

**UNIVERSIDADE FEDERAL DE SANTA CATARINA**

**FERRAMENTA PARA EXTRAÇÃO DE WEB FORMS**

**LEONARDO BRES DOS SANTOS**

**Florianópolis - SC**

**2012/2**

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

**FERRAMENTA PARA EXTRAÇÃO DE WEB FORMS**

**LEONARDO BRES DOS SANTOS**

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

**Florianópolis - SC**

**2012/2**

**LEONARDO BRES DOS SANTOS**

**FERRAMENTA PARA EXTRAÇÃO DE WEB FORMS**

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

Orientador(a): Carina Friedrich Dorneles

Banca examinadora

Luciana de O. Rech

Ronaldo dos Santos Mello

*"As oportunidades multiplicam-se à medida que são agarradas."  
(Sun Tzu)*

# SUMÁRIO

<b>LISTA DE FIGURAS</b>	<b>7</b>
<b>LISTA DE TABELAS</b>	<b>8</b>
<b>LISTA DE EQUAÇÕES</b>	<b>9</b>
<b>RESUMO</b>	<b>10</b>
<b>1 INTRODUÇÃO</b>	<b>12</b>
<b>2 DEEP WEB</b>	<b>17</b>
2.1 Visão geral	17
2.2 Coleta de dados	18
2.3 Coleta de formulários web	19
<b>3 TRABALHOS RELACIONADOS</b>	<b>22</b>
<b>3.1 DEEPBOT</b>	<b>22</b>
3.1.1 ARQUITETURA	23
3.1.2 DEFINIÇÕES DE DOMÍNIOS	23
3.1.3 TESTES	25
<b>3.2 LABELLEX</b>	<b>26</b>
3.2.1 DEFINIÇÃO DO PROBLEMA E SOLUÇÃO	26
3.2.2 GERANDO MAPEAMENTOS CANDIDATOS	27
3.2.3 EXTRAINDO CARACTERÍSTICAS	28
3.2.4 APRENDENDO A IDENTIFICAR MAPEAMENTOS	28
3.2.5 TESTES	29
<b>3.3 EXTRAINDO DADOS POR TRÁS DE FORMULÁRIOS WEB</b>	<b>30</b>
3.3.1 PLANOS DE SUBMISSÃO DOS FORMULÁRIOS	31
3.3.2 TESTES	32
<b>4 IFRIT</b>	<b>34</b>
<b>4.1 VISÃO GERAL</b>	<b>34</b>
<b>4.2 DETECÇÃO DOS FORMULÁRIOS</b>	<b>35</b>
4.2.1 ALGORITMO	37
<b>4.3 EXTRAÇÃO DOS RÓTULOS</b>	<b>41</b>
4.3.1 ALGORITMO	41

<b>4.4</b>	<b>IMPLEMENTAÇÃO</b>	<b>46</b>
4.4.1	A APLICAÇÃO	46
4.4.2	API	48
4.4.3	WEB CRAWLER	49
<b>4.5</b>	<b>EXPERIMENTOS</b>	<b>53</b>
4.5.1	VARIÁVEIS	53
4.5.2	MÉTRICAS	54
4.5.3	RESULTADOS	55
4.5.4	DISCUSSÃO	58
4.5.5	COMPARAÇÃO	60
<b>5</b>	<b>CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS</b>	<b>63</b>
<b>6</b>	<b>REFERÊNCIA BIBLIOGRÁFICA</b>	<b>66</b>

## LISTA DE FIGURAS

Figura 1- Exemplo de formulário.....	13
Figura 2 - Código HTML da criação de um formulário com a utilização do elemento <form>.....	13
Figura 3 - Código HTML da criação de um formulário com a utilização de JavaScript.....	14
Figura 4 - Analogia entre a Deep Web e um iceberg (Megan). .....	17
Figura 5 - Fluxo de passos executados por um crawler(Liu, 2011, p. 313).....	21
Figura 6 - Arquitetura do crawler Ifrit.....	35
Figura 7 - Exemplo de formulário e uso da numeração DEWEY.....	39
Figura 8 - Pseudo código ilustrando a navegação na árvore de elementos .....	40
Figura 9 - Exemplo de extração e associação de rótulos .....	42
Figura 10 - Exemplo da extração de componentes de rótulos.....	43
Figura 11 - Exemplo de agregação de rótulos em sequência.....	45
Figura 12 - Diagrama de classes da API do Ifrit.....	46
Figura 13 - Diagrama de classes do Web Crawler .....	47
Figura 14 - Diagrama de classes mostrando a conexão entre API e Web Crawler .....	47
Figura 15 - Resultados das métricas por campo do formulário .....	55
Figura 16 - Resultados das métricas por associação entre rótulo e campo.....	56
Figura 17 - Resultados gerais por métrica - Campo/Formulario .....	57
Figura 18 - Resultados gerais por métrica - Rotulo/Campo.....	58

## *LISTA DE TABELAS*

Tabela 1 - Matriz do cálculo de distância entre componentes de rótulos .....	43
Tabela 2 - Problemas.....	58
Tabela 3 - Falhas.....	59
Tabela 4 - Comparação entre o Ifrit e outros web crawlers .....	60



## LISTA DE EQUAÇÕES

Equação 1 - Fórmula para o cálculo de distância entre os elementos.....	44
Equação 2 - Fórmula para cálculo de precisão por campo do formulário .....	54
Equação 3 - Fórmula para cálculo de revocação por campo do formulário .....	54
Equação 4 - Fórmula para cálculo de medida-f por campo do formulário.....	54
Equação 5 - Fórmula para cálculo de precisão por associação entre rótulo e campo do formulário.....	55
Equação 6 - Fórmula para cálculo de revocação por associação entre rótulo e campo do formulário.....	55
Equação 7 - Fórmula para cálculo de medida-f por associação entre rótulo e campo do formulário.....	55

## LISTA DE REDUÇÕES

API - Application Programming Interface

IDE - Integrated Development Environment

HTML - *HyperText Markup Language*

HTTP - Hypertext Transfer Protocol

SQL - Structured Query Language

## RESUMO

Como a Internet cresce muito rápido em termos de número de endereços eletrônicos disponíveis na rede, existe um imenso trabalho por parte das empresas que oferecem o serviço de busca para tornar possível a captura de todos os endereços disponíveis. Este trabalho aborda a questão do reconhecimento de formulários em páginas web, seus campos e respectivos rótulos. Este tipo de operação é fundamental quando se quer identificar a área de serviço de uma determinada página, ou seja, saber se a página trata de aluguel de carros ou venda de imóveis ou venda de passagens aéreas, etc.

Para resolver o problema proposto - identificação de formulários e seus rótulos em páginas web - foi desenvolvido um *Web Crawler* batizado de Ifrit, que é capaz de navegar de uma página para outra a procura de formulários web. Existem dois grandes problemas tratados neste trabalho: (i) o reconhecimento de formulários construídos sem o uso da *tag* FORM do HTML e (ii) a extração dos rótulos associados aos elementos do formulário.

**Palavras-chave:** *Web Crawler* focado; *Web Form*; Motor de Busca.

# 1 INTRODUÇÃO

Mesmo sem conhecer todos os endereços das páginas existentes na Internet, é possível, através de serviços de busca, encontrar a informação que se está procurando. Para tornar isto possível, muito esforço foi empenhado na criação de motores de busca extremamente rápidos e amplos, que conseguem encontrar praticamente qualquer tipo de informação que esteja na Internet. Então, torna-se possível, através de um site de busca, como o Google.com por exemplo, encontrar informações sobre um determinado assunto sem conhecer o endereço eletrônico que contém a informação desejada.

Para manter os arquivos de indexação de páginas de um motor de busca atualizados, usa-se um *WEB Crawler*, uma espécie de robô, que varre a Internet indo de uma página para outra, armazenando os endereços visitados (Liu, 2011, p. 311). Os motores de busca mais comuns como o Google, buscam manter, de uma forma geral, uma base de dados com o maior número de informação possível sobre as páginas contidas na Internet. Além de saber qual é o endereço de uma página, um motor de busca tem interesse em saber também o conteúdo da mesma, para que em uma busca, seja possível informar quais informações são fornecidas por determinada página. Desta forma, neste trabalho é abordado a extração de informações referentes a formulários web.

O problema tratado aqui é a diversidade de maneiras diferentes de se criar uma página na *web* que contenha um formulário. A maneira mais comum de se fazer isso usando a linguagem HTML, é colocar as informações sobre os campos do formulário dentro de um elemento do tipo FORM. Sendo assim é relativamente simples pensar em um algoritmo que consiga identificar se uma página da *web* contém ou não um formulário, pois basta verificar a existência de elemento HTML do tipo FORM. No entanto, esta solução não abrange todos os casos, então, faz-se necessário a criação de um mecanismo que consiga identificar os formulários que não estejam relacionadas com o elemento HTML do tipo FORM. Além da disposição dos componentes, os rótulos também podem estar em muitas posições diferentes em relação ao elemento ao qual ele está vinculado, e isto é um dos maiores problemas a tratados pelo Ifrit.

**Figura 1 - Exemplo de formulário**

Afim de contextualizar o leitor sobre a criação de formulários *web* e para mostrar as características dos formulários que são extraídos pelo Ifrit, é abordado agora duas maneiras distintas de criação de formulários, uma com o uso da *tag* FORM e outra com o uso de Javascript. A Figura 1 representa um pequeno formulário contendo três campos e um botão responsável por enviar as informações para o servidor. Os exemplos a seguir implementam este mesmo formulário e mostram as diferenças entre as duas abordagens.

```

1  <form action="/exemplo/cadastro">
2      <div>
3          <div>
4              <input type="checkbox" value="Componente 1" name="componente_1">
5          </div>
6          <div>
7              <div>
8                  <span>Componente 2</span> <br> <select name="componente_2">
9                      <option>Opção 1</option>
10                     <option>Opção 2</option>
11                     <option>Opção 3</option>
12                 </select>
13             </div>
14             <span>Componente 3</span> <br> <input type="text"
15                 name="componente_3">
16         </div>
17         <input type="submit">
18     </div>
19 </form>

```

**Figura 2 - Código HTML da criação de um formulário com a utilização do elemento <form>.**

Na Figura 2, onde foi utilizado o elemento FORM do HTML, está representado uma maneira simples de se implementar o comportamento do formulário. A simplicidade aqui está no fato de o *browser* se tornar responsável por enviar os valores dos campos do formulário para o servidor, isentando o

desenvolvedor de fazer isto. Os pontos importantes aqui são: (i) linha 1, elemento FORM com o atributo *action* que representa a URL do servidor; (ii) atributo *name* dos componentes, que representam a chave para recuperar o valor do componente quando a requisição chegar ao servidor e (iii) linha 17, elemento INPUT do tipo *submit* representando um botão que ao ser clicado envia o formulário para o servidor.

```
24 <div>
25   <div>
26     <input id="componente_1" type="checkbox" value="Componente 1"
27       name="componente_1">
28   </div>
29   <div>
30     <div>
31       <span>Componente 2</span> <br> <select id="componente_2"
32         name="componente_2">
33         <option>Opção 1</option>
34         <option>Opção 2</option>
35         <option>Opção 3</option>
36       </select>
37     </div>
38     <span>Componente 3</span> <br> <input id="componente_3"
39       type="text" name="componente_3">
40   </div>
41   <input type="button" onclick="cadastrar();">
42 </div>
43
44 <script type="text/javascript">
45   function cadastrar() {
46     var url = "http://" + location.host + "/exemplo/cadastro";
47     $.get(url, {
48       componente_1 : $("#componente_1").attr("checked"),
49       componente_2 : $("#componente_2").val(),
50       componente_3 : $("#componente_3").val()
51     }, function(data) {
52       alert("Cadastro efetuado com sucesso!");
53     });
54   }
55 </script>
```

**Figura 3 - Código HTML da criação de um formulário com a utilização de JavaScript.**

Uma outra maneira de se realizar a mesma atividade é apresentada na Figura 3, onde foi utilizado o JavaScript para se realizar a requisição que envia os dados do formulário para o servidor. A vantagem é que se tem maior controle sobre a requisição e não existe a necessidade de recarregar toda a página a cada requisição, diminuindo a quantidade de informação que deve circular pela rede. Os pontos importantes neste caso são: (i) ausência do elemento FORM; (ii) cada componente do formulário contém o atributo id que é utilizado para recuperar o valor preenchido no campo; (iii) linha 41, elemento INPUT do tipo *button* que representa o botão responsável por enviar as

informações do formulário para o servidor, que é feito através da função `cadastrar()` indicada no atributo `action` e (iv) linha 44, elemento `SCRIPT` contendo a função JavaScript `cadastrar`. É possível verificar que para realizar a requisição é necessário buscar individualmente os campos contidos no formulário, linhas 48-50. Para exemplificar que este tipo de requisição proporciona maior controle por parte do programador, pode-se observar a linha 51, que contém uma função que é executada quando a requisição é bem sucedida, este tipo de controle não pode ser feito quando se utiliza o elemento `FORM`.

Foram apresentados as duas principais maneiras de se criar um formulário em uma página *web*. É importante ficar claro que o `lfr` é focado em procurar por formulários construídos sem o uso da *tag* `FORM` do HTML, ou seja, o formulário mostrado na Figura 3. Também é importante dizer que este tipo de construção, com uso de Javascript, é mais dinâmico e sendo assim, mais difícil de se identificar.

Atualmente, existem *Web Crawler's* focados, por exemplo o `DeepBot` (Álvarez, Raposo, Pan, Cacheda, Bellas, & Carneiro, 2007), que fazem a extração de formulários desenvolvidos da maneira mais comum, ou seja, com a utilização do elemento `FORM` do HTML. Esses mecanismos verificam no documento HTML a existência de um elemento do tipo `FORM`, e se o campo for encontrado, então o *Crawler* armazena o endereço da página que contém o formulário.

Para que se consiga realmente extrair informações relevantes de um formulário não basta apontar a sua existência, é necessário saber que tipo de dado está sendo inserido em cada elemento. Para isto é necessário conhecer não só o componente de um formulário, mas também o rótulo que está associado a ele.

Como proposta para a resolução do problema de identificação de formulários na *web* que não estejam contidos em elementos do tipo `FORM`, é proposto um *Web Crawler* focado que implementa uma heurística que consiga, na maioria dos casos, identificar, com base nas *tag's* HTML utilizadas, se uma página da *web* contém ou não um formulário. Já para o problema do reconhecimento dos rótulos, é proposto uma segunda heurística que calcula a

distância entre os elementos e os rótulos afim de se encontrar o relacionamento entre eles.

Especificamente, os principais objetivos do trabalho são:

1. Definição de uma heurística para detecção de *Web Forms* construídos sem a *tag* FORM do HTML.
2. Definição de uma heurística para extração dos rótulos relacionados aos campos dos formulários encontrados.
3. Implementação de um *Crawler* focado e das heurísticas previamente definidas.
4. Realização de experimentos para testar a abordagem definida.
5. Criação do *script* SQL para inserir em um banco de dados já existente as informações coletadas pelo *Crawler*.

A estrutura deste trabalho está organizada como segue: o Capítulo 2 apresenta uma introdução aos conceitos de *Deep Web* e *Crawler*, o Capítulo 3 apresenta os trabalhos que estão correlacionados, o Capítulo 4 fala sobre as definições, características e todo o processo de desenvolvimento do *Crawler* focado Ifrit, alvo deste trabalho. Ainda no Capítulo 4 são comentados os resultados obtidos nos experimentos realizados para validar a abordagem. E por fim o Capítulo 0 faz o fechamento do trabalho expondo sua conclusão.



## 2 DEEP WEB

Este capítulo trata da fundamentação teórica necessária para o entendimento do restante deste trabalho. Primeiro uma visão geral sobre a *Deep Web* e suas características. Em seguida, uma explicação sobre a coleta de dados e *crawlers*. Ao final, o objetivo principal deste trabalho, a coleta de formulários *web*.

### 2.1 Visão geral

A *Deep Web* é caracterizada pelas informações contidas na *web* que não podem ser acessadas diretamente. Geralmente o acesso a estas informações é feito a partir de uma consulta em um formulário *web* (Bergman, 2001), desta forma os formulários *web* representam uma interface de pesquisa para um banco de dados na *Deep Web*. Páginas que se encaixam nesta descrição, ou seja, que contém informação "escondida" geralmente são construídas de maneira dinâmica a partir de uma consulta executada por um formulário. Isto quer dizer que a mesma página pode apresentar toda a informação contida no servidor, ou apenas uma parte, isto vai depender das informações preenchidas no formulário que originou esta página.

