Universidade Federal de Alagoas
Instituto de Computação

Trabalho de Conclusão de Curso

Framework para Criação de Blog *Crawlers* Baseados em Contexto

Rafael Ferreira Leite de Mello
rafaelflmello@gmail.com

Orientadores:
Ig Ibert Bittencourt
Evandro de Barros Costa

Maceió, Dezembro de 2009

Rafael Ferreira Leite de Mello

Framework para Criação de Blog *Crawlers* Baseados em Contexto

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciências da Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientadores:

Ig Ibert Bittencourt

Evandro de Barros Costa

Maceió, Dezembro de 2009

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciências da Computação do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Ig Ibert Bittencourt - Orientador
Instituto de Computação
Universidade Federal de Alagoas

Evandro de Barros Costa - Orientador
Instituto de Computação
Universidade Federal de Alagoas

Henrique Pacca Loureiro Luna - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Patrick H. S. Brito - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Robério José Rogério dos Santos - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Maceió, Dezembro de 2009

Resumo

Com o grande crescimento da Web, foram criados inúmeros mecanismos para interação entre os usuários. Tal fenômeno ficou conhecido como Web 2.0, onde o conhecimento é gerado através da interação dos usuários, fazendo uso da inteligência coletiva. Sob uma perspectiva da Web 2.0, diversas ferramentas colaborativas foram implementadas e disponibilizadas, dentre elas podemos destacar: wikis, fóruns, blogs, entre outros. Um exemplo clássico na geração de conhecimento através destas ferramentas colaborativas equivale aos blogs. Atualmente, há mais de 133 milhões de blogs e a cada dia são criados centenas deles. Além disto, a atividade nos blogs dobra a cada duzentos dias, sendo este fenômeno social conhecido como Blogosfera. A partir do momento que o conhecimento é gerado na Blogosfera, diariamente, as potencialidades de aplicações e decisões que podem ser tomadas através destas informações tornam-se inúmeras. Entretanto, utilizar as informações disponíveis na Blogosfera torna-se impraticável se executadas de forma manual. Com isso, mostra-se fundamental a extração de informações úteis através de abordagens computacionais.

Diante deste cenário, este trabalho propõe um *framework* para construção de blog *crawlers* para recuperar informação na blogosfera. Este *framework* disponibiliza serviços de pré-processamento, indexação, extração de textos a partir de HTML, entre outros. Além disto, foi proposto um algoritmo para extração de texto a partir de páginas HTML. O trabalho apresentada um estudo de caso da instanciação do *framework* e avalia o algoritmo proposto utilizando as métricas precisão e *recall*.

Abstract

With the growth of the Web, were created many mechanisms for interaction among the users. This phenomenon is known as Web 2.0, where knowledge is generated through the interaction of users, making use of collective intelligence. From a perspective of Web 2.0, various collaborative tools have been implemented and made available, among then we highlight: wikis, forums, blogs, among others. A classic example in generation of knowledge through these collaborative tools is blogs. Currently, there are more than 133 million blogs and every day are created hundreds of them. Furthermore, the activity on blogs doubles every two hundred days, this social phenomenon is known as the blogosphere. Once that knowledge is generated in the Blogosphere daily, the applications and decisions that can be taken through this information become numerous. However, using available information in the Blogosphere becomes unworkable to do manually. Thus, is essential to extract useful information through computational approaches.

In this scenario, this paper proposes a framework for building blog crawlers to retrieve information in the Blogosphere. This framework provides services for preprocessing, indexing, extraction of text from HTML, among others. Moreover, it was proposed an algorithm for extracting text from HTML pages. Was also presented a case study of the framework's instantiation and the proposed algorithm was evaluated using precision and recall metrics.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Capítulo 1

Introdução

Com o grande crescimento da Web, foram criados inúmeros mecanismos para interação entre os usuários. Tal fenômeno ficou conhecido como Web 2.0, onde o conhecimento é gerado através da interação dos usuários, fazendo uso da inteligência coletiva [Seiji Isotani 2009]. Sob uma perspectiva da Web 2.0, diversas ferramentas colaborativas foram implementadas e disponibilizadas, dentre elas podemos destacar: wikis, fóruns, blogs, entre outros. Um exemplo clássico na geração de conhecimento através destas ferramentas colaborativas equivale aos blogs. Atualmente, há mais de 133 milhões de blogs e a cada dia são criados centenas deles [Technorati 2008]. Ainda de acordo com tal levantamento a atividade nos blogs dobra a cada duzentos dias, sendo este fenômeno social conhecido como Blogosfera.

Blogosfera é o nome dado a toda a comunidade de blogs na Internet, é parte da infraestrutura pela qual idéias são desenvolvidas e transmitidas. Os blogs são, essencialmente, um texto publicado dos pensamentos de um autor, enquanto a blogosfera é um fenômeno social. A partir do momento que o conhecimento é gerado na Blogosfera, diariamente, as potencialidades de aplicações e decisões que podem ser tomadas através destas informações tornam-se inúmeras. Por exemplo, é possível descobrir a opinião de vários usuários sobre um determinado filme ou produto. Entretanto, utilizar as informações disponíveis na Blogosfera torna-se impraticável se executadas de forma manual. Com isso, mostra-se fundamental a extração de informações úteis através de abordagens computacionais.

Por esta razão, vários estudos para trabalhar com o problema de automatização estão sendo desenvolvidos. Técnicas como processamento de linguagem natural [Abney 1991], recuperação de informação [Manning et al. 2008] e extração de informação [Gaizauskas and Wilks 1998] são algumas abordagens que procuram obter informações interessantes de textos de forma automática.

Com relação a obtenção de documentos, a área de recuperação de informação(RI) se destaca em relação às demais [Hotho et al. 2005], pois a mesma se preocupa em identificar textos relevantes para uma determinada busca dentro de uma grande coleção de textos[1].

[1]Existem vários algoritmos que recuperam e organizam dados como: PageRank [Brin and Page 1998],

Todavia, apesar de possuir várias técnicas para recuperação de documentos, em alguns casos, como na blogosfera, a busca, utilizando apenas RI, nem sempre obtém bons resultados. Diante disto, algumas empresas começaram a investir em soluções alternativas que pudessem ser acopladas às técnicas de RI existentes. Na Blogosfera, uma destas soluções é contextualizar os *posts*[2] dos blogs. Para isso, empresas como: Technorati [Technorati 2009b], Icerocket [Icerocket 2009], Blogcatalog [Blogcatalog 2009], Delicious [Company 2009], entre outras, criaram serviços de marcação(*tag*). Com este serviço, o nível de semântica associada a cada texto publicado nos blogs aumentam consideravelmente [Mathes 2004] [Golder and Huberman 2005]. Por exemplo, um post relacionado a uma aula de matemática é marcado com a *tag* educação, ou seja, está inserida no contexto de educação. Desta forma, quando o usuário pesquisar por *posts* interessantes, pode-se definir o contexto que deseja encontrá-lo.

É importante destacar, que antes dos usuários realizarem as buscas, há necessidade de indexar e armazenar os textos dos blogs. Tal mecanismo é realizado por uma entidade de software conhecido como web *crawlers* [Arasu et al. 2001]. Especificamente no contexto de blogs, os web *crawler* são chamados de blog *crawler*.

Entretanto, para criação de blog *crawler* para recuperar informação na blogosfera deve-se levar em consideração vários aspectos, como: i) idioma do blog; ii) empresa indexadora[3]; iii) técnicas de pré-processamento e indexação que serão aplicadas aos textos do blog; iv) empresa estruturadora[4], entre outros.

Percebe-se então que construir um blog *crawler* é um processo que possui as seguintes características: i) alta complexidade; ii) possui pontos de variabilidade; iii) deve ser extensível; iv) precisa ser simples de usar. Para resolver estes problemas, a solução encontrada foi utilizar *framework* [Johnson and Foote 1988], que é um conjunto de classes que incluem um projeto abstrato para soluções de uma família de problemas relacionados, ou seja, utilizando esta técnica de programação o sistema desenvolvido pode ser estendido para atender a muitas aplicações que possuem o mesmo fim.

Dentro deste contexto, este trabalho propõe um *framework* para construção de blog *crawlers* baseados em contexto. Ou seja, que utiliza as marcações para aumentar o nível semântico dos blogs. Este *framework* disponibiliza serviços de pré-processamento, indexação, extração de textos a partir de HTML, entre outros. Para exemplificar a instanciação dele, foi criada uma aplicação baseada no sistema de marcação da *Technorati*.

Anchor Text [Brin and Page 1998], Kleinberg's HITS [Kleinberg 1999], entre outras.

[2] Textos publicados no blog.

[3] Technorati, Icerocket, entre outros

[4] Facebook, Blogger, entre outros

1.1 Problemática

Construir blog *crawlers* não é uma tarefa fácil de ser realizada, pois muitas variáveis estão envolvidas com este processo. Além disto, para conseguir bons resultados de recuperação de informação na Blogosfera normalmente é necessário utilizar mais que as técnicas de RI tradicionais. Uma solução encontrada para este problema foi utilizar marcação (*tags*) para aumentar o nível de precisão nas buscas. Em [Shirky 2005] é descrito um trabalho deste tipo, porém ele compreende apenas blogs corporativos[5]. Existem outras abordagens que tentam resolver problemas específicos dos blogs *crawlers*([Hurst and Maykov 2009][Glance et al. 2004]), contudo todas focam em domínios específicos e não se preocupam com o reuso da solução.

Além do problema dos blog *crawlers*, outro desafio é extrair os textos completos dos blogs para análise. Na Web existem APIs públicas para trabalhar com blogs, disponibilizadas por empresas como a *Technorati*, porém nenhuma disponibiliza serviços que retornem os conteúdos dos posts, no máximo retornam resumos deles. Então, para construir um blog *crawler* seria preciso criar um mecanismo para obter apenas o texto do blog em meio as páginas HTML, tendo apenas informações como: resumo do *post*, link do blog, título da publicação, entre outros.

1.2 Objetivo

O objetivo deste trabalho é propor um *framework* para construção de blog *crawlers* baseados em contexto. Ou seja, que utiliza as marcações para aumentar o nível semântico dos blogs. Este *framework* disponibiliza serviços de pré-processamento, indexação, extração de textos a partir de HTML, entre outros. Além disto, também é proposto um algoritmo para extração do texto do blog a partir de uma página HTML.

1.3 Relevância

A partir do desenvolvimento deste trabalho, espera-se a obtenção dos seguintes ganhos:

- Criação rápida de aplicações para recuperação de informação em blogs baseadas em contexto;
- Criação de um arcabouço facilmente extensível para trabalhar com mineração de texto em blogs;
- Extrair textos dos blogs a partir de páginas HTML.

[5]Blogs utilizados por empresas normalmente para um sistema interno.

1.4 Estrutura do Trabalho

Este trabalho está dividido como segue:

- No Capítulo 2, são apresentados os conceitos relacionados ao tema deste trabalho. A teoria básica acerca de blogs, recuperação de informação, web *crawlers*, projeto de software com reuso e uma breve explicação sobre alguns diagramas que serão utilizados para explicar o *framework* proposto;
- No Capítulo 3, são apresentados alguns trabalhos relacionados ao *framework* proposto e características das ferramentas utilizadas para realização deste trabalho;
- No Capítulo 4, são apresentadas as propostas do trabalho, o framework e o algoritmo;
- No Capítulo 5, é mostrado um estudo de caso relacionado à instanciação de uma aplicação com o *framework* proposto e resultados da utilização do algoritmo para extração de texto a partir de páginas HTML;
- No Capítulo 6, são apresentadas as considerações finais deste trabalho.

Capítulo 2

Fundamentação Teórica

O objetivo deste capítulo é apresentar a fundamentação teórica referente ao foco deste trabalho, fazendo uma pequena introdução sobre conceitos correlacionados. O capítulo divide-se em duas partes, na primeira são levantados aspectos tecnológicos envolvidos com o trabalho, já a segunda trata de questões relacionadas a engenharia de software. Na Seção 2.1 serão discutidas algumas motivações para utilizar os blogs e o que está acontecendo na blogosfera. Na seção seguinte serão apresentados conceitos sobre recuperação de informação que serão utilizados na proposta do trabalho. Na Seção 2.3 serão discutidos aspectos relacionados aos Web *Crawlers*, suas definições e características. As últimas duas seções tratam de definições ligadas à engenharia de software. Na Seção 2.4, são abordadas características relacionadas a projetos de software com reuso, com enfoque em *framework* 2.4.1 e padrões de projetos 2.4.2. Por fim, na Seção 2.5 serão detalhados alguns diagramas que foram utilizados para descrever o *framework* proposto.

2.1 Blogosfera

Blogosfera é o nome dado a toda a comunidade de blogs na internet, é parte da infra-estrutura pela qual idéias são desenvolvidas e transmitidas. Os blogs são, essencialmente, apenas o texto publicado dos pensamentos de um autor, enquanto a blogosfera é um fenômeno social.

