;

import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.extractor.database.dao.AlternativaDao;
import br.ufsc.tcc.extractor.database.dao.PerguntaDao;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Pergunta;

/**
 * Classe de mais alto nível responsável por lidar com operações do
banco de dados
 * relacionadas a classe/tabela Pergunta.
 *
 * @author Gilney N. Mathias
 */
public class PerguntaManager {
    private PerguntaDao perguntaDao;
    private AlternativaDao alternativaDao;

    public PerguntaManager(BasicConnection c){
        perguntaDao = new PerguntaDao(c);
        alternativaDao = new AlternativaDao(c);
    }

    public void save(Pergunta p) throws Exception {
        perguntaDao.save(p);
        for(Alternativa a : p.getAlternativas()){
            alternativaDao.save(a);
        }
        for(Pergunta child : p.getFilhas()){
            this.save(child);
        }
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/database/manager/QuestionarioManager.java

```
package br.ufsc.tcc.extractor.database.manager;

import java.util.ArrayList;
import org.json.JSONArray;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.database.dao.FiguraDao;
import br.ufsc.tcc.extractor.database.dao.GrupoDao;
import br.ufsc.tcc.extractor.database.dao.QuestionarioDao;
import br.ufsc.tcc.extractor.model.Figura;
import br.ufsc.tcc.extractor.model.Grupo;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;
```

```java
/**
 * Classe de mais alto nível responsável por lidar com operações do
banco de dados
 * relacionadas a classe/tabela Questionario.
 *
 * @author Gilney N. Mathias
 */
public class QuestionarioManager {
    private QuestionarioDao questionarioDao;
    private GrupoDao grupoDao;
    private PerguntaManager perguntaManager;
    private FiguraDao figuraDao;

    public QuestionarioManager(BasicConnection c) {
        questionarioDao = new QuestionarioDao(c);
        grupoDao = new GrupoDao(c);
        perguntaManager = new PerguntaManager(c);
        figuraDao = new FiguraDao(c);

        // Carrega os dados do banco de dados
        FormaDaPerguntaManager.loadFormas(c);
    }

    public void save(Questionario q) throws Exception {
        questionarioDao.save(q);
        for(Grupo g : q.getGrupos()){
            grupoDao.save(g);
        }
        for(Pergunta p : q.getPerguntas()){
            perguntaManager.save(p);
        }
        for(Figura f : q.getFiguras()){
            figuraDao.save(f);
        }
    }

    public void cleanDatabase() {
        try{
            this.questionarioDao.clean();
        }catch(Exception e){
            CommonLogger.error(e);
        }
    }

    public void deleteLinks(JSONArray links){
        try{
            ArrayList<String> dbLinks =
this.questionarioDao.getAllLinks();
            for(int i = 0; i<links.length(); i++){
                String link = (String) links.get(i);
                if(dbLinks.contains(link))
                    this.questionarioDao.remove(link);
            }
        }catch(Exception e){
            CommonLogger.error(e);
        }
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/impl/ChoiceInputExtractor.java

```
package br.ufsc.tcc.extractor.extractor.impl;

import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.model.MyNodeType;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.builder.RulesChecker;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.extractor.IPerguntaExtractor;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Figura;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class ChoiceInputExtractor implements IPerguntaExtractor {
    private Questionario currentQ;
    private Pergunta currentP;
    private RulesChecker checker;

    public ChoiceInputExtractor(Questionario currentQ, Pergunta
currentP, RulesChecker checker) {
        this.currentQ = currentQ;
        this.currentP = currentP;
        this.checker = checker;
    }

    @Override
    public int extract(Cluster desc, List<MyNode> nodes, int
currentI) {
        if(checker.checkIfTextIsAbove(nodes, currentI)){
            //Ex: https://www.nbrii.com/customer-survey-questions-
template/
            return this.extractWithTextAbove(desc, nodes, currentI);
        }else if(this.checker.isImageCheckboxOrRadioInput(nodes,
currentI)){
            //Ex: https://survey.zoho.com/surveytemplate/Events-
Entertainment%20Evaluation%20Survey [+/-]
            return this.extractWithImageOnly(nodes, currentI);
        }else
            return this.extractSimplePattern(desc, nodes, currentI);
    }

    private int extractSimplePattern(Cluster desc, List<MyNode>
nodes, int currentI) {
        MyNode img = null, input = null, text = null, tmp = null,
lastDescNode = desc.last();
        boolean isImgInputQuestion = false,
                isTextImgQuestion = false,
                isImgQuestion = false;
        String txt = "", commonPrefix = "";

        input = nodes.get(currentI);


this.currentP.setForma(FormaDaPerguntaManager.getForma(input.getType
().toString()));
        if(input.isA("CHECKBOX_INPUT"))
```

```
            CommonLogger.debug("\tCheckbox Input:");
        else
            CommonLogger.debug("\tRadio Input:");

        // Faz verificações sobre imagens no meio das alternativas
        img = nodes.get(currentI-1);
        text = nodes.get(++currentI);
        tmp = nodes.get(currentI+1);
        // Perguntas com imagens pode seguir o padrão:
        //      img -> input -> text -> img -> input -> text ...
        // Ex: https://www.survio.com/modelo-de-pesquisa/avaliacao-
de-um-e-shop
        isImgInputQuestion = img.isImage() && tmp.isImage();

        if(!isImgInputQuestion) {
            img = tmp;
            tmp = currentI+4 < nodes.size() ?
nodes.get(currentI+4) : null;
            // Ou pode seguir o padrão:
            //      input -> text -> img -> input -> text -> img
            // Ex: http://www.objectplanet.com/opinio/s/s?s=259
            isTextImgQuestion = img.isImage() && tmp != null &&
tmp.isImage();
            if(isTextImgQuestion)
                img = nodes.get(++currentI);
            else
                img = null;
            tmp = nodes.get(currentI+1);
        }

        isImgQuestion = isImgInputQuestion || isTextImgQuestion;
        while(input != null &&
                (input.isA("CHECKBOX_INPUT") ||
input.isA("RADIO_INPUT")) &&
                text.getType() == MyNodeType.TEXT &&
                (!isImgQuestion || (img != null && img.isImage()))){

            if(!this.checker.areCompAndTextNear(input, text))
                break;

            // Verifica o prefixo comum para ver se terminou esta
pergunta
            if(commonPrefix.isEmpty()) {
                commonPrefix =
lastDescNode.getDewey().getCommonPrefix(input.getDewey());
            }else {
                String cTmp =
lastDescNode.getDewey().getCommonPrefix(input.getDewey());
                if(!cTmp.equals(commonPrefix))
                    break;
            }

            txt = text.getText();
            Object dono = null;

            if(tmp != null){
                //Ex: https://www.surveymonkey.com/r/CAHPS-Health-
Plan-Survey-40-Template [pergunta 8]
                if(text.getType() == MyNodeType.TEXT &&
tmp.getType() == MyNodeType.TEXT &&
```