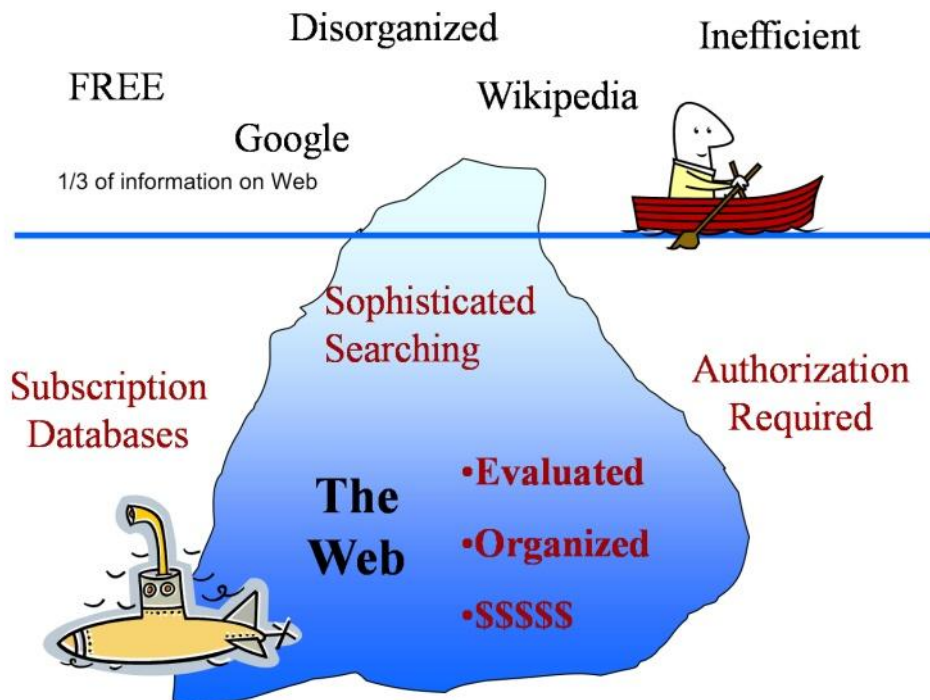


Figura 4 - Analogia entre a Deep Web e um iceberg (Megan).

A Figura 4 representa a maneira como a *Deep Web* está organizada. Para entender melhor a *Deep Web* faz-se uma analogia com um *iceberg*, onde a camada que não pode ser vista representa a *Deep Web* e a camada visível representa a *surface web* ou *web* visível. A *Deep Web* é muito mais extensa do que a camada visível da *web* e é ali onde é possível encontrar informações mais valiosas. O grande problema consiste em acessar esta camada de informação.

## **2.2 Coleta de dados**

Esta seção aborda uma das ferramentas utilizadas na busca de informação na *web*, os *Crawlers*. Além disso, esta seção fala também sobre os principais tipos de *crawlers*, suas características e a maneira como eles funcionam.

Os *Crawlers*, também conhecidos como robôs (Liu, 2011, p. 311), são programas que conseguem baixar automaticamente páginas *Web*. Como a informação na *Web* está espalhada entre bilhões de páginas armazenadas em servidores em todo mundo, os usuários que navegam na *Web* podem seguir *hyperlinks* para acessar informações se deslocando de uma página para outra. Um *crawler* pode visitar muitos sites para coletar informações que podem ser analisadas em um local central, seja on-line (enquanto o *crawler* está visitando a página) ou off-line (depois que o conteúdo da página for armazenado).

Um *crawler* comum não tem capacidade para diferenciar as páginas que visita, ou seja, toda página que o *crawler* visitar, será armazenada. Algumas vezes o que se deseja é obter informação de um grupo específico de páginas. Para isso é necessário que o *crawler* seja desenvolvido com um algoritmo que consiga classificar as páginas que ele visita, e desta forma garantir que só as páginas que interessam serão armazenadas. *Crawlers* construídos com este objetivo são chamados de *Crawlers* focados (Liu, 2011). Neste trabalho, foi desenvolvido o Ifrit, um *crawler* focado, que é capaz de buscar por formulários *web* e extrair os rótulos contidos nestes formulários.

O grande problema com a extração de dados na *Deep Web* é que os dados não podem ser mapeados para uma URL, ou seja, os dados não fazem parte do conteúdo estático de uma página. O conteúdo da *Deep Web* é visível

apenas através de consultas feitas a partir de um formulário web, que serve de interface para o acesso ao banco de dados que está escondido na *Web*.

Sendo os formulários *web* uma porta de acesso à *Deep Web*, pode-se dizer então que *crawlers* voltados para a extração de dados na *Deep Web* devem ser focados na descoberta de formulários. Existem algumas maneiras diferentes de se procurar por um formulário em um página *web*: (i) a maneira mais simples é procurar por *tags* tipo FORM do HTML; (ii) é possível também fazer uma comparação com termos relevantes em um domínio, afim de extrair apenas páginas referentes a um determinado assunto (Esta abordagem pode ser usada também em outras situações, não apenas na busca por formulários); (iii) pode-se também procurar por *templates* estruturais de formulários *web* (o caso do Ifrit) ou ainda aplicar técnicas de *machine learning* (aprendizado de máquina) para detectar formulários com características relevantes através de uma amostra de páginas.

De maneira geral, depois de superado o obstáculo que é o reconhecimento de formulários, foca-se então na extração dos dados da *Deep Web*. Como já foi comentado, um formulário serve como porta de acesso ao banco de dados "escondido" na *web*. Sendo assim, para se ter acesso a todos os dados contidos em uma página, é necessário fazer consultas ao banco de dados, utilizando as opções permitidas pelo formulário em questão. Nesta tarefa, entram muitas abordagens diferentes para se fazer o preenchimento automatizado de formulários. Mais sobre este assunto será abordado no capítulo 3-TRABALHOS RELACIONADOS.

### **2.3 Coleta de formulários web**

Esta seção apresenta um passo a passo na execução de um *crawler* que tem uma estrutura simples. A Figura 5 apresenta o fluxo de passos geralmente executados por um *crawler* deste tipo.

O elemento chamado **(4) frontier** é responsável por manter uma lista de URLs que serão usadas pelo *crawler* para acessar as páginas. No início da execução o **(4) frontier** deve receber do usuário, a lista de **(1) seed URLs**, estas são as

primeiras URLs a serem usadas pelo *crawler*, no entanto, novos valores serão adicionados ao **(4) frontier** no decorrer da execução.

Na sequência, o *crawler* deve consumir uma URL do **(4) frontier** e acessar a página *web* através de uma requisição HTTP. Neste ponto, podem existir diferentes implementações dependendo do objetivo do *crawler*. Geralmente o conteúdo da página é interpretado, os *links* contidos na página são extraídos e inseridos no **(4) frontier** e o conteúdo da página é armazenado em um repositório. Deve existir um critério de parada que termina a execução do *crawler*, ou reinicia o processo em uma nova página. *Crawlers* focados só devem armazenar o conteúdo da página se ela atender aos critérios definidos pelo usuário ou pelo desenvolvedor da aplicação.

É possível que as páginas não sejam armazenadas, todo o procedimento pode ser executado *online*. No momento em que o *crawler* consegue acesso ao conteúdo da página, é feita a análise e sendo o resultado positivo, o *crawler* armazena as informações sobre o formulário e seus componentes e segue para a próxima página da lista. Outras implementações podem executar estes procedimentos de maneira *offline*, onde os processos de visitar as páginas e tratar as informações são executados em momentos distintos. Esta abordagem permite otimizar o processamento das páginas através do paralelismo da execução.

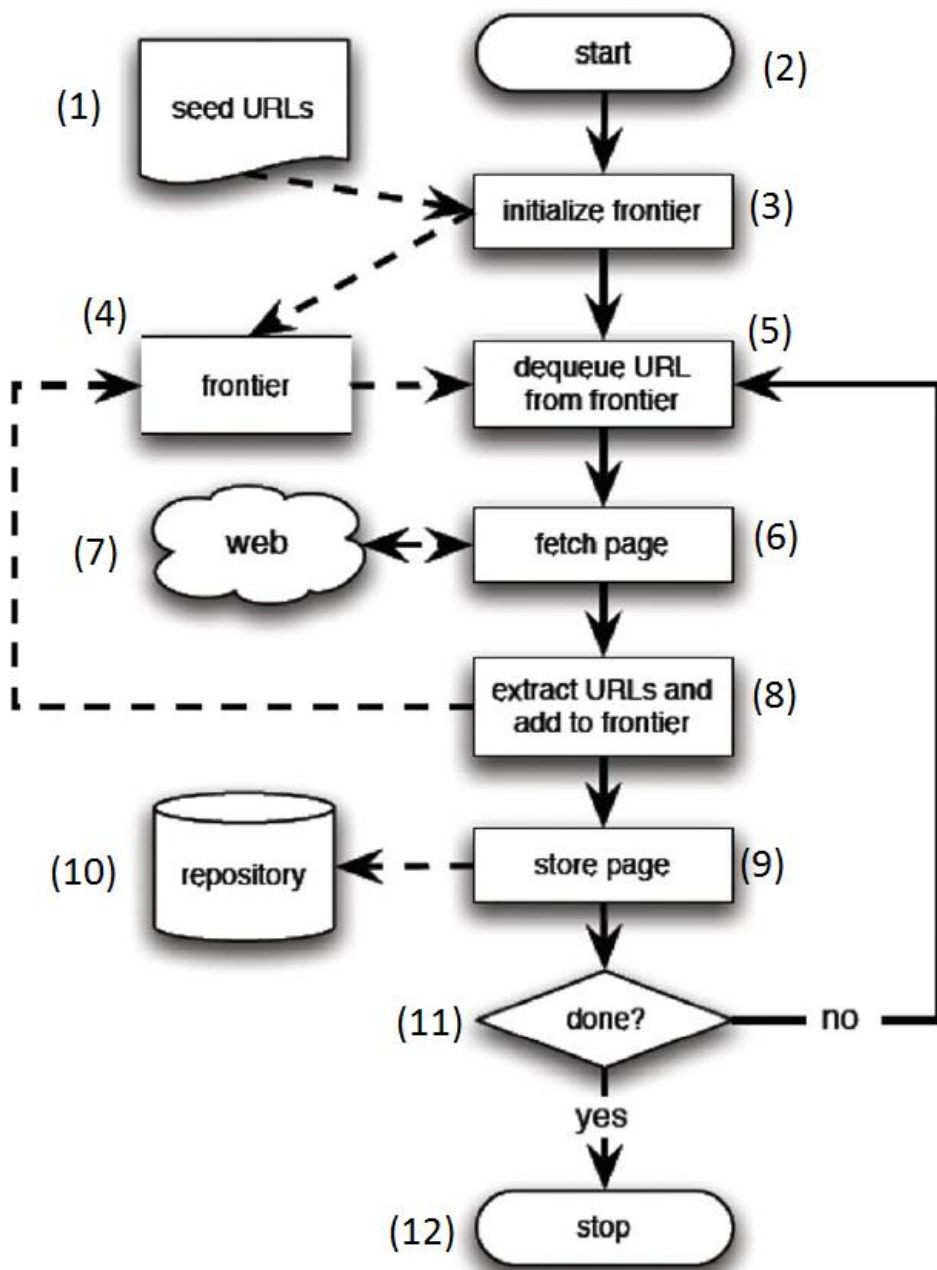


Figura 5 - Fluxo de passos executados por um crawler (Liu, 2011, p. 313)

## 3 TRABALHOS RELACIONADOS

Este capítulo faz um resumo dos principais trabalhos que estão relacionados ao desenvolvimento do *Crawler* focado Ifrit. São apresentados suas principais características, semelhanças e diferenças em relação à abordagem proposta.

### 3.1 DEEPBOT

A ferramenta DeepBot (Álvarez, Raposo, Pan, Cacheda, Bellas, & Carneiro, 2007) é um *web crawler* focado em domínios específicos, desenvolvido pelo Departamento de Informação e Comunicações Tecnológicas da Universidade de Corunha, na Espanha. Na visão de seus criadores, como *web crawlers* convencionais não conseguem alcançar a *deep web*, a execução desta tarefa possui dois grandes desafios:

- Busca de informações do lado servidor da *deep web*. Os *web crawlers* convencionais não sabem como submeter os formulários que acessam bancos de dados ocultos.
- Busca de informações do lado cliente na *deep web*. Muitas páginas constroem seu conteúdo de maneira dinâmica, e a maioria dos *crawlers* não conseguem tratar este tipo de tecnologia.

Para tratar a busca do lado servidor, o DeepBot possui um mecanismo que recebe as definições de domínios que descrevem tarefas de coleta de dados. Desta forma, o DeepBot encontra automaticamente formulários relevantes e faz requisições pré definidas ao servidor. Para tratar o problema de busca do lado cliente, usa-se um conjunto de mini-browsers que simula um navegador e consegue desta forma executar o código que gera o conteúdo dinâmico da página.

### 3.1.1 ARQUITETURA

Web *crawlers* trabalham com URL's para acessar as páginas, mas existe um problema nessa abordagem quando se está tratando páginas com conteúdo dinâmico que necessite de estados pré definidos como *cookies*, ou dados de uma sessão, por exemplo. Para lidar com este problema, o sistema DeepBot armazena um objeto seção que contém as informações necessárias para acessar determinada página. Faz-se o uso de mini *web browsers*, construídos a partir de API's de grandes *browsers*, que conseguem executar os códigos responsáveis por gerar o conteúdo dinâmico das páginas. Para especificar a seqüência de navegação no *browser* usou-se a linguagem NSEQL (Pan, Raposo, Álvarez, Hidalgo, & Viña, 2002), que é uma linguagem que permite representar uma lista de eventos que devem ser seguidos para se encontrar a página desejada .

O funcionamento do *DeepBot* pode ser explicado da seguinte forma: o sistema entra em uma nova página, faz uma varredura no HTML procurando por formulários que se encaixem na descrição dos domínios configurados, e assim que um formulário relevante é encontrado, o sistema faz uma série de interações pré configuradas para aquele domínio. A definição de domínios será tratada a seguir.

### 3.1.2 DEFINIÇÕES DE DOMÍNIOS

Um domínio é formado por uma lista de atributos, cada um associado a um nome, um pseudônimo e um índice, uma lista de *queries*, cada uma composta por um atributo do domínio e um valor e um limiar relevante. Um atributo representa um campo do formulário e seus pseudônimos representam as possíveis variações para o mesmo nome. O valor do índice representa a importância que o campo tem em determinado domínio. As *queries* representam as consultas que se deseja executar nos formulários, cada consulta tem uma lista de pares de atributos e valores. O limiar relevante é um valor usado para determinar se um formulário é relevante para o domínio.

## Processando formulários

Para cada domínio, o sistema tenta combinar os atributos com os campos do formulário usando distância visual ou similaridade textual. Em seguida, é possível determinar se o formulário encontrado é relevante ao domínio. Em caso positivo, são executadas as *queries* definidas para o domínio, a fim de encontrar outras URL's que serão incluídas na base de dados. Nesta fase, é necessário determinar quais rótulos estão relacionados com quais campos do formulário. Em seguida são relacionados os rótulos do formulário com os atributos do domínio.

Para inferir o relacionamento entre campos e rótulos, usa-se a distância visual entre os dois elementos. Com o uso da API do *browser* é possível conseguir a coordenada dos elementos na tela. Elas são comparadas para que seja possível encontrar as menores distâncias e desta forma relacionar campos com seus rótulos. Quando existem rótulos com distâncias iguais, dá-se preferência àqueles dispostos à esquerda e não à direita, e acima e não abaixo.

Assim que os componentes do formulário tenham sido associados a um texto, é possível iniciar a fase de inferência da relevância do formulário para o domínio. Para isso é necessário relacionar os textos dos campos com os atributos do domínio.

Os campos de um formulário foram categorizados em dois grandes grupos: campos limitados, que possuem um número de opções restritas, como um COMBOBOX por exemplo, e campos ilimitados, que possuem opções infinitas. A idéia básica para gerar o *ranking* de similaridade é a comparação dos rótulos dos componentes com o atributos do domínio. Quando se trata de um campo limitado, o sistema leva em consideração também as possíveis opções aceitas pelo campo e as *queries* definidas para o atributo. As métricas propostas para a computação de similaridade dos textos combina TFIDF e o algoritmo de Jaro-Winkler (Cohen, Ravikumar, & Fienberg, 2003).

Para a fase de inferência da relevância do formulário para o domínio, um conjunto de atribuições entre campos do formulário e atributos de domínio é



gerado. Cada atribuição possui um valor de confiança expressado por um número que vai de 0 a 1. Para definir a relevância de um formulário são somados os valores de confiança de cada campo e esta soma deve ser maior do que o valor de limiar atribuído ao domínio.

Sendo o formulário relevante, então o sistema deve executar uma série de submissões para conseguir extrair toda a informação disponível no servidor onde está hospedado o formulário. O termo submissão é usado para representar o ato de enviar as informações preenchidas no formulário para o servidor, um exemplo desta ação pode ser o botão "salvar" de um formulário de cadastro. Como algumas páginas utilizam linguagens de *script* para tratar a submissão do formulário, o sistema deve procurar por componentes *INPUT* do tipo *SUBMIT*, *IMAGE* ou *BUTTON* e executar um clique no componente na expectativa de que o formulário seja submetido. Não obtendo resposta positiva, o sistema então procura por *links* que estejam associados com códigos *scripts* e executa um clique nestes elementos também. Se mesmo assim não for possível submeter o formulário, o sistema gera a submissão do formulário de maneira forçada, via JAVA Script por exemplo.