Segundo a wikipédia [Wikipédia 2009a] o blog (uma contração da expressão "Web log") é um site cuja estrutura permite a atualização rápida a partir de acréscimos dos chamados artigos, ou *posts*. Estes são, em geral, organizados de forma cronológica inversa, tendo como foco a temática proposta do blog, podendo ser escritos por um número variável de pessoas, de acordo com a política do blog.

A *Technorati*[1], um dos maiores agregadores de blogs da Web, faz anualmente um levantamento sobre o estado da Blogosfera e suas tendências. Usaremos as pesquisas de

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

2008[Technorati 2008] e 2009[Technorati 2009a] para definir pontos interessantes sobre o atual estado dos blogs.

Em 2008 a pesquisa da *Technorati* chegou a conclusões interessantes sobre o tamanho da blogosfera. Todos os estudos comprovam que os blogs são um fenômeno global que atingiu a Web. Na figura 2.1 [Technorati 2008] podemos perceber fatos relevantes como a grande quantidade de *posts* que são escritos diariamente, novecentos mil, e existem mais de 133 milhões de blogs indexados pela *technorati* entre os anos de 2002 e 2008. De acordo com a mesma estimativa a atividade nos blogs dobram a cada duzentos dias.

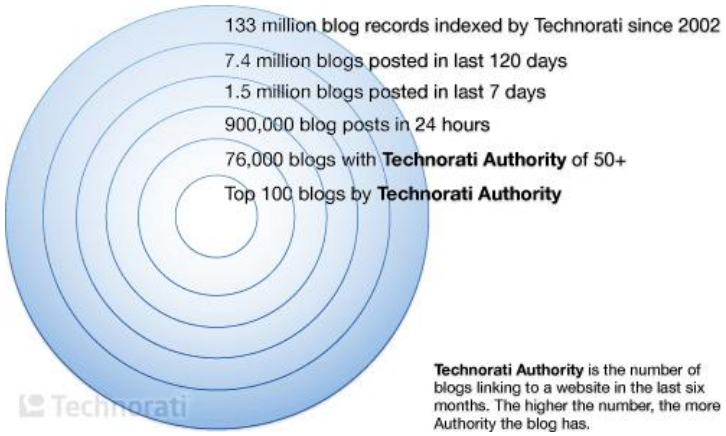

Figura 2.1: Pequeno demonstrativo do tamanho da blogosfera

Duas perguntas interessantes foram respondidas na pesquisa de 2009[Technorati 2009a]: i)Quem são os blogueiros?; e ii)Quais os motivos para escrever um blog? Elas serão mais respondidas nas subseções a seguir.

2.1.1 Quem são os blogueiros?

A palavra blogueiros se refere às pessoas que escrevem em blogs, segundo pesquisa de [Technorati 2008], os blogueiros em geral, fazem parte de um grupo altamente educado e rico. Quase metade de todos os blogueiros examinados ganharam um diploma de pós-graduação, e a maioria tem uma renda familiar de US $ 75.000 ou superior por ano. Além destes dados, outras características interessantes são:

- Dois terços são do sexo masculino;
- 60% são 18-44;
- A maioria são mais ricos e educados do que a população em geral;
- 75% têm diploma universitário;
- 40% têm pós-graduação;

- profissionais independentes que são blogueiros são mais ricos: cerca de metade tem uma renda familiar anual de 75.000 dólares e um terço superaram o nível de 100.000 dólares.

Algumas destas características indicam que as pessoas que estão utilizando os blogs são qualificadas, com isso os textos publicados fornecem informações interessantes.

2.1.2 Quais os motivos para escrever um blog?

Troca de experiência de auto-expressão continuam a ser as principais motivações para blogueiros, e 70% dos inquiridos dizem que a satisfação pessoal é uma maneira de medir o sucesso de seu blog. Entre os blogs profissionais, no entanto, a principal métrica de sucesso é o número de visitantes únicos. Blogueiros comuns tendem a escrever sobre reflexões e opiniões pessoais, enquanto os profissionais tendem a ser mais atuais.

A ascensão do blogueiro profissional continua, 70% dos funcionários em tempo parcial e trabalhadores autônomos estão postando mais do que nunca, enquanto os blogueiros comuns são um pouco menos. O principal motivo para diminuir a atividade dos blogueiros foi o aumento do trabalho e dos compromissos familiares (64%). Blogueiros descrevem significativos impactos positivos em suas vidas pessoais, mas os blogueiros mais experientes têm de carreira e os impactos positivos do negócio. 70% dizem que eles são mais conhecidos em sua indústria por causa de seu blog. A diversidade da blogosfera, Figura 2.2 [Technorati 2009a], refletem na questão do assunto da postagem - mesmo tendo 23 opções, incluindo a maioria dos grandes campos de instrução, 30% dos inquiridos dizem que seu assunto principal é "Outro".

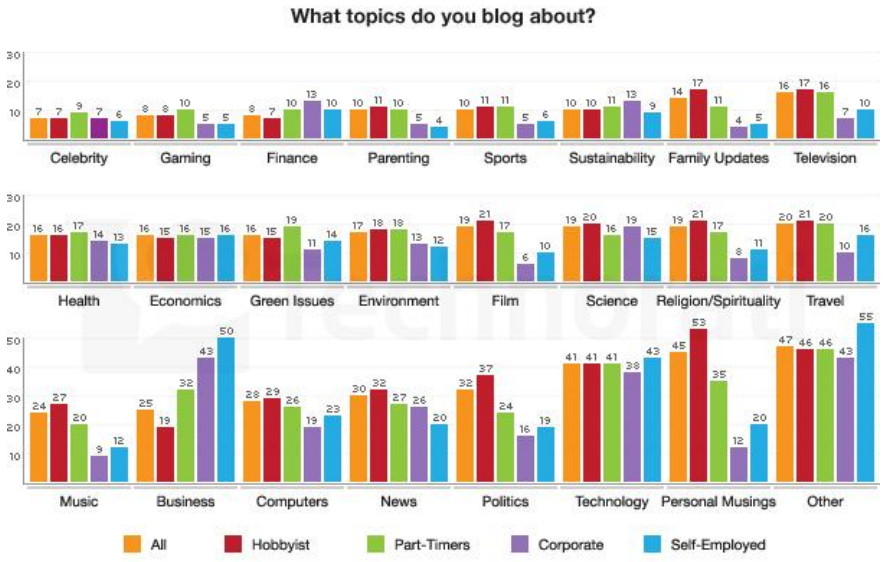

Figura 2.2: Assuntos blogados

2.2 *Recuperação de informação*

Recuperação de informação [Manning et al. 2008] pode ser definida como a procura de textos interessantes para uma determinada busca dentro de uma grande coleção de textos. Durante muito tempo essa área foi pouco explorada pela comunidade científica, contudo após a popularização da Web, grandes empresas, como a Google e a Yahoo surgiram e vários investimentos nessa área foram feitos.

As áreas de pesquisa na recuperação da informação envolvem modelagem, classificação e categorização da informação, arquitetura de sistemas, filtragem, linguagens de consulta, entre outros, e são categorizadas pelo tamanho no qual eles são utilizados, como por exemplo: pesquisa na Web, recuperação de informações pessoais e pesquisas em empresas, instituições ou um domínio específico [Baeza-Yates and Ribeiro-Neto 1999].

O objetivo da recuperação da informação, nestas aplicações, é representar eficientemente a coleção de documentos que o usuário pretende interagir. Uma estrutura de dados muito utilizada para representar tal coleção é a indexação invertida [Manning et al. 2008] [Hatcher and Gospodnetic 2004]. A idéia do índice invertido é construir uma tabela de referência onde cada termo ocorrido no documento é usado como um item para a lista de documentos que cada termo ocorre. Ele funciona como um índice remissivo de um livro, ou seja, guarda as páginas que contém as palavras e não o contrário. Então na busca ao invés de procurar página por página até encontrar a palavra, ele vai direto para a palavra que já contém as páginas em que ela ocorre. Além disso, outro fator importante que deve ser armazenado é a freqüência de um termo em determinado documento. Quando concluído o processo, o índice invertido recebe o nome de dicionário.

Contudo, um problema é facilmente identificado neste processo: existem palavras que têm pouco poder representativo no documento, por exemplo os artigos no português, e outras palavras podem conter a mesma semântica, como as palavras no masculino e no feminino, no plural e no singular. Para resolver esses problemas, antes de indexar, os documentos devem ser submetidos a um pré-processamento. Existem várias técnicas que procuram processar o texto antes de indexá-los, entre elas estão[Hotho et al. 2005]:

- **Análises léxicas**: divide o documento em *tokens* e verifica palavra por palavra, trata a retirada de símbolos não pertencentes à linguagem trabalhada, por exemplo retirar código HTML(*HyperText Markup Language*) do texto, caracteres inválidos, espaços em branco, entre outros;

- **Filtragem**: retira palavras do dicionário e dos documentos. O método mais conhecido é a remoção de *stop words*. A idéia desse método é retirar palavras que contém pouco valor de informação. Por exemplo, no português os artigos (a, o, um, uma...) possuem pouco valor para o texto, além disso palavras que aparecem muito na

coleção de documentos também poder ser consideradas *stop words* pois não irão acrescentar valor significativo para as buscas;

- ***Lemmatization***: métodos que tentam mapear os verbos para formas primitivas e nomes para o singular, contudo esta técnica ainda apresenta algumas dificuldades, pois ela mantém no texto a palavra e cria uma estrutura, normalmente são usadas tags, que contém as formas reduzidas, ou seja, ela não retira do texto a palavra original, apenas marca aquela palavra com uma tag que vai conter uma palavra reduzida relacionada a original. Por esse motivo normalmente é utilizada a técnica de *Stemming*;

- ***Stemming***: tenta mapear a forma básica das palavras, como tirar o plural ou reduzir o verbo obtendo o radical, por exemplo no inglês os verbos são transformados para a forma primitiva, *traveling* e *traveled* são transformados em *travel*;

- **Remoção de spam**: técnicas para encontrar e retirar do conjunto de documentos textos que possuem pouco valor para a pesquisa ou domínio utilizado pelo usuário.

Depois do processo de pré-processamento e criação do dicionário, utilizando o processo de índice invertido, é necessário criar mecanismos de buscas e consultas, como por exemplo o modelo booleano ou o modelo estatístico [Manning et al. 2008]. Com este mecanismo o usuário poderá recuperar qualquer documento dentro que já esteja no dicionário através de uma *query*. A Figura 2.3, mostra um fluxo de um processo básico de recuperação de informação que foi tratado ao longo desta seção.

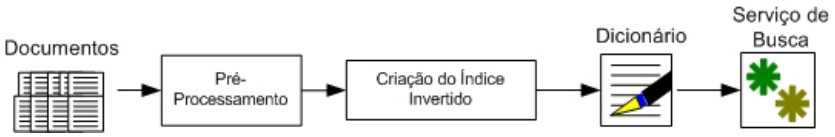

Figura 2.3: Fluxo de um Processo Básico de RI

Ferramentas como o Bow [McCallum 1996] e Lemur [Lemur 2009] possuem diversas funcionalidades de recuperação de informação, contudo o Lucene [Lucene 2001] é a ferramenta mais conhecida e completa que disponibiliza funcionalidades de recuperação de informação em diversos idiomas.

2.3 Web *Crawler*

Os sistemas de busca vem se tornando cada dia mais indispensáveis para obter informação relevante da grande quantidade de dados que a Web disponibiliza. Os motores de busca armazenam imensas coleções de páginas com ajuda dos Web *crawlers*, que são responsáveis

por percorrer a Web através de *links* e colher dados que normalmente são indexados para que o usuário possa executar consultas eficientemente [Arasu et al. 2001].

Segundo Gautam Pan [Pant et al. 2004] Web *crawler* é um programa de computador que explora a estrutura de grafo da Web para mover uma página para outra. Como já foi dito, a maior motivação para idealizar os Web *crawlers* foi recuperar as páginas da Web e adicionar elas ou a representação delas em um repositório local para facilitar a busca do usuário. Na sua forma mais simples um *crawler* parte de uma página e a partir daí utiliza *links* externos para atingir as outras páginas.

Por detrás desta simples descrição, encontra-se uma série de questões relacionadas com as conexões de rede, *spider traps*, análise de páginas HTML, entre outros.

Se a Web fosse uma coleção estática de páginas, logo chegaria o momento em que os *crawlers* não seriam mais necessários, visto que todas as páginas já estariam ligadas a um repositório. Contudo, a Web é uma entidade dinâmica que evolui com taxas e velocidades diferentes. Por isso existe a necessidade de continuar usando os *crawlers*. Eles normalmente são os responsáveis por deixar o banco de dados de sistemas de busca sempre atualizado.

A seguir, serão descritos detalhes sobre o fluxo atividades e arquitetura básica de um Web *crawler*.

2.3.1 Fluxo de atividades

A Figura 2.4 [Pant et al. 2004] mostra o fluxo básico de atividades de um *crawler*, que ele é detalhado a seguir.

