Anotador automático de texto

Autor:	Josep Massagués Vidal, Estudante 702215 na Universidade Aberta
Orientador:	Dr. Vitor Rocio, Professor auxiliar na Universidade Aberta
Data de finalização:	2010-05-14

Tabela de conteúdos

Tabela de ilustrações

Agradecimentos

Não quero deixar passar a oportunidade para agradecer a ajuda indispensável do orientador do projecto, o Dr. Vitor Rocio, com indicações claras e respostas atempadas.

Também quero agradecer aos restantes professores pela contribuição directa ou indirecta para conseguir completar este projecto.

E “last, but not least”, agradecer aos meus colegas de curso que ofereceram sempre o seu apoio e ajuda.

Capítulo 1 - Objectivos

1.1 - Introdução

Neste capítulo são introduzidos os objectivos do projecto tal e como foram propostos inicialmente.

1.2 - Objectivos

O objectivo do projecto é construir um anotador automático de texto que identifique entidades nomeadas ("named entities") que não serão mais que os termos definidos na Wikipédia. Por exemplo, pegando num excerto das notícias de hoje do google, o texto devia vir anotado assim:

> Em vésperas de se assinalar o <ne type="event">Dia Internacional da Mulher</ne>, <ne type="person">Kathryn Bigelow</ne> tornou-se a primeira mulher a receber a estatueta de ouro do Festival Internacional de <ne type="location">Hollywood</ne>, vulgarmente conhecido por Óscar pela direcção do filme "<ne type="work">Estado de Guerra</ne>" que ...

1.3 - Entidades Nomeadas

Antes de continuar, é relevante fazer uma breve explicação do que são as entidades nomeadas e a sua importância.

As entidades nomeadas fazem parte dum esforço para a compreensão do texto dentro dos estudos para o processamento da linguagem natural. O seu propósito é etiquetar e classificar de forma objectiva fragmentos de texto. Representam um primeiro passo na criação de sistemas de conhecimento automáticos, que permitam elaborar consultas e extrair informação inteligente e estruturada a partir de informação em linguagem natural. Um dos esforços neste sentido é o de a Web Semântica, proposta e impulsionada por Tim Berners-Lee dentro do World Wide Web Consortium.

Uma explicação mais elaborada sobre o que são as entidades nomeadas, a sua história e futuro está disponível no relatório “Named Entity: History and Future”, por Satoshi Sekine da New York University[1].

A organização mais simples e popular para as entidades nomeadas consiste nos seguintes tipos gerais: Organização, Pessoa, Data, Local, Percentagem, Dinheiro, etc. No entanto, esta classificação é excessivamente simples e é possível estendê-la ou adicionar outros sub-tipos para refinar ainda com mais detalhe a informação. Algumas das classificações de entidades nomeadas propostas estão disponíveis em diferentes sítios Web[2].

Também se encontram disponíveis algumas versões de demonstração dos programas desenvolvidos[3].

1.4 - Conclusão

Foram apresentados os objectivos da proposta e alguma informação de fundo relativamente às entidades nomeadas que representam o núcleo da proposta.

Os objectivos foram interpretados como a criação da capacidade de processar um artigo e identificar termos ou blocos de palavras com recurso à Wikipédia de forma a identificar as categorias a que correspondem.

Partindo destes objectivos genéricos, procedeu-se a elaborar uma lista mais formal de requisitos.

[1] http://cs.nyu.edu/~sekine/papers/NEsurvey200402.pdf
[2] http://www.ldc.upenn.edu/Catalog/docs/LDC2005T33/BBN-Types-Subtypes.html
http://nlp.cs.nyu.edu/ene/version6_1_0eng.html

[3] http://l2r.cs.uiuc.edu/~cogcomp/LbjNer.php
http://xldb.di.fc.ul.pt/Rembrandt/?do=sendtext
http://lingpipe-demos.com:8080/lingpipe-demos/ne_en_news_muc6/textInput.html

Capítulo 2 - Requisitos

2.1 - Introdução

São enumerados e comentados os requisitos que se podem deduzir dos objectivos propostos. Os requisitos são divididos em funcionais e não funcionais. Os primeiros determinam as funções que a aplicação deverá fornecer. Os segundos referem-se a restrições que a aplicação deverá respeitar.

2.2 - Requisitos

2.2.1 Requisitos funcionais

1. A aplicação deve identificar entidades nomeadas (named entities) em notícias.
2. A identificação das entidades nomeadas será feito com ajuda dos recursos disponíveis na Wikipédia em língua portuguesa (http://pt.wikipedia.org/wiki).
3. O conceito de termo pode referir-se a um conjunto de palavras.
4. Para não sobrecarregar os resultados, só os termos com maior destaque serão considerados. Neste contexto, serão considerados termos de destaque aqueles que comecem com uma letra maiúscula.
5. O resultado do processo de anotação deverá conter os termos embebidos numa etiqueta que os identifique claramente. Esta etiqueta deverá listar como atributos as categorias da página da Wikipédia onde aparece o termo e o tipo de entidade nomeada.
6. O principal foco é a anotação de artigos. Deverá, portanto, conseguir consultar Feeds (fluxos RSS) de notícias.

2.2.2 Requisitos não funcionais

Não foram estipulados quaisquer requisitos não funcionais.

2.3 - Conclusão

Foram apresentados os principais requisitos que se depreendem dos objectivos propostos para o projecto.

A seguinte fase passa por transformar estes requisitos genéricos em especificações mais concretas, passando assim do domínio do problema ao domínio da solução.

Capítulo 3 - Especificações

3.1 - Introdução

Neste capítulo são enumeradas as especificações mais destacadas quanto às funcionalidades da aplicação.

3.2 - Especificações

1. A aplicação constará de clientes e um servidor num ambiente Web.
2. Deverá ser compatível com os principais navegadores Web actuais.
3. O servidor constará dum servidor Web e um conjunto de scripts em PHP. Opcionalmente poderá ser necessário um servidor de bases de dados MySQL.
4. O cliente constará duma página Web. Para maior interactividade deverá fazer uso de facilidades Javascript (nomeadamente o AJAX).
5. O utilizador deverá poder guardar as suas Feeds.
6. O utilizador deverá, para cada Feed, poder visualizar todos os cabeçalhos dos artigos disponíveis.
7. O utilizador deverá, para cada cabeçalho, poder visualizar o conteúdo do artigo correspondente devidamente anotado.
8. O utilizador deverá poder baixar o artigo como um ficheiro normal e gravá-lo localmente.
9. Um artigo anotado será aquele cujos termos principais estão devidamente identificados com as correspondentes categorias na Wikipédia. A aplicação deverá, portanto, cruzar os termos do artigo com as páginas disponíveis na Wikipédia para conseguir identificar as mesmas.
10. Os termos anotados no artigo estarão identificados da seguinte forma “<span class=”WikiCat NE_<tipo> WikiCat_<categoria1> WikiCat_<categoria2> ...>Termo”, que constitui um marcador HTML válido e que facilita a identificação e processamento com Javascript.