### **3.1.3 TESTES**

Para validar a proposta, foram feitos testes em três domínios diferentes: Compra de livros, compra de músicas e compra de vídeos. Para a definição dos domínios foram feitas visitas a dez sites de cada domínio e com base na experiência adquirida na visita dessas páginas, foram inferidos os atributos e índices.

Após definidos os domínios, a ferramenta DeepBot foi executada em vinte páginas de cada domínio e a partir de análise manual dessas mesmas páginas foi possível fazer a comparação dos resultados obtidos com o DeepBot. Para quantificar os resultados, obtidos foram utilizadas métricas para recuperação de informação: Precisão, Revocação e Medida-F. Os resultados obtidos foram promissores, algumas das métricas alcançaram 100% de acerto.

Alguns dos erros encontrados foram causados em sua maioria pelo fato de existir formulários com apenas dois campos, o que neste caso impossibilita que a soma dos valores de confiança ultrapassem o valor de limiar do domínio. Outro problema foi causado pela forma que o sistema foi desenvolvido, pois o sistema espera que todos os campos tenham um texto associado, mas alguns campos com respostas limitadas, como um COMBOBOX, não possuem um rótulo associado, pois o rótulo é representado pelo valor selecionado no próprio componente.

## **3.2 LABLEX**

O LabelEx (Nguyen, Nguyen, & Freire, 2008) é outro *crawler*, desenvolvido na Universidade de Utah, EUA, para resolver o problema da recuperação do conteúdo encontrado nas inúmeras bases de dados existentes na *deep web*. A proposta se baseia em uma abordagem de aprendizado que consiga automaticamente extrair os rótulos dos componentes contidos nos formulários. O problema chave aqui é a identificação correta de um rótulo.

Abordagem anteriores confiam em heurísticas para a detecção dos rótulos dos formulários, mas a maioria não consegue tratar os formulários gerados dinamicamente, ou seja, aqueles formulários que são gerados com o uso de uma linguagem de *script*, onde os componentes podem ser adicionados ao corpo do formulário mesmo após a página ser carregada no *browser*, este tipo de ação é disparado geralmente por alguma ação do usuário, um clique, por exemplo. Além do mais é muito difícil criar um heurística que consiga trabalhar de maneira satisfatória em todos os domínios, devido a grande variação entre estilos de construção de páginas. A estrutura desenvolvida consegue aprender novos *layouts* de formulários e isso garante maior eficácia no reconhecimento dos rótulos.

### **3.2.1 DEFINIÇÃO DO PROBLEMA E SOLUÇÃO**

O problema de extração de rótulos foi encarado como uma tarefa de aprendizado. Dado um conjunto de formulários web, é necessário aprender os

padrões usados para alocar os rótulos. A abordagem trata o problema de extração em duas etapas, primeiro existe a fase de treino e depois a fase de extração.

Na fase de treino, um conjunto de mapeamentos entre rótulos e elementos são criados manualmente. O sistema guarda características desta associação, tanto do rótulo quanto do elemento. Cada associação correta é marcada como um exemplo positivo e as demais são marcadas como exemplos negativos. O primeiro classificador, Naive Bayes (Zhang H. , 2004) é criado a partir destes dados e o segundo classificador, árvore de decisão, é criado a partir do resultado do primeiro.

A fase de extração é muito semelhante à fase de treino: um conjunto de mapeamentos é criado e suas características são extraídas. Após este passo, o conjunto com as associações é enviado para o módulo de poda que remove as associações incorretas. O resultado deste passo é enviado para um seletor de mapeamentos que faz uma nova análise nos dados para selecionar somente os mapeamentos considerados corretos. Existem outros passos após este que servem para analisar os elementos que não foram associados com algum rótulo e elementos que foram associados a mais de um rótulo.

### **3.2.2 GERANDO MAPEAMENTOS CANDIDATOS**

Dado um determinado formulário web, o LabelEx cria um conjunto de associações entre elementos e rótulos vizinhos. Para garantir maior eficácia, ele considera também os textos que se encontram fora de elementos HTML do tipo FORM. Como entrada para o algoritmo Naive Bayes poderiam ser usadas todas as possíveis combinações entre elementos e rótulos, mas isso seria um desperdício já que para ser entendido pelos usuários um rótulo deve estar próximo do elemento a qual está associado. Por este motivo serão geradas somente as associações entre elementos vizinhos. Para que dois elementos estejam perto o suficiente, o rótulo deve estar a uma distância menor ou igual, para cima ou para baixo, de  $x \cdot \text{altura}$  do componente e a uma distância menor

ou igual, para a esquerda ou direita, de  $y$ \*largura do componente. Foi identificado que bons valores para  $x$  e  $y$  são respectivamente 3 e 5.

O conjunto dos mapeamentos candidatos é gerado considerando também os componentes tratados dinamicamente. Os elementos vizinhos são definidos e suas associações mapeadas. Este passo gera um número grande de associações entre um componente e muitos rótulos. Por este motivo foi criado o passo de poda citado anteriormente.

### **3.2.3 EXTRAINDO CARACTERÍSTICAS**

Para que um humano consiga interpretar um formulário, os elementos relacionados devem estar próximos uns dos outros. Trabalhos anteriores conseguiram encontrar padrões usados para posicionar um rótulo dependendo do tipo de componente a qual ele está associado. Por exemplo, um rótulo associado a um campo de texto está geralmente posicionado acima ou a esquerda do mesmo. O objetivo do LabelEx é aprender estes padrões automaticamente.

Um conjunto de características é extraído das associações feitas para que cada mapeamento possa ser avaliado. Exemplos de informações recolhidas são tamanho da fonte, o tipo do componente, similaridade textual entre o rótulo e o elemento, distância entre os elementos, posição relativa entre o rótulo e o componente, etc.

### **3.2.4 APRENDENDO A IDENTIFICAR MAPEAMENTOS**

O número de associações erradas, entre elementos e rótulos, é muito grande. Este fato acaba por atrapalhar a construção de um classificador que consiga a eficácia pretendida. Por este motivo, o LabelEx trabalha com dois tipos de classificadores. O primeiro faz a poda das associações incorretas (Naive Bayes) e após este passo fica mais fácil a construção de um classificador (Árvore de decisão J48) para identificar as associações corretas. Esta

abordagem em dois passos se mostrou mais efetiva do que abordagens monolíticas.

Mesmo com ótimos resultados, o aplicativo não consegue ser seguro na inferência de rótulos, quando se trata de casos onde existem mais de uma boa opção de rótulo ou quando a aplicação não consegue encontrar uma opção. Então foi desenvolvida uma estrutura que possibilita o aprendizado a partir de experiências passadas. O princípio de que os termos mais comuns em um domínio são boas alternativas para rótulos quando comparados com termos pouco frequentes deve ser usado para classificadores específicos de um determinado domínio e não para classificadores genéricos.

### **3.2.5 TESTES**

Para validar a abordagem foi necessário fazer experimentos com formulários reais. Partindo de um conjunto de aproximadamente três mil páginas, foram executados testes sobre uma amostra destas páginas, considerando um acerto de 95% na representatividade da amostra. Os resultados foram validados utilizando as métricas de Revocação, Precisão e Medida-F. Foi necessário criar uma amostra, pois para executar o teste é preciso fazer a avaliação manual de todos os formulários e seria muito trabalhoso executar a avaliação manual em todas as páginas do conjunto de teste.

Como uma tentativa de responder a pergunta: "É possível criar um classificador genérico para qualquer domínio?" foi criado um classificador usando como treino amostras de todos os domínios usados no teste. Este classificador se mostrou muito bom, provando que as diferenças de *layout* de formulários se repetem em diferentes domínios. Viabilizando a construção de um classificador genérico.

O desempenho da abordagem foi também analisada em comparação a outras soluções existentes atualmente, usando dois softwares para fazer esta comparação, o HSP (Zhang, He, & Chang, 2004) e o IEXP (He, Meng, Yu, & Wu, 2004). O resultado foi que a metodologia usada na construção do LabelEx se mostrou superior àquelas usadas nas soluções anteriores. Uma explicação

para isso é que ambos HSP e IEXP usam um conjunto de regras fixo, então eles não são capazes de trabalhar sobre aquilo que não conhecem.

A abordagem baseada no aprendizado trouxe ótimos resultados na extração de rótulos de componentes de formulários. O esforço ficou concentrado em superar o problema de variação entre *layouts* encontrados na construção de um formulário. Foram realizados testes inclusive em comparação com outras soluções existentes atualmente, e o LabelEx se mostrou superior com uma margem considerável.

A metodologia levou em consideração que um elemento está associado a somente um rótulo e que um rótulo pode estar associado a mais de um elemento. Para o futuro é interessante elevar esta premissa para relacionamentos N:M. Outro ponto a melhorar é a estratégia usada para encontrar as associações corretas, a solução leva em consideração somente as informações individuais do componente e do rótulo. Para o futuro é interessante usar informações do formulário como um todo.

### **3.3 EXTRAINDO DADOS POR TRÁS DE FORMULÁRIOS WEB**

O objetivo é encontrar uma maneira estatística de preencher um formulário web para se ter acesso às informações contidas na base de dados do servidor relacionado ao formulário e ao mesmo tempo fazer isto de uma maneira eficiente. Os softwares existentes nesta área buscam geralmente preencher as informações do formulário com base em informações inseridas previamente. Estes produtos servem para facilitar uma compra na Internet por exemplo, onde o usuário não vai precisar preencher seu dados pessoais. Geralmente estão associados a somente um determinado domínio da web, impedindo o uso em outros domínios.

A abordagem proposta é complementar a um *software* já existente, o HiWE (*HiddenWeb Exposer*) (Raghavan & Garcia-Molina, 2000). O propósito do HiWE é preencher de maneira automatizada os formulários encontrados na *web*. Como existem muitos desafios na criação de uma ferramenta totalmente automatizada, o HiWE foi construído para depender da interação humana na

decisão de alguns pontos chave. Além do preenchimento de formulários, outra grande contribuição do HiWE foi a sua abordagem para o reconhecimento de elementos em um formulário através da posição visual do componente e não através do código HTML. Como incremento ao HiWE, a abordagem desenvolvida trata também da eliminação de registros duplicados e através de técnicas estatísticas tenta aumentar a eficiência do *crawler* a partir da inferência de que não existem mais informações que possam ser recuperadas de um mesmo formulário.

A abordagem trata de, primeiramente, fazer uma análise no código HTML da página que contém o formulário. Em seguida é gerado um requisição HTTP que é equivalente a preencher os campos do formulário. Desta forma, a ferramenta envia a requisição ao servidor e analisa o resultado obtido fazendo a remoção de registros repetidos, contornando páginas com erro e respostas inesperadas. Em algumas páginas, existe uma requisição padrão que traz todas as informações contidas no servidor, no entanto, muitas vezes é necessário submeter o formulário com variação nas opções escolhidas para cada campo afim de conseguir toda a informação contida no banco de dados por trás do formulário.

### **3.3.1 PLANOS DE SUBMISSÃO DOS FORMULÁRIOS**

A estratégia desenvolvida procura extrair toda a informação (ou uma parte significativa) contida no servidor. Nesse tipo de tarefa existem dois problemas: (i), o processo demanda tempo e (ii), muito provavelmente toda a informação contida no servidor será recuperada antes de que todas as possíveis submissões sejam feitas. Existem três passos que são executados pelo *software*: submissão da requisição padrão, comparação do resultado da requisição padrão com uma amostra do site para decidir se as informações retornadas são suficientemente abrangentes e submeter, de maneira exaustiva, as variações possíveis no formulário.

Para definir que já se tem um percentual aceitável da informação contida no servidor, e desta forma evitar consultas desnecessárias, existem algumas

configurações que podem ser feitas, como: decidir o número de consultas que devem ser feitas ao servidor, o número de bytes recebidos, o número máximo de consultas no qual não se teve alguma informação nova e o percentual de informação que deve ser recuperada do servidor. Estas regras definem os critérios de parada do processo de recuperação de informação.

Cada campo do formulário foi considerado como um fator no espaço de pesquisa. Para saber se a consulta padrão retornou toda a informação contida no servidor, foi desenvolvido esta regra: seja  $N$  o número total de possibilidades diferentes para se preencher o formulário (sem contar os campos abertos), e  $c$  o valor correspondente ao número de opções contidas no campo com maior número de opções e seja  $C$  o valor máximo entre  $\log N$  e  $c$ . Então  $C$  vai representar o número de consultas que deverão ser feitas para confirmar se a consulta padrão realmente trouxe toda a informação possível. É necessário tomar cuidado com a escolha das consultas que serão usadas para não deixar opções de fora e ao mesmo tempo usar a mesma opção várias vezes.

Se não for possível comprovar que a consulta padrão trouxe todos os dados possíveis, o algoritmo continua executando consultas com outras combinações de valores. É nesta hora que a configuração do usuário é utilizada.

### **3.3.2 TESTES**

A maior parte do trabalho foi feito com base em simulações, então para validar a abordagem foi necessário executar a ferramenta em algumas páginas. Foram usadas quinze páginas diferentes como ambiente de validação. Foi feita uma avaliação manual em cada página afim de verificar se a consulta padrão era capaz de trazer todas as informações e das páginas avaliadas, seis delas não passaram no teste, ou seja, a consulta padrão não é capaz de fornecer toda a informação encontrada no servidor.

Os resultados obtidos foram encorajadores. É necessário melhorar a escolha dos parâmetros usados em cada consulta para maximizar os resultados, ou seja, encontrar uma maior quantidade de informação com o



menor número de consultas possível. Existe um grupo de pesquisa trabalhando com integração e extração de informação através de ontologias, então a meta é conseguir incluir os componentes com respostas abertas, como um campo de texto por exemplo. Seria possível preencher esses campos e a partir daí fazer a busca de maneira específica. Até agora foi tratado somente a busca de informação sem levar em consideração o tipo da informação. Mas é necessário agora melhorar a qualidade de informação extraída.

## 4 IFRIT

Esta seção trata do desenvolvimento da *crawler*, onde são explicados os passos utilizados para a criação da lógica de processamento e geração de informação utilizada no desenvolvimento do *web crawler* focado Ifrit. São dois os principais problemas tratados no desenvolvimento deste trabalho: a detecção dos formulários encontrados em páginas da Internet; e a associação entre os componentes dos formulários e seus respectivos rótulos. Toda a metodologia desenvolvida é detalhada a seguir, assim como os passos que foram utilizados para se chegar a ela.

### 4.1 VISÃO GERAL

Esta seção faz um apanhado geral sobre a arquitetura desenvolvida para o *web crawler* focado Ifrit, apresentando as diferentes camadas da aplicação e suas respectivas contribuições, no reconhecimento e na extração dos formulários e seus rótulos.

A Figura 6 apresenta a estrutura do *crawler* Ifrit. A API foi o primeiro módulo a ser desenvolvido e certamente é o mais importante, toda a lógica de reconhecimento e extração de formulários e rótulos está implementada ali. Para que o módulo API funcione, é necessário que ele receba como entrada uma URL e as configurações feitas pelo usuário no arquivo API.txt. Para fazer o trabalho de navegação entre as páginas e a extração das URL's que abastecem a API, foi desenvolvido o módulo Crawler, que implementa a lógica de um *web crawler*, além da interface com o usuário. O Crawler recebe como entrada as configurações feitas no arquivo Crawler.txt, que representa, entre outras coisas, as configurações da quantidade total de páginas a ser visitada e o *browser* que deve ser usado para acessar as páginas. Existe ainda o módulo DAO, responsável pela conexão com o banco de dados e inserção das informações coletadas no processo.