```
this.checker.checkDistForTextsOfAlternative(text, tmp)){
                    txt = txt + tmp.getText();
                    ++currentI;
                    tmp = currentI+1 < nodes.size() ?
nodes.get(currentI+1) : null;
                }

                CommonLogger.debug("\t\t{}", txt);

                if(tmp != null && (tmp.isA("TEXT_INPUT") ||
tmp.isA("TEXTAREA")) &&
                        this.checker.areCompAndTextNear(tmp, text)){
                    Pergunta tmpPerg = new Pergunta(txt);
                    dono = tmpPerg;

                    if(tmp.isA("TEXT_INPUT")) {

tmpPerg.setForma(FormaDaPerguntaManager.getForma("TEXT_INPUT"));
                        CommonLogger.debug("\t\t\tCom Text Input.");
                    }else {
                        //Ex: http://www.surveymoz.com/s/course-
evaluation-survey-example

tmpPerg.setForma(FormaDaPerguntaManager.getForma("TEXTAREA"));
                        CommonLogger.debug("\t\t\tCom Textarea.");
                    }

                    tmpPerg.setQuestionario(this.currentQ);
                    this.currentP.addFilha(tmpPerg);
                    ++currentI;
                }
            }else
                CommonLogger.debug("\t\t{}", txt);

            if(dono == null){
                Alternativa tmpAlt = new Alternativa(txt);
                dono = tmpAlt;
                this.currentP.addAlternativa(tmpAlt);
            }

            if(isImgQuestion && img != null && img.isImage()){
                Figura fig = new Figura(img.getAttr("src"),
img.getAttr("alt"));
                fig.setDono(dono);
                this.currentQ.addFigura(fig);
                CommonLogger.debug("\t\t\tLegenda: {}",
fig.getLegenda());
            }

            if(currentI+1 < nodes.size() && isImgInputQuestion)
                img = nodes.get(++currentI);
            else
                img = null;
            if(currentI+2 < nodes.size()){
                input = nodes.get(++currentI);
                text = nodes.get(++currentI);
                if(isTextImgQuestion && currentI+1 < nodes.size())
                    img = nodes.get(++currentI);
                if(currentI+1 < nodes.size())
                    tmp = nodes.get(currentI+1);
```

```
            }else
                input = null;
        }
        if(input != null){
            if(isImgQuestion && img != null) --currentI;
            currentI -= 2;
            if(this.currentP.getAlternativas().size() == 0)
                currentI += 1;
        }
        if(this.currentP.getAlternativas().size() == 0)
            this.currentP.setForma(null);

        return currentI;
    }

    private int extractWithTextAbove(Cluster desc, List<MyNode>
nodes, int currentI) {
        // A principio não se preocupa com input text
        MyNode input = null, text = null, img = null, lastDescNode =
desc.last();
        boolean isImgQuestion = false;
        String txt = "", commonPrefix = "";

        input = nodes.get(currentI);


this.currentP.setForma(FormaDaPerguntaManager.getForma(input.getType
().toString()));
        if(input.isA("CHECKBOX_INPUT"))
            CommonLogger.debug("\tCheckbox Input [text above]:");
        else
            CommonLogger.debug("\tRadio Input [text above]:");

        text = nodes.get(currentI-1);
        img = nodes.get(currentI-2);
        // Perguntas com imagens seguem o padrão:
        //      img -> text -> input -> img -> text -> input ...
        isImgQuestion = img.isImage() &&
(nodes.get(currentI+1).isImage());
        while(input != null &&
                (input.isA("CHECKBOX_INPUT") ||
input.isA("RADIO_INPUT")) &&
                text.getType() == MyNodeType.TEXT &&
                (!isImgQuestion || (img != null && img.isImage()))){

            if(!this.checker.areCompAndTextNear(input, text))
                break;

            // Verifica o prefixo comum para ver se terminou esta
pergunta
            if(commonPrefix.isEmpty()) {
                commonPrefix =
lastDescNode.getDewey().getCommonPrefix(input.getDewey());
            }else {
                String cTmp =
lastDescNode.getDewey().getCommonPrefix(input.getDewey());
                if(!cTmp.equals(commonPrefix))
                    break;
            }

            txt = text.getText();
```

```
            CommonLogger.debug("\t\t{}", txt);
            Alternativa tmpAlt = new Alternativa(txt);
            this.currentP.addAlternativa(tmpAlt);

            if(isImgQuestion){
                Figura fig = new Figura(img.getAttr("src"),
img.getAttr("alt"));
                fig.setDono(tmpAlt);
                this.currentQ.addFigura(fig);
                CommonLogger.debug("\t\t\tLegenda: {}",
fig.getLegenda());
            }

            if(currentI+1 < nodes.size() && isImgQuestion)
                img = nodes.get(++currentI);
            else
                img = null;
            if(currentI+2 < nodes.size()){
                text = nodes.get(++currentI);
                input = nodes.get(++currentI);
            }else
                input = null;
        }

        if(input != null){
            if(isImgQuestion && img != null) --currentI;
            currentI -= 2;
        }
        if(this.currentP.getAlternativas().size() == 0)
            this.currentP.setForma(null);

        return currentI;
    }

    private int extractWithImageOnly(List<MyNode> nodes, int
currentI) {
        MyNode img = null,
                input = nodes.get(currentI);
        int backup_currentI = currentI, i = 1;


this.currentP.setForma(FormaDaPerguntaManager.getForma("IMAGE_"
+input.getType().toString()));
        if(input.isA("CHECKBOX_INPUT"))
            CommonLogger.debug("\tImage Checkbox Input:");
        else
            CommonLogger.debug("\tImage Radio Input:");

        img = nodes.get(++currentI);
        while(input != null &&
                (input.isA("CHECKBOX_INPUT") ||
input.isA("RADIO_INPUT")) &&
                img.isImage()){
            if(!this.checker.areCompAndTextNear(input, img))
                break;

            Alternativa tmpAlt = new Alternativa();
            Figura fig = new Figura(img.getAttr("src"),
img.getAttr("alt"));
```