3.3 - Conclusão

Foram listadas as principais especificações da aplicação. A partir delas é já possível

esboçar o desenho da aplicação e elaborar alguns diagramas.

A seguinte fase consiste na elaboração dalguns desses diagramas, análise e desenho da aplicação.

Capítulo 4 - Análise e desenho

4.1 - Introdução

Neste capítulo é feita a análise e o desenho da solução.

4.2 - Análise

A aplicação será desenhada com uma interface simples via Web. Todas as funcionalidades de utilizador serão implementadas com recurso a Javascript e AJAX.

Do lado servidor estarão disponíveis bibliotecas em PHP para fornecer os serviços.

O utilizador tem a possibilidade de escolher os Feeds preferidos, os quais são guardados no servidor como um simples ficheiro de texto.

Também existem as “Stop Words”, termos individuais que devem ser ignorados no processo de anotação, e que são guardados num ficheiro de texto.

É usada uma base de dados MySQL para armazenar algumas tabelas que o projecto Wikipédia disponibiliza.[4]

4.3 - Desenho

4.3.1 Diagrama de casos de utilização

O diagrama de casos de utilização ilustra as principais interacções do utilizador com a aplicação.

[4] Ver capítulo sobre recursos usados para maior informação

Ilustração 4-1: Diagrama de casos de utilização

4.3.2 Diagrama de classes

Neste diagrama de classes são representadas e detalhadas as classes principais e desenvolvidas propositadamente para a aplicação. Existem outras classes mas que fazem parte de pacotes externos, nomeadamente o pacote SimplePie e o pacote simple_html_dom[5].

[5] Mais informação no capítulo sobre recursos

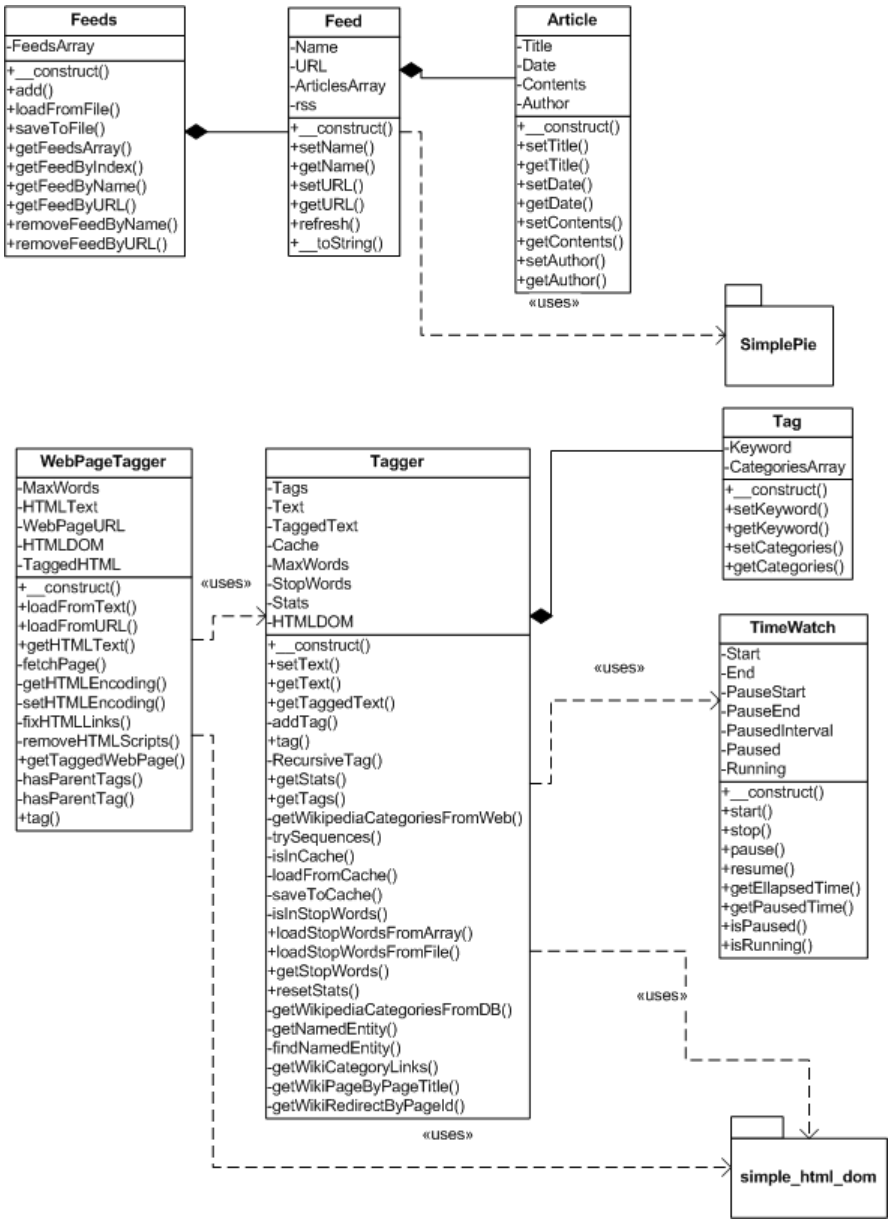

Ilustração 4-2: Diagrama de classes

4.3.3 Diagramas de actividade

Só é mostrado o diagrama da actividade correspondente à anotação duma página Web, e que representa o núcleo do valor da aplicação.

Não são tidos em conta condicionais nem ciclos por uma questão de simplicidade.