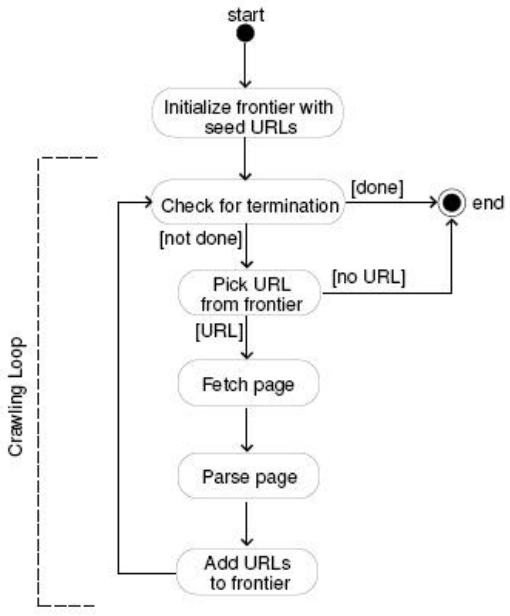

Figura 2.4: Fluxo básico de um *crawler*

- **Inicializar o *frontier* com as primeiras URIs**: O *frontier* é a lista de "coisas a fazer"do *crawler* e contém uma lista de URIs que ainda não forão visitadas. Para esta etapa pode ser implementado, por exemplo, um esquema de escalonamento *Fist In Fist Out*(FIFO) [Tanenbaum 2007] para ordenar as URIs que serão acessadas;
- **Verifica fim e procura próximo**: Verifica se o *frontier* está vazio, caso não esteja retorna a próxima URI a ser analisada;
- ***Fetch page***: Obtém a página, normalmente através de uma requisição feita por meio de um cliente HTTP. Trata erros de conexão e determina algumas características das páginas, como a última atualização;
- ***Parse page***: Faz um *parser* na página para extrair informações úteis e possivelmente guiar o próximo passo do *crawler*, além disso, procura também eliminar informações pouco expressivas ou inúteis;
- **Adiciona URI no *frontier***: Adiciona URIs, obtidas no passo anterior, ao *frontier*.

2.3.2 Arquitetura

A figura 2.5 [Shkapenyuk and Suel 2002] apresenta os componentes básicos de uma arquitetura de um *crawler*.

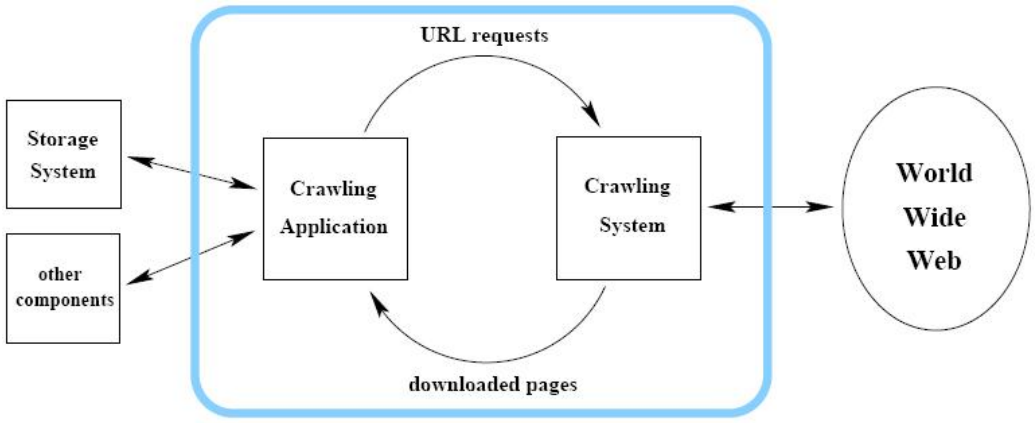

Figura 2.5: Arquitetura básica de um *crawler*

Nesta arquitetura existem dois componentes fundamentais que são os *crawlers*, de aplicação e de sistema. O *crawler* de sistema tem a responsabilidade de ir até a Web e fazer o *download* das páginas. Por outro lado, o de aplicação é responsável por passar a lista de páginas que devem ser baixadas e fazer as devidas operações com os dados obtidos pelo *crawler* de sistema.

2.3.3 *Crawlers* sociais

Cada dia é maior a utilização de ferramentas sociais como: wikis, emails, fóruns, blogs, ferramentas de relacionamento, entre outros. Estas ferramentas possuem informações que podem ser valiosas se bem utilizadas, por exemplo, uma ferramenta de relacionamento descreve algumas características dos usuários que podem ser usadas para traçar o perfil dele e criar mecanismos de individualização de serviços.

Contudo, para isso é necessário mecanismos para obter informações dessas ferramentas. Aí que entra os *crawlers* sociais [Ying Ding and Yan 2008], eles são similares aos *crawlers* descritos anteriormente, mas possuem uma diferença fundamental, eles são projetados para colher informações específicas das ferramentas e não a sua página completa. Na seção 3 serão abordados alguns *crawlers* sociais que realizam atividades específicas nas ferramentas da Web 2.0.

2.4 Projeto de software com reuso

O processo de projeto, na maioria das disciplinas de engenharia, tem como base o reuso de componentes. No desenvolvimento do software cada vez mais é necessária uma abordagem parecida, o software é considerado um ativo e o reuso dele é essencial para aumentar o retorno de seus custos de desenvolvimento. Requisitos como menor custo de produção e manutenção do software, maior rapidez na entrega e aumento da qualidade, só podem ser atendidos pelo reuso generalizado e sistemático do software [Sommerville 2004].

A engenharia de software baseado no reuso é uma abordagem para desenvolvimento que tenta maximizar o reuso do software já existente. As unidades de software reutilizadas podem ser, por exemplo [Sommerville 2004]:

- **Reuso de sistemas de aplicações**: Todo sistema de aplicação pode ser reutilizado pela sua incorporação, sem mudanças, em outros sistemas ou pelo desenvolvimento de famílias de aplicações;
- **Reuso de componente**: Os componentes de uma aplicação, que variam em tamanho incluindo desde subsistemas até objetos isolados, podem ser reutilizados;
- **Reuso de funções**: Os componentes de software que implementam uma única função, como uma função matemática, podem ser reutilizados.

As principais vantagens de utilizar sistemas baseados em reuso são apresentadas a seguir [Sommerville 2004]:

- **Maior confiabilidade**: Componentes reutilizados normalmente são experimentados e testados em diferentes ambientes;

- **Redução de riscos do processo**: Recorrendo a um componente já existente, serão menores as incertezas sobre os custos relacionados ao reuso desse componente do que sobre custos de desenvolvimento;
- **Uso efetivo de especialistas**: Especialistas podem fazer componentes reusáveis que englobem seus conhecimentos;
- **Conformidade com os padrões**: Alguns padrões podem ser implementados com um conjunto de componentes-padrão;
- **Desenvolvimento acelerado**: O reuso de componentes acelera a produção, porque o tempo de desenvolvimento e o de validação devem ser reduzidos.

Por outro lado também existem alguns problemas relacionados a esse tipo de desenvolvimento [Sommerville 2004]:

- **Aumento nos custos de manutenção**: Se o código fonte de um componente não estiver disponível, então o custo de manutenção pode aumentar, uma vez que os elementos reutilizados no sistema podem se tornar crescentemente incompatíveis com as mudanças do sistema;
- **Síndrome do "não-foi-inventado-aqui"**: Alguns engenheiros de software preferem reescrever componentes, pois acreditam que podem fazer melhor que o componente reutilizável;
- **Manutenção de uma biblioteca de componentes**: Implementar uma biblioteca de componentes e assegurar que os desenvolvedores de software utilizem essa biblioteca pode ser dispendioso;
- **Encontrar e adaptar componentes reutilizáveis**: Os engenheiros de software precisam ter uma razoável certeza de poder encontrar um componente, antes de incluírem a rotina de busca de componentes como parte de seu processo normal de desenvolvimento.

Dentro do reuso baseado em componentes duas abordagens, muitas vezes complementares, são muito utilizadas, o *framework* e os padrões de projeto. A seguir serão detalhadas estas abordagens.

2.4.1 Framework

Existem várias definições de *framework*, sendo que as mais famosas são as de Ralph E. Johnson [Johnson and Foote 1988] [Johnson 1997], que dizem: i) Um framework é um conjunto de classes que incluem um projeto abstrato para soluções de uma família de

problemas relacionados; ii) Um framework é um conjunto de objetos que colaboram para realizar um conjunto de responsabilidades para um domínio de aplicação, um esqueleto de uma aplicação que pode ser customizada pelo desenvolvedor.

Destas definições podemos listar algumas características de um *framework*:

- Reusa o projeto de software e não apenas o código;
- Não é uma aplicação, e sim, um arcabouço para construir aplicações num domínio específico;
- É um arcabouço voltado para um determinado domínio;
- Resolve problemas de uma mesma natureza;
- São aplicações incompletas.

Além das características citadas a partir das definições acima, outras devem ser ressaltadas, são elas [Sauvé 2000] [Fayad et al. 1999]:

- **Reusabilidade**: Este é o propósito final do *framework*, mas para ser reusável antes tem que ser usável, por isso ele tem que ser bem documentado e fácil de usar;
- **Extensibilidade**: Deve conter funcionalidade abstrata (sem implementação) que deve ser completada. Desta forma, inúmeras aplicações podem ser instanciadas com base no mesmo *framework*;
- **Completude**: Precisa endereçar o domínio do problema pretendido;
- **Inversão de controle**: A inversão do controle é uma característica fundamental na arquitetura de um *framework*. Ela permite determinar que conjunto de métodos de uma aplicação específica deve ser chamado pelo *framework*. Quem define o controle de fluxo é o *framework*.

Os *frameworks* podem ser classificados de várias maneiras, por exemplo, em relação a onde ele é usado [Sauvé 2000]:

- **Framework de suporte**: Provê serviços de nível de sistema operacional, como acesso a arquivos ou computação distribuída;
- **Framework de aplicação**: São também conhecidos como frameworks horizontais. Esses fornecem funcionalidades que são aplicadas a uma variedade de problemas. São utilizados em mais de um domínio. Por exemplo um *framework* para construção de interface GUI;

- **Framework de domínio**: São também conhecidos como frameworks verticais. Esses fornecem funcionalidades que são aplicadas num domínio específico. Por exemplo *framework* para construir aplicações de controle de manufatura.

Eles também podem ser classificados em relação à técnica de extensão [Fayad et al. 1999]:

- ***White Box***: Estão fortemente ligado às características das linguagens OO, para realizar a customização utilizam-se de herança. Requer um bom entendimento do *framework* para criar uma aplicação;
- ***Black Box***: São instanciados a partir de algum tipo de configuração, como XML por exemplo. Esses frameworks confiam principalmente em composição(classes ou componentes) e delegação para realizar customização. Não requer entendimento de detalhes internos para produzir uma aplicação;
- ***Gray Box***: São *frameworks* projetados para evitar as desvantagens apresentadas por *frameworks white box* e *black box*, permitindo certo grau de flexibilidade e extensibilidade sem expor informações internas desnecessárias.

Para projetar um framework é necessário estar atento a algumas propriedades, como [Clements and Northrop 2001]:

- **Núcleo do framework**: Conjunto de classes que juntas colaboram para implementar uma arquitetura de família de sistemas;
- **Pontos de extensão**: Admite pontos de extensão, normalmente, na forma de classes abstratas ou interfaces;
- **Controle de Fluxo da Aplicação**: O fluxo é definido pelas classes que estão no núcleo, e elas são responsáveis por invocar as classes que estendem os pontos de extensão;
- ***Hot Spots***: São as partes flexíveis de um *framework*, estes pontos são passíveis de extensão. *Hot spots* são invocados pelo *framework*, ou seja, classes (implementadas pelo programador da aplicação) recebem mensagens de uma classe do framework (*frozen spot*). Isso geralmente é implementado através de herança e de métodos abstratos;
- ***Frozen Spots***: Partes fixas de um *framework*, fornece os serviços já implementados do *framework*. Normalmente realizam chamadas indiretas aos *hot spots*.

Algumas vantagens de utilizar os *frameworks* são [Sauvé 2000] [Sommerville 2004]: i) menos código para projetar e escrever; ii) redução de custos e do tempo de desenvolvimento; iii) código mais confiável e robusto; iv) manutenção reduzida e evolução

facilitada; v) melhor consistência e compatibilidade entre aplicações; vi) estabilização do código (menos defeitos) devido ao uso em várias aplicações.

Por outro lado também podemos citar algumas desvantagens de se trabalhar com *frameworks* [Sauvé 2000] [Sommerville 2004]: i) requer mais esforço para construir; ii) benefícios são realizados em longo prazo; iii) precisa modificar o processo de desenvolvimento e criar novos incentivos; iv) requer documentação, manutenção e suporte.

A documentação do *framework* é um dos fatores que podem definir o sucesso dele. Ela precisa se adaptar a diferentes públicos, para isso deve ter diferentes níveis de abstração. Quatro tópicos não podem deixar de ser documentados [Markiewicz and Lucena 2001]: i) propósito, que contém uma breve descrição do framework e o domínio do problema para o qual foi desenvolvido; ii) como usar o framework, garante a reutilização do framework, descreve como utiliza-lo; iii) propósito das aplicações, exemplos que ajudem a entender melhor o framework; iv) design, deve conter as classes, seus relacionamentos e colaborações.