```
            tmpAlt.setDescricao(fig.getLegenda().isEmpty() ? i+"" :
fig.getLegenda());
            currentP.addAlternativa(tmpAlt);

            fig.setDono(tmpAlt);
            this.currentQ.addFigura(fig);
            CommonLogger.debug("\t\t{}", fig);

            i++;
            if(currentI+2 < nodes.size()){
                input = nodes.get(++currentI);
                img = nodes.get(++currentI);
            }else
                input = null;
        }
        if(input != null)
            currentI -= 2;
        if(currentI == backup_currentI)//Não salvo nenhuma imagem
            this.currentP.setForma(null);
        return currentI;
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/impl/ChoiceInputWithHeaderExtractor.java

```
package br.ufsc.tcc.extractor.extractor.impl;

import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.builder.RulesChecker;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.extractor.IPerguntaExtractor;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class ChoiceInputWithHeaderExtractor implements
IPerguntaExtractor {
    private Pergunta currentP;

    public ChoiceInputWithHeaderExtractor(Questionario currentQ,
Pergunta currentP, RulesChecker checker) {
        this.currentP = currentP;
    }

    @Override
    public int extract(Cluster desc, List<MyNode> nodes, int
currentI) {
        MyNode input = nodes.get(currentI);
        String type = input.getType().toString();
        Cluster head = desc;
        int j = 0;


this.currentP.setForma(FormaDaPerguntaManager.getForma(type));
        if(input.isA("CHECKBOX_INPUT"))
            CommonLogger.debug("\tCheckbox Input [with header]:");
        else
```

```
            CommonLogger.debug("\tRadio Input [with header]:");

        while(input != null && input.isA(type) && j < head.size()){
            String text = head.get(j).getText();
            CommonLogger.debug("\t\t{}", text);

            Alternativa alt = new Alternativa(text);
            this.currentP.addAlternativa(alt);

            if(currentI+1 < nodes.size())
                input = nodes.get(++currentI);
            else
                input = null;
            j++;
        }
        if(input != null)
            currentI--;
        return currentI;
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/impl/MultiCompExtractor.java

```
package br.ufsc.tcc.extractor.extractor.impl;

import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.model.MyNodeType;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.builder.RulesChecker;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.extractor.IPerguntaExtractor;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class MultiCompExtractor implements IPerguntaExtractor {
    private Questionario currentQ;
    private Pergunta currentP;
    private RulesChecker checker;

    public MultiCompExtractor(Questionario currentQ, Pergunta
currentP, RulesChecker checker) {
        this.currentQ = currentQ;
        this.currentP = currentP;
        this.checker = checker;
    }

    @Override
    public int extract(Cluster desc, List<MyNode> nodes, int
currentI) {
        MyNode input = null, lastInput = null;
        MyNodeType multiCompType = null;
        int i = 1;


currentP.setForma(FormaDaPerguntaManager.getForma("MULTI_COMP"));
        CommonLogger.debug("\tMulti Comp:");

        input = nodes.get(currentI);
```

```
        multiCompType = input.getType();
        while(input != null && input.getType() == multiCompType){
            if(lastInput != null &&
                    !this.checker.areCompAndTextNear(lastInput,
input))
                break;

            String description = input.getAttr("placeholder");
            if(description.isEmpty())
                description = ""+ (i++);

            Pergunta tmpPerg = new Pergunta(description);

tmpPerg.setForma(FormaDaPerguntaManager.getForma(multiCompType.toStr
ing()));
            tmpPerg.setQuestionario(currentQ);
            this.currentP.addFilha(tmpPerg);
            CommonLogger.debug("\t\tText: {} - Comp: {}",
tmpPerg.getDescricao(), input.getText());

            lastInput = input;
            if(currentI+1 < nodes.size())
                input = nodes.get(++currentI);
            else
                input = null;
        }
        //Se input != null então o loop passo da pergunta e entro na
proxima e por isso,
        //deve-se voltar o index para o final da pergunta
        if(input != null)
            --currentI;
        //Se não tem nenhuma filha/alternativa quer dizer que o loop
não
        //completo nenhuma vez
        if(this.currentP.getFilhas().size() == 0)
            this.currentP.setForma(null);

        return currentI;
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/impl/ RatingExtractor.java

```
package br.ufsc.tcc.extractor.extractor.impl;

import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.model.MyNodeType;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.builder.RulesChecker;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.extractor.IPerguntaExtractor;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class RatingExtractor implements IPerguntaExtractor {
    private Pergunta currentP;
    private RulesChecker checker;
```

```
    public RatingExtractor(Questionario currentQ, Pergunta currentP,
RulesChecker checker) {
        this.currentP = currentP;
        this.checker = checker;
    }

    @Override
    public int extract(Cluster desc, List<MyNode> nodes, int
currentI) {
        MyNode input = null, lastInput = null;
        int i = 1;


currentP.setForma(FormaDaPerguntaManager.getForma("RATING"));
        CommonLogger.debug("\tRating:");

        input = nodes.get(currentI);
        while(input != null && input.getType() ==
MyNodeType.RADIO_INPUT){
            if(lastInput != null &&
                    !this.checker.areCompAndTextNear(lastInput,
input))
                break;

            Alternativa tmpAlt = new Alternativa(""+ (i++));
            this.currentP.addAlternativa(tmpAlt);
            CommonLogger.debug("\t\t{}", tmpAlt.getDescricao());

            lastInput = input;
            if(currentI+1 < nodes.size())
                input = nodes.get(++currentI);
            else
                input = null;
        }
        if(input != null)
            --currentI;
        if(this.currentP.getAlternativas().size() == 0)
            this.currentP.setForma(null);

        return currentI;
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/impl/SelectExtractor.java

```
package br.ufsc.tcc.extractor.extractor.impl;

import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.builder.RulesChecker;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.extractor.IPerguntaExtractor;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class SelectExtractor implements IPerguntaExtractor {
```

```
    private Pergunta currentP;
    private RulesChecker checker;

    public SelectExtractor(Questionario currentQ, Pergunta currentP,
RulesChecker checker) {
        this.currentP = currentP;
        this.checker = checker;
    }

    @Override
    public int extract(Cluster desc, List<MyNode> nodes, int
currentI) {
        MyNode opt = null, text = null;

        if(currentI+2 >= nodes.size()) return currentI;

        if(checker.isSelectGroup(nodes, currentI)){

currentP.setForma(FormaDaPerguntaManager.getForma("SELECT_GROUP"));
            CommonLogger.debug("\tSelect Group:");
            return this.extractSelectGroup(nodes, currentI);
        }


currentP.setForma(FormaDaPerguntaManager.getForma("SELECT"));
        CommonLogger.debug("\tSelect:");

        opt = nodes.get(++currentI);
        text = nodes.get(++currentI);
        while(opt != null && opt.isA("OPTION")){
            if(text.isA("OPTION")){
                if(currentI+1 < nodes.size()){
                    opt = text;
                    text = nodes.get(++currentI);
                    continue;
                }else{
                    opt = null;
                    break;
                }
            }

            if(!this.checker.areCompAndTextNear(opt, text))
                break;

            Alternativa tmpAlt = new Alternativa(text.getText());
            this.currentP.addAlternativa(tmpAlt);
            CommonLogger.debug("\t\t{}", tmpAlt.getDescricao());

            if(currentI+2 < nodes.size()){
                opt = nodes.get(++currentI);
                text = nodes.get(++currentI);
            }else
                opt = null;
        }
        if(opt != null) {
            //Ex: http://www.123contactform.com/js-form--37229.html
[perg 8]
            if(opt.isA("OPTION"))
                currentI -= 1;
            else
                currentI -= 2;
```