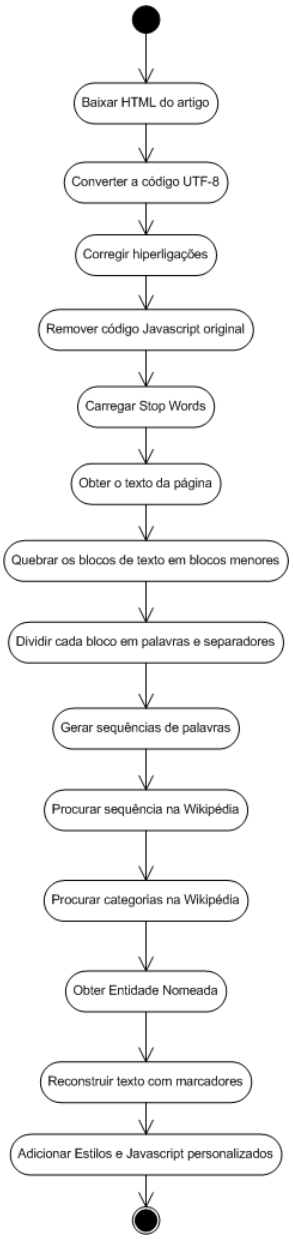

Ilustração 4-3: Diagrama de actividade para etiquetar um artigo

4.3.4 Diagrama de instalação

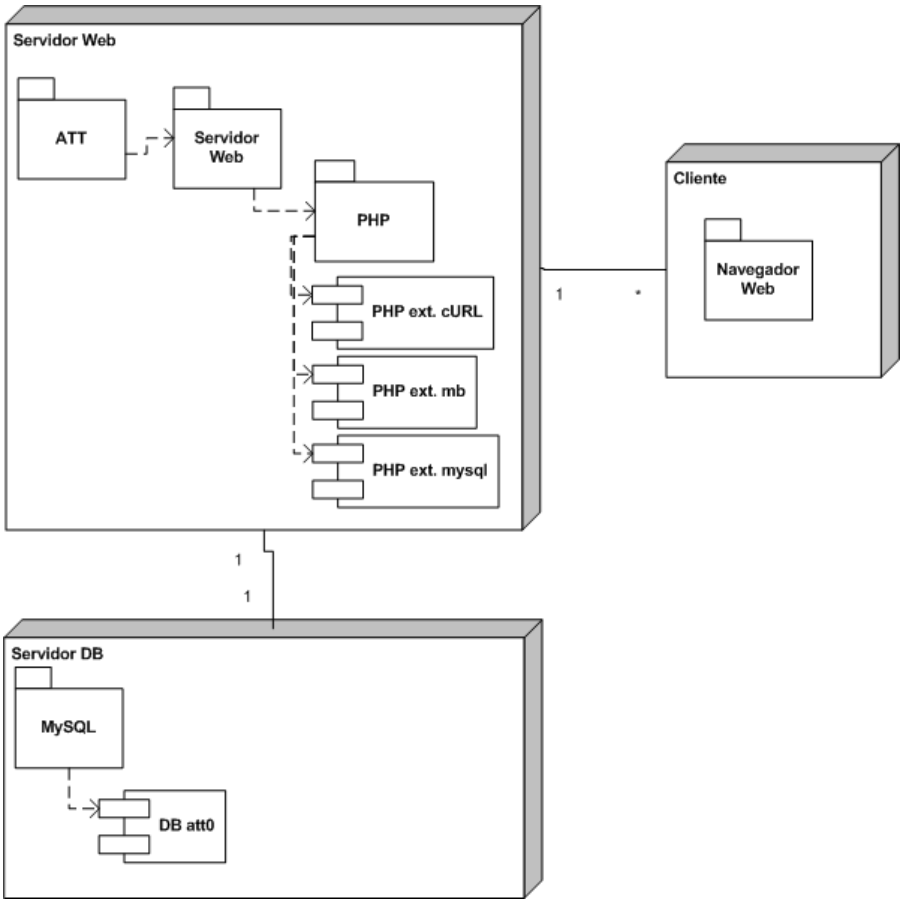

Ilustração 4-4: Diagrama de instalação

4.4 - Explicação da implementação

Nesta secção é feita uma descrição mais detalhada do processo de obtenção das etiquetas com entidades nomeadas a partir de um documento/artigo em HTML.

Cada subsecção tem correspondência com o diagrama de actividade da Ilustração 4-3: Diagrama de actividade para etiquetar um artigo.

4.4.1 Quebrar os blocos de texto em blocos menores

Após obter o artigo em HTML, o método WebPageTagger::tag() executa ainda algumas “limpezas” e chama o método Tagger::tag() para cada bloco de texto reconhecível. São feitas algumas validações para evitar processar texto dentro de certos elementos, como hiperligações, campos de formulários, etc., e focar a análise só no texto que tem mais probabilidades de se corresponder com o corpo do artigo.

```
/**
     * It carries the bulk of operations to generate the tagged text from the plaintext:
     * changing the HTML text's encoding, removing scripts, fixing links and adding
     * some custom styles and javascript.
     *
     * @return bool True if successful
     */
    public function tag() {
…
        // Find all text occurrences and tag their bits with Wikipedia categories
        $Texts = $Body->find('text');

        // Create the tagger object
        $Tagger = new Tagger(null, $this->MaxWords);
        // Load stop words
        $Tagger->loadStopWordsFromFile($CONFIG['DataDir'].DS.'stopwords.txt');
        foreach ($Texts as $Text) {
            // Skip empty text occurrences
            if (trim($Text->text())=='') continue;
            // Skip items with the HREF attribute
            if (isset($Text->href)) continue;
            // Skip items with the SRC attribute
            if (isset($Text->src)) continue;
            // Skip items with the TYPE attribute
            if (isset($Text->type)) continue;
            // Skip text under following tags. Should skip on 'form' as well, but some journals
            // happen to use it ... :-(
            if ($ParentTag=$this->hasParentTags($Text,
                    array('a','script','style','input','textarea','button','select'))!==false) {
                continue;
            }
            // Proceed to tag this text
            $Tagger->setText($Text->text());
            $Tagger->tag();
            $Text->innertext = $Tagger->getTaggedText();
        }
…
```

Ilustração 4-5: Método WebPageTagger::tag()

4.4.2 Dividir cada bloco em palavras e separadores

O método Tagger::tag() invocado anteriormente, essencialmente chama o método Tagger::RecursiveTag(), cuja função é tomar um bloco de texto e dividi-lo recursivamente baseado numa sequência de símbolos.

```
    /**
     * Recursively breaks the text into tokens by symbols.
     * A default sensible array of symbols is already set, and caution should
     * be exercised when using something different. It is suggested that tests
     * are run to verify how adequate they are for each situation.
     * Notice that a blank space is (should be) the last symbol.
     *
     * @param string $Text
     * @param array $Symbols
     * @return string
     */
    private function RecursiveTag( $Text ,
            $Symbols=array( "\r", "\n", "\t", '\'','"', '“', '”','|', '=', '(', ')', '[', ']', '{',
                '}', ',', ';', ':', '.', '!', '?', '\\', '/', ' ')) {
…
```

Ilustração 4-6: Método Tagger::RecursiveTag()

É importante destacar que os símbolos podem ser caracteres individuais ou sequências longas, oferecendo alguma flexibilidade à análise. A ordem em que são declarados os símbolos é importante, e deveriam aparecer, preferencialmente, ordenados do mais externo ao mais interno. A lista usada tem como base as experiências feitas com artigos reais.