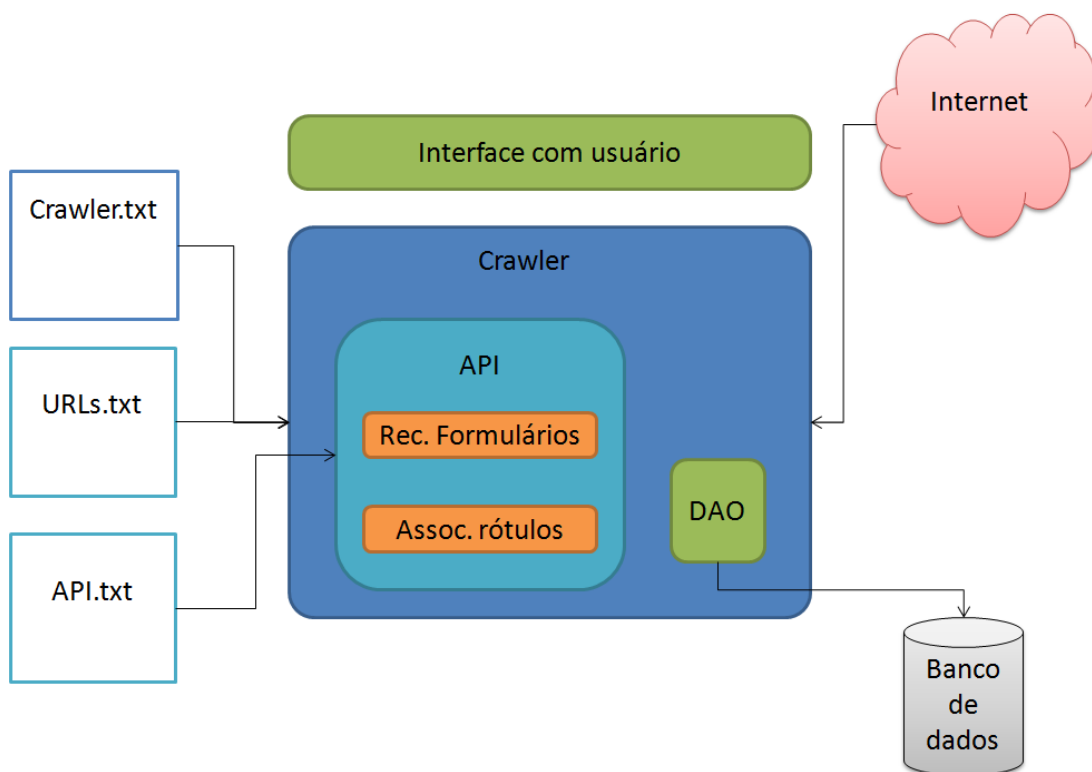


Figura 6 - Arquitetura do crawler Ifrit

## 4.2 DETECÇÃO DOS FORMULÁRIOS

Para resolver o problema da descoberta de um formulário em uma página qualquer da web, foi necessário definir as características que o código HTML deve apresentar para que seja possível inferir de maneira segura a existência de um formulário. O primeiro passo no desenvolvimento deste trabalho foi encontrar as peculiaridades de um formulário, lembrando que o objetivo do Ifrit é encontrar formulários que estão construídos sem o uso do elemento FORM do HTML. O algoritmo foi desenvolvido usando parâmetros configuráveis, descritos a seguir. Desta forma, todos os quesitos apresentados podem ser modificados de acordo com a flexibilidade que se deseja permitir na busca.

**(i) Forma de submissão do formulário** - trata de identificar o elemento responsável pela submissão das informações contidas no formulário.

Uma das características de um formulário é a forma de submissão dos dados preenchidos pelo usuário. Muitas vezes se usa um botão, entretanto, pelo fato de existirem muitas maneiras de se construir um elemento que

represente um botão, foi necessário desenvolver um algoritmo que conseguisse encontrar essas diferentes construções e classificá-las como sendo o elemento responsável pela submissão do formulário. Algumas páginas utilizam uma imagem que suporta um evento de clique e isto representa um botão, outras utilizam um elemento do tipo link e outras utilizam apenas elementos do tipo DIV. Estas construções, com exceção do botão construído com o uso de DIV, foram incluídas na rotina.

**(ii) Distância entre os elementos** - existe mais de um tipo de configuração referente à distância suportada pelo algoritmo. Essas configurações tratam de delimitar a distância máxima permitida entre os elementos e a densidade de elementos contidos em um formulário.

Foram definidos dois parâmetros: a distância máxima entre cada elemento do formulário e o elemento que representa o container; e a densidade máxima dos componentes de um formulário, que é calculada pela distância de um elemento do formulário e o próximo elemento do formulário, que esteja acima ou abaixo deste. Para entender esta definição de distância é necessário entender que o código HTML é tratado como sendo uma estrutura de dados do tipo grafo. Então existem nodos, que são os elementos, e arestas que são as ligações entre os elementos. Desta forma a distância é o número de arestas entre um elemento e outro. Nestas configurações de distância, cada elemento que não respeite o limite imposto não é considerado pertencente ao formulário em questão.

**(iii) Elementos visíveis e não visíveis** - esta configuração foi criada para dar mais flexibilidade ao *crawler*, pois existem algumas páginas que contém elementos que só são mostrados quando o usuário executa uma determinada ação. É importante enfatizar, que os elementos tratados por esta configuração são aqueles que já estão carregados na página, mas não estão sendo mostrados. Um exemplo disso são os formulários que permitem uma busca simples e uma busca avançada.

É possível escolher entre levar em consideração somente os elementos que estão visíveis na página ou considerar todos os componentes, ou seja, aqueles componentes que só aparecem quando o usuário efetua alguma

ação específica. Neste caso, são considerados invisíveis os componentes que estão no código HTML mas não estão sendo apresentados, por exemplo. Os casos em que o componente não está presente no código HTML e que é injetado no código da página, através de um evento JavaScript, não são tratados neste trabalho.

Para um bom entendimento deste capítulo, é necessário a descrição do termos usados nos próximos parágrafos:

- **elemento** - é usado para fazer referência a qualquer tipo de construção suportada pelo HTML.
- **componente** - refere-se a um elemento que representa um campo do formulário, um CHECKBOX, um TEXTBOX, um COMBOBOX entre outros.
- **rótulo** - também se refere a um elemento, mas considera-se um rótulo como um elemento de texto que está associado a um componente do formulário.
- **container** - outro elemento do HTML. Um container é um elemento de agrega outros elementos, um DIV por exemplo. No texto o container está referenciando o elemento mais externo do formulário, ou seja, aquele que comporta todos os outros.

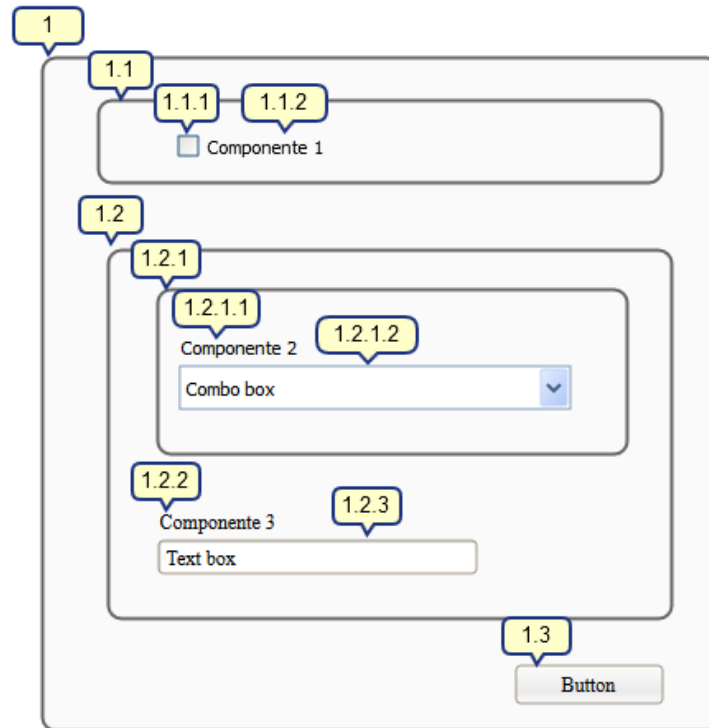
#### 4.2.1 ALGORITMO

O primeiro passo executado pelo algoritmo é a construção de uma árvore de elementos que mantém a mesma estrutura contida na página. A árvore de elementos serve como base para que o restante do processamento seja feito, ou seja, o código HTML de uma página é organizado de forma que os elementos formem uma árvore com os seus diferentes níveis.

A árvore de elementos é feita a partir da API do Ifrit, criada especialmente para este trabalho. Neste caso, a API fornece os métodos usados para percorrer a árvore e avaliar os elementos contidos no código HTML, além de fornecer os dados que são usados pela segunda parte da

tarefa, que é a inferência de um rótulo para cada componente contido no formulário.

A medida que a árvore vai sendo construída, cada novo elemento adicionado à estrutura recebe um ID que o acompanhará pelo resto do processamento. Este ID serve não só para identificar o elemento mas também para informar sua localização na árvore. O conjunto de ID's segue também uma estrutura onde o ID do elemento "pai" é usado para compor o ID do elemento "filho". Para exemplificar este processo pode-se pensar na lógica usada para enumerar os diferentes títulos e subtítulos de um artigo. Neste caso, o primeiro título terá o ID "1" e seu primeiro subtítulo terá o ID "1.1", já o segundo título será o "2" e seus subtítulos serão "2.1" e "2.2", por exemplo. A estrutura usada para construir os ID's para a árvore de elementos é conhecida como "Numeração *DEWEY*" (Tatarinov, 2002). A Figura 7 mostra um exemplo da utilização prática deste conceito, ela contém um formulário com seus elementos já indexados. Para entender a Figura 7 é necessário estar familiarizado com os elementos contidos na estrutura de um código HTML. Basicamente este formulário é formado por um *CHECKBOX*, um *COMBOBOX*, um *TEXTBOX* e um botão, além dos rótulos e dos *DIV's* representados pelos retângulos.



**Figura 7 - Exemplo de formulário e uso da numeração DEWEY**

O algoritmo que verifica a existência de um formulário faz, basicamente, uma varredura pela árvore de elementos e uma série de testes para saber se as configurações apresentadas anteriormente estão sendo atendidas. A Figura 8 apresenta um pseudo código que ilustra a navegação pela árvore de elementos. Este processo é feito de maneira recursiva, então o algoritmo se repete para cada nodo contido na árvore.

```

1 #Declaração de classe
2 treeNode
3
4 #Declaração de variável
5 childrens : List (treeNode)
6
7 #Declaração de método
8 begin isForm(formElements : List (treeNode))
9
10     #Verifica se o elemento que está executando é um elemento de formulários
11     if isFormElement(this)
12         #Adiciona o elemento que está executando à lista dos elementos de formulários encontrados
13         formElements.add this
14
15     #Percorre os filhos do node
16     for each treeNode in childrens
17         #chama recursivamente o método inicial
18         isForm(formElements)
19
20     #Ao final, verifica se foram encontrados elementos em quantidade suficiente para inferir que
21     #existe um formulário e se estes elementos respeitam as regras estabelecidas.
22     if verifyRules(formElements)
23         return true
24     else
25         return false
26
27 end is Form

```

**Figura 8 - Pseudo código ilustrando a navegação na árvore de elementos**

Basicamente a função `isForm`, linha 8, é executada para cada nodo contido na árvore de elementos e o que ela faz é verificar se existem elementos suficientes para que seja possível afirmar a existência de um formulário. O método `verifyRules`, linha 22, confronta os dados obtidos, linha 11, com o conjunto de regras estabelecidas e os elementos que não estiverem dentro do padrão esperado são removidos da lista de elementos do formulário.

Ao final da execução desta rotina, é possível saber se o código HTML usado na execução possui uma estrutura de elementos que pode ser considerada um formulário. Além da lista de elementos contidos no formulário, é necessário armazenar também os possíveis rótulos e seus respectivos ID, que são usados como entrada para a próxima parte do algoritmo que é a extração de rótulos. Tendo um resultado positivo nesta etapa ou seja, sendo possível encontrar um formulário, o *crawler* então passa para a fase de inferência de rótulos. Nesta etapa, são associados componentes e rótulos de maneira que ao final cada componente tenha associado a si um rótulo. Isto é claro, se houverem rótulos suficientes para todos os componentes.



### **4.3 EXTRAÇÃO DOS RÓTULOS**

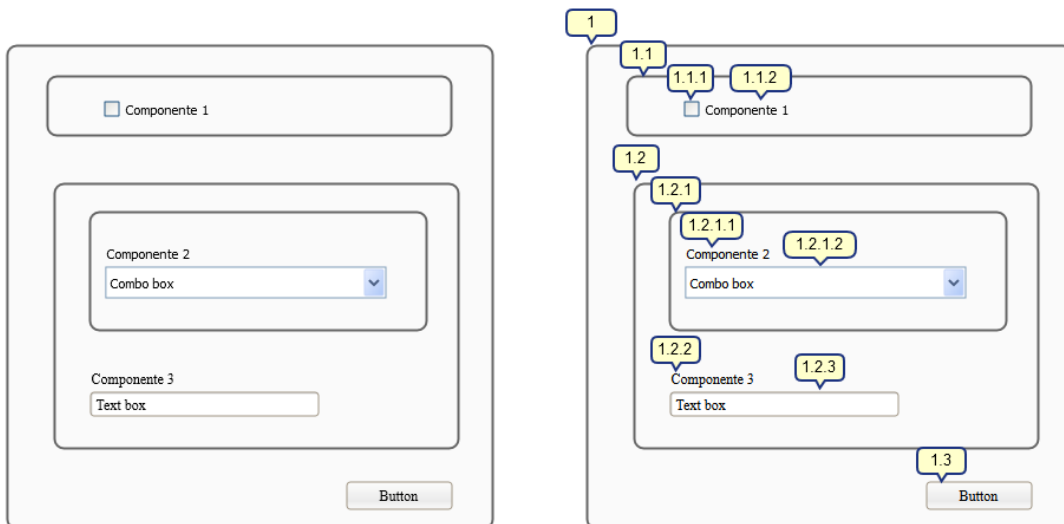
Nesta parte da execução, os componentes do formulário e os rótulos são associados com base em seus ID's. A entrada para este processamento é formada basicamente por duas listas de elementos, uma representando os componentes do formulário e a outra representando os possíveis rótulos. A lógica utilizada aqui é bem simples: basicamente se deseja encontrar para cada componente do formulário, um rótulo correspondente. Não é levado em consideração que alguns componentes não possuem um rótulo, ou ainda, compartilham seu rótulo com outros componentes, o método simplesmente atribui um rótulo diferente para cada elemento do formulário.

A lógica neste ponto é simples, mas eficaz. O algoritmo tenta encontrar o rótulo que está mais próximo de determinado elemento para fazer a associação. Este cálculo de distância é feito utilizando a mesma linha de pensamento usada no módulo que busca reconhecer um formulário, com o acréscimo de algumas particularidades envolvidas no cálculo, explicadas no decorrer do texto.

#### **4.3.1 ALGORITMO**

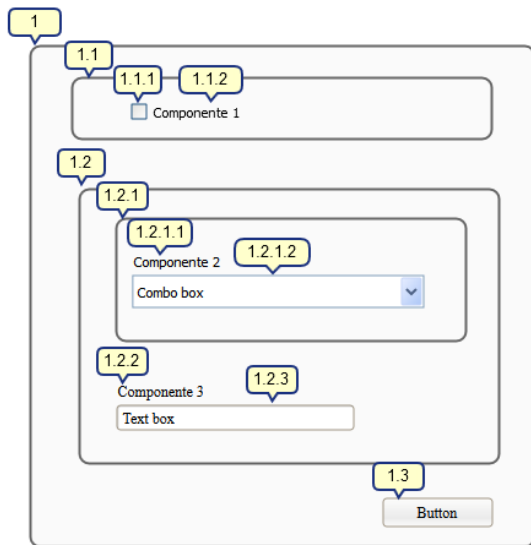
O algoritmo usado no módulo de inferência de rótulo trata basicamente com os ID's dos elementos. O que ele faz é formar pares com os elementos - rótulos e componentes do formulário - que estão mais próximos uns dos outros.

A Figura 9 apresenta um exemplo do processo de extração e associação dos rótulos contidos em um formulário. Do lado esquerdo pode-se ver um pequeno formulário e do lado direito sua representação aplicando a numeração *DEWEY* após o processo de criação de árvore de elementos explicada anteriormente. É possível notar que cada elemento do formulário recebe uma identificação e que esta identificação por si só é capaz de dizer a posição (em relação aos níveis da árvore) de cada elemento. Esta representação de identificação é muito importante para o resto do processamento pois possibilita, de uma maneira simples, fazer o cálculo da distância entre os elementos.



**Figura 9 - Exemplo de extração e associação de rótulos**

Junto com o processo de construção da árvore de elementos, todos os elementos da página são visitados em uma busca em profundidade ou seja, os elementos são visitados na mesma ordem de seus ID's. Cada vez que o mecanismo de busca retorna para um nó já visitado significa que já se tem a informação dos componentes filhos do nó atual. Então é possível fazer os testes para tentar reconhecer um formulário. Isto significa que a medida que os elementos são visitados pelo mecanismo de busca, são feitos testes para saber se com as informações encontradas até o momento é possível reconhecer um formulário. Todos os elementos que estiverem respeitando as regras impostas na configuração serão adicionados à lista de elementos válidos. A mesma coisa é feita com os elementos de texto. No exemplo, são encontrados quatro elementos de formulário (ID's: 1.1.2, 1.2.1.2, 1.2.3 e 1.3) e três elementos de texto (ID's: 1.1.2, 1.2.1.1 e 1.2.2). A Figura 10 apresenta o resultado desta etapa.



- Elemento:
- CheckBox 1.1.1
  - ComboBox 1.2.1.2
  - TextBox 1.2.3
  - Button 1.3

- Texto:
- Componente1 1.1.2
  - Componente2 1.2.1.1
  - Componente3 1.2.2

**Figura 10 - Exemplo da extração de componentes de rótulos**

O primeiro passo executado nesta parte visa preparar as informações que serão usadas no decorrer da execução. Para encontrar os pares de elementos, é necessário conhecer a distância entre todos os elementos envolvidos, então a primeira coisa que o algoritmo faz é construir uma matriz que fornece a distância de cada componente em relação a cada rótulo. A Tabela 1 apresenta a matriz gerada para o exemplo.