```
        }
        if(this.currentP.getAlternativas().size() == 0)
            this.currentP.setForma(null);

        return currentI;
    }

    private int extractSelectGroup(List<MyNode> nodes, int currentI)
{
        MyNode opt = null, text = null;
        Pergunta p = new Pergunta((currentP.getFilhas().size()+1)
+"");
        p.setPai(currentP);

        p.setForma(FormaDaPerguntaManager.getForma("SELECT"));
        CommonLogger.debug("\t\tSelect:");

        opt = nodes.get(++currentI);
        text = nodes.get(++currentI);
        while(opt != null && opt.isA("OPTION") &&
                (text.isA("TEXT") || text.isA("OPTION"))){
            if(text.isA("OPTION")){
                if(currentI+1 < nodes.size()){
                    opt = text;
                    text = nodes.get(++currentI);
                    continue;
                }else{
                    opt = null;
                    break;
                }
            }

            if(!this.checker.areCompAndTextNear(opt, text))
                break;

            Alternativa tmpAlt = new Alternativa(text.getText());
            p.addAlternativa(tmpAlt);
            CommonLogger.debug("\t\t\t{}", tmpAlt.getDescricao());

            if(currentI+2 < nodes.size()){
                opt = nodes.get(++currentI);
                text = nodes.get(++currentI);
            }else
                opt = null;
        }
        if(opt != null)
            currentI -= 2;
        if(p.getAlternativas().size() > 0)
            this.currentP.addFilha(p);

        if(currentI+2 < nodes.size()){
            text = nodes.get(currentI+1);
            if(text.isText()){
                if(text.getText().matches(RulesChecker.DATE_REGEX1))
                    currentI++;
                else
if(text.getText().matches("(?i)"+RulesChecker.DATE_REGEX2)){
                    p.setDescricao(text.getText());
                    currentI++;
                }
            }
```

```
            opt = nodes.get(currentI+1);
            if(opt.isA("SELECT")){
                currentI++;
                return this.extractSelectGroup(nodes, currentI);
            }
        }

        if(this.currentP.getFilhas().size() == 0)
            this.currentP.setForma(null);

        return currentI;
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/impl/SimpleMatrixExtractor.java

```
package br.ufsc.tcc.extractor.extractor.impl;

import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.model.MyNodeType;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.builder.RulesChecker;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.extractor.IPerguntaExtractor;
import br.ufsc.tcc.extractor.model.Alternativa;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

/**
 * Classe responsável pela extração de 'simple matrices'. </br>
 * Uma 'simple matrix' é uma matriz que possui simplesmente
componentes em sequencia. </br>
 * A quantidade de componentes em sequencia deve bater com a
quantidade de textos no header. </br>
 * Exemplo: </br>
 * <pre>Descrição da pergunta
 *   header
 *
 *   descrição da primeira subpergunta
 *      sequencia de componentes (alternativas) desta subpergunta
 *   descrição da segunda subpergunta
 *      sequencia de componentes (alternativas) desta subpergunta
 *   etc...</pre>
 *
 * @author Gilney N. Mathias
 *
 */
public class SimpleMatrixExtractor implements IPerguntaExtractor {
    private Questionario currentQ;
    private Pergunta currentP;

    public SimpleMatrixExtractor(Questionario currentQ, Pergunta
currentP, RulesChecker checker) {
        this.currentQ = currentQ;
        this.currentP = currentP;
    }

    @Override
```

```
    public int extract(Cluster desc, List<MyNode> nodes, int
currentI) {
        MyNode input = null;
        MyNodeType lastCompType = null;
        Cluster lastMatrixHead = desc;
        //Esta matriz possui componentes mistos?
        boolean isMix = false;
        int j = 0;

        CommonLogger.debug("\tSimple Matrix:");
        input = nodes.get(currentI);
        while(input != null && j < lastMatrixHead.size() &&
input.isComponent()){
            if(!isMix && lastCompType != null && lastCompType !=
input.getType())
                isMix = true;
            lastCompType = input.getType();

            String text = lastMatrixHead.get(j).getText();
            CommonLogger.debug("\t\tText: {} - Comp: {}", text,
input.getText());

            if(lastCompType == MyNodeType.RADIO_INPUT ||
lastCompType == MyNodeType.CHECKBOX_INPUT){
                Alternativa alt = new Alternativa(text);
                this.currentP.addAlternativa(alt);
            }else{
                Pergunta tmpPerg = new Pergunta(text);

tmpPerg.setForma(FormaDaPerguntaManager.getForma(lastCompType.toStri
ng()));
                tmpPerg.setQuestionario(currentQ);
                this.currentP.addFilha(tmpPerg);
            }

            if(currentI+1 < nodes.size())
                input = nodes.get(++currentI);
            else
                input = null;
            j++;
        }
        if(input != null)
            currentI--;

        if(isMix)

this.currentP.setForma(FormaDaPerguntaManager.getForma("MIX_COMP_GRO
UP"));
        else

this.currentP.setForma(FormaDaPerguntaManager.getForma(lastCompType.
toString()));

        return currentI;
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/impl/SingleInputExtractor.java

```
package br.ufsc.tcc.extractor.extractor.impl;
```

```
import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.builder.RulesChecker;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.extractor.IPerguntaExtractor;
import br.ufsc.tcc.extractor.model.FormaDaPergunta;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class SingleInputExtractor implements IPerguntaExtractor {
    @SuppressWarnings("unused")
    private Questionario currentQ;
    private Pergunta currentP;
    private RulesChecker checker;

    public SingleInputExtractor(Questionario currentQ, Pergunta
currentP, RulesChecker checker) {
        this.currentQ = currentQ;
        this.currentP = currentP;
        this.checker = checker;
    }

    @Override
    public int extract(Cluster desc, List<MyNode> nodes, int
currentI) {
        String type = nodes.get(currentI).getType().toString();
        CommonLogger.debug("\tInput [{}].", type);
        currentP.setForma(FormaDaPerguntaManager.getForma(type));

        int ret = checker.checkCompositeInput(nodes, type,
currentI);
        switch(ret){
        case 0://Check [ ] : [ ] (: [ ])?
            currentI += 2;
            break;
        case 1://Check ( [ ] ) [ ]
            currentI += 2;
            break;
        case 2://Check [ ] (/|-) [ ] (/|-) [ ]
            currentI += 4;
            break;
        case 3:{//Check [ ] (/|-) Month [ ] (/|-) Day [ ] Year
            FormaDaPergunta forma =
FormaDaPerguntaManager.getForma(type);
            CommonLogger.debug("\tInput [{}].", type+"_GROUP");

currentP.setForma(FormaDaPerguntaManager.getForma(type+"_GROUP"));

            String txt = nodes.get(currentI+2).getText();
            Pergunta p = new Pergunta(txt, forma);
            currentP.addFilha(p);
            CommonLogger.debug("\t\t{}", txt);

            txt = nodes.get(currentI+5).getText();
            p = new Pergunta(txt, forma);
            currentP.addFilha(p);
            CommonLogger.debug("\t\t{}", txt);
```