Quando o símbolo a usar é o último, é feita uma última divisão do bloco. Se o número de palavras é zero ou a primeira palavra não é capitalizada (o mais comum quando são incluídos nomes próprios ou títulos), o bloco é ignorado. Isto permite limitar significativamente o número de tentativas para identificar entidades nomeadas. Ao mesmo tempo, restringe a universalidade da aplicação. Caso se optar por expandir as entidades potencialmente reconhecíveis, esta parte deveria ser revista. Eventualmente poderia ser retirada qualquer restrição se o sistema tiver uma capacidade de resposta aceitável.

O último símbolo deveria ser sempre um espaço. Este aspecto é contraditório com o facto de permitir a sua inclusão numa lista de símbolos opcionais, mas foi feito para permitir a maior flexibilidade possível. Na realidade poderia ser mais razoável impedir a sua inclusão na lista (ou filtrar as suas ocorrências) e utilizar o espaço após tratar o último símbolo

O processamento dos símbolos pode parecer um tanto complicado e convoluto, mas é preciso ter em conta que o objectivo não é só o de conseguir dividir os blocos como o de conseguir reconstruí-los posteriormente para obter um formato idêntico ao original. Idealmente o processamento deveria ser transparente.

4.4.3 Gerar sequências de palavras e Procurar sequências na Wikipédia

O método Tagger::RecursiveTag() invoca o método Tagger::trySequences() com a lista de palavras disponíveis e o número máximo de palavras a considerar para possíveis entidades nomeadas.

```
    /**
     * Given an array of words, it creates sequences of them
     * (from longest to shortest) and looks for related categories in Wikipedia.
     *
     * @param mixed $TextArray Array of words to use or just a single word
     * @param string $Glue String used to concatenate the words in each sequence
     * @return int Number of words in longest sequence
     */
    private function trySequences( $TextArray, $Glue=' ' ) {
        // If it is not an array, convert it to one
        if (!is_array($TextArray)) {
            $TextArray[0] = $TextArray;
        }

        $Counter = 0 + count($TextArray);

        for ($i=$Counter;$i>0;$i--) {
            // Build a sequence
            $Sequence = implode( $Glue, $TextArray);
            // Find related categories
            if ($this->db===false) {
                $Cats = $this->getWikipediaCategoriesFromWeb($Sequence);
            } else {
                $Cats = $this->getWikipediaCategoriesFromDB($Sequence);
            }
            if ($Cats!='') {
                // Success
                break;
            } else {
                // Remove last word and try again with a shorter sequence
                array_pop($TextArray);
            }
        }
```

Ilustração 4-7: Método Tagger::trySequences()

Se não encontrar nenhuma ocorrência, reduz o número e volta a tentar até ficar sem palavras ou até acertar.

A procura na Wikipédia pode ser feita do site na Internet ou numa base de dados se uma ligação à mesma estiver configurada e disponível.

A geração de sequências e a procura destas na Wikipédia são passos logicamente distintos mas fisicamente englobados no mesmo método.

4.4.4 Procurar categorias na Wikipédia

Como visto no método anterior, existem dois métodos que permitem obter as categorias da Wikipédia: um obtém as categorias directamente do site na Internet e o outro as obtém de uma base de dados.

```
    /**
     * Generates a string tagged with the Wikipedia related categories (if any).
     *
     * @param string $Text Combination of words being proposed
     * @return string Tagged text or empty string
     */
    private function getWikipediaCategoriesFromWeb( $Text ) {
…
```

Ilustração 4-8: Tagger:getWikipediaCategoriesFromWeb()

```
    /**
     * Generates a string tagged with the Wikipedia related categories (if any).
     *
     * @param string $Text Combination of words being proposed
     * @return string Tagged text or empty string
     */
    private function getWikipediaCategoriesFromDB( $Text ) {
…
```

Ilustração 4-9: Tagger::getWikipediaCategoriesFromDB()

Nos dois casos, é retornado o texto devidamente etiquetado com categorias se este existe na Wikipédia.

Existem duas diferenças principais:

- O site da Wikipédia poderá redireccionar a uma outra página. Por exemplo, se procurarmos “Terreiro do Paço” em http://pt.wikipedia.org/wiki/Terreiro_do_Paço, somos redireccionados para uma outra página com “Praça do Comércio” em http://pt.wikipedia.org/wiki/Praça_do_Comércio. Este redireccionamento é feito automaticamente no site, mas tem que ser feito “manualmente” quando se usa a base de dados.
- A identificação de entidades é feita quando a consulta é na própria base de dados e não quando é no site. O tempo de acesso ao site é de 500ms por consulta (aproximadamente), e a identificação de entidades obriga diversas consultas, adicionalmente à obtenção regular de categorias. Os tempos totais podem exceder os vários minutos por artigo, o que torna este processamento muito inconveniente.

4.4.5 Obter entidade nomeada

A tentativa de identificação da entidade nomeada só é feita quando o processo é suportado por uma base de dados.

O método Tagger: findNamedEntity faz uma busca recursiva de alguma categoria que resulte numa entidade nomeada. Começa com uma lista de categorias e tenta encontrar uma que gere um resultado positivo. Se não encontrar nenhum, procura novas categorias ligadas a cada uma das categorias originais.