Por fim, é importante distinguir *framework* de outras abordagens de reuso, como bibliotecas de classes e padrões de projeto.

Framework x bibliotecas de Classes

Uma biblioteca de classes possui características como: cada classe é única e independente das outras, os clientes chamam funções e não existe controle de fluxo. Por outro lado o framework possui: classes com dependências/colaborações estão embutidas, os clientes chamam funções da "aplicação"e existe controle de fluxo de execução.

Framework x Padrões de Projeto

A diferença entre framework e padrão de projeto é: i) Padrões de projeto são mais abstratos do que *frameworks*. Um *framework* inclui código, um padrão de projeto não; ii) Padrões de projeto são elementos arquiteturais menores do que *frameworks*. Um *framework* típico contém vários padrões de projeto mas o contrário nunca ocorre; iii) Padrões de projeto são menos especializados do que *frameworks*. *Frameworks* sempre têm um domínio de aplicação particular enquanto padrões de projeto não ditam uma arquitetura de aplicação particular.

2.4.2 Padrões de projetos

Projetar *software* orientado a objeto é uma tarefa difícil, e desenvolver programas reutilizáveis é ainda mais difícil. Para isso é necessário utilizar várias técnicas para resolver um problema específico e ao mesmo tempo deixar a solução genérica suficiente para resolver problemas da mesma natureza e rapidamente satisfazer futuras exigências. Designers ex-

perientes afirmam que é praticamente impossível conseguir uma boa solução para projetos reutilizáveis e flexíveis em um primeiro momento [Gamma et al. 1995].

Contudo, designers experientes produzem boas soluções. Os padrões de projetos são um conjunto de boas práticas de programação que já foram utilizadas em vários sistemas e foram catalogadas por programadores e designers experientes. Então utilizar essas práticas é recomendável, principalmente quando se é principiante.

Em geral, um padrão tem quatro elementos essenciais [Gamma et al. 1995]:

1. **Nome**: É um identificador que procura descrever o problema, suas soluções e conseqüências em poucas palavras;

2. **Problema**: Descreve quando o padrão deve ser aplicado, o seu problema e o contexto. Às vezes o problema vai incluir uma lista de condições que devem ser atendidos antes que ela faz sentido aplicar o padrão;

3. **Solução**: A solução descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações. A solução não descreve um projeto ou implementação em particular, porque um padrão é como um modelo que pode ser aplicado em muitas situações diferentes. Em vez disso, o modelo fornece uma descrição abstrata de um problema de projeto e como uma disposição geral dos elementos para resolvê-lo;

4. **Conseqüências**: As conseqüências são os resultados da aplicação do padrão. Estas são consideradas um atributo crítico para avaliar alternativas de projeto e para a compreensão dos custos e benefícios da aplicação do padrão. Esta seção deve incluir o impacto do padrão sobre a flexibilidade, extensibilidade e portabilidade de um sistema.

Os padrões de projetos podem ser divididos em vários tipos, entre eles os mais usados são [Gamma et al. 1995]:

- **Padrões de criação**: Como o próprio nome diz, este conjunto de padrões estão ligados ao processo de criação dos objetos. Deixa o sistema independendo da maneira de como os objetos estão sendo criados, representados e compostos. Especifica como, quando, onde e por quem objetos devem ser criados. Alguns exemplos são: *Factory Method, Singleton, Abstract Factory*, entre outros;

- **Padrões estruturais**: Padrões estruturais estão preocupados com a forma como classes e objetos são compostos para formar estruturas mais complexas. Alguns exemplos são: *Facade, Composite, Decorator*, entre outros;

- **Padrões comportamentais**: Os padrões comportamentais estão preocupados com algoritmos e a atribuição de responsabilidades/dependências entre objetos. Alguns exemplos: *Observer, Command, Strategy, Template Method*, entre outros.

A seguir serão explicados os padrões utilizados neste trabalho.

Factory Method

O *Factory Method* faz parte do tipo criacional, ou seja, está ligado ao processo de criação dos objetos. A seguir são mostradas algumas características deste padrão:

- **Problema**: Como criar um objeto sem usar a diretiva *new* diretamente? Como saber qual classe instanciar?
- **Intenção**: Definir uma interface para criar um objeto, mas deixar as subclasses decidirem qual classe instanciar. *Factory Method* permite que uma classe de adiar a instanciação para subclasses;
- **Aplicabilidade**: Deve-se usar o *Factory Method* quando: i) uma classe não conhece antecipadamente a classe dos objetos que deve criar; ii) uma classe quer que suas sub-classes especifiquem os objetos que criam;
- **Consequências**: i) Aumenta a flexibilidade, criação de objetos dentro de uma classe com um método de fábrica é sempre mais flexível do que criar um objeto diretamente; ii) paraleliza a hierarquia de classes, acontecendo quando uma classe delega algumas das suas responsabilidades para outra classe.

Singleton

O padrão *Singleton* também faz parte dos padrões de criação, entretanto o objetivo dele é bem diferente do *Factory Method*. Este padrão está preocupado em controlar o número de instâncias de um objeto. Segue mais detalhes sobre o padrão:

- **Problema**: Como garantir que uma classe só possui uma instância e um único ponto de acesso?
- **Intenção**: Garantir a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto;
- **Aplicabilidade**: Deve-se usar o *Singleton* quando: i) deve existir exatamente uma instância de uma classe, e ela deve ser acessível aos clientes a partir de um ponto de acesso conhecido; ii) quando a instância única deve ser extensível através de subclasses, e os clientes devem ser capazes de usar uma instância estendida sem alterar seu código;

- **Conseqüências**: i) Controle ao acesso da única instância; ii) reduz o *namespace*, evita poluir o código com várias variáveis globais; iii) permite variar o número de instâncias, o padrão permite facilmente permitir mais que uma instância de uma classe *Singleton*.

Facade

O *Facade* faz parto dos padrões estruturais, com ele o programador aumenta o nível de abstração em relação as classes envolvidas com o padrão. Abaixo estão descritos mais detalhes sobre este padrão:

- **Problema**: Qual classe do sub-sistema deve ser chamada?
- **Intenção**: Fornecer uma interface unificada para um conjunto de interfaces em um subsistema; *Facade* define uma interface de nível mais alto que torna o subsistema mais fácil de usar.
- **Aplicabilidade**: Deve-se usar o *Facade* quando: i) É necessário prover uma interface simples para um sistema complexo; ii) existem muitas dependências entre os clientes e as implementações de classes abstratas; iii) É necessário adicionar uma camada ao subsistema;
- **Conseqüências**: i) Isola os clientes dos componentes internos do sistema; ii) promove baixo acoplamento entre o subsistema e os clientes; iii) não impede clientes de utilizarem componentes diretamente, caso seja necessário.

Command

O *Command* é um padrão comportamental, a grande vantagem de utilizar este padrão é a diminição o acoplamento entre o objeto que chama a operação e o objeto que executa a operação. Mais detalhes sobre o padrão a seguir:

- **Problema**: Qual classe do subsistema deve ser chamada?
- **Intenção**: Encapsular uma requisição como um objeto, permitindo que os clientes parametrizem diferentes requisições, filas ou fazer o registro de log de requisições e dar suporte operações que podem ser desfeitas;
- **Aplicabilidade**: i) Parametrizar objetos por uma ação a ser executada; ii) especificar, enfileirar e executar solicitações em tempos diferentes, um objeto *Command* poder ter o ciclo de vida independente da requisição do cliente; iii) suporte para desfazer operações; iv) estruturar um sistema em torno de operações de alto nível, como transações por exemplo; v) reduzir acoplamento entre as requisições dos clientes e o objetos que as executam;

- **Conseqüências**: i) Reduz o acoplamento (dependência) entre o objeto que chama a operação e o objeto que executa a operação; ii) pode ser estendido e manipulado por outros objetos; iii) é mais fácil de acrescentar novas operações ou novos comandos, pois você não tem que mudar as classes existentes.

2.5 Diagramas

Esta seção tem por objetivo explicar cinco tipos de diagramas que foram usados para descrever o *framework* proposto, são eles: Diagrama de classe conceitual, diagrama de classe de implementação, diagrama de caso de uso, diagrama de *features* e diagrama de seqüência.

2.5.1 Diagrama de classe conceitual e de implementação

O diagrama de classe faz parte do conjunto de modelos propostos pela *Unified Modeling Language* UML [Rumbaugh et al. 2004], ele é uma representação da estrutura e relações das classes de um sistema, normalmente orientado a objetos [Booch et al. 2005].

O diagrama conceitual, como o próprio nome diz, está preocupado em mostrar os conceitos relacionados ao sistema, possuindo uma visão bastante simples, por isso é utilizado na fase de análise. Nesta fase do processo o que importa é o entendimento do sistema. A Figura 4.1 mostra um exemplo deste diagrama.

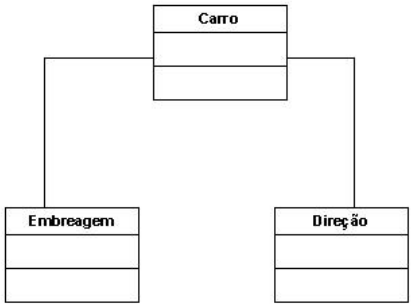

Figura 2.6: Diagrama de Classes Conceitual - Carro

Por outro lado, o diagrama de classes de implementação, mostra uma visão mais concreta do sistema, sendo dirigido para o desenvolvedor. Ele é construído na fase de projeto e se preocupa com a criação dos métodos e atributo das classes, pacotes do sistema, possíveis utilizações de padrões de projeto, entre outras coisas pertinentes a implementação do sistema. Na Figura 2.7 é apresentado o diagrama de implementação do exemplo anterior.

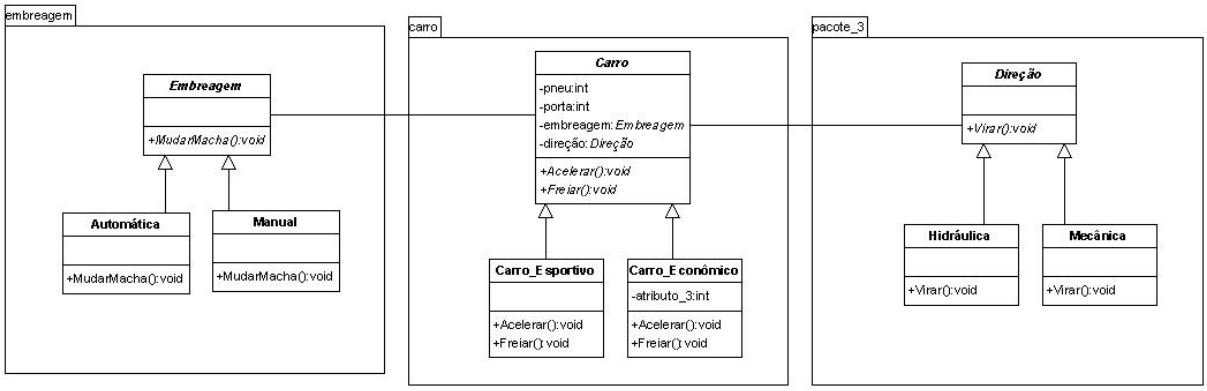

Figura 2.7: Diagrama de Classes de Implementação - Carro

2.5.2 Diagrama de caso de uso

Assim como o diagrama de classes, este diagrama também faz parte da UML. Ele descreve as funcionalidades propostas para um novo sistema que será projetado. O diagrama de caso de uso [Cockburn 2000] é um documento narrativo que descreve a seqüência de eventos de um ator que usa um sistema para completar um processo.

Este diagrama engloba quatro componentes:

- **Ator**: É um papel que tipicamente estimula/solicita ações/eventos do sistema e recebe reações. Cada ator pode participar de vários casos de uso;
- **Casos de uso**: Normalmente representadas por elipses, representam os requisitos funcionais do sistema;
- **Relacionamentos**: Representam os relacionamentos entre atores e funcionalidades;
- **Sistema**: Representação gráfica do sistema a ser modelado.

Os relacionamentos entre atores só podem ser do tipo herança, ou seja, um ator herda todas as funcionalidades de outro e pode ter novas funcionalidades. Já entre os casos de uso existem três tipos de relacionamentos:

- **Inclusão(*Include*)**: Se um caso de uso inicia ou inclui o comportamento de outro, dizemos que ele usa o outro;
- **Extensão(*Extends*)**: Define pontos de extensão que adicionam comportamento a um caso de uso base descrevendo uma variação do comportamento normal. O caso de uso base pode ser executado mesmo sem a extensão;
- **Generalização**: Indica um caso de base que possui diferentes especializações e inclui comportamento ou sobrescreve o caso de uso base.