```
            txt = nodes.get(currentI+7).getText();
            p = new Pergunta(txt, forma);
            currentP.addFilha(p);
            CommonLogger.debug("\t\t{}", txt);

            currentI += 7;
            break;
        }case 4:{//Check [ ] Dollars . [ ] Cents
            FormaDaPergunta forma =
FormaDaPerguntaManager.getForma(type);
            CommonLogger.debug("\tInput [{}].", type+"_GROUP");

currentP.setForma(FormaDaPerguntaManager.getForma(type+"_GROUP"));

            String txt = nodes.get(currentI+1).getText();
            Pergunta p = new Pergunta(txt, forma);
            currentP.addFilha(p);
            CommonLogger.debug("\t\t{}", txt);

            txt = nodes.get(currentI+4).getText();
            p = new Pergunta(txt, forma);
            currentP.addFilha(p);
            CommonLogger.debug("\t\t{}", txt);

            currentI += 4;
            break;
        }
        default:
            break;
        }
        return currentI;
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/impl/TextAreaExtractor.java

```
package br.ufsc.tcc.extractor.extractor.impl;

import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.extractor.builder.RulesChecker;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.extractor.IPerguntaExtractor;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public class TextAreaExtractor implements IPerguntaExtractor {
    private Pergunta currentP;

    public TextAreaExtractor(Questionario currentQ, Pergunta
currentP, RulesChecker checker) {
        this.currentP = currentP;
    }

    @Override
    public int extract(Cluster desc, List<MyNode> nodes, int
currentI) {
        CommonLogger.debug("\tTextarea");
```

```
this.currentP.setForma(FormaDaPerguntaManager.getForma("TEXTAREA"));
        return currentI;
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/IPerguntaExtractor.java

```
package br.ufsc.tcc.extractor.extractor;

import java.util.List;
import br.ufsc.tcc.common.model.Cluster;
import br.ufsc.tcc.common.model.MyNode;

public interface IPerguntaExtractor {
    public int extract(Cluster desc, List<MyNode> nodes, int
currentI);
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/extractor/ PerguntaExtractorFactory.java

```
package br.ufsc.tcc.extractor.extractor;

import br.ufsc.tcc.extractor.builder.RulesChecker;
import br.ufsc.tcc.extractor.extractor.impl.ChoiceInputExtractor;
import
br.ufsc.tcc.extractor.extractor.impl.ChoiceInputWithHeaderExtractor;
import br.ufsc.tcc.extractor.extractor.impl.MultiCompExtractor;
import br.ufsc.tcc.extractor.extractor.impl.RatingExtractor;
import br.ufsc.tcc.extractor.extractor.impl.SelectExtractor;
import br.ufsc.tcc.extractor.extractor.impl.SimpleMatrixExtractor;
import br.ufsc.tcc.extractor.extractor.impl.SingleInputExtractor;
import br.ufsc.tcc.extractor.extractor.impl.TextAreaExtractor;
import br.ufsc.tcc.extractor.model.Pergunta;
import br.ufsc.tcc.extractor.model.Questionario;

public abstract class PerguntaExtractorFactory {

    public static IPerguntaExtractor getExtractor(String extractor,
Questionario currentQ,
            Pergunta currentP, RulesChecker checker) {
        switch(extractor) {
        case "SELECT":
            return new SelectExtractor(currentQ, currentP, checker);
        case "CHOICE_INPUT_WITH_HEADER":
            return new ChoiceInputWithHeaderExtractor(currentQ,
currentP, checker);
        case "CHECKBOX_INPUT":
        case "RADIO_INPUT":
            return new ChoiceInputExtractor(currentQ, currentP,
checker);
        case "TEXT_INPUT":
        case "NUMBER_INPUT":
        case "EMAIL_INPUT":
        case "DATE_INPUT":
        case "TEL_INPUT":
        case "TIME_INPUT":
        case "URL_INPUT":
            return new SingleInputExtractor(currentQ, currentP,
checker);
```

```
        case "TEXTAREA":
            return new TextAreaExtractor(currentQ, currentP,
checker);
        case "RANGE_INPUT":
            return null;
        case "SIMPLE_MATRIX":
            return new SimpleMatrixExtractor(currentQ, currentP,
checker);
        case "MULTI_COMP":
            return new MultiCompExtractor(currentQ, currentP,
checker);
        case "RATING":
            return new RatingExtractor(currentQ, currentP, checker);
        default:
            return null;
        }
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/main/Main.java

```
package br.ufsc.tcc.extractor.main;

import org.json.JSONArray;
import org.json.JSONObject;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import br.ufsc.tcc.common.crawler.CrawlerController;
import br.ufsc.tcc.common.database.connection.BasicConnection;
import
br.ufsc.tcc.common.database.manager.PossivelQuestionarioManager;
import br.ufsc.tcc.common.util.CommonConfiguration;
import br.ufsc.tcc.common.util.CommonLogger;
import br.ufsc.tcc.common.util.CommonUtil;
import br.ufsc.tcc.extractor.builder.QuestionarioBuilder;
import br.ufsc.tcc.extractor.crawler.Crawler;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;
import br.ufsc.tcc.extractor.database.manager.QuestionarioManager;
import br.ufsc.tcc.extractor.util.Configuration;

public class Main {

    public static void main(String[] args) {
        // Carrega as configurações do projeto
        System.out.println("Carregando arquivo de configuracao...");
        CommonConfiguration.setInstance(new Configuration());

        // Inicializa a aplicação
        start();
        // simpleTest();
    }

    private static void start(){
        try{
            JSONObject configs =
CommonConfiguration.getInstance().getCrawlerConfigs();
            // Para o extrator o maxDepth tem que ser igual a 0 pois
a
            // biblioteca de Crawler é usada apenas para percorrer
as seeds
```

```
            // passadas e não para explorar ainda mais as mesmas
            configs.put("maxDepthOfCrawling", 0);

            final CrawlerController controller = new
CrawlerController(configs);

            // Seeds do banco de dados
            System.out.println("Carregando seeds do banco de
dados...");

if(CommonConfiguration.getInstance().loadSeedsFromCrawler()){
                PossivelQuestionarioManager.loadLinksAsASet();

                for(String seed :
PossivelQuestionarioManager.getLinksAsASet()){
                    controller.addSeed(seed);
                }
            }

            // Seeds do arquivo de configuração
            System.out.println("Carregando seeds do arquivo de
configuracao...");
            JSONArray seeds =
CommonConfiguration.getInstance().getSeeds();
            if(seeds != null){
                seeds.forEach((seed) ->
controller.addSeed((String)seed));
            }

            //Limpando banco de dados
            System.out.println("Limpando banco de dados do
extrator...");
            BasicConnection conn = new
BasicConnection(CommonConfiguration.getInstance().getExtractorDataba
seConfigs());
            QuestionarioManager qManager = new
QuestionarioManager(conn);

if(CommonConfiguration.getInstance().loadSeedsFromCrawler())
                qManager.cleanDatabase();
            else
                qManager.deleteLinks(seeds);
            conn.close();
            qManager = null;