O método Tagger::getNamedEntity é o responsável por estabelecer uma correspondência entre uma categorias e o nome de uma entidade.

```
    /**
     * Returns the named entitty clasification, by matching the name with a set of possible strings.
     * This is a very weak method and requires lots of hand-tuning to get it right.
     * Unfortunately there is no fixed taxonomy in Wikipedia to guarantee 100% accurate results.
     *
     * @param <type> $Subject A page name as supplied by Wikipedia
     * @return mixed String with the Named Entity Classification (PESSOA,LOCALIZAÇÃO,ORGANIZAÇÃO,
     *          DESPORTO,CULTURA,TECNOLOGIA,ARQUITETURA,CIÊNCIA,GEOGRAFIA,MISC
     */
    private function getNamedEntity( $Subject ) {
        $Response = false;
        $Found = false;

        // PERSON
        if (!$Found) {
            if (!$Found) $Found =(stripos($Subject, 'pessoa')===0);
                …
            if ($Found) $Response = 'PESSOA';
        }

        // LOCATION
```

Cada nome de entidade tem uma série de categorias associadas. A correspondência é simples e directa, comparando a categoria com nomes de categorias ou partes das mesmas pré-determinadas. A comparação de parte do nome é feita para ser o mais abrangente possível, tendo em conta género e número.

Os nomes de entidades usados são aleatórios, existem alguns conjuntos mais comuns que outros, mas não foram detectadas regras fixas quanto às mesmas.

Na literatura disponível são tidas em conta a detecção de datas e quantidades monetárias. Esta detecção seria possível, introduzindo ainda mais critérios e funções especializadas que testassem determinados formatos pré-determinados. No entanto, este esforço deveria ir acompanhado de uma reformulação na divisão do texto em blocos, que está longe de ser simples (seria preciso reformular, entre outras coisas, o uso de símbolos para dividir o texto).

Quanto ao uso da Wikipédia para obter entidades nomeadas, não existe uma taxonomia definida na Wikipédia. No seu lugar, um artigo tem categorias associadas e portais associados (ou outras classificações). Cada categoria pode fazer parte de outras categorias. Para uma descrição mais detalhada dos tipos de páginas possíveis, consultar http://meta.wikimedia.org/wiki/Help:Namespace.

Para obter uma classificação mais rigorosa, seria necessário uma edição manual de uma lista contendo todas as categorias na Wikipédia e da entidade que lhe corresponde, naqueles casos em que a associação pode ser feita com toda a certeza.

Com suficientes recursos, este esforço é possível (na actualidade existem 86.959

categorias entre as 2.206.786 páginas). Aliás, tendo em conta que não é muito provável que o número de categorias mude drasticamente com o tempo, seria razoável fazer um esforço inicial para fazer essa classificação. Cada nova actualização da lista de categorias seria comparada com a anterior e só as diferenças iriam requerer uma manutenção.

Ainda, o número de categorias mais importantes, ou prioritárias, para edição poderia ser significativamente reduzido se a tarefa se limitasse a aquelas categorias que referenciem mais páginas.

4.4.6 Resumo

A revisão dos métodos nucleares do processo mostra que a etiquetagem “ao vivo” de artigos é possível, embora há muito trabalho envolvido e há espaço para muitas alterações e melhorias.

O texto e formatação usados nos artigos de jornal estão longe de ser simples e coerente. Os sinais de pontuação são muito diversos e usados de forma inconsistente. Exemplos das dificuldades:

Pontos (.): podem ser usados para separar frases, mas também podem ser usados em abreviaturas. Exemplo: “D. José”, pode fazer parte de um contexto como “… o desembarco foi no dia D. José encontrava-se no escritório …” ou um outro como “D. José era conhecido na aldeia …”. A mesma sequência de termos é ambígua.

Vírgulas (,): podem ser usadas para separar sequências, mas também podem ser usadas como separadores de decimais. Também podem formar parte de títulos de filmes ou obras de teatro.

Aspas e variantes ("”,’,',«,»): Podem ser usadas para citações, mas também para enquadrar títulos e, no caso dos apóstrofes, podem fazer parte dum nome.

Sinais de interrogação e exclamação (!,?): Podem ser usadas de forma normal ou podem fazer parte de nomes (ex. Yahoo!) e de títulos.

Traços (-): Podem fazer parte de verbos, de palavras compostas e de nomes próprios e títulos.

Foram enumerados só algumas das dificuldades no processamento do texto mais evidentes. Sem criar um programa para processamento de linguagem natural, não há um método garantido para a divisão dum bloco de texto nos seus blocos essenciais e que permitam a correcta delimitação de entidades nomeadas.

Nestas circunstâncias, optou-se pela divisão dum bloco de texto com base num conjunto de caracteres razoável, tal e como indicado na definição do método Tagger::RecursiveTag.

4.5 - Conclusão

Neste capítulo foi apresentada a análise e alguns dos diagramas principais utilizados como estrutura da aplicação.

O seguinte capítulo explica as ferramentas e recursos utilizados na construção da solução.

Capítulo 5 - Recursos e ferramentas usadas

5.1 - Introdução

Neste capítulo são apresentados os diversos recursos externos e ferramentas usadas para completar o projecto.

5.2 - Recursos

5.2.1 Os dumps da Wikipédia

A Wikipédia disponibiliza Dumps de algumas tabelas da sua base de dados. Para as tabelas em português, estes encontram-se disponíveis em http://dumps.wikimedia.org/ptwiki/latest/.

A Wikipédia é um projecto suportado pela plataforma Mediawiki. Um diagrama do layout da sua base de dados está disponível na página http://www.mediawiki.org/wiki/Manual:Database_layout.

A ilustração 5 Layout da base de dados da Mediawiki mostra a parte da imagem que é de interesse para este projecto.

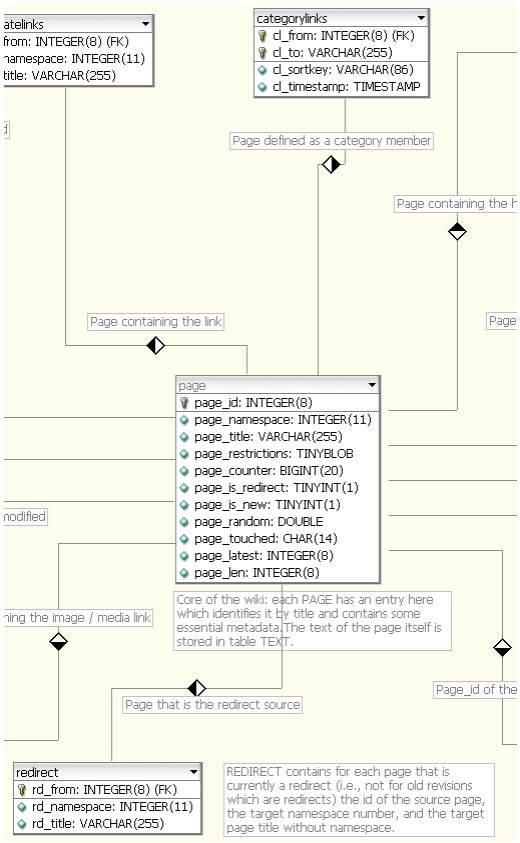

Ilustração 5-1: Layout da base de dados da Mediawiki

Existem 3 tabelas essenciais: page, redirect e categorylinks.