A Figura 2.8 [Booch et al. 2005] mostra um exemplo de um diagrama de caso de uso que modela as funcionalidades e atores de um celular. Nela podemos encontrar dois autores (*Cellular network* e *User*) e cinco casos de uso (*Place phone call, place conference call, recive phone call, use scheduler* e *receive additional call*). O único relacionamento entre casos de uso existente é o *extends*. O relacionamento entre *Place phone call* e *placa conference call* significa que o último caso de uso possui todos os atributos do primeiro e ainda adiciona algum, em outras palavras, *placa conference call* extende *Place phone call*.

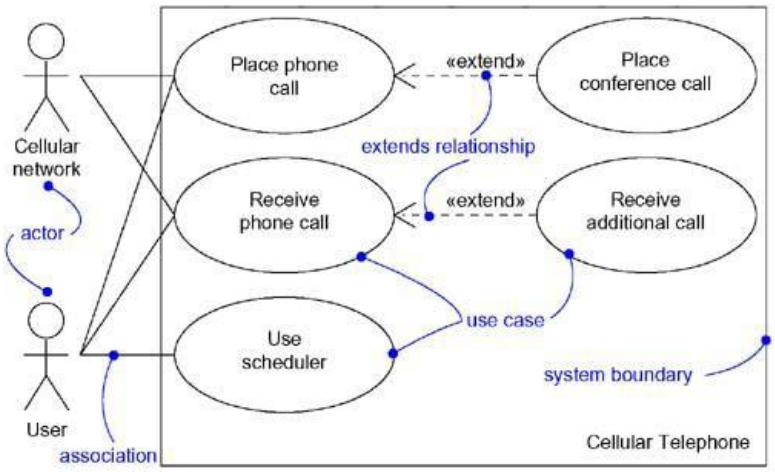

Figura 2.8: Diagrama de Caso de Uso - Celular

2.5.3 Diagrama de seqüência

Diagrama de seqüência, assim como os outros, é um diagrama usado em UML, representando a seqüência de processos num programa de computador. Como um projeto pode ter uma grande quantidade de métodos em classes diferentes, pode ser difícil determinar a seqüência global do comportamento. O diagrama de seqüência representa essa informação de uma forma simples e lógica.

Um diagrama de seqüência mostra uma interação como um gráfico bidimensional. A dimensão vertical é o eixo do tempo. A dimensão horizontal mostra as funções de classificação que representam objetos individuais na colaboração. Cada função do classificador é representada por uma coluna vertical, linha de vida. Durante o tempo que um objeto existe, o papel é representado por uma linha tracejada. Durante o tempo de uma ativação de um procedimento em que o objeto é ativo, a linha de vida é desenhada como uma linha dupla.

A mensagem é mostrada como uma seta de linha de vida de um objeto para o de outro. As setas estão organizadas em seqüência de tempo previsto no diagrama. A Figura 2.9 [Rumbaugh et al. 2004] mostra um diagrama de seqüência típico com mensagens assíncronas.

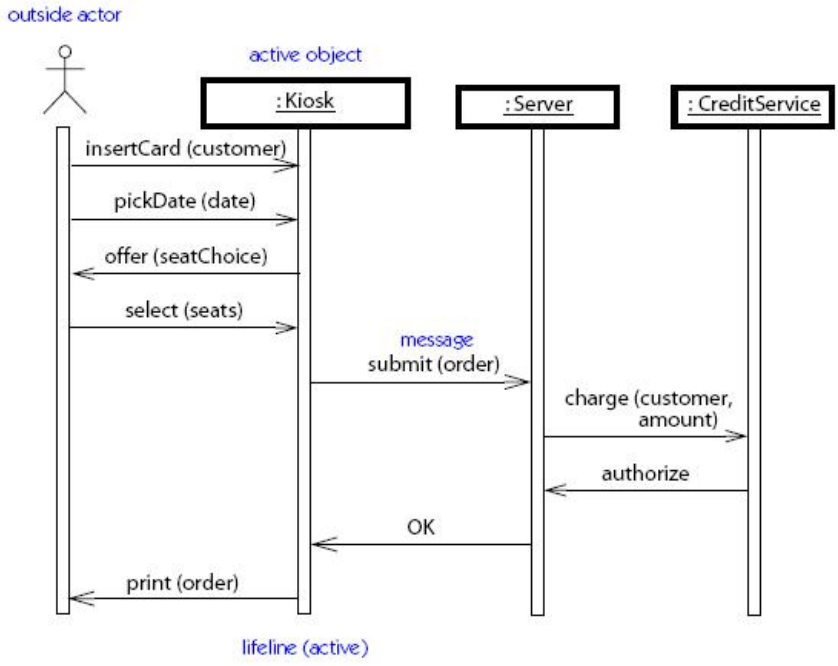

Figura 2.9: Diagrama de Seqüência

2.5.4 Modelo de *features*

Uma *feature* é uma propriedade do sistema que é relevante para algum *stakeholder* e é usado para capturar semelhanças e variabilidades entre os sistemas [Czarnecki et al. 2005]. Modelagem de *feature* é a atividade de identificar semelhanças e variabilidades dos produtos de uma linha de produtos, ou aplicações em *frameworks*, em termos de características e organizá-los em um modelo. Um diagrama de *feature* representa uma decomposição hierárquica de recursos, normalmente incluindo dois tipos de relacionamentos: agregação e generalização. O relacionamento de agregação é usado se um recurso pode ser decomposto em um conjunto de sub-funções, e a relação de generalização é usada quando um recurso pode ser especializado em outras mais específicas com informações adicionais [Lee and Kang 2004].

Os relacionamentos básicos entre as *features* podem ser de três tipos:

- Mandatário: Para existir uma instância da *feature* é obrigatório que exista uma instância da outra;
- Alternativo: Para existir uma instância da *feature* pode existir, no máximo, uma instância de um conjunto de *features*, podendo exitir nenhuma também;
- Opcional: Pode existir ou não uma instância da *feature* dependente, e caso exista, pode ser mais que uma.

Por exemplo, na Figura 4.2 [Lobo et al. 2007], para existir uma instância de *ATM System*, necessariamente precisam existir instâncias de *User Identification* e *Balance* as-

sociadas a ela(relacionamento mandatário). Por outro lado para existir o *User Identification* é necessário que seja escolhido apenas um dos subtipos *Touch Screen* e *Card Reader*(relacionamento alternativo). Por fim, para existir uma instância de *Balance* é necessário ter uma de *Display*(relacionamento mandatário) e pode ter ou não uma instância de *Printer*(relacionamento opcional).

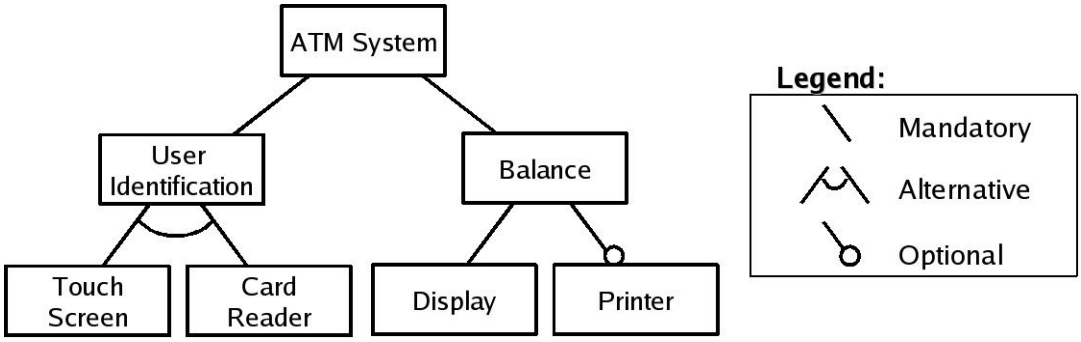

Figura 2.10: Modelo de *Features*

Capítulo 3

Trabalhos Relacionados e Tecnologias Utilizadas

Neste capítulo serão abordados alguns *crawlers* sociais que realizam atividades específicas nas ferramentas da web 2.0, em seguida serão descritos três blog *crawlers*, e será apresentada uma explanação sobre o trabalho proposto, comparando-o com os trabalhos relacionados que foram apresentados, principalmente a respeito de suas contribuições. Além disto, as ferramentas utilizadas para desenvolver o projeto serão caracterizadas.

3.1 *Crawlers* sociais

Uma das ferramentas mais populares da Web social são os wikis, que são softwares colaborativos permitem a edição coletiva dos documentos usando um sistema que não necessita que o conteúdo tenha que ser revisto antes da sua publicação [Wikipédia 2009c]. *Jeff Stuckman* e *James Purtilo* [Stuckman and Purtilo 2009] fizeram a proposta de um wiki crawler que recupera automaticamente e analisa wikis. Foi feito um estudo nas 151 páginas mais populares em execução na Mediawiki[1].

Outro serviço que está sendo largamente usado são as redes de relacionamentos [Wikipédia 2009b], sites como *Orkut, Friendster, LinkedIn, Facebook*, fazem parte desta rede. Elas são utilizadas para os mais variados propósitos, como aumentar a quantidade de amigos, procurar emprego, achar pessoas que tem interesse em comum, entre outros. Diante disto, podemos perceber que informações interessantes podem ser obtidas através deste serviço. Com isso, estão sendo desenvolvidos trabalhos para criação de *crawlers* que obtenham tais informações.

Em [Benevenuto et al. 2009] é proposto um estudo baseado em detalhar o fluxo de dados dos cliques. Foram coletados dados de 37.024 usuários que acessaram quatro populares redes sociais: Orkut, MySpace, Hi5, e LinkedIn. Os dados foram coletados a partir

[1] http://www.mediawiki.org/wiki/MediaWiki

de um site agregador de redes sociais no Brasil, que permite aos utilizadores ligar-se a várias redes sociais, com uma única autenticação. Este estudo revela as principais características das redes sociais, como a freqüência as pessoas se conectam as redes sociais e por quanto tempo, bem como os tipos e seqüências de atividades que os usuários conduta nesses sites.

Já em [Motoyama and Varghese 2009], foi desenvolvido desenvolvemos um sistema de busca e *matching* de indivíduos em redes sociais. Foi feito uma avaliação da eficácia da técnica proposta e os resultados foram comparados com os publicados em compete.com[2].

3.2 Blog *crawlers*

No contexto de blogs existem *crawlers* que foram desenvolvidos para melhor utilizar o poder desta ferramenta. Como já foi discutido, os blogs possuem grande potencial, que pode ser usado em diversas áreas, como: educação, comércio, ciência, entre outros.

Em *Social Streams Blog Crawler* [Hurst and Maykov 2009], *Matthew Hurst* e *Alexey Maykov* propõe uma arquitetura para blog *crawlers* que procura resolver alguns pontos específicos, como desempenho, escalabilidade e rápida atualização. Este *crawler* conseguiu alcançar desempenho interessantes, entretanto os resultados baseiam-se somente em busca sintática o que prejudeca a análise do contexto e a qualidade do resultado final.

Outro blog *crawler* que está sendo desenvolvido é o *BLOGalyzer* [Eckert 2008], que tem como objetivo agregar vários blogs para estudar a propagação do fenômeno meme [Dawkins 1990] na Blogosfera. Este *crawler* utiliza as APIs de busca do Google e do Yahoo, para recuperar os blogs que serão indexados. Este trabalho também não faz um tratamento semântico das buscas.

Natalie Glance, Matthew Hurst e *Takashi Tomokiyo* em *BlogPulse: Automated Trend Discovery for Weblogs* [Glance et al. 2004] descrevem uma aplicação para analisar coleções de blogs, utilizando técnicas de aprendizagem de máquina e processamento de linguagem natural. Este sistema disponibiliza informações, como: frases mais citadas, pessoas chave, blogs mais acessados, entre outros. O sistema de busca e um analisador de tendências são duas ferramentas que ainda estão em desenvolvimento.

3.3 Comparação entre os blog *crawlers* e a proposta do trabalho

Os três blog *crawlers* supracitados possuem características bem específicas, o primeiro busca resolver, principalmente, problemas de desempenho. O segundo faz parte de um trabalho bastante direcionado, estudo sobre a propagação do meme na Blogosfera, e o

[2] http://compete.com/

último se preocupa em descobrir aspectos interessantes sobre a blogosfera, como: frases mais citadas, pessoas chave, blogs mais acessados.

Apesar de todos explorarem aspectos importantes e interessantes da blogosfera, eles não estão ligados a questão da melhor recuperação dos blogs. Para [Glance et al. 2004] a recuperação dos blogs é muito importante, porém são utilizadas métricas que não aumentam o nível da descrição dos blogs, ou seja, para este trabalho um blog interessante é, por exemplo, um blog bastante acessado.

Por outro lado, a proposta deste trabalho é aumentar o nível semântico dos blogs para melhorar a resposta das buscas feitas por usuários. Para isso são utilizados serviços de marcação. Com isso, a recuperação de informação na Blogosfera deixa de ser puramente sintática ou baseada em métricas como blog mais acessado.

3.4 Tecnologias Utilizadas

Esta seção tem por objetivo analisar, de forma rápida, as ferramentas utilizadas para realização deste projeto.