            // Inicia o crawling
            System.out.println("Inicializando a aplicacao...");
            controller.start(Crawler.class);
            System.out.println("Encerrando a aplicacao...");
        }catch(Exception e){
            CommonLogger.error(e);
        }
    }

    @SuppressWarnings("unused")
    private static void simpleTest(){
        String path = "";

        BasicConnection conn = new
BasicConnection(CommonConfiguration.getInstance().getExtractorDataba
seConfigs());;
```

```
        FormaDaPerguntaManager.loadFormas(conn);
        QuestionarioBuilder qBuilder = new QuestionarioBuilder();

        Document doc = null;
        try{
            if(path.startsWith("cache/")){
                String html = CommonUtil.readResource(path);
                doc = Jsoup.parse(html);
            }else{
//              System.setProperty("javax.net.debug", "all");
                System.setProperty("https.protocols",
"TLSv1,TLSv1.1,TLSv1.2");

                path = path.replaceAll("%20", " ");
                doc = Jsoup.connect(path)
                    .validateTLSCertificates(false)
                    .get();
            }
            Element root = doc.select("body").get(0);
            qBuilder.setCurrentLink(path);
            qBuilder.build(root, doc.title());
        }catch(Exception e){
            CommonLogger.error(e);
        }finally{
            conn.close();
        }
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/model/Alternativa.java

```
package br.ufsc.tcc.extractor.model;

public class Alternativa {
    // idAlternativa
    private long id;
    // DESCRICAO
    private String descricao;
    // Pergunta a qual esta alternativa faz parte
    private Pergunta pergunta;

    // Construtores
    public Alternativa(){
        this("");
    }

    public Alternativa(String descricao){
        this.id = -1;
        this.descricao = descricao;
        this.pergunta = null;
    }

    // Getters e Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getDescricao() {
```

```
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public Pergunta getPergunta() {
        return pergunta;
    }

    public void setPergunta(Pergunta pergunta) {
        this.pergunta = pergunta;
    }

    // Demais métodos
    /**
     * Cria um clone simplificado deste objeto. Apenas copia o
{@link Alternativa#id}
     * e {@link Alternativa#descricao} sem se preocupar com a
Pergunta e Figura que
     * esta Alternativa possa fazer parte/possuir.
     *
     * @return      Um clone simplificado desta alternativa.
     */
    public Alternativa clone() {
        Alternativa c = new Alternativa();
        c.setDescricao(this.descricao);
        c.setId(this.id);
        return c;
    }

    @Override
    public String toString() {
        return this.getDescricao();
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/model/Figura.java

```
package br.ufsc.tcc.extractor.model;

public class Figura {
    // idFigura
    private long id;
    // imagem
    private String image_url;
    // Legenda
    private String legenda;
    ////////////////// Dono da figura \\\\\\\\\\\\\\\\\
    // Irá guardar uma Pergunta, P, uma Alternativa, A, ou
    // um Questionario, Q, que será usado para setar os campos
    // dono e idDono do banco de dados.
    private Object dono;

    // Construtores
    public Figura(){
        this("", "");
    }

    public Figura(String image_url, String legenda){
        this.id = -1;
```

```
        this.image_url = image_url.replaceAll("(\r)?\n", "");
        this.legenda = legenda;
        this.dono = null;
    }

    // Getters e Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getLegenda() {
        return legenda;
    }

    public void setLegenda(String legenda) {
        this.legenda = legenda;
    }

    public String getImage_url() {
        return image_url;
    }

    public void setImage_url(String image_url) {
        this.image_url = image_url;
    }

    public Object getDono() {
        return dono;
    }

    /**
     * Seta o dono da Figura. </br>
     * Este dono pode ser uma Pergunta, uma Alternativa ou um
Questionário.
     *
     * @param dono  O dono desta Figura.
     */
    public void setDono(Object dono) {
        if(dono instanceof Questionario ||
                dono instanceof Pergunta ||
                dono instanceof Alternativa){
            this.dono = dono;
        }else
            this.dono = null;
    }

    // Demais métodos
    @Override
    public String toString() {
        return this.getLegenda() +" - "+ this.getImage_url();
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
```

```
            return false;
        Figura other = (Figura) obj;
        return this.toString().equals(other.toString());
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/model/FormaDaPergunta.java

```
package br.ufsc.tcc.extractor.model;

public class FormaDaPergunta {
    // idFormaDaPergunta
    private long id;
    // descricao
    private String descricao;

    // Construtores
    public FormaDaPergunta(){
        this(-1, "");
    }

    public FormaDaPergunta(long id, String descricao){
        this.id = id;
        this.descricao = descricao;
    }

    // Getters e Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    // Demais métodos
    @Override
    public String toString() {
        return this.getDescricao();
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/model/Grupo.java

```
package br.ufsc.tcc.extractor.model;

public class Grupo {
    // idGrupo
    private long id;
    // ASSUNTO
    private String assunto;
    // Questionario ao qual este grupo faz parte
    private Questionario questionario;
```

```
    // Construtores
    public Grupo(){
        this("");
    }

    public Grupo(String assunto){
        this.id = -1;
        this.assunto = assunto;
        this.questionario = null;
    }

    // Getters e Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getAssunto() {
        return assunto;
    }

    public void setAssunto(String assunto) {
        this.assunto = assunto;
    }

    public Questionario getQuestionario() {
        return questionario;
    }

    public void setQuestionario(Questionario questionario) {
        this.questionario = questionario;
    }