A tabela “page” contém os títulos das páginas. A tabela “redirect” contém as redirecções (um determinado título de página existe, mas é redireccionado para um outro título). A tabela “categorylinks” contém todas as categorias às quais pertence uma determinada página.

Estas tabelas estão disponíveis como “ptwiki-latest-page.sql.gz”, “ptwiki-latest-redirect.sql.gz” e “ptwiki-latest-categorylinks.sql.gz”. É necessário baixar estes ficheiros do site anteriormente referido e importá-los dentro do MySQL.

5.2.2 Biblioteca para ler Feeds

Embora o formato dos Feeds RSS não pareça muito complicado, existem muitas variantes e pontos a ter em conta. Neste caso optou-se por recorrer à biblioteca SimplePie[6].

Esta biblioteca permite aceder aos artigos disponibilizados por um Feed RSS de forma simples e ordenada.

5.2.3 Biblioteca para processar DOM

[6] http://simplepie.org/

Após algumas pesquisas, os resultados rapidamente se concentraram na biblioteca PHP Simple HTML DOM Parser[7].

Esta biblioteca permite aceder a um texto formatado com HTML como uma árvore DOM.

5.2.4 As fontes dos artigos

Foram utilizados os Feeds disponibilizados pelos sites dos seguintes jornais online:

- Jornal de Notícias, http://jn.sapo.pt/paginainicial/
- Diário de Notícias, http://dn.sapo.pt/
- Público, http://publico.pt/
- SOL, http://www.sol.pt/
- Jornal de Negócios, http://www.jornaldenegocios.pt/

Apesar de os Feeds estarem normalizados e existir ferramentas que permitem o seu consumo de forma organizada, tal não existe para os artigos publicados pelos jornais. As páginas disponibilizadas ao consumidor estão cheias de scripts, formulários e tudo aquilo relacionado com a publicidade, ficando os artigos propriamente ditos mal encaixados entre os restantes elementos.

Realmente não há nada que nos possa preparar para este trabalho exceptuando uma metodologia de experiência e erro, e testar o máximo número de fontes para blindar a nossa aplicação contra todas as possíveis variações de formatos e identificar as partes onde o texto relevante se encontra.

5.3 - Ferramentas

No desenvolvimento foi usado um portátil com o sistema operativo Windows 7 Professional 64 bits.

O ambiente de desenvolvimento (para HTML, CSS, Javascript, PHP) foi o Netbeans 6.8[8].

A plataforma usou o servidor Web Apache[9] e o servidor de base de dados MySQL[10], disponibilizados através do pacote XAMPP[11].

[7] http://simplehtmldom.sourceforge.net
[8] http://netbeans.org/
[9] http://httpd.apache.org/
[10] http://www.mysql.com/
[11] http://www.apachefriends.org/en/xampp.html

5.4 - Conclusão

Foram apresentados os diversos recursos e ferramentas usados para o desenvolvimento do projecto.

No seguinte capítulo é indicada a abordagem e metodologia.

Capítulo 6 - Abordagem / metodologia

6.1 - Introdução

Neste capítulo é apresentada a metodologia seguida.

6.2 - Metodologia

Foi seguida uma metodologia ágil, primando a produção do código e documentando o mesmo o melhor possível, enquanto se mantinha o projecto sempre perto dum estado funcional.

Primou a reutilização de código adoptando bibliotecas já prontas e disponíveis livremente.

6.3 - Código de escrita

Foi seguida uma norma de escrita de código em todo o projecto:

- Comentários estilo Javadoc
- Nomes de variáveis capitalizadas
- Nomes de funções capitalizadas, exceptuando quando começa com um verbo
- Por uma questão prática, documentação e nomes de variáveis e funções estão em inglês
- As únicas excepções correspondem ao código escrito nas bibliotecas externas ou a uma biblioteca de utilidades que não faz uso de classes

6.4 - Testes

Foram criados e executados testes unitários da maioria das classes, o que ajudou a identificar erros bem cedo.

No entanto, a maior parte do tempo foi dedicado aos testes de integração e quando confrontado com casos reais.

6.5 - Conclusão

A metodologia escolhida funcionou e ajudou a manter os pés na terra de forma permanente. Isto também ajudou a evitar aventuras que conduzissem a derrapagens de tempo no projecto.

Capítulo 7 - Descrição do trabalho efectuado

7.1 - Introdução

Neste capítulo é explicado como foi organizado o trabalho e algumas considerações adicionais.

É fácil uma pessoa se perder com teorias e desenhos no desenvolvimento, especialmente quando as tecnologias a usar também são novas para nós. A abordagem passou por ler com cuidado o objectivo e conferir os requisitos com o orientador.

A partir dali foram elaboradas as especificações de forma muito geral.

Foram elaborados alguns desenhos iniciais, mas sem muito detalhe, só mesmo um esquema inicial para poder ter uma primeira guia.

Foi colocado muito peso no desenvolvimento rápido, a documentação do próprio código, e a identificação de potenciais problemas o mais cedo possível.

Tudo para conseguir ter sempre um protótipo a funcionar e sobre o qual eram adicionadas novas funcionalidades e corrigidos erros de forma iterativa.

7.2 - Cronograma

A seguinte tabela mostra o cronograma previsto no início do projecto.

Actividade	Início	Fim	Duração (dias)
Levantamento de necessidades concretas junto com o cliente (neste caso o próprio orientador) para garantir que os objectivos e funcionalidades estão bem definidos logo à partida.	16-Mar	21-Mar	6
Análise da estrutura do site Wikipedia, e descrição das fontes de dados pretendidas.	22-Mar	23-Mar	2
Definição das funcionalidades a implementar, com ajuda dos Use Cases em UML.	24-Mar	26-Mar	3
Definição dos objectos principais e as suas interacções, com os respectivos diagramas UML.	27-Mar	31-Mar	5

Definição da estrutura de dados em SQL necessária para suportar todas as funcionalidades pretendidas.	1-Abr	2-Abr	2
Escolha da plataforma e linguagem de programação.	3-Abr	3-Abr	1
Escolha do sistema de gestão de bases de dados.	4-Abr	4-Abr	1
Codificação e realização de testes unitários.	5-Abr	16-Mai	42
Integração e testes finais.	17-Mai	23-Mai	7
Tempo para imprevistos	24-Mai	31-Mai	8
Relatório final do projecto com eventuais “Lessons learned”.	1-Jun	16-Jun	17

No fim, o cronograma real acabou por divergir pouco do previsto, encurtando-o em 2 semanas.