3.4.1 Lucene

O Lucene [Lucene 2001] é um software que contém uma biblioteca de códigos reutilizáveis desenvolvido pela organização Apache. Entre outras características, o Lucene fornece um alto desempenho na indexação e eficiente algoritmo de pesquisa e pode realizar muitas tarefas tais como, por exemplo:

- Pesquisa por classificação: os melhores resultados aparecem primeiro;
- Muitos tipos de consultas: consultas por frases, consultas com operadores léxicos avançados, consultas por proximidade;
- Pesquisa por atributos (título, autor, conteúdo);
- Pesquisa por faixa de dados;
- Ordenação por campo;
- Atualizações e pesquisas simultâneas.

Para o Lucene não importa a origem dos dados, seu formato ou mesmo a linguagem em que foi escrito, desde que esses dados possam ser convertidos para texto. Isto significa que o Lucene pode ser utilizado para indexar e buscar dados gravados em: arquivos, páginas web em servidores remotos, documentos gravados no sistema de arquivos local, arquivos

textos, documentos Microsoft Word, documentos HTML, arquivos PDF, ou qualquer outro formato do qual possa ser extraído informação textual.

Além disso, o Lucene é desenvolvido em Java, existem várias aplicações que o utilizam, é código aberto e possui uma vasta documentação em vários idiomas, incluindo um livro [Hatcher and Gospodnetic 2004] que detalha vários aspectos da ferramenta.

3.4.2 Tecnhorati API

A *Tecnhorati* [Technorati 2009b] foi fundada para ajudar os blogueiros a ter sucesso em procurar e divulgar na seus posts na Blogosfera. Fundada como o primeiro motor de busca do blog, *Technorati* expandiu-se para uma completa empresa de serviços de comunicação que prestam serviços para os blogs e sites de mídia social e conecta eles com os anunciantes que querem se juntar à conversa.

O motor de busca de blog, Technorati.com, indexa milhões de posts em tempo real. O local tornou-se a fonte definitiva para as principais notícias, opiniões, fotos e vídeos emergentes sobre notícias, entretenimento, tecnologia, estilo de vida, esportes, política e negócios. Technorati.com não está interessada somente na autoridade e influência dos blogs, mas também nos índices mais abrangentes e atuais sobre quem e o que é mais popular na Blogosfera.

Para que desenvolvedores possam utilizar esses serviços em seus programas a *Tecnhorati* disponibiliza uma API(*Application Programming Interface*) que funciona no estilo REST(*Representational State Transfer*), que é uma técnica de engenharia de software para sistemas hipermídia distribuídos como a *World Wide Web*. A requisição à API deve ser feita através de um cliente HTML e o seu retorno é um XML padronizado pela própria *Tecnhorati*.

3.4.3 *HTTP Client*

O HTTP(*Hyper-Text Transfer Protocol*) é talvez o protocolo mais importante usado na Internet hoje. Web Services, rede de eletrodomésticos e o crescimento da computação em rede continuam a expandir o papel do protocolo HTTP além do acesso via web browsers. Embora o pacote java.net forneça a funcionalidade básica para acessar recursos via HTTP, não oferece a flexibilidade total ou funcionalidade necessária por muitas aplicações. O pacote *Jakarta Commons HttpClient* visa preencher esta lacuna [Apache 2005].

Projetado para ser extensível, proporcionando suporte robusto para o protocolo HTTP base, o componente *HttpClient* podem ser de interessante para construir qualquer aplicação com cliente HTTP, tais como navegadores web, clientes de serviço Web, ou sistemas que alavancam ou prorrogar o protocolo HTTP para comunicação distribuída.

Há muitos projetos que usam o *HttpClient* para fornecer funcionalidades HTTP. Al-

guns deles são código aberto com páginas do projeto que você pode encontrar na web, enquanto outros são de código fechado. A Licença *Apache Source* proporciona a máxima flexibilidade para a reutilização de códigos e binários. Algumas características do *HTTP Client* são:

- Implementado em Java;
- Implementa todos os métodos HTTP (*GET, POST, PUT, DELETE, HEAD, OPTIONS,* e *TRACE*) de forma extensível e orientada a objeto;
- Dá suporte ao protocolo HTTPS;
- Faz conexões transparentes através dos *proxies* HTTP;
- Implementa o padrão Command para dar suporte a requisições paralelas e para otimizar o reuso das conexões.

3.4.4 Google API - idiomas

O Google disponibiliza várias APIs para desenvolvedores utilizarem programas testados e consolidados por milhões de usuários entre elas está a API de idioma [Google 2009].

A API AJAX de idioma disponibiliza serviços de tradução e detecção de idioma dos blocos de texto dentro de uma página Web ou através de uma requisição REST. Além disso, é possível habilitar a transliteração, que é o processo de conversão fonética de uma palavra escrita em um script para outro, de qualquer campo ou área de texto da sua página web. A API de idioma foi criada para ser simples e fácil de usar, para traduzir e detectar idiomas durante o uso, quando não houver traduções off-line disponíveis.

A API é fácil de utilizar e os resultados são altamente satisfatórios, por isso ela foi usada para detecção de idiomas no trabalho proposto.

3.4.5 MySQL

O MySQL [Sun 1995] é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês *Structured Query Language*) como interface. Atualmente este sistema é largamente utilizado por várias empresas. Algumas características importantes:

- Portabilidade, suporta vários tipos de plataforma;
- Ótimo desempenho;
- Estável;
- É um software livre;

- Replicação facilmente configurável;
- Possui uma larga documentação, incluindo muitos tutoriais.

3.4.6 Hibernate

O Hibernate [Middleware 2009] é um Framework para o mapeamento objeto-relacional escrito na linguagem Java. Este programa facilita o mapeamento dos atributos entre uma base tradicional de dados relacionais e o modelo oriantado a objeto de uma aplicação, mediante o uso de arquivos (XML) ou de anotações para estabelecer esta relação.

A grande vantagem de utilizar o Hibernate é que ele permite que você desenvolva classes persistentes seguindo o paradigma orientado a objetos, incluindo associação, herança, polimorfismo, composição, coleções, entre outros. Hibernate permite que você expresse consultas em sua própria extensão SQL (HQL), assim como em SQL nativo, ou com um objeto chamado *Criteria*.

Utilizando o Hibernate, pode-se contar com as vantagens de inúmeros SGBDs com as facilidades de estar usando um programa orientado a objeto. Com isso o tempo de desenvolvimento diminui.

Capítulo 4

Framework Proposto

Esta capítulo apresenta de forma detalhada as características do *framework* proposto, assim como o funcionamento do algoritmo desenvolvido para extração de texto a partir de páginas HTML.

4.1 Descrição do *framework*

O sistema é um *framework*, caixa branca, que permite a criação de blog crawlers baseados em contexto, ou seja, baseados em marcação feita através de tags. Nas próximas seções a proposta será detalhada.

4.1.1 Requisitos e visão geral

O *framework* disponibiliza serviços como: pré-processamento, indexação, extração de textos das páginas HTML, serviços de utilidades (manipulação de arquivos, requisições http, manipulação de arquivos XML, entre outros) e comunicação com a persistência.

Os *Hot Spots* do *framework* são os serviços:

- **Pré-Processamento**: Serviços que tratam o texto do blog para retirar informações com pouca relevância para a análise. Já estão implementados pré-processamentos de quatro tipos: i) *CleanHTML* [Hotho et al. 2005], responsável por limpar todos os HTMLs de um texto; ii) *EnglishFiltering* [Frakes and Baeza-Yates 1992], técnica de filtragem (remoção de stop words) para língua inglesa; iii) *EnglishStemming* [Porter 1980], técnica de stemming (reduz a palavra para a sua forma primitiva) para língua inglesa; iv) *WhiteSpace*, retira espaços brancos desnecessários;
- **Indexação**: Serviços que vão indexar o texto para facilitar uma futura busca. Já está implementado o serviço *LuceneIndexing*, que provê indexação utilizando a ferramenta Lucene [Lucene 2001];

- **Extração de textos das páginas HTML**: Serviço que extrai o texto do blog/post propriamente dito de uma página HTML. Já esta implementado o *SummaryStrategy*, algoritmo proposto no trabalho;
- **Comunicação com a persistência**: Serviços que vão ser responsáveis por operações como: salvar, recuperar, atualizar as instâncias em um banco de dados. Já está implementado um serviço de comunicação com banco de dados MySQL.
- **Pacote Application e Pacote TagParser**: Pacotes que precisam ser estendidas pelo usuário para criar uma aplicação utilizando o *framework*.

Os *Frozen Spots* são:

- **Serviços de utilidades**: Implementa serviços básicos para o *framework*, como: manipulação de arquivos, requisições HTTP, manipulação de arquivos XML, entre outros;
- **Blog Crawler**: Núcleo do framework. Possui os passos necessários para o processo de crawler dos blogs;
- **Pacote Data**: Possui o formado de dados dos blogs, como eles são estruturados.

4.1.2 Diagramas de classes conceitual

Para facilitar a explicação da proposta, a Figura 4.1 mostra o diagrama conceitual do sistema. Nele pode-se identificar o núcleo do *framework*, a classe *BlogCrawler* é o principal *frozen spot* da aplicação, responsável por gerenciar todo o processo do crawler. Outro ponto fixo é o conjunto de classes de utilidades, que disponibilizará serviços básicos, na figura está representada pela interface *Utilities*. Da mesma forma, existem esboços dos *hot spots*, por exemplo, as classes *Indexing* e *Preprocessing*, a primeira é responsável por disponibilizar serviços de indexação, enquando a segunda serviços de pré-processamento. Entretanto, alguns pontos de variação ainda não foram identificados nesta etapa. Por fim, as classes *Application* e *TagParser* estão diretamente ligadas ao núcleo, principalmente a classe *TagParser* a qual o núcleo tem uma relação de dependência forte, estas classes são a parte que o usuário tem que instanciar, obrigatoriamente, para criar uma aplicação usando o *framework* proposto. A classe *Item* é o tipo de dados utilizado pelo sistema para manupular os blogs. Esta classe contém, por exemplo, atributos como link do blog, resumo do post, texto completo dos post, entre outros. Esta classe ficou isolada pois ela está ligada à todas as outras, por isso neste diagrama foi colocada apenas uma observação, no diagramas de classes de implementação este aspecto será melhor detalhado posteriormente.

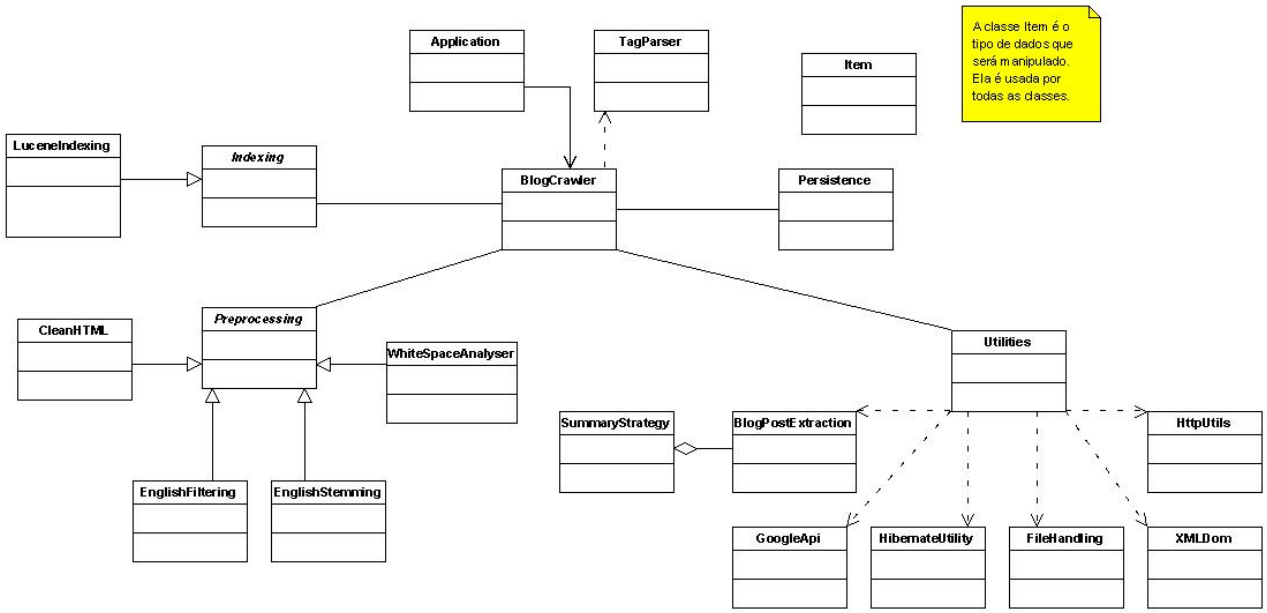

Figura 4.1: Diagrama de Classes Conceitual

4.1.3 Modelo de features

O modelo de *features*, vide Figura 4.2, facilita a percepção dos *hot spots* e suas características.