**Tabela 1 - Matriz do cálculo de distância entre componentes de rótulos**

	CheckBox 1.1.1	ComboBox 1.2.1.2	TextBox 1.2.3
Componente1 1.1.2	1	-1.1.2	-1.1
Componente2 1.2.1.1	1.1	0	-2.1
Componente3 1.2.2	1.1	1.2	0

Os valores de distância contidas na Tabela 1 foram calculados com o uso da fórmula de distância apresentadas na Equação 1.

$$\frac{\text{Id Texto} - \text{Id Componente}}{\text{Coeficiente}}$$

#### **Equação 1 - Fórmula para o cálculo de distância entre os elementos**

Existem alguns pontos importantes sobre esta fórmula que serão explicados a seguir:

- O caractere "." é totalmente ignorado no cálculo.
- Começando da esquerda para a direita, todos os valores iguais nos dois ID's são desconsiderados até que se encontre o primeiro valor diferente, exemplo: sendo o valor para o ID do texto "1.1.1" e o valor para o ID do componente "1.1.2.1", fazendo a remoção dos valores repetidos temos "1" como ID do texto e "2.1" como ID do componente. É importante enfatizar que este procedimento de remoção de valores repetidos só é feito nos primeiros valores, ou seja, nenhum valor é removido à direita da primeira posição com valores diferentes.
- Id do texto sempre é posto à frente do ID do componente do formulário. Isto faz com que o resultado fique negativo quando o ID do texto é menor do que o ID do componente, o que quer dizer que o texto está posicionado (na árvore de elementos) antes do componente do formulário.
- Sempre que o resultado do cálculo de distância é igual a "-1", este é substituído por "0" ou seja, é a menor distância possível entre os elementos. Isso quer dizer que o componente de texto está imediatamente acima do componente do formulário e que o *crawler* irá considerar os dois, texto e formulário, associados.

Tendo em mãos a matriz de distâncias, encontra-se então os menores valores de distância, sem considerar o sinal, e estes são associados aos elementos relacionados. A Tabela 1 mostra que as menores distâncias estão

presentes na diagonal principal da matriz, ou seja, pode-se observar que o algoritmo associa o "Checkbox 1.1.1" com o rótulo "Componente1 1.1.2" com distância igual a 1, o "ComboBox 1.2.1.2" com o rótulo "Componente2 1.2.1.1" tendo distância igual a 0 e o "TextBox 1.2.3" com o rótulo "Componente3 1.2.2." tendo distância igual a 0. Observando novamente a Figura 10, é possível perceber que a associação foi bem sucedida, ou seja, cada componente recebeu o rótulo que lhe é de direito.

Afim de não considerar as características inerentes ao uso de formatação visual dos elementos de texto, o algoritmo agrupa os elementos de texto que possuem ID's em sequência, ou seja, quando o resultado do cálculo de distância é "0". Um exemplo de situação onde esta atitude é necessária é apresentado na Figura 11, onde existe uma palavra em negrito em um rótulo. Na linha 4 do quadro à direita, o efeito de negrito da palavra "Mês" é feito através do elemento "b" do HTML, o restante do rótulo "\_de nascimento" está dentro de outro elemento do HTML, o SPAN. Sendo assim cada um destes textos irá ocupar um nó diferente na árvore de elementos e ambos receberão diferentes ID's. Analisando a imagem do componente é possível dizer que os dois elementos compõem apenas um rótulo "Mês de aniversário", por isso o algoritmo concatena todos os elementos de texto que estão em sequência na árvore de elementos.



```
1 <html>
2 <body>
3 <div>
4     <b>Mês</b><span> de nascimento</span>
5         <select style="display: block;">
6             <option>Selecione</option>
7             .
8             .
9             .
10        </select>
11 </div>
12 </body>
13 </html>
```

Figura 11 - Exemplo de agregação de rótulos em sequência

## 4.4 IMPLEMENTAÇÃO

Todo o *crawler* foi desenvolvido utilizando a linguagem de programação JAVA da Oracle e a *IDE (Integrated Development Environment) open source* Eclipse (Eclipse Foundation).

### 4.4.1 A APLICAÇÃO

A aplicação está dividida em dois módulos:

- A API, que consiste na lógica de tratamento do código HTML e JavaScript, na descoberta de um formulário e seus elementos e na inferência de rótulos para cada elemento do formulário. A Figura 12 apresenta o diagrama de classes da módulo API.

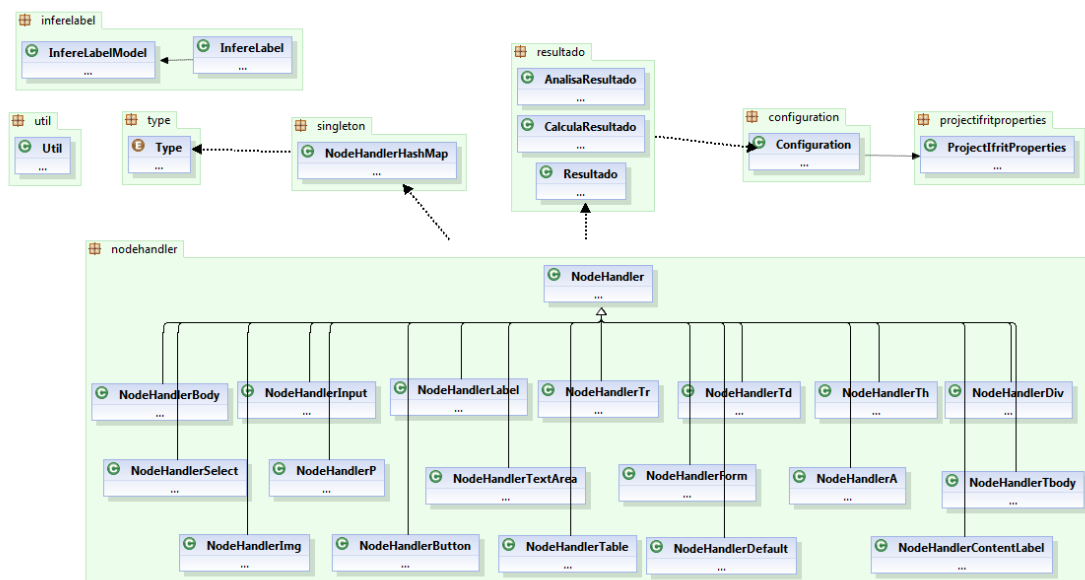


Figura 12 - Diagrama de classes da API do Ifrit

- *Web Crawler*, que é a parte da aplicação responsável por encontrar as páginas que alimentam a API. A parte de interface com o usuário também está inclusa neste módulo. A Figura 13 mostra o diagrama de classes do módulo *Web Crawler* do Ifrit.

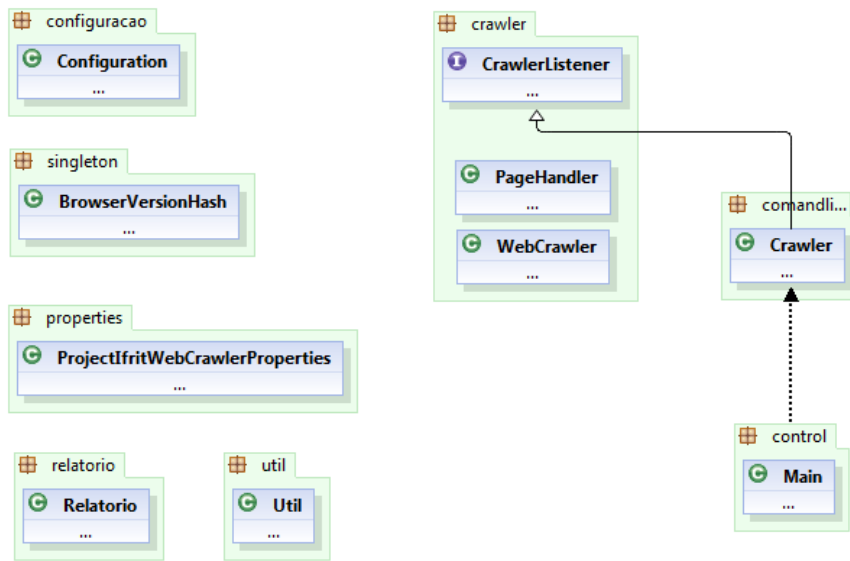


Figura 13 - Diagrama de classes do Web Crawler

Na Figura 14, é possível ver o ponto de ligação entre os dois módulos - API e *Web Crawler*. Basicamente a ligação é feita pelas classes contidas no pacote crawler do *Web Crawler* e as classes dos pacotes resultado e inferelabel da API.

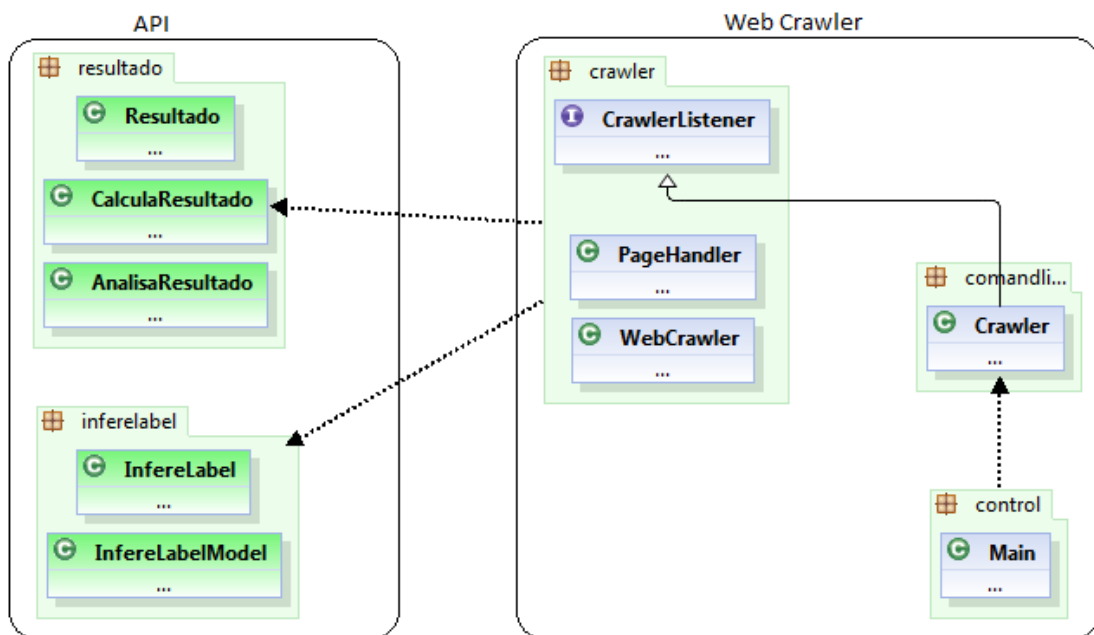


Figura 14 - Diagrama de classes mostrando a conexão entre API e Web Crawler

O *crawler* implementado para o Ifrit trabalha a partir de uma busca em largura, então ele vai funcionar da maneira como foi explicado anteriormente: na primeira iteração, o *crawler* visita a página representada pela semente, na segunda iteração ele visita todos os links contidos na semente e assim por diante. Por ser um *crawler* focado, o Ifrit guarda apenas as informações que lhe é relevante, ou seja, as informações sobre os formulários.

#### 4.4.2 API

Todo o motor central, ou seja, o *core* da aplicação, está contido no módulo API e basicamente são dois os problemas resolvidos por este módulo:

- Inferência sobre a existência de um formulário dada uma página da web. Este é o problema que motivou a execução deste trabalho, conseguir identificar automaticamente diferentes tipo de formulários em páginas na Internet.
- Inferência de rótulos para cada elemento contido no formulário, caso este exista. Para tornar possível a extração de informação relevante de formulários *web* é necessário que se conheça o tipo de informação que determinado campo está tentando recolher. Por isso a importância de se conhecer o "nome" ou rótulo do campo, pois assim é possível saber do que se trata determinado campo.

A API do Ifrit foi desenvolvida utilizando o HTMLUnit (**Sourceforge, 2012**), que é um *browser* sem interface gráfica, criado para poder ser incorporado em aplicações escritas em JAVA. O HTMLUnit é uma aplicação de código aberto e está sob a Apache Licence V2.0 (**Apache, 2012**). Optou-se por usar uma API externa no desenvolvimento do Ifrit para que fosse possível focar os esforços na implementação da lógica da aplicação e não no tratamento de elementos HTML. Outras API's foram testadas mas nenhuma delas era tão completa quanto a HTMLUnit, que consegue tratar códigos em JavaScript e é muito simples de se utilizar.



### 4.4.3 WEB CRAWLER

Este módulo trata principalmente da interface com o usuário (feita através de linha de comando) e do *crawler* em si. Existem muitos *web crawlers* disponíveis na rede, mas optou-se por construir um especialmente para o Ifrit. A motivação para isso veio da necessidade de se ter uma aplicação exclusiva e focada nos objetivos deste trabalho, descritos na introdução. Este módulo da aplicação faz a interface entre o usuário e o *core* propriamente dito, além de recolher as URL's que servem como entrada para o algoritmo de reconhecimento de formulários.

O desenvolvimento do *crawler* apresentou algumas complicações em relação a memória da aplicação. Deve ser um requisito básico de todo *web crawler* que ele seja capaz de ser executado por longos períodos de tempo sem a supervisão de um operador ou algo do gênero. Mas os primeiros testes com a aplicação demonstraram que não era possível navegar por muitas páginas sem que a aplicação apresentasse erros de estouro de memória.

Assim como foi apresentado no início deste capítulo, na Figura 6, o *crawler* Ifrit pode ser configurado para atuar de acordo com as diferenças encontradas nas diversas páginas da web.

As configurações de entrada relevantes estão nos arquivos API.TXT e CRAWLER.TXT. A seguir, são apresentados os pontos principais da configuração. No CRAWLER.txt estão as configurações a respeito do processo de busca de páginas, exemplo:

1. Número de páginas que devem ser visitadas
  - max\_pages\_to\_visit=200
2. Opção de mostrar ou não as mensagens do log da API HtmlUnit (true/false)
  - logger=false
3. Opção de gerar ou não os arquivos de log (true/false)
  - report=true
4. Opção de fazer a leitura das páginas iniciais pelo arquivo de entrada URLs.txt (O arquivo URLs.txt deve conter apenas URL's)
  - use\_input\_file=true

5. Nome do arquivo de entrada
  - `input_file_name=URLS.txt`

No arquivo `API.txt`, estão as configurações sobre o *parser* do HTML, inferência de rótulos e formulários. Os itens a seguir servem como delimitadores, todos os formulários que estiverem dentro dos parâmetros são considerados, os demais não.

1. Número mínimo de botões permitidos (serve para delimitar a quantidade de botões que o formulário deve ter para que ele seja reconhecido, o mesmo vale para o item 2)
  - `qtd_min_submit=1`
2. Número máximo de botões permitidos
  - `qtd_max_submit=2`
3. Número mínimo de componentes de entrada (está sendo considerado um componente de entrada, qualquer componente do formulário) . Esta configuração serve para delimitar a quantidade de componentes que um formulário deve ter para que seja reconhecido. O mesmo vale para o item 4.
  - `qtd_min_input=4`
4. Número máximo de componentes de entrada (Neste caso, com o valor em branco, não será considerado um número máximo de inputs)
  - `qtd_max_input=`
5. Opção de calcular o número de componentes de entrada separadamente (`true/false`) (Serve quando se deseja um tipo de formulário muito específico, onde deve existir um número limitados de determinados elementos). Quando esta configuração está ativa (valor igual a `true`), os valores dos itens 3 e 4 não são considerados, já os itens 6 e 7, são.
  - `usar_qtd_tipo_input=false`
6. Número mínimo de componentes do tipo `<input type=text>` (somente será considerado esta configuração quando `usar_qtd_tipo_input` igual a `true`)

- `qtd_min_input_text=1`

7. Número máximo de componentes do tipo `<input type=password>`

- `qtd_max_input_password=0`

Existem outras configurações para delimitar a quantidade dos componentes separadamente, assim como os itens 6 e 7, que não estão sendo apresentadas aqui. Os itens omitidos seguem a mesma lógica, mas são destinados aos outros componentes de formulários: TEXT AREA, SELECT, RADIO BUTTON, e assim por diante.

8. Distância máxima entre um componente e o *container* do formulário

- `distancia_max_parent=12`

9. Distância máxima para que o algoritmo continue considerando um componente, mesmo que não existam outros componentes acima deste. Se o algoritmo percorrer uma distância maior do que a especificada neste item sem encontrar um componente de formulário, todos os componentes encontrados até o momento são descartados. Isto só não acontece quando o algoritmo diz que os componentes encontrados são suficientes para confirmar a presença de um formulário.

- `distancia_max_sem_form=2`

10. Opção de considerar somente os elementos que estão visíveis na página (`true/false`). Esta configuração é capaz de excluir da análise os elementos que não estão aparecendo na página.