7.3 - Enquadramento nas disciplinas

As principais disciplinas que contribuem ao conhecimento necessário para o desenvolvimento do projecto são as seguintes: Programação, Análise de Sistemas, Programação por Objectos, Fundamentos de Bases de Dados, Desenvolvimento do Software, Sistemas e Serviços Web, Gestão de Projectos Informáticos.

7.4 - Conclusão

Foi feita uma breve descrição do trabalho efectuado e exposto o cronograma seguido.

No último capítulo são tiradas as conclusões finais.

Capítulo 8 - Conclusões

8.1 - Introdução

Neste capítulo final são apresentadas as conclusões e lições aprendidas.

Essencialmente a aplicação responde aos objectivos propostos e foi desenvolvida dentro das datas previstas, embora com um custo pessoal superior ao esperado.

8.2 - Possíveis melhoras

8.2.1 Uso do sistema de anotação originalmente proposto (<ne type="…">…</ne>).

Porque foi escolhido um cliente Web e o recurso ao Javascript para mostrar informações sobre as categorias, optei por usar uma anotação com …. Esta opção poderia não ser necessária.

O Javascript permite criar um elemento "ne" no DOM do documento, de forma que as etiquetas <ne></ne> seriam reconhecidas posteriormente por qualquer ferramenta de manipulação do DOM, mais concretamente a biblioteca jQuery. Assim, "type" seria reconhecido como mais um atributo e seria possível mostrar o seu conteúdo.

Esta melhora permitiria usar uma nomenclatura específica e eventualmente compatível com outras usadas noutros sistemas.

8.2.2 Tempos de processamento muito elevados

Quando não se utiliza uma base de dados, a aplicação ainda consegue trabalhar executando consultas directamente contra o site da Wikipédia (embora nesse caso não é feita qualquer tentativa de identificar o tipo de entidades nomeadas, por causa do custo de tempo associado).

O uso da linguagem PHP do lado servidor, faz que a execução das tarefas seja linear, sem grandes opções para criar *threads* que aumentem o paralelismo de execução. Isto faz que o processo de procura de correspondências de potenciais termos na Wikipédia

seja feito com um pedido de cada vez. As estatísticas recolhidas na aplicação permitem observar que o tempo total de processamento é quase directamente proporcional ao número de pedidos feitos via http, e que o processamento de cada um destes pedidos demora uns 500ms.

Uma possível solução é minimizar o número de pedidos. Isto pode ser feito delimitando ao máximo o texto do artigo que é realmente relevante. Esta aproximação é difícil porque o anotador acaba por ter que lidar com páginas Web completas e muito complexas. Não existe um padrão de publicação entre diferentes agências, assim, alguns colocam o texto relevante do artigo dentro de etiquetas enquanto outros usam etiquetas genéricas ou até inserem o artigo dentro de etiquetas <form></form>. O melhor que se conseguiu fazer foi eliminar blocos a processar num método de trial and error. Por exemplo são ignorados textos dentro de campos <input></input> ou que façam parte duma ligação <a> (embora alguns também inserem hiperligações no meio do texto).

Não sendo possível limitar o âmbito de análise, o seguinte passo foi limitar os termos candidatos. Testes com diferentes critérios rapidamente revelou que a maior parte dos termos relevantes começam com letras maiúsculas, e abrir estes critérios para aceitar letras minúsculas e números aumentava significativamente o número de pedidos a Wikipédia, bem como o número de termos irrelevantes.

Assim sendo, a única alternativa viável é acelerar o processo de obter as categorias da Wikipédia.

A aplicação actual implementa os pedidos directos via HTTP ao servidor da Wikipédia, os quais são feitos sequencialmente. O PHP não oferece facilidades para aplicações *multi-threaded*, embora existam algumas experiências para implementar esta funcionalidade de forma um tanto limitada (ver http://www.alternateinterior.com/2007/05/multi-threading-strategies-in-php.html e http://www.alternateinterior.com/2007/05/communicating-with-threads-in-php.html). Alternativamente, poderia utilizar-se a capacidade da extensão cURL do PHP para executar diferentes pedidos em simultâneo (http://www.php.net/manual/en/function.curl-multi-init.php, http://www.php.net/manual/en/function.curl-multi-exec.php, etc.).

Esta última opção iria requerer uma alteração ao algoritmo de análise do texto, que deveria ser adaptado para o trabalho em paralelo. Isto pode ser conseguido ao preço de criar uma elevada redundância, mas dado que o processamento local é muito inferior ao processamento remoto, provavelmente iria compensar. Esta adaptação passaria por dividir o texto em termos com um determinado número de palavras e gerar todas as possíveis sequências que cumpram os nossos critérios de termos candidatos. Por ex. para um texto de tipo “A B C D E F G H” e dividindo-o em termos de 5 palavras, geraríamos os seguintes termos candidatos: “A B C D E”, “B C D E F”, “C D E F G” e “D E F G H”. Os 4 pedidos seriam lançados simultaneamente a Wikipédia. É claro que se o primeiro termo candidato fosse válido, imediatamente todos os termos seguintes com palavras em comum seriam eliminados.

É preciso notar que o tempo de processamento é proporcional ao número de palavras no artigo e ao número máximo de palavras por entidade: Tempo = [número de palavras por entidade]*[número de palavras por artigo]

Trata-se, portanto, de um tempo polinomial e susceptível de melhoras por optimizações pontuais.

8.2.3 Interface de utilizador

A interface geral poderia ser mais rica e apelativa.

Seria importante incorporar funções que permitissem controlar a actualização da base de dados cada vez que houvesse novas versões das tabelas na Wikipédia.

A identificação das entidades nomeadas poderia ser mais elaborada, ao estilo da aplicação REMBRANDT.

8.2.4 Tipos de entidades nomeadas

O actual código faz uma identificação de entidades nomeadas muito pouco sofisticada: só compara os inícios das categorias com certas palavras e devolve um determinado tipo.

Os tipos usados no código não seguem qualquer padrão, entre outros motivos porque também não me pareceu que houvesse um único padrão aceite pela comunidade, fora dalguns tipos básicos, e porque os tipos devem ser derivados das categorias da Wikipédia.