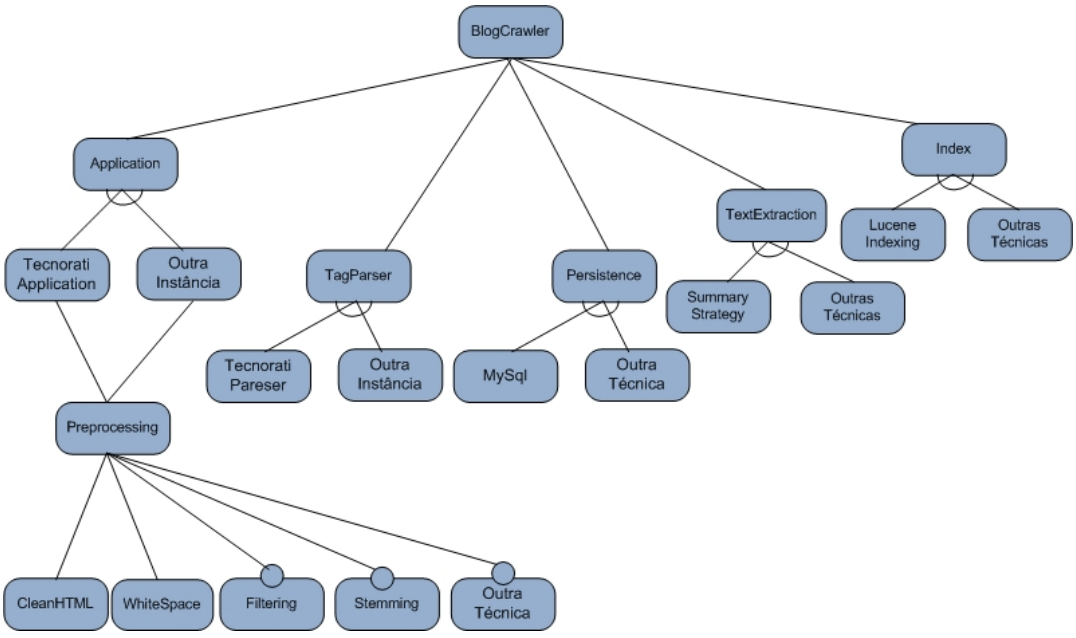

Figura 4.2: Modelo de Features

Os pontos interessantes deste diagrama devem ser citados, como seguem:

- Os atriutos *Application*, *TagParser*, *Persistence*, *TextExtraction* e *Index*, são mandatários, ou seja, é necessário instância de todos eles;

- Os subtipos das classes supracitadas são todos alternativos, ou seja, deve ser instanciado apenas um objeto de cada atributo;
- Nos atributos filhos de *Preprocessing*, existem dois mandatários e os outros são opcionais, ou seja, podem ser instanciados quantos forem necessários, se for o caso não precisão ser criados.

O modelo de *features* pode definir o modelo de decisões. Esse modelo apresenta possíveis instanciações e configurações do *framework* no contexto de diferentes cenários(produtos).

4.1.4 Diagrama de caso de uso

No diagrama de caso de uso, vide Figura 4.3, as funcionalidades que o usuário pode acessar são melhor visualizadas, assim como as dependências de algumas dessas atividades.

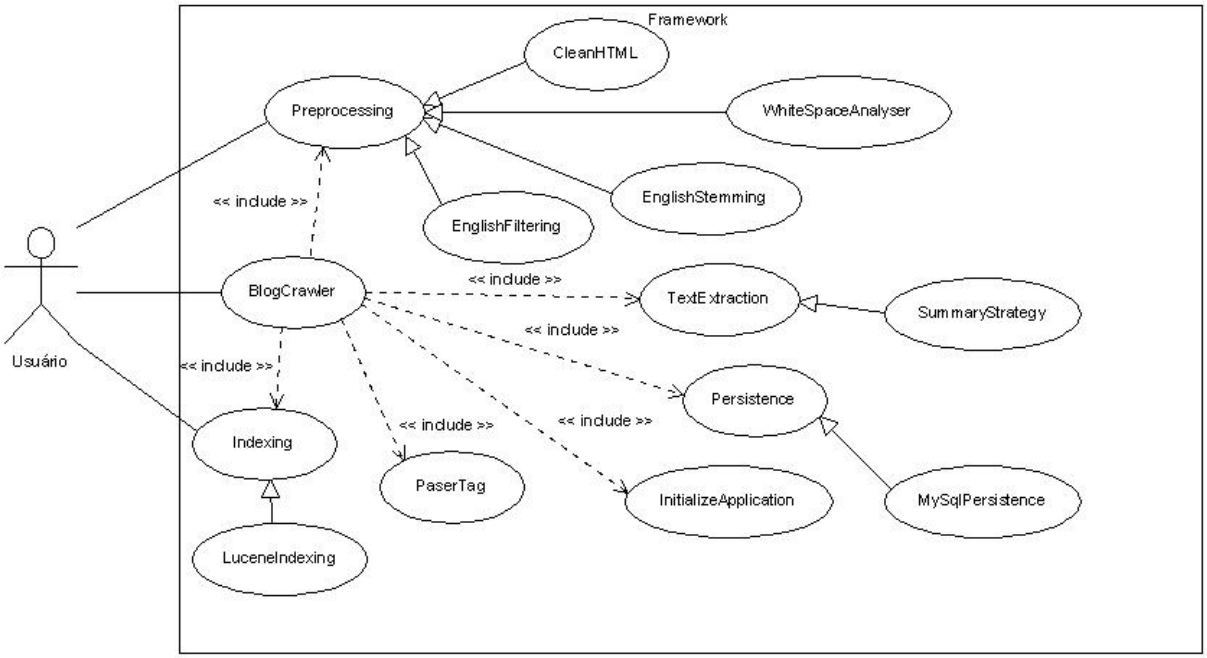

Figura 4.3: Diagrama de Caso de Uso

Como o diagrama mostra, o usuário pode utilizar diretamente três serviços do sistema: pré-procesamento, indexação e criação do blog *crawler*. Sendo que destes, o último é o principal objetivo do *framework*. Os outros facilitam a criação de outras aplicações, por exemplo, o *framework* pode ser utilizado para pré-processar arquivos de texto apenas com uma chamada de método. Estes serviços não possuem pré-requisitos.

Por outro lado, o serviço de criação de blog *crawler* necessita de alguns pré-requisitos, representados na Figura 4.3 como os casos de uso *include*, para ser executado. Para utilizar este serviço os seguintes serviços deverão ser instanciados:

1. *Indexing*;
2. *ParserTag*;
3. *InitializeApplication*;
4. *Persistence*;
5. *TextExtraction*;
6. *Preprocessing*.

4.1.5 Diagramas de classes de implementação

Este diagrama acrescenta aspectos de implementação no diagrama de classes conceitual, aumentando a granularidade do mesmo para facilitar o trabalho do desenvolvedor. Como o diagrama ficaria muito grande, ele foi dividido em várias partes que serão explicadas ao longo do trabalho. A Figura 4.4, mostra a visão de pacotes. Apesar de ser uma visão alto nível do sistema, aspectos importantes são identificados, como as dependências entre os pacotes.

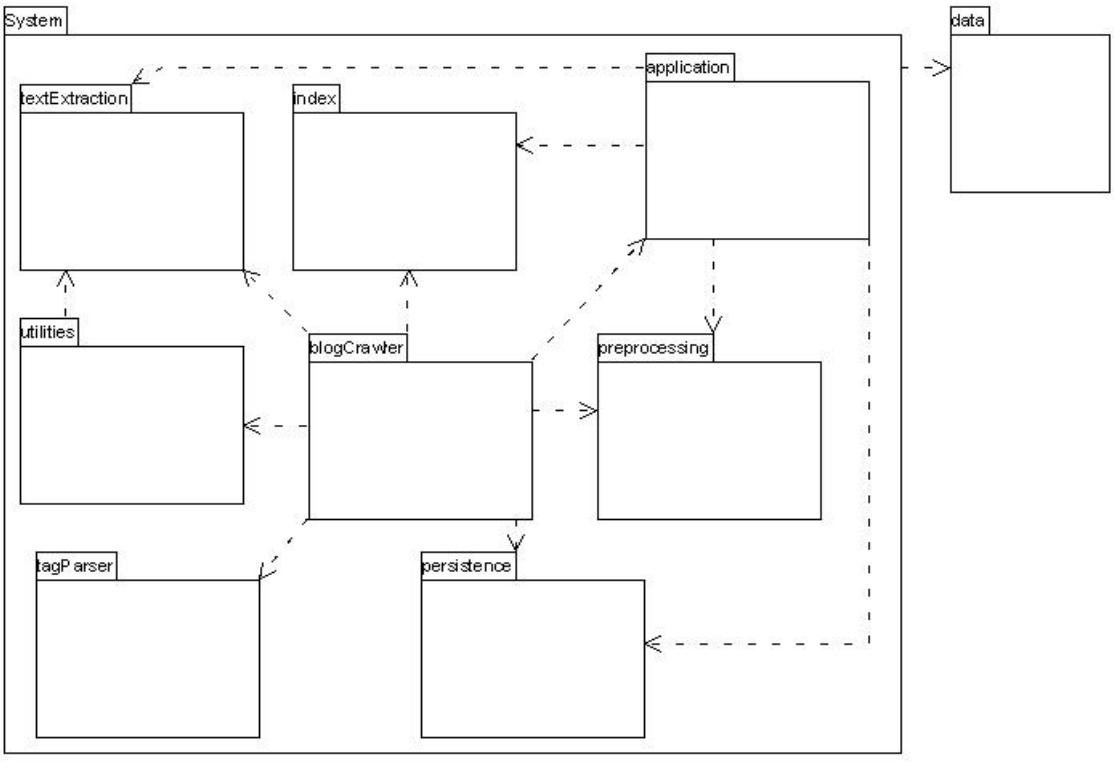

Figura 4.4: Diagrama de Classes de Implementação - Visão de Pacotes

Pode-se identificar, ainda neste diagrama, que o pacote *blogCrawler* é o núcleo do *framework*, pois todos os outros pacotes estão ligados a ele. Outro fator importante é que

o pacote *data* está ligado ao pacote *system*, isto se deve ao fato de que este pacote contém o tipo de dados que será trafegado pelas classes da aplicação.

As Figuras 4.5 e 4.6, representam o diagrama dos principais *hot spots* do *framework*, pois para criar uma aplicação eles têm que ser estendidas pelo usuário, o que será melhor explicado na Seção 4.1.7.

A classe *BlogCrawler* possiu uma dependência de ambos os pacotes, mais precisamente do controlador de cada um. É importante frisar que os padrões *Factory Method*, atráves das classes *ApplicationFactory* e *TagParserFactory*, e *Command*, implementado nas classes *ApplicationControler* e *TagParserControler*, são aplicados em todos os *hot spots*(mais detalhes sobre os padrões serão comentados no apêndice A).

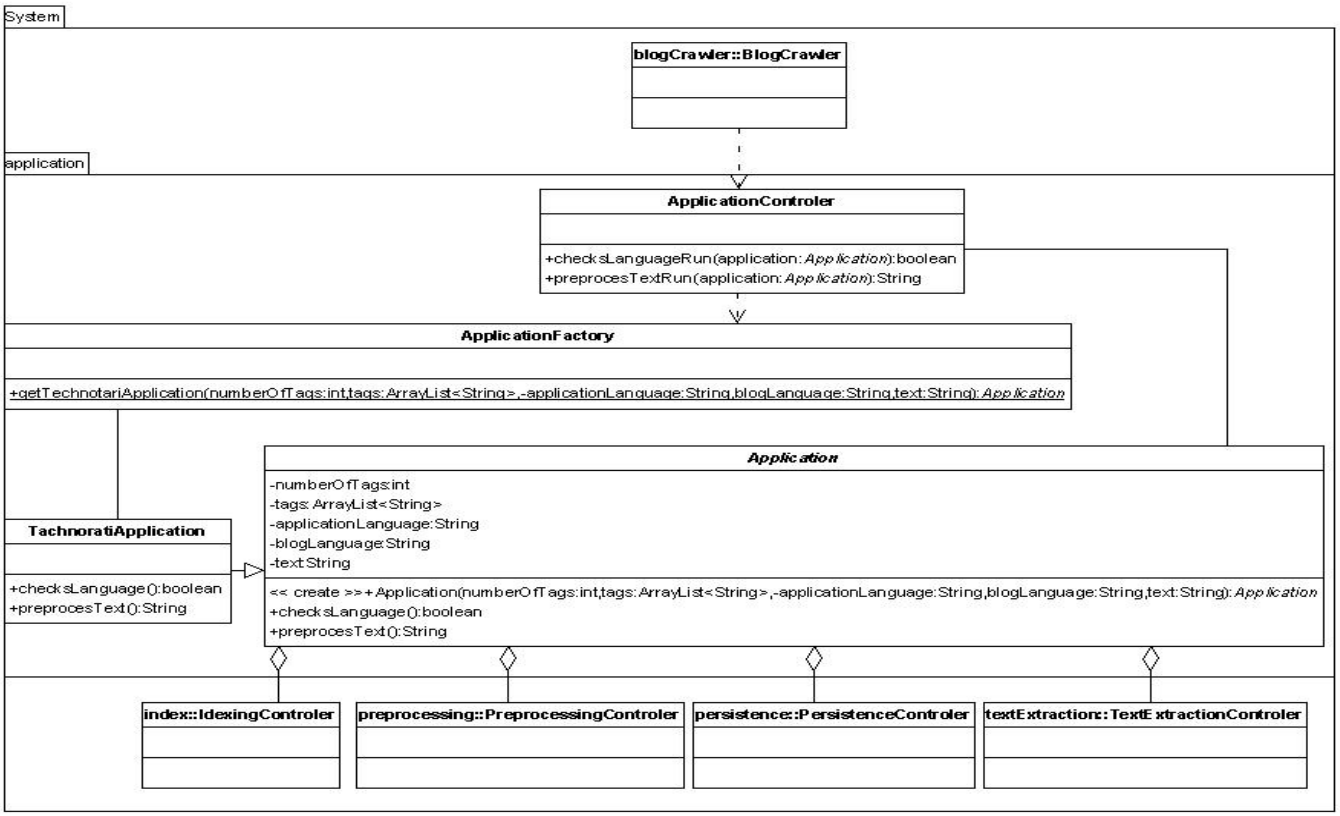

Figura 4.5: Diagrama de Classes de Implementação - *Application*

A diferença entre estes dois pacotes é que o *application* possui variáveis que instanciam outros *hot spots*. Com isso as funcionalidade de cada pacote são melhor caracterizadas. Desta forma, o pacote *application* vai ser utilizado para interagir com os outros *hot spots* do *framework*, o pacote *tagParser* vai ser responsável por recolher informações sobre o blog(mais detalhes serão dados no capítulo 5).