    @Override
    public String toString() {
        return this.getAssunto();
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/model/Pergunta.java

```
package br.ufsc.tcc.extractor.model;

import java.util.ArrayList;
import br.ufsc.tcc.common.util.CommonLogger;
import
br.ufsc.tcc.extractor.database.manager.FormaDaPerguntaManager;

public class Pergunta {
    // idPergunta
    private long id;
    // DESCRICAO
    private String descricao;
    // TipoPergunta
    //  ABERTO, FECHADO ou MULTIPLA_ESCOLHA
    private String tipo;
    // Forma desta pergunta
    private FormaDaPergunta forma;
```

```
    // Pergunta pai, se tiver uma
    private Pergunta pai;
    // Questionario ao qual esta pergunta faz parte
    private Questionario questionario;
    // Grupo que esta pergunta faz parte, se tiver um
    private Grupo grupo;
    // Se esta pergunta for uma pergunta pai,
    // ela terá uma ou mais perguntas filhas
    private ArrayList<Pergunta> filhas;
    // Lista de alternativas desta pergunta
    private ArrayList<Alternativa> alternativas;

    // Construtores
    public Pergunta(){
        this("", null);
    }

    public Pergunta(String descricao){
        this(descricao, null);
    }

    public Pergunta(String descricao, FormaDaPergunta forma){
        this.id = -1;
        this.descricao = descricao;
        this.setForma(forma);
        this.pai = null;
        this.questionario = null;
        this.grupo = null;
        this.filhas = new ArrayList<>();
        this.alternativas = new ArrayList<>();
    }

    // Getters e Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        tipo = tipo.toUpperCase();
        if(tipo.matches("(ABERTO|FECHADO|MULTIPLA_ESCOLHA)"))
            this.tipo = tipo;
        else
            CommonLogger.info("Pergunta:setTipo()> Tipo nao
permitido ({}).", tipo);
    }
```

```
    public FormaDaPergunta getForma() {
        return forma;
    }

    /**
     * Seta a forma desta pergunta.<br>
     * <b>Side effect</b>: seta o tipo desta pergunta utilizando o
método {@link #convertFormaToTipo()}.
     *
     * @param forma
     */
    public void setForma(FormaDaPergunta forma) {
        this.forma = forma;
        this.convertFormaToTipo();
    }

    public Pergunta getPai() {
        return pai;
    }

    public void setPai(Pergunta pai) {
        this.pai = pai;
    }

    /**
     * Retorna o questionário que esta pergunta faz parte.<br>
     * Caso esta pergunta seja filha de outra, é retornado o
questionário ao
     * qual o pai desta pergunta faz parte.
     *
     * @return          Questionário que esta pergunta faz parte.
     */
    public Questionario getQuestionario() {
        if(this.pai != null)
            return this.pai.getQuestionario();
        return questionario;
    }

    public void setQuestionario(Questionario questionario) {
        this.questionario = questionario;
    }

    /**
     * Retorna o grupo que esta pergunta faz parte.<br>
     * Caso esta pergunta seja filha de outra, é retornado o grupo
ao
     * qual o pai desta pergunta faz parte.
     *
     * @return          Grupo que esta pergunta faz parte.
     */
    public Grupo getGrupo() {
        if(this.grupo == null && this.pai != null)
            return this.getPai().getGrupo();
        return grupo;
    }

    public void setGrupo(Grupo grupo) {
        this.grupo = grupo;
    }
```

```
    public ArrayList<Pergunta> getFilhas() {
        return filhas;
    }

    public void setFilhas(ArrayList<Pergunta> filhas) {
        this.filhas = filhas;
    }

    /**
     * Adiciona a pergunta {@code p} a lista de perguntas filhas
desta
     * pergunta.<br>
     * <b>Side effect</b>: Seta também a pergunta pai da pergunta
{@code p} para
     * apontar para esta pergunta.
     *
     * @param p
     */
    public void addFilha(Pergunta p){
        p.setPai(this);
        this.filhas.add(p);
    }

    public ArrayList<Alternativa> getAlternativas() {
        return alternativas;
    }

    public void setAlternativas(ArrayList<Alternativa> alternativas)
{
        this.alternativas = alternativas;
    }

    /**
     * Adiciona a alternativa {@code a} a lista de alternativas
desta
     * pergunta.<br>
     * <b>Side effect</b>: Seta também a pergunta da alternativa
{@code a} para
     * apontar para esta pergunta.
     *
     * @param a
     */
    public void addAlternativa(Alternativa a){
        a.setPergunta(this);
        this.alternativas.add(a);
    }

    // Demais métodos
    /**
     * Verifica se esta pergunta é da {@code forma} passada.
     *
     * @param forma
     * @return              <b>TRUE</b> caso esta pergunta seja da
{@code forma}
     *                      passada ou, <br>
     *                      <b>FALSE</b> caso contrario.
     */
    public boolean isA(String forma){
        return this.getForma() ==

FormaDaPerguntaManager.getForma(forma.toUpperCase());
```

```
    }

    /**
     * Seta o tipo desta pergunta usando como base a forma dela.
     */
    public void convertFormaToTipo(){
        if(this.getForma() == null) {
            this.tipo = "";
            return;
        }

        switch(this.getForma().toString()){
        case "SELECT":
        case "SELECT_GROUP":
        case "IMAGE_RADIO_INPUT":
        case "IMAGE_RADIO_INPUT_GROUP":
        case "IMAGE_RADIO_INPUT_MATRIX":
        case "RADIO_INPUT":
        case "RADIO_INPUT_GROUP":
        case "RADIO_INPUT_MATRIX":
        case "RANGE_INPUT":
        case "RANGE_INPUT_GROUP":
        case "RATING":
        case "RATING_GROUP":
            this.tipo = "FECHADO";
            break;
        case "IMAGE_CHECKBOX_INPUT":
        case "IMAGE_CHECKBOX_INPUT_GROUP":
        case "IMAGE_CHECKBOX_INPUT_MATRIX":
        case "CHECKBOX_INPUT":
        case "CHECKBOX_INPUT_GROUP":
        case "CHECKBOX_INPUT_MATRIX":
            this.tipo = "MULTIPLA_ESCOLHA";
            break;
        case "TEXT_INPUT":
        case "TEXT_INPUT_GROUP":
        case "TEXT_INPUT_MATRIX":
        case "NUMBER_INPUT":
        case "NUMBER_INPUT_GROUP":
        case "NUMBER_INPUT_MATRIX":
        case "EMAIL_INPUT":
        case "EMAIL_INPUT_GROUP":
        case "EMAIL_INPUT_MATRIX":
        case "DATE_INPUT":
        case "DATE_INPUT_GROUP":
        case "DATE_INPUT_MATRIX":
        case "TEL_INPUT":
        case "TEL_INPUT_GROUP":
        case "TEL_INPUT_MATRIX":
        case "TIME_INPUT":
        case "TIME_INPUT_GROUP":
        case "TIME_INPUT_MATRIX":
        case "URL_INPUT":
        case "URL_INPUT_GROUP":
        case "URL_INPUT_MATRIX":
        case "TEXTAREA":
        case "TEXTAREA_GROUP":
        case "TEXTAREA_MATRIX":
            this.tipo = "ABERTO";
            break;
        case "MIX_COMP_GROUP":
```

```
        case "MIX_COMP_MATRIX":
        case "MULTI_COMP":
        case "MULTI_COMP_GROUP":{
            String tipo = "", tmp = "";
            for(Pergunta p : this.filhas) {
                tmp = p.getTipo();
                if(tmp.equals("ABERTO"))
                    tipo = "ABERTO";
                else if(tmp.equals("MULTIPLA_ESCOLHA")
&& !tipo.equals("ABERTO"))
                    tipo = "MULTIPLA_ESCOLHA";
                else if(tipo.matches("FECHADO|"))
                    tipo = tmp;
            }
            this.tipo = tipo;
            break;
        }default:
            this.tipo = "";
        }
    }

    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        String tmpTxt = "";
        builder.append(this.getDescricao() +" ["+ this.getForma() +
                " / "+ this.getTipo()+"]\n");
        builder.append("\tGrupo: " +this.getGrupo()+ "\n");
        builder.append("\tAlternativas [" +this.alternativas.size()+
"]:\n");
        for(Alternativa a : this.getAlternativas()){
            tmpTxt = a.toString().replaceAll("\n", "\n\t\t   ");
            builder.append("\t\t" +tmpTxt+ "\n");
        }
        builder.append("\tFilhas [" +this.filhas.size()+ "]:\n");
        for(Pergunta filha : this.getFilhas()){
            tmpTxt = filha.getDescricao().replaceAll("\n", "\n\t\t
");
            builder.append("\t\t" +tmpTxt+" [" +filha.getForma()+
"]\n");
            for(Alternativa a : filha.getAlternativas()){
                tmpTxt = a.toString().replaceAll("\n", "\n\t\t\t
");
                builder.append("\t\t\t"+ tmpTxt +"\n");
            }
        }
        builder.append("\n");
        return builder.toString();
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/model/Questionario.java