- `apenas_visivel=false`

11. Opção de considerar um elemento do tipo `<a>` como sendo um botão. Este item serve para que os links de uma página sejam tratados como possíveis botões, ou seja, elementos que são responsáveis por enviar as informações para o servidor.

- `a_botao=true`

12. Atributo `class` do botão. Quando o algoritmo está considerando `<a>` como botão, esta propriedade será testada no atributo `class` de um link. Esta configuração serve para decidir se um link está fazendo o trabalho de um botão ou não. Diferentes valores podem ser cadastrados, basta separá-los com um espaço em branco

- `class_a_botao=button`

13. Opção de considerar um elemento do tipo `<img>` como um botão. Mesma lógica do item 11.

- `img_botao=true`

Todo o desenvolvimento deste trabalho foi feito tendo como base cinco páginas, então todos os problemas encontrados nestas páginas foram tratados de alguma maneira. As páginas utilizadas (acessadas no mês de março do ano de 2012) serão apresentadas a seguir:

1. <http://www.decolar.com.br>
2. <http://www.expedia.com/daily/cars/default.asp>
3. <http://www.whsmith.co.uk/CatalogAndSearch/AdvancedSearch.aspx?cat=Books&uw=false>
4. <http://www.buybooksontheweb.com/advancedsearch.aspx>
5. <http://www.worldcat.org/advancedsearch>

O método de desenvolvimento usado foi o seguinte: desenvolver a aplicação com todos os requisitos básicos e ir testando diferentes parâmetros de configuração em cada uma das páginas. Como essas páginas apresentam diferentes características, uma configuração que serve para uma delas, não serve para outra. Então foi necessário algumas vezes fazer até mudanças no código do *software*.

Uma das alterações que surgiu a partir dos testes foi possibilitar o reconhecimento de uma imagem como sendo um elemento do tipo botão, ou seja, um elemento que pode receber um clique e disparar determinada ação na página. Esta configuração é representada pelo campo "img\_botao" do arquivo de configuração. Outro exemplo é o campo "class\_a\_botao", que permite a configuração dos possíveis nomes de classes do CSS que podem ser encontradas em elementos "a" do HTML que representem botões.

Enfim, o desenvolvimento do aplicativo acabou sendo um processo iterativo, em que várias modificações foram necessárias para tornar o

comportamento do *crawler* aceitável frente a todas as páginas apresentadas anteriormente, utilizando o mesmo arquivo de configuração.

## **4.5 EXPERIMENTOS**

Nesta seção, são apresentados os experimentos realizados a fim de verificar a qualidade da abordagem utilizada na resolução do problema proposto: detecção de formulários e associação de rótulos com campos de formulários. Existem principalmente dois pontos para serem avaliados nesta abordagem, o primeiro problema que precisou de uma solução foi a identificação de formulários a partir do código fonte de um determinada página. Tendo esta informação em mãos, vem a tona outro problema: identificar os rótulos de cada campo, pois esta informação é capaz de nos dizer o que o campo representa.

Para a execução com experimentos, foram selecionadas e analisadas páginas de cinco domínios diferentes: aluguel ou compra de veículos, reserva de hotéis, compra de passagens aéreas e busca por empregos. Algumas das páginas se incluem em mais de um desses domínios. Por isso foi criado mais uma categoria. A categoria viagens inclui então as páginas que oferecem o serviço de reserva de hotel, aluguel de veículos e compra de passagens aéreas, totalizando cinquenta páginas. Após executar o *crawler*, os resultados obtidos foram comparados com os dados resultantes da avaliação manual.

### **4.5.1 VARIÁVEIS**

Na avaliação dos dados obtidos, foram usadas métricas padronizadas de Recuperação de Informação: Precisão, Revocação e Medida-F (Rijsbergen, 1975). Além das métricas, foram definidas as variáveis que serviram de entrada para as métricas.

- **CampoFormulario/lfrit:** Conjunto com os campos encontrados pelo lfrit no formulário.
- **CampoFormulario/Real:** Conjunto com os campos reais do formulário.

- **RotuloCampo/lfrit:** Conjunto com as associações entre campo/rótulo encontrados pelo lfrit. O *crawler* sempre vai atribuir um rótulo para um componente, desde que exista pelo menos um rótulo candidato. Então, vão existir casos em que o componente real não possui um rótulo, mas o *crawler* vai inferir um. Estes casos serão considerados como erros, ou seja, não foi feita uma atribuição correta.

- **RotuloCampo/Real:** Conjunto com as associações entre campo/rótulo reais do formulário. São considerados somente o número de componentes que possuem um rótulo.

#### 4.5.2 MÉTRICAS

$$PrecisãoCampoFormulario = \frac{|CampoFormularioLfrit \cap CampoFormularioReal|}{|CampoFormularioLfrit|}$$

Equação 2 - Fórmula para cálculo de precisão por campo do formulário

$$RevocaçãoCampoFormulario = \frac{|CampoFormularioLfrit \cap CampoFormularioReal|}{|CampoFormularioReal|}$$

Equação 3 - Fórmula para cálculo de revocação por campo do formulário

$$MedidaFCampoFormulario = \frac{2 \times PrecisãoCampoFormulario \times RevocaçãoCampoFormulario}{PrecisãoCampoFormulario + RevocaçãoCampoFormulario}$$

Equação 4 - Fórmula para cálculo de medida-f por campo do formulário

$$PrecisãoRotuloCampo = \frac{|RotuloCampoIfrit \cap RotuloCampoReal|}{|RotuloCampoIfrit|}$$

Equação 5 - Fórmula para cálculo de precisão por associação entre rótulo e campo do formulário

$$RevocaçãoRotuloCampo = \frac{|RotuloCampoIfrit \cap RotuloCampoReal|}{|RotuloCampoReal|}$$

Equação 6 - Fórmula para cálculo de revocação por associação entre rótulo e campo do formulário

$$MedidaFRotuloCampo = \frac{2 \times PrecisãoRotuloCampo \times RevocaçãoRotuloCampo}{PrecisãoRotuloCampo + RevocaçãoRotuloCampo}$$

Equação 7 - Fórmula para cálculo de medida-f por associação entre rótulo e campo do formulário

### 4.5.3 RESULTADOS

Abaixo serão apresentados os valores calculados para cada métrica. Os valores estão ordenados em ordem crescente para facilitar o entendimento.

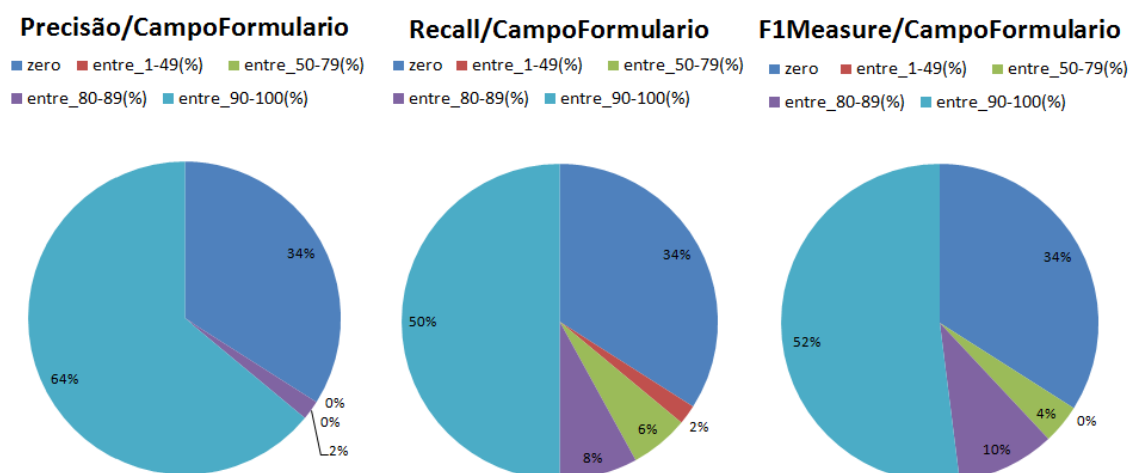


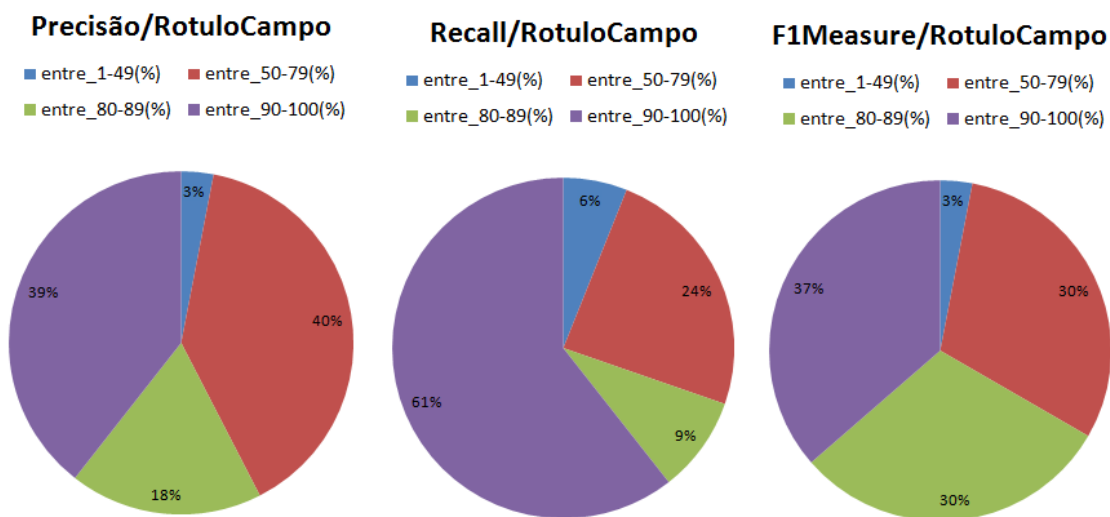
Figura 15 - Resultados das métricas por campo do formulário

A Figura 15 apresenta os valores para todas as métricas referentes ao reconhecimento dos campos do formulário. Os resultados de Precisão por campo do formulário mostram que 32 das 50 páginas avaliadas, ou seja, 64%, apresentou resultado satisfatório. Isso significa que em 64% dos casos o *crawler* acerta em sua inferência de campo para um formulário.

Para a métrica de Revocação por campo do formulário, os valores mostram que a metade dos casos houve acerto de 100% no reconhecimento dos campos contidos nos formulários avaliados, e considerando o valor de 80% aceitável, temos um número ainda maior, onde 29 das 50 páginas mostraram um resultado aceitável, ou seja 58%.

Para os resultados da Medida-F, considerando o valor de 80% como satisfatório, 62% das páginas analisadas apresentam um bom resultado.

Para os valores referentes a rótulos, foram considerados somente os casos em que o formulário foi reconhecido pela aplicação, pois nos outros casos o *crawler* nem chegou a inferir os rótulos para os componentes.



**Figura 16 - Resultados das métricas por associação entre rótulo e campo**

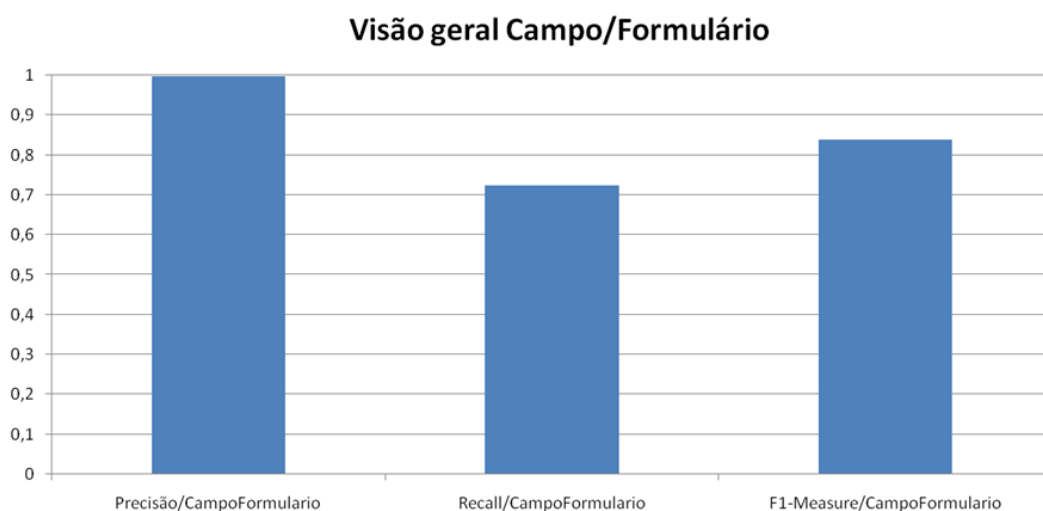
A Figura 16 mostra os valores obtidos para a métrica de Precisão na associação de componentes e rótulos, mostrando que o *crawler* erra em 42%



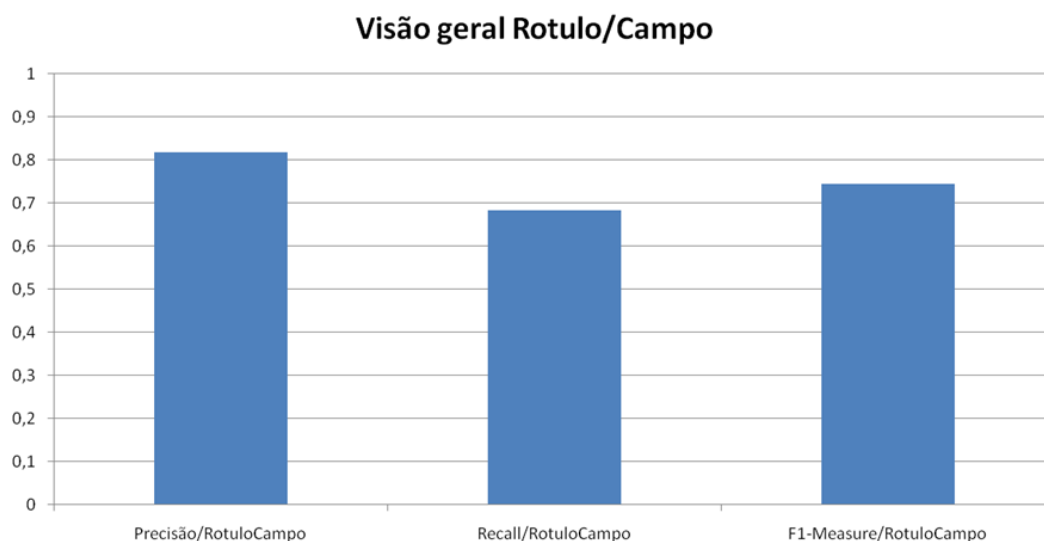
dos casos. Já nos resultados de Revocação para a associação entre componentes e rótulos pode-se ver que 66% dos formulários avaliados, apresentaram um resultado satisfatório. Isto quer dizer que em 66% dos casos o *crawler* consegue associar corretamente 80% dos campos contidos no formulário. No caso da Medida-F, o comportamento geral do *crawler* é satisfatório em 63% dos casos.

A seguir são apresentados os valores para cada métrica considerando a totalidade dos dados. Este tipo de informação foi considerado importante para que seja possível visualizar o comportamento geral da aplicação. É possível perceber que o *crawler* Ifrit apresentou bons resultados na execução dos testes. Quase 100% dos componentes reconhecidos pela aplicação realmente faziam parte do formulário.

As Figura 17 e Figura 18 apresentam os valores por métrica considerando o número total dos valores, ou seja, o número total de componentes, o número total de rótulos, o número total de formulários, etc. Pode-se perceber que ambas as medidas ficaram acima de 80%, sendo assim um bom resultado. Apenas o valor de Revocação para o reconhecimento dos campos que apresentou valor inferior a 80%, isso devido aos formulários que não foram reconhecidos.



**Figura 17 - Resultados gerais por métrica - Campo/Formulario**



**Figura 18 - Resultados gerais por métrica - Rotulo/Campo**

#### 4.5.4 DISCUSSÃO

Com a execução dos experimentos, foram encontradas maneiras diferentes de implementações e estilos de apresentação dos componentes na tela. Isto foi de grande valia pois proporcionou uma avaliação mais ampla da implementação do *crawler*. Segue abaixo as principais características encontradas nas páginas usadas nos testes e um comentário dos seus efeitos no comportamento da aplicação.

**Tabela 2 - Problemas**

<b>PROBLEMAS</b>		
(Características referentes à configuração que afetaram o desempenho do <i>crawler</i> )		
<b>1</b>	<b>Formulários com número de elementos &lt; 4</b>	Foram encontradas um total de oito páginas que contém formulários com menos do que quatro elementos. Estes formulários não foram reconhecidos pelo <i>crawler</i> pois ele foi configurado desta forma para impedir o reconhecimento de

		formulários de login. Poderia ser usada a configuração de elementos específicos, onde é possível excluir formulários com a ocorrência de campos de senha.
2	<b>Incapacidade de reconhecer um botão para o formulário</b>	Link representando botão que não pode ser reconhecido. Para corrigir este problema é necessário a adição, no arquivo de configuração, no campo "class_a_botao", do nome do atributo <i>class</i> que o link possui.