Uma das dificuldades é o facto de nem todos os artigos estarem identificados com as categorias correctas: uns têm categorias a mais e outros têm categorias a menos.

Exemplo: José Sócrates faz parte da categoria “Políticos de Portugal”, mas “Aníbal Cavaco Silva” ou “Pedro Passos Coelho” não fazem parte dessa mesma categoria, apesar de claramente ser parte dela.

Algumas das categorias são tão genéricas que induzem qualquer processamento em erro.

Para além disto tudo, há muitos termos que nem estão definidos. Por exemplo, o termo “Ministério de Educação” não existe.

Uma melhor identificação poderia ser atingida com uma análise mais pormenorizada ao sistema de categorias, talvez tentando criar uma árvore que permitisse simplificar o processo de associar categorias com tipos, reduzindo essa associação a umas poucas categorias.

8.3 - Lições aprendidas

8.3.1 Incorporar funções de diagnóstico

Nomeadamente, em relação com o ponto anterior relativamente a tempos elevados de processamento, o uso de estatísticas ou funções de diagnóstico para monitorizar o desempenho são importantes para focar os esforços nas áreas críticas.

Durante muita parte do desenvolvimento, fazias as consultas contra o site da Wikipédia, um processo que era muito demorado. Isto foi assim por uma questão de evitar ter que manter uma base de dados e ter que lidar com processos de importação e actualização de tabelas, mantendo a solução o mais simples possível.

As estatísticas recolhidas acabaram por pesar e optei por incluir uma base de dados que acelerou o tempo de processamento de 150-300s para 5-10s.

8.3.2 Uso de sistemas de cache

O uso de um sistema, embora rudimentar, de cache no código, permite reduzir o número de consultas à base de dados ou ao site da Wikipédia.

É especialmente eficiente quando trabalhamos com artigos cujo vocabulário é reduzido e terá, portanto, muitas palavras repetidas.

8.3.3 Novas tecnologias

Um dos principais objectivos quando assumi este projecto foi o de aproveitar para aprender a trabalhar com a biblioteca jQuery e a fazer chamadas AJAX para aumentar as capacidades de interacção duma página Web.

Neste aspecto considero que tive sucesso. Se bem não domino ainda estas tecnologias (e duvido que alguma vez consiga), aprendi a tirar partido delas e compreendi algumas das limitações (especialmente a impossibilidade de carregar uma página externa num iframe … directamente, embora foi possível criando uma outra página que actuava como proxy, e que me permitia manipular a página original).

Chegados a este ponto, acho que a aplicação poderia ter melhor performance se tivesse experimentado outros caminhos, mas esses são os riscos de usar tecnologias que ainda não nos são familiares e de ter um tempo limite para desenvolver o projecto.

8.3.4 Testes, testes, testes

Os diferentes navegadores vão ganhando maior adopção dos padrões W3, mas ainda há diferenças significativas sobre que elementos suportam e quais não. Tive situações em que a aplicação funcionava perfeitamente num deles e não nos outros. O teste simultâneo em todos eles devia ser feito com cada alteração, e não esperar ao fim para ter que corrigir partes significativas do código.

8.3.5 Informar-se bem sobre o âmbito do projecto

Logo à partida achei que tinha compreendido o que se estava a pedir e parti logo para a planificação do projecto.

Só mais tarde é que procurei literatura e me informei sobre o que realmente são entidades nomeadas e o seu papel em diferentes áreas. Se o tivesse feito, teria encontrado outras aplicações como REMBRANDT[12], um programa de Reconhecimento de Entidades Mencionadas Baseado em Relações e ANálise Detalhada do Texto. Isto me teria dado uma melhor ideia de qual o objectivo e também teria servido como termo de comparação.

O aspecto gráfico da aplicação é razoável, embora, em minha defesa, tenha que dizer que a interface de REMBRANDT disponível ao público não é consistente entre navegadores e tem falhas, para além de não ser capaz de lidar com textos compridos ou complexos como aqueles que a minha aplicação consegue processar.

Para concluir a comparação, a aplicação desenvolvida por mim suporta a gestão de Feeds e a etiquetagem transparente de artigos de forma muito mais cómoda para o utilizador.

Os tempos, proporcionalmente à quantidade de texto a processar, são maiores no REMBRANDT, embora tenha que supor que é devido a que é feita uma análise mais elaborada ao texto e que provavelmente a versão disponível no site não está optimizada.

8.3.6 Sistemas de webhosting baratos

Para testar num ambiente de produção coloquei a aplicação num servidor de webhosting partilhado.

Apesar de ter acesso ao shell (via SSH), a importação dos Dumps da Wikipédia foi complicada pelo facto de a qualidade da ligação entre o servidor Web e o servidor de bases de dados ser de qualidade duvidosa. Isto provocou que o processo de importação fosse repetidamente cortado sem nunca ter a certeza sobre se o processo de importação foi completado na totalidade ou não.

Foi necessário usar as funcionalidades de Backup/Restore disponibilizadas pelo fornecedor, um processo que limita as possibilidades de actualizações interactivas da base de dados.

[12] http://xldb.di.fc.ul.pt/Rembrandt/?do=home

8.3.7 Conclusão

Neste capítulo final foram apresentadas as conclusões, lições aprendidas e possíveis melhoras que poderiam ser aplicadas.

As lições aprendidas são especialmente úteis porque são suficientemente genéricas para ser aplicadas em futuros projectos.

Quanto às possíveis melhoras, algumas ideias também são genéricas, mas seriam passos a aplicar caso se quisesse desenvolver ainda mais a aplicação.

Anexo A: Referências

Sekine, Satoshi “Named Entity: History and Future”, New York University, 2004

Brunstein, Ada “ANNOTATION GUIDELINES FOR ANSWER TYPES”, http://www.ldc.upenn.edu/Catalog/docs/LDC2005T33/BBN-Types-Subtypes.html, 2002

Sekine, Satoshi “Definition of Sekine’s Extended Named Entity”, http://nlp.cs.nyu.edu/ene/version6_1_0eng.html, 2003

http://l2r.cs.uiuc.edu/~cogcomp/LbjNer.php, Cognitive Computation Group, University of Illinois, 2010

Nuno Cardoso, “REMBRANDT - Reconhecimento de Entidades Mencionadas Baseado em Relações e ANálise Detalhada do Texto”. In Cristina Mota & Diana Santos (eds.). Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O Segundo HAREM. Linguateca. 2008. In Portuguese.