```
package br.ufsc.tcc.extractor.model;

import java.util.ArrayList;
import br.ufsc.tcc.common.model.MyNode;

public class Questionario {
    // idQuestionario
    private long id;
    // ASSUNTO
```

```
    private String assunto;
    // LINK_DOCUMENTO
    private String link_doc;
    // Lista de perguntas do questionario
    private ArrayList<Pergunta> perguntas;
    // Lista de grupos do questionario
    private ArrayList<Grupo> grupos;
    // Lista de figuras encontradas no questionario
    private ArrayList<Figura> figuras;

    // Construtores
    public Questionario(){
        this("");
    }

    public Questionario(String link_doc){
        this.id = -1;
        this.assunto = "";
        this.link_doc = link_doc;

        this.perguntas = new ArrayList<>();
        this.grupos = new ArrayList<>();
        this.figuras = new ArrayList<>();
    }

    // Getters e Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getAssunto() {
        return assunto;
    }

    public void setAssunto(String assunto) {
        this.assunto = assunto;
    }

    public String getLink_doc() {
        return link_doc;
    }

    public void setLink_doc(String link_doc) {
        this.link_doc = link_doc;
    }

    public ArrayList<Pergunta> getPerguntas() {
        return perguntas;
    }

    public void setPerguntas(ArrayList<Pergunta> perguntas) {
        this.perguntas = perguntas;
    }

    /**
     * Adiciona a pergunta {@code p} a lista de perguntas deste
     * questionário.<br>
```

```
     * <b>Side effect</b>: Seta também o questionário da pergunta
{@code p} para
     * apontar para este questionário.
     *
     * @param p
     */
    public void addPergunta(Pergunta p){
        p.setQuestionario(this);
        this.perguntas.add(p);
    }

    public ArrayList<Grupo> getGrupos() {
        return grupos;
    }

    public void setGrupos(ArrayList<Grupo> grupos) {
        this.grupos = grupos;
    }

    /**
     * Adiciona o grupo {@code g} a lista de grupos deste
     * questionário.<br>
     * <b>Side effect</b>: Seta também o questionário do grupo
{@code g} para
     * apontar para este questionário.
     *
     * @param p
     */
    public void addGrupo(Grupo g){
        g.setQuestionario(this);
        this.grupos.add(g);
    }

    public ArrayList<Figura> getFiguras() {
        return figuras;
    }

    public void setFiguras(ArrayList<Figura> figuras) {
        this.figuras = figuras;
    }

    public void addFigura(Figura f){
        this.figuras.add(f);
    }

    // Demais métodos
    public boolean hasFigura(Figura fig) {
        return this.figuras.contains(fig);
    }

    public boolean hasFigura(MyNode fig) {
        if(!fig.isImage()) return false;
        String src = fig.getAttr("src").replaceAll("(\r)?\n", ""),
                alt = fig.getAttr("alt");
        for(Figura f : this.figuras) {
            if(f.getImage_url().equals(src) &&
                    f.getLegenda().equals(alt))
                return true;
        }
        return false;
    }
```

```
    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        builder.append("Questionario: " +this.getAssunto()+ "\n");
        builder.append("\tLink: " +this.getLink_doc() + "\n");
        builder.append("\tFiguras:\n");
        for(Figura fig : this.figuras){
            builder.append("\t\t" +fig+ "\n");
        }
        builder.append("\tPerguntas:\n");
        for(int i = 0; i<this.perguntas.size(); i++){
            Pergunta p = this.perguntas.get(i);
            builder.append("<<"+(i+1)+">> " +p);
        }
        return builder.toString();
    }
}
```

questionnaires_extractor/br/ufsc/tcc/extractor/util/Configuration.java

```
package br.ufsc.tcc.extractor.util;

import org.json.JSONException;
import org.json.JSONObject;
import br.ufsc.tcc.common.util.CommonConfiguration;

public class Configuration extends CommonConfiguration {

    // Construtores
    public Configuration() {
        configsPath = "./extractor_configs.json";
        this.loadConfigs();
        this.validateConfigs();
    }

    // Demais métodos
    @Override
    protected void validateParameters() throws JSONException {
        JSONObject p = configs.getJSONObject("parameters");
        validatingPath = "parameters";

        p.getInt("minQuestionsOnQuestionnaire");
        p.getInt("maxWordsInAGroupDescription");
        p.getInt("maxTextClustersBetweenQuestions");
        p.getString("phrasesToIgnoreRegex");

        validateIntParameter(p, "distBetweenNearNodes",
                "height", "maxHeight", "width");

        validateIntParameter(p,
"distBetweenTextsInsideQuestionnaire",
                "height");

        validateIntParameter(p, "distBetweenCompAndText",
                "height", "maxHeight");

        validateIntParameter(p, "distBetweenDescAndQuestion",
                "height", "maxHeight");

        validateIntParameter(p, "distBetweenGroupAndFirstQuestion",
                "height", "width");
```

```
        validateIntParameter(p,
"distBetweenDescAndComplementaryText",
                "height", "maxHeight", "width");

        validateIntParameter(p,
"distBetweenTextsInQuestionWithSubQuestions",
                "height", "width");

        validateIntParameter(p, "distBetweenPartsOfDescription",
                "height", "maxHeight", "width");

        validateIntParameter(p, "distBetweenTextsOfSameAlternative",
                "height", "maxHeight", "width");

        validateIntParameter(p,
"distBetweenHeaderAndFirstAlternative",
                "height", "width");

        validateIntParameter(p,
"distBetweenEvaluationLevelsAndDesc",
                "height");

        p.getInt("maxSpacesAndNewLinesInEvaluationLevels");
        p.getString("evaluationLevelsWordsRegex");
    }

    @Override
    protected void validateDatabaseConfig() {
        JSONObject dbObj = configs.getJSONObject("database");
        validatingPath = "database";

        dbObj.getBoolean("loadSeedsFromCrawler");

        JSONObject tmp = dbObj.getJSONObject("crawler");
        validatingPath += ".crawler";

        tmp.getString("dbms");
        tmp.getString("name");
        tmp.getString("host");
        tmp.getString("login");
        tmp.getString("password");

        tmp = dbObj.getJSONObject("extractor");
        validatingPath = validatingPath.substring(0,
validatingPath.indexOf(".")+1) + "extractor";

        tmp.getString("dbms");
        tmp.getString("name");
        tmp.getString("host");
        tmp.getString("login");
        tmp.getString("password");
    }
}
```