**Tabela 3 - Falhas**

<b>FALHAS</b>		
(Problemas referentes à implementação)		
1	<b>Incapacidade de reconhecer componentes representados por imagens</b>	Uma das páginas analisadas apresentou um formulário formado por vinte e sete componentes, mas a aplicação foi capaz de reconhecer somente nove destes elementos. O motivo pelo qual isto aconteceu foi o uso de uma imagem para representar um <i>RADIO BUTTON</i> . A página foi construída usando imagens de <i>RADIO BUTTONS</i> e quando se efetua um clique sobre o elemento, a imagem muda para uma imagem de um <i>RADIO BUTTON</i> no estado de selecionado. É uma maneira totalmente ineficiente de implementação, mas como já foi dito anteriormente, existem muitas maneiras de se chegar ao mesmo resultado quando se trata de implementação de páginas na <i>web</i> . O Ifrit não é capaz de reconhecer elementos representados por imagens. Apenas botões podem ser reconhecidos, e isso se possuírem ação de clique cadastrada.

2	<b>Incapacidade de reconhecer mais do que um formulário por página</b>	Algumas páginas possuem mais do que um formulário e o Ifrit não está apto a tratar este tipo de situação. Isso acontece porque o <i>crawler</i> procura pelo formulário a medida que vai navegando pelos nós da árvore HTML e ele para o processo assim que encontra um formulário.
3	<b>Incapacidade de tratar elementos que não tem rótulo associado, ou que dividem o mesmo rótulo</b>	Alguns elementos realmente não tem um rótulo associado ou as vezes dividem o rótulo com outros elementos. O Ifrit não consegue tratar este tipo de situação.

#### 4.5.5 COMPARAÇÃO

Nesta subsecção é apresentada uma comparação entre os *crawlers* referenciados nos trabalhos relacionados e o Ifrit.

Tabela 4 - Comparação entre o Ifrit e outros web crawlers

#	CARACTERÍSTICAS	Ifrit	DeepBot	LabelEx
1	Executa JavaScript	Sim	Sim	Sim
2	Construído a partir de um mini web <i>browser</i>	Sim	Sim	Sim
3	Permite escolher a implementação do <i>browser</i> que será usado para navegar nas páginas	Sim	Não	Não
4	Permite a definição de domínios específicos	Não	Sim	Sim
5	Armazena informações de sessão e <i>cookies</i>	Não	Sim	Não
6	Trata a página sem que seja preciso fazer o <i>download</i> do conteúdo da página	Sim	Não	Não
7	Considera formulários que não estão dentro da tag <code>&lt;form&gt;</code>	Sim	Não	Não
8	Determina relação entre campos do formulário e rótulos	Sim	Sim	Sim
9	Utiliza informação visual da página para encontrar o relacionamento entre componentes	Não	Sim	Sim

	e rótulos			
10	Consegue tratar casos em que um campo não tem um rótulo associado	Não	Não	Sim
11	Auto-aprendizagem	Não	Não	Sim
12	Abordagem simples	Sim	Não	Não

Os pontos analisados são os itens 3, 6, 7 e 12 que se referem às características que destacam o Ifrit dos outros trabalhos analisados. O item 3 se refere à capacidade de escolher (dentro de um número fechado de opções) a implementação do browser que será usada para abrir o conteúdo das páginas que o Ifrit visitar. Esta característica é importante pois cada browser tem a sua implementação da especificação do HTML, então algumas páginas podem apresentar um comportamento diferente dependendo do browser usado na navegação. O Ifrit não faz o download do conteúdo das páginas que visita, toda a análise é feita em tempo de execução, então não é necessário ter espaço em disco para realizar varreduras na web. Como as páginas da web mudam frequentemente, estamos considerando esta característica do Ifrit como sendo uma vantagem.

O item 7 pode ser considerado o mais importante de todos pois foi a motivação inicial para a realização deste trabalho. Ao invés de apenas procurar por *tags* do HTML, como fazem os *crawlers* comuns, o Ifrit foi construído para analisar o conteúdo das páginas que visita e a partir de uma heurística, definir a existência de formulários. Desta forma, não se fica preso a uma única configuração HTML, visto que existem muitas maneiras diferentes de se implementar a mesma página HTML.

O item 12 foi considerado uma vantagem, pois o trabalho foi construído em um tempo muito curto e com uma lógica mais simples das que haviam sido empregadas até então, e mesmo assim, foi obtido um bom resultado na análise das páginas HTML.

Já os itens 9, 10 e 11 podem ser considerados os pontos a serem melhorados. Sobre o item 9, a partir da análise visual do posicionamento dos elementos é possível afinar o resultado do processo de descoberta de

formulários. Sobre o item 10, um número considerável de páginas apresentam elementos que não tem um rótulo associado, então tratar estes casos seria uma boa melhoria. O item 12 auto-aprendizagem, ajuda muito a longo prazo. As tendências na construção de páginas vão sendo alteradas, e se o *crawler* não for capaz de aprender os novos padrões utilizados será necessário fazer ajustes na lógica da aplicação.

## 5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este trabalho abordou a questão do reconhecimento de formulários em páginas web, seus campos e respectivos rótulos. Para resolver o problema proposto - identificação de formulários e seus rótulos em páginas web - foi desenvolvido um *Web Crawler*, batizado de Ifrit, que é capaz de navegar de uma página para outra a procura de formulários web. Dois grandes problemas foram tratados neste trabalho: (i) o reconhecimento de formulários construídos sem o uso da *tag* FORM do HTML e (ii) a extração dos rótulos associados aos elementos do formulário. Ao todo, foram cinco os objetivos deste trabalho, a seguir são feitas considerações sobre cada item.

1. Definição de uma heurística para detecção de *Web Forms* construídos sem a *tag* FORM do HTML:

Através da análise dos elementos de uma página HTML, foi possível desenvolver uma heurística que consegue inferir a existência de formulários web em páginas na Internet. O algoritmo percorre todos os elementos da página que se deseja analisar, monta uma árvore de elementos, atribuí um identificador para cada elemento (Numeração *DEWEY*" (Tatarinov, 2002)) e através da análise da distância e da disposição dos elementos HTML, infere sobre a existência de um formulário.

2. Definição de uma heurística para extração dos rótulos relacionados aos campos dos formulários encontrados:

Com o uso da mesma árvore de elementos criada pelo primeiro algoritmo, encontra-se os possíveis rótulos contidos na página. Uma lista com os possíveis rótulos e uma segunda lista com os elementos do formulário são as entradas para o segundo algoritmo. Este por sua vez, forma pares de elementos e rótulos, levando em consideração a distância entre esses elementos. Desta forma foi possível extrair os rótulos dos componentes de formulários web.

3. Implementação de um *Crawler* focado e das heurísticas previamente definidas:

Tendo as heurísticas já definidas, foi possível desenvolver o *Crawler* focado Ifrit. A aplicação foi desenvolvida com a linguagem Java e o uso de bibliotecas de tratamento de códigos HTML. O *Crawler* faz o trabalho de navegar entre diferentes páginas da *web*, verificar a existência de formulários e fazer a extração dos rótulos.

4. Realização de experimentos para testar a abordagem definida:

Os experimentos foram realizados com base em métricas padronizadas de Recuperação de Informação: Precisão, Revocação e Medida-F (**Rijsbergen, 1975**). Cinquenta páginas diferentes, contendo formulários, foram usadas nos experimentos com o *crawler*. O Ifrit apresentou um resultado satisfatório. No geral, houveram 70% de acerto, tanto no reconhecimento de campos de formulários quanto na associação entre campos e rótulos.

5. Criação do *script SQL* para inserir em um banco de dados já existente as informações coletadas pelo *Crawler*:

O *crawler* foi desenvolvido para conectar um banco de dados e inserir nele as informações sobre os formulários encontrados, entre elas: as URL's das páginas, os campos dos formulários e os rótulos.

A seguir é apresentado um resumo, em forma de itens, das contribuições deste trabalho e dos pontos que podem ser melhorados.

### **Contribuições:**

1. **Crawler para coleta de formulários:** Desenvolvimento do *crawler* focado Ifrit, que é capaz de analisar o conteúdo HTML de uma página e checar a existência de formulários web, além de fazer a associação entre os rótulos e os elementos do formulário.
2. **Resultados para o projeto Wf-Sim** - Que tem como objetivo criar um Processador de Consultas por Similaridade para Formulários na Web:



- Possibilidade de encontrar na *web* formulários construídos sem o uso da *tag* FORM do HTML.
- Alimentação da base de dados do projeto com dados sobre os formulários encontrados, e seus campos.

### 3. **Publicação de 2 artigos:**

- L.B. dos Santos, C.F. Dorneles e R.S. Mello. Uma Abordagem para Detecção e Extração de Rótulos em Formulários Web. Simpósio Brasileiro de Banco de Dados. Short Paper, 2012.
- 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.

### **Pontos a serem melhorados:**

1. Permitir o reconhecimento de mais do que um formulário por página.
2. Tratar os elementos que não possuem rótulos associados.
3. Fazer a análise para reconhecimento de um formulário levando em consideração a posição visual dos elementos.

## 6 REFERÊNCIA BIBLIOGRÁFICA

Álvarez, M., Raposo, J., Pan, A., Cacheda, F., Bellas, F., & Carneiro, V. (2007). *Crawling the Content Hidden Behind Web Forms*. Corunha, Espanha.

Apache. (2012). *Apache License*. Retrieved 08 19, 2012, from <http://www.apache.org/licenses/LICENSE-2.0.html>

Bergman, M. K. (2001). *The Deep Web: Surfacing Hidden Value*. *BrightPlanet*.

Cohen, W., Ravikumar, P., & Fienberg, S. (2003). A comparison of String Distance Metrics for Name-Matching Tasks. *In Proceedings of IJCAI-03 Workshop*.

Eclipse Foundation. (n.d.). Retrieved 06 24, 2012, from Eclipse: <http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/indigosr2>

He, H., Meng, W., Yu, C., & Wu, Z. (2004). Automatic extraction of web search interfaces for interface schema integration. *In Proceedings of WWW*.

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

Megan. (n.d.). *deja entendu*. Retrieved 11 05, 2012, from <http://coneyislanddreams.wordpress.com/2011/03/29/crawling-into-the-deep-web/>

Nguyen, H., Nguyen, T., & Freire, J. (2008). *Learning to Extract Form Labels*. Salt Lake City, UT, EUA.

Oracle. (n.d.). *VisualVM download*. Retrieved 06 24, 2012, from <http://visualvm.java.net/download.html>

Pan, A., Raposo, J., Álvarez, M., Hidalgo, J., & Viña, A. (2002). Semi-Automatic Wrapper Haneration for Commercial Web Sources. Corunha, Espanha.

Raghavan, S., & Garcia-Molina, H. (2000, 12). Crawling the Hidden Web. *Technical Report 2000-36* .

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

Sourceforge. (2012). *HtmlUnit*. Retrieved 06 24, 2012, from <http://htmlunit.sourceforge.net/>

Zhang, H. (2004). The Optimality of Naive Bayes. Fredericton, New Brunswick, Canada.

Zhang, Z., He, B., & Chang, K. (2004). Understanding web query interfaces: best-effort parsing with hidden syntax. *In Proceedings of ACM SIGMOD* .

## Anexo A

# FERRAMENTA PARA EXTRAÇÃO DE WEB FORMS

Leonardo Bres dos Santos  
Universidade Federal de Santa Catarina  
leonardobres@gmail.com

**Abstract.** *Deep Web volume continues to increase as well as the interest to discover and extract Web hidden database data and schemata. This is motivated by applications that intend to provide unified search over several Web forms or the hidden content of Web databases. On considering this context, this paper presents an approach for detecting and extracting labels in Web forms. For detecting a Web form, we propose an algorithm that analyzes HTML tags and identifies if a Web page contains a form or not. We also developed an algorithm for label extraction based on the distance between a form field and page labels in order to find out the relationship between them. Some preliminary experiments demonstrate the effectiveness of the developed algorithms.*

**Resumo.** O volume de dados da *Deep Web* permanece aumentando, assim como o interesse pela descoberta e extração de dados e de esquemas de bancos de dados escondidos na Web. Tal fato é motivado pelo surgimento de aplicações cujo objetivo é permitir buscas unificadas sobre diversos formulários na Web e sobre o conteúdo escondido de bancos de dados na Web. Neste contexto, este artigo apresenta uma abordagem para a detecção e a extração de rótulos em formulários Web. Para a detecção de formulários é proposto um algoritmo que analisa *tags* HTML e identifica se uma página Web contém ou não um formulário. Já para o problema da extração dos rótulos, é proposto um algoritmo baseado na distância entre um campo de um formulário e os rótulos da página a fim de encontrar uma associação correta entre eles. Alguns experimentos preliminares demonstram a eficácia dos algoritmos desenvolvidos.

*Categories and Subject Descriptors: H.Information Systems [H.m. Miscellaneous]: Databases*

*General Terms: Dados na web, Deep Web, Extração de informação*

*Keywords: Web form, extração de rótulos*

## 1. INTRODUÇÃO

Uma imensa quantidade de dados digitais encontra-se disponível atualmente em milhões de bancos de dados na Web em diversos domínios do conhecimento, como venda de passagens aéreas, livrarias virtuais e compra/venda de veículos. O acesso a estes bancos de dados é possível somente através de formulários presentes em páginas Web, também chamados de "pontos de entrada" para o banco de dados. Estes formulários exibem alguns atributos do banco de dados (campos do formulário) sobre os quais o usuário especifica filtros e então submete consultas ao banco de dados. Esses bancos de dados na Web são denominados "bancos de dados escondidos" (ou *Deep Web*), uma vez que o seu esquema e o seu conteúdo não estão completamente visíveis ao usuário [Madhavan et al. 2009].

A disponibilidade crescente de bancos de dados escondidos na Web tem motivado a comunidade de pesquisa em Banco de Dados a desenvolver soluções que permitam o gerenciamento desses dados com vistas principalmente a atividades de integração e busca, como por exemplo, encontrar Web sites que ofereçam veículos para venda e que me permitam definir consultas integradas a vários sites por determinadas marcas e modelos. Uma problemática neste contexto é a descoberta e a posterior extração de rótulos. Rótulos representam a identificação de atributos (nomenclatura de campos) presentes em formulários de acesso a um banco de dados escondido. O tratamento desta problemática é bastante relevante, pois serve de base para o reconhecimento do esquema de bancos escondidos e para viabilizar subseqüentes mecanismos de busca. Trabalhos relacionados populares na literatura tratam essa questão através de abordagens consideradas complexas em termos de processamento e/ou conhecimento sobre o domínio requerido para a extração [Nguyen et al. 2008; Álvarez et al. 2007; Raghavan and Garcia-Molina 2001].

Este artigo apresenta Ibrit, uma abordagem simples, porém efetiva, para detecção e extração automática de rótulos centrada na análise estrutural do código HTML presente em páginas de formulários Web. Esta análise se baseia em um esquema de numeração para a hierarquia de *tags* HTML e um algoritmo que associa componentes de texto a campos do formulário, garantindo uma descoberta eficiente de rótulos presentes nestes formulários. Experimentos preliminares comprovam a boa acurácia da abordagem.

O restante deste artigo está organizado conforme segue. A seção 2 descreve sucintamente os principais trabalhos relacionados e suas limitações. A seção 3 detalha a abordagem proposta. A seção 4 apresenta os resultados experimentais com a execução do Ifrit e a seção 5 é dedicada às considerações finais.

## 2. TRABALHOS RELACIONADOS

Algumas abordagens se propõem a extrair rótulos de formulários Web. LabelEx [Nguyen et al. 2008] é um *crawler* desenvolvido para resolver o problema da recuperação do conteúdo em bancos de dados escondidos. A proposta se baseia em uma técnica de aprendizado que extrai automaticamente os rótulos dos atributos contidos nos formulários. O problema chave é a identificação correta de um rótulo, pois existem inúmeras maneiras diferentes de posicionar um rótulo no espaço de definição de um atributo em um formulário Web. Para sanar este problema, LabelEx tenta aprender novos layouts de formulários e então extrair rótulos com comportamentos similares em termos de representação. Mesmo assim, a técnica não garante alta acurácia, pois depende de boas amostras de páginas para fins de aprendizado.

A ferramenta DeepBot [Álvarez et al. 2007] é um Web *crawler* focado em domínios específicos. A definição do domínio é pré-definida e composta basicamente por uma lista de atributos, cada um associado a um nome, pseudônimos e um limiar relevante. Os pseudônimos representam as possíveis variações para o nome do atributo e o limiar representa a importância que o atributo possui no domínio. No processo de extração, a ferramenta tenta combinar os atributos com os campos do formulário usando distância visual ou similaridade textual. A partir disso, é possível determinar se o formulário encontrado é relevante ao domínio. Sua principal deficiência, entretanto, é a dependência (criação e manutenção) desta base de conhecimento por domínio.

A abordagem descrita em [Raghavan and Garcia-Molina 2001], assim como o DeepBot, também efetua processamento do layout para calcular a distância de pixels entre o campo e os rótulos candidatos para fins de extração. O problema aqui é a complexidade do processo baseado em distância visual por pixels, que geralmente é alta.

## 3. IFRIT

Esta seção apresenta o extrator Ifrit. Ifrit adota uma abordagem mais simples, porém efetiva, que as existentes nos principais trabalhos relacionados, tratando dois problemas: (i) a detecção dos formulários encontrados em páginas Web; e (ii) a associação entre campos dos formulários e seus respectivos rótulos.

### 3.1 Visão Geral

A Figura 1 apresenta uma visão geral dos módulos que compoem a arquitetura do Ifrit. A API é o módulo mais importante, pois toda a lógica de reconhecimento e extração de formulários e rótulos está implementada ali. Para que o módulo API funcione, é necessário que ele receba como entrada uma URL e as configurações feitas pelo usuário no arquivo API.txt. Para realizar a navegação entre as páginas e a extração das URL's que irão abastecer a API, foi desenvolvido o módulo Crawler, que implementa a lógica de um Web crawler, além da interface com o usuário.

O *Crawler* recebe como entrada as configurações feitas no arquivo Crawler.txt, que basicamente mantém a quantidade total de páginas a ser visitada e o browser que deve ser usado para acessar as páginas. Por `_m`, o módulo DAO é responsável pela conexão com o banco de dados e inserção das informações coletadas no processo. O foco deste artigo é no módulo API. As seções a seguir detalham, respectivamente, os processos de detecção de formulários e extração de rótulos.

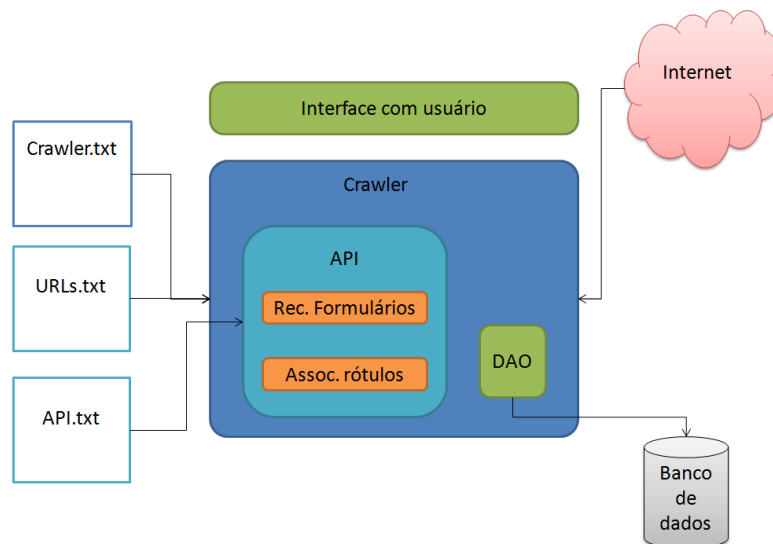


Fig. 1. Visão geral do funcionamento do Ifrit

### 3.2 Detecção de Formulários

O processo de detecção de formulários leva em conta os seguintes conceitos:

- Elemento: indica qualquer elemento suportado pelo HTML;
- Componente: elemento que representa um campo do formulário, como um *Checkbox*, *Textbox* e *Combobox* ;
- Rótulo: texto associado a um componente do formulário;
- Container : elemento HTML que agrega outros elementos, como um *Div*.

Para resolver o problema da descoberta de um formulário em uma página qualquer da Web, foi necessário definir as características que o código HTML deve apresentar para que seja possível inferir de maneira segura a existência de um formulário. O primeiro passo no desenvolvimento da proposta foi encontrar as peculiaridades de um formulário. Para isso, um algoritmo foi desenvolvido com o uso de parâmetros configuráveis, descritos a seguir. Estes parâmetros podem ser modificados de acordo com a flexibilidade que se deseja permitir na busca:

- forma de submissão do formulário: identifica o elemento responsável pela submissão das informações contidas no formulário;
- distância entre os elementos: considera a existência de mais de um tipo de configuração referente à distância suportada pelo algoritmo. Essas configurações delimitam a distância máxima permitida entre os elementos e a densidade de elementos contidos em um formulário;
- elementos visíveis e não visíveis: trata elementos não visíveis para o usuário. Esta configuração foi criada para dar mais flexibilidade ao *crawler*, pois existem algumas páginas que contêm elementos que só são mostrados quando o usuário executa uma determinada ação. Enfatiza-se que os elementos tratados por esta configuração são aqueles que já estão carregados na página, mas não estão sendo mostrados. Um exemplo disso são formulários que permitem uma busca simples e uma busca avançada.

Uma das principais características de um formulário é a forma de submissão dos dados preenchidos, que normalmente é feita através de botões. Existem muitas maneiras diferentes de se construir um elemento que representa um botão. Assim sendo, foi necessário detectar essas diferentes construções e classificá-las como sendo o elemento responsável pela submissão do formulário. Algumas páginas utilizam uma imagem que suporta um evento de clique, outras utilizam um elemento do tipo link e outras utilizam apenas elementos do tipo *Div*. Todas estas construções são consideradas.

Outra característica importante é a distância entre os elementos que compõem um formulário. Para este quesito, foram definidos dois parâmetros: (i) a distância máxima entre cada elemento do formulário e o elemento que representa o container ; e (ii) a densidade máxima dos componentes de um formulário, que é calculada pela distância de um elemento do formulário e o próximo elemento do formulário, que esteja acima ou abaixo deste. Esta definição de distância é possível, pois um documento HTML pode ser tratado como um grafo que possui nodos (elementos) e arestas (ligações hierárquicas entre os elementos). Desta forma, a distância é o número de arestas entre um elemento e outro, e um elemento que não respeite o limite imposto não é considerado pertencente ao formulário em questão.

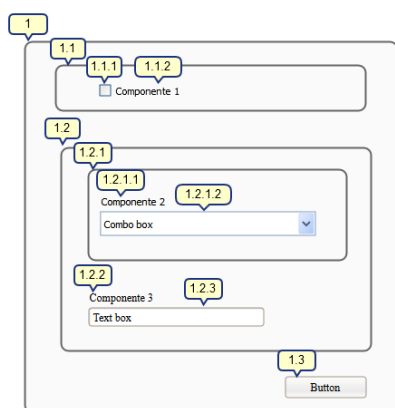
Além disso, é possível escolher entre considerar apenas os elementos que estão visíveis na página ou considerar todos os componentes (visíveis e invisíveis). Componentes invisíveis são aqueles que estão no código HTML mas não estão sendo apresentados, por exemplo.

O primeiro passo executado pelo algoritmo é a construção de uma árvore de elementos que mantém a mesma estrutura contida na página. A árvore de elementos serve como base para que o restante do processamento seja feito, ou seja, o código HTML de uma página é organizado de forma que os elementos formem uma árvore com os seus diferentes níveis.

A medida que a árvore vai sendo construída, cada novo elemento adicionado à estrutura recebe um ID, que o acompanha pelo resto do processamento, utilizando o esquema de numeração Dewey [Tatarinov et al. 2002]. Este esquema permite não apenas identificar o elemento, mas também para informar sua localização na árvore, uma vez que o ID do elemento pai é usado para compor o ID do elemento \_lho. A Figura 2 mostra um exemplo da utilização deste conceito. O algoritmo que verifica a existência de um formulário faz, basicamente, uma varredura pela árvore de elementos e uma série de testes para saber se as configurações apresentadas anteriormente estão sendo atendidas. O processamento é feito de maneira recursiva para cada nodo contido na árvore.

### 3.3 Extração de Rótulos

O processo de extração de rótulos faz uma verificação com base nos ID's dos componentes e dos rótulos para determinar se é possível associar os elementos. A entrada para este processamento são basicamente duas listas de elementos: uma representando os componentes do formulário e a outra representando os possíveis rótulos. Nesta etapa, basicamente se deseja encontrar, para cada componente do formulário, um rótulo correspondente. O algoritmo tenta encontrar o rótulo que está mais próximo de determinado elemento para então proceder a associação. Este cálculo de distância é feito utilizando a mesma ideia do algoritmo construído para reconhecimento de formulários, com algumas particularidades envolvidas no cálculo, explicadas a seguir.



- Elemento:**
- CheckBox 1.1.1
  - ComboBox 1.2.1.2
  - TextBox 1.2.3
  - Button 1.3

- Texto:**
- Componente1 1.1.2
  - Componente2 1.2.1.1
  - Componente3 1.2.2

Fig. 2. Exemplo de formulário e uso da numeração Dewey



O algoritmo usado no módulo de inferência de rótulo trata basicamente com os ID's dos elementos. Ele define pares com os elementos - rótulos e componentes do formulário - que estão mais próximos uns dos outros. A Figura 2 apresenta um exemplo do processo de extração e associação dos rótulos contidos em um formulário. Do lado esquerdo pode-se ver um pequeno formulário e do lado direito sua representação aplicando a numeração DEWEY após o processo de criação de árvore de elementos explicada anteriormente. É possível notar que cada elemento do formulário recebe uma identificação e que esta identificação por si só é capaz de dizer a posição (em relação aos níveis da árvore) de cada elemento. Este esquema de identificação é muito importante para o resto do processamento, pois possibilita, de uma maneira simples, a execução do cálculo da distância entre os elementos.

Junto com o processo de construção da árvore de elementos, todos os elementos da página são visitados em uma busca em profundidade, ou seja, os elementos são visitados na mesma ordem de seus ID's. Cada vez que o processo de busca volta para um nodo já visitado significa que já se tem a informação dos componentes \_lhos do nó atual. Neste ponto, é possível fazer os testes para tentar reconhecer um formulário. Isto significa que a medida que os elementos vão sendo visitados pelo mecanismo de busca, testes são realizados para saber se, com as informações encontradas até o momento, é possível reconhecer um formulário. Em suma, todos os elementos que estiverem respeitando as regras impostas na configuração são adicionados à lista de elementos válidos. O mesmo raciocínio é aplicado a elementos de texto. No exemplo da Figura 2, são encontrados quatro elementos de formulário (ID's: 1.1.2, 1.2.1.2, 1.2.3 e 1.3) e três elementos de texto (ID's: 1.1.2, 1.2.1.1 e 1.2.2).

O primeiro passo executado pelo processo de extração visa preparar as informações que serão usadas no decorrer do processo. Para encontrar os pares de elementos, é necessário conhecer a distância entre todos os elementos envolvidos. Assim sendo, a primeira tarefa do algoritmo é construir uma matriz que fornece a distância de cada componente em relação a cada rótulo. A Tabela I apresenta a matriz gerada para o exemplo da Figura 2.

Através da matriz de distâncias é possível encontrar os menores valores de distância e então associar os elementos relacionados. A Tabela I mostra que as menores distâncias estão presentes na diagonal principal da matriz, ou seja, pode-se observar que o algoritmo associa o "Checkbox 1.1.1" com o rótulo "Componente1 1.1.2" (distância igual a 1), o "ComboBox 1.2.1.2" com o rótulo "Componente2 1.2.1.1" (distância igual a 0) e o "TextBox 1.2.3" com o rótulo "Componente3 1.2.2." (distância igual a 0). Observando novamente a Figura 2, é possível perceber que a associação foi bem sucedida, ou seja, cada componente recebeu o rótulo correto.

Table I. Matriz do cálculo de distância entre componentes de rótulos

	Checkbox 1.1.1	Combobox 1.2.1.2	Textbox 1.2.3
Componente 1	1	-1.1.2	-1.1
Componente 2	1.1	0	-2.1
Componente 3	1.1	1.2	0

#### 4. EXPERIMENTOS PRELIMINARES

Experimentos preliminares foram realizados a \_m de comprovar a qualidade da abordagem de extração de formulários Web proposta. Para tanto, foram selecionadas e analisadas cinquenta páginas de cinco domínios diferentes: aluguel/compra de veículos, reserva de hotéis, compra de passagens aéreas e busca por empregos. Como algumas das páginas se incluem em mais de um desses domínios, foi criada mais uma categoria (viagens), que contempla páginas que oferecem o serviço de reserva de hotel, aluguel de veículos e compra de passagens aéreas.

Após executar o processo de extração, os resultados obtidos foram comparados com os dados resultantes da avaliação manual. Para facilitar a análise manual das páginas, foram ignorados os componentes que não estavam visíveis na página. Para avaliar os dados obtidos,

utilizou-se as métricas clássicas Precisão (*precision*), Revocação (*recall*) e Medida-F (*F-measure*), cujos valores são apresentados na Figura 3 considerando a totalidade dos dados.

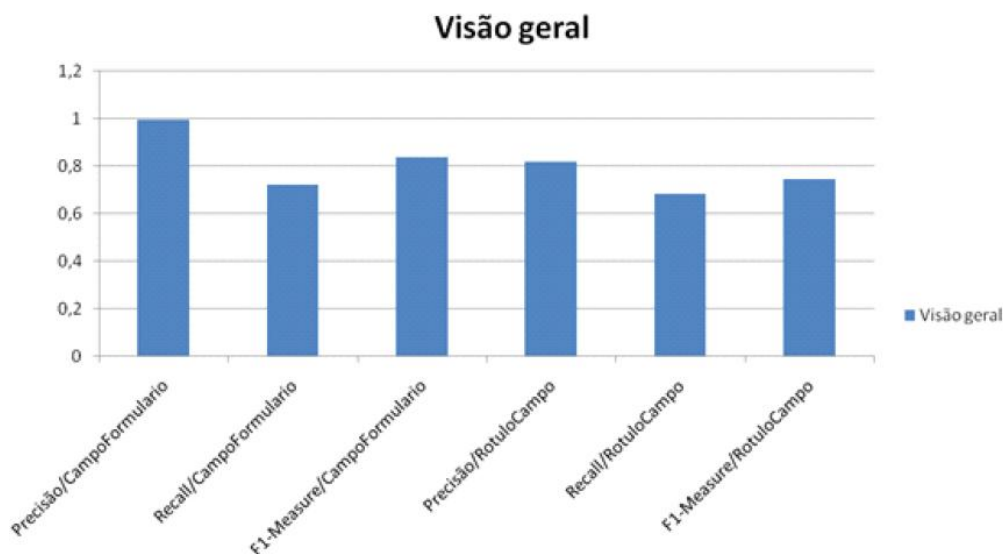


Fig. 3. Visão geral dos resultados dos experimentos

A conclusão é que Ibrit apresentou bons resultados na execução dos testes, ou seja, quase 100% dos componentes reconhecidos pela aplicação realmente faziam parte do formulário. Já o reconhecimento dos rótulos teve um desempenho inferior, pois o grande número de possibilidades de se apresentar um formulário torna muito difícil encontrar um padrão que sirva para todos os casos. Mesmo assim, a Medida-F foi superior a 70%, não caracterizando um resultado muito negativo.

## 5. CONCLUSÃO

Este artigo apresenta uma proposta para detecção e extração de rótulos presentes em formulários Web, motivado pela demanda crescente de soluções para o gerenciamento de bancos de dados escondidos na Web e a carência de abordagens simples e efetivas para a realização de tais atividades. Com a execução dos experimentos, foi possível confirmar a grande heterogeneidade de implementações e estilos de apresentação dos componentes de um formulário Web, constatando-se a grande dificuldade na elaboração de uma proposta geral, que funcione para todos os casos.

As principais dificuldades encontradas nas páginas utilizadas nos experimentos foram: (i) formulários com menos de 3 componentes, que são fontes de dados de acesso a Deep Web, e não a formulários de *login*; (ii) reconhecimento da forma de submissão de formulários; e (iii) a presença de imagens que substituem rótulos de componentes. Como trabalhos futuros, pretende-se fazer melhorias nos algoritmos de forma a resolver esses problemas detectados. Além disso, pretende-se realizar experimentos com maior volume de páginas Web e também experimentos comparativos com o extrator do sistema DeepPeep [Barbosa et al. 2010] a fim de avaliar qualidade de resultados e desempenho.

## REFERENCES

- Álvarez, M., Raposo, J., Pan, A., Cacheda, F., Bellas, F., and Carneiro, V. Crawling the content hidden behind web forms. In Proceedings of the 2007 international conference on Computational science and Its applications - Volume Part II. ICCSA'07. Springer-Verlag, Berlin, Heidelberg, pp. 322\_333, 2007.
- Barbosa, L., Nguyen, H., Nguyen, T. H., Pinnamaneni, R., and Freire, J. Creating and exploring web form repositories. In SIGMOD Conference. pp. 1175\_1178, 2010.
- Madhavan, J., Afanasiev, L., Antova, L., and Halevy, A. Y. Harnessing the deep web: Present and future. In CIDR, 2009.

Nguyen, H., Nguyen, T., and Freire, J. Learning to extract form labels. *Proc. VLDB Endow.* 1 (1): 684\_694, Aug., 2008.

Raghavan, S. and Garcia-Molina, H. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases. VLDB '01.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 129\_138, 2001.

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.