Frisa-se ainda que já estão implementados no sistema, tanto no pacote *application* quanto no *TagParser*, classes que criam uma aplicação utilizando blogs indexados pela *Technorati*, como será mostrado no capítulo 5.

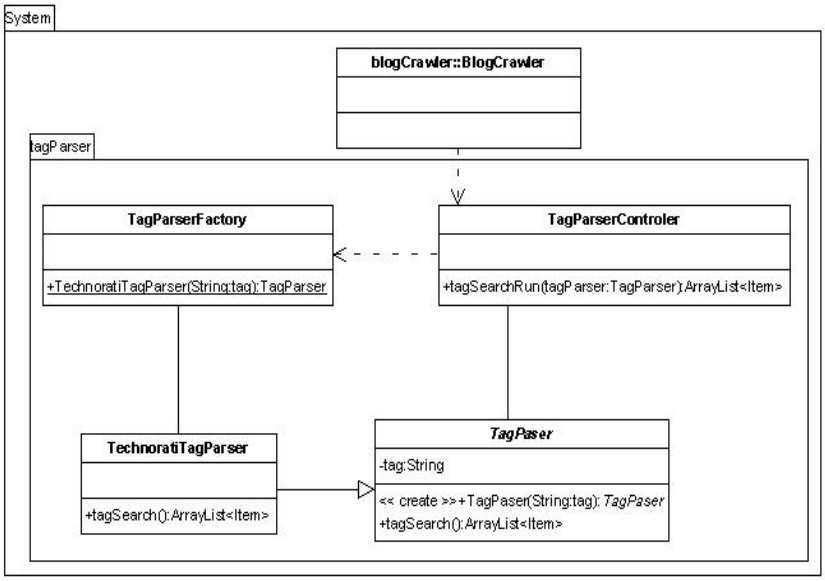

Figura 4.6: Diagrama de Classes de Implementação - *TagParser*

Os diagramas a seguir, Figuras 4.7, 4.8, 4.9 e 4.10, mostram os *hot spots* inerentes ao sistema. Assim como os dois pacotes descritos acima, eles possuem os padrões *Factory Method* e *Command*. Maiores detalhes dos padrões de projeto utilizados estão descritos no Apêndice A.

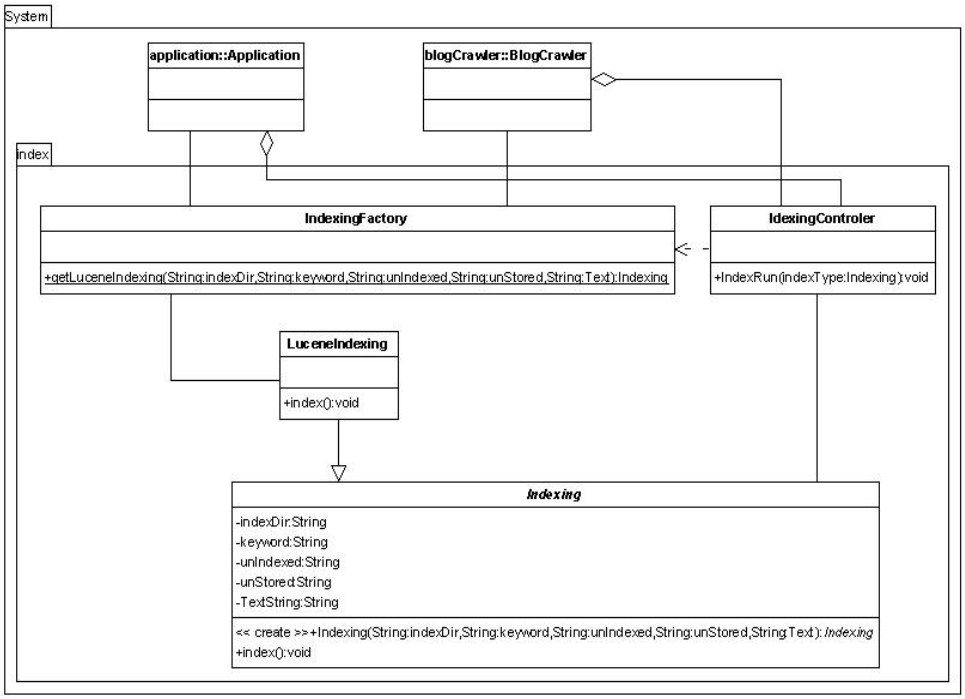

Figura 4.7: Diagrama de Classes de Implementação - *Index*

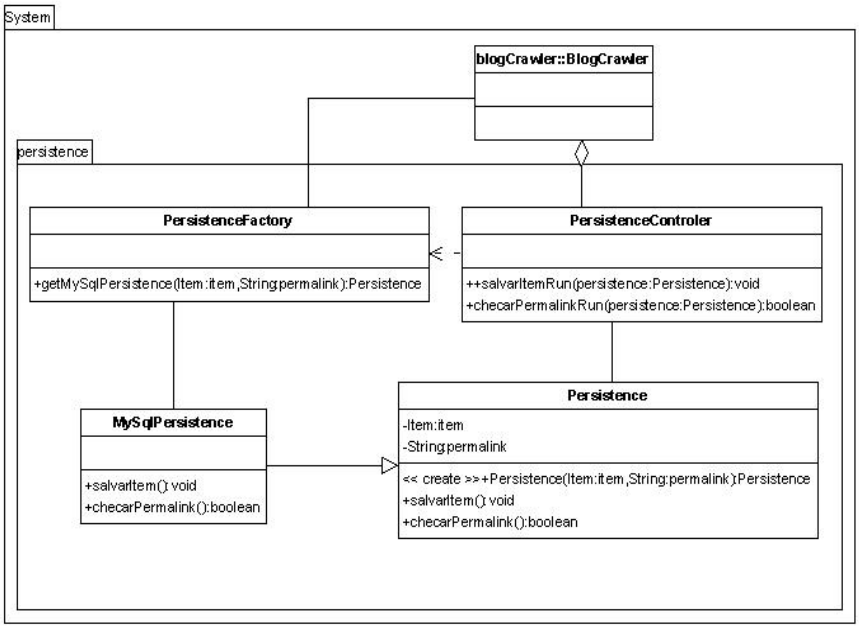

Figura 4.8: Diagrama de Classes de Implementação - *Persistence*

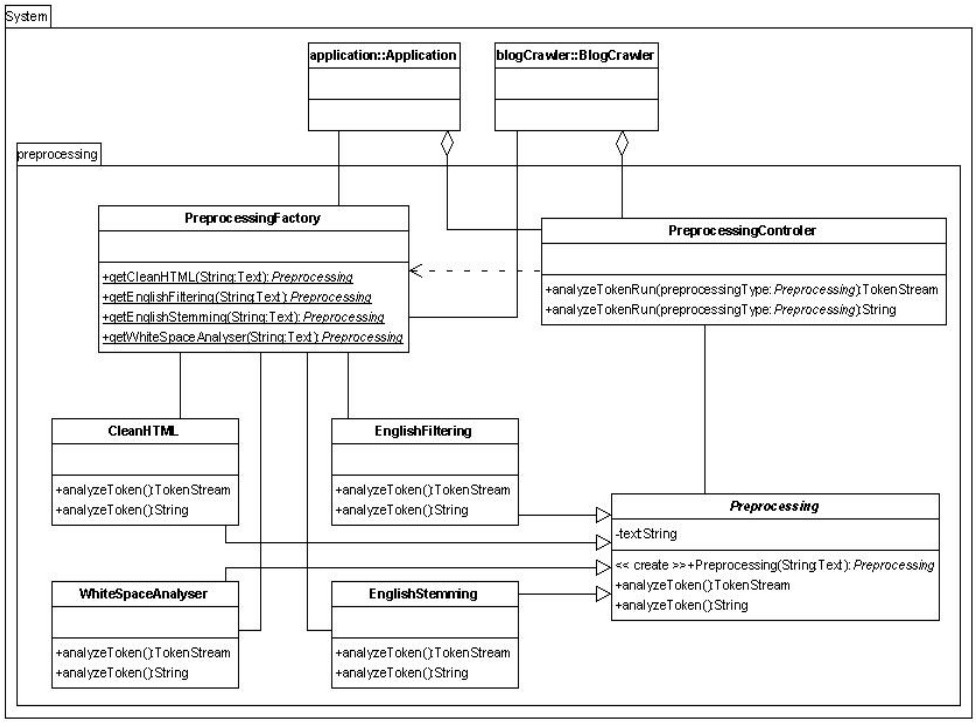

Figura 4.9: Diagrama de Classes de Implementação - *Preprocessing*

Os serviços de indexação e pré-processamento podem ser usados diretamente pelo usuário para outras aplicações que envolvam mineração de texto, entretanto foi considerado desnecessário deixar o serviço de persistência como um serviço de acesso livre para o usuário, visto que seus métodos são bastante direcionados para o *framework* proposto.

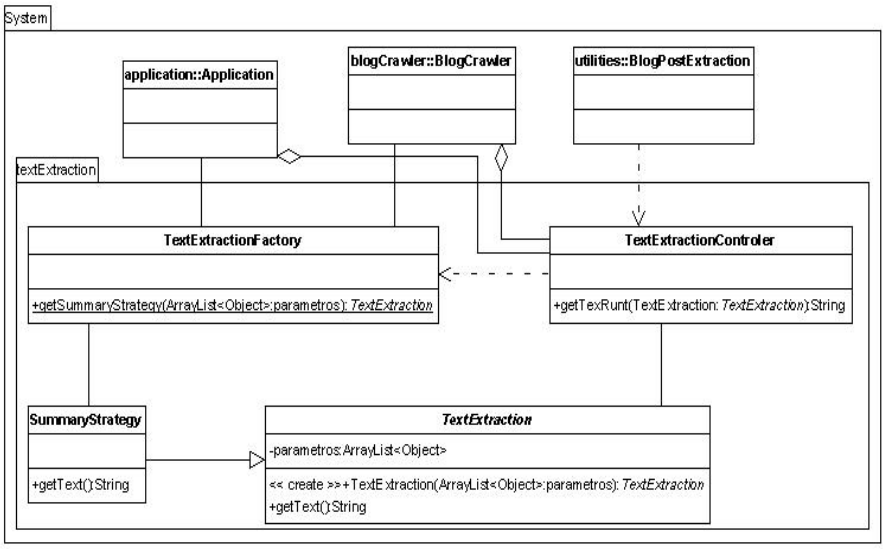

Figura 4.10: Diagrama de Classes de Implementação - *TextExtraction*

As Figuras 4.7, 4.8, 4.9 e 4.10 mostram os serviços (descritos em 4.1.1) que estão disponibilizados em cada pacote, são eles: i)Pacote *index*: serviço de indexação com a ferramenta lucene(*LuceneIndexing*); ii)pacote *persistence*: serviço de persistência utilizando o MySQL(*MySqlPersistence*); iii)patoce *preprocessing*: serviços para limpar HTMLs do texto(*CleanHTML*), filtragem (*EnglishFiltering*) e *stemming*(EnglishStemming) em textos em inglês e retirada de espaços desnecessários do texto(*WhiteSpaceAnalyzer*); iv)pacote *textExtraction*: serviço de extração de texto a partir de uma página HTML (*SummaryStrategy*).

Além de já possuir todos esses serviços implementados, novos podem ser facilmente incluídos no *framework*, por exemplo para criar um novo serviço de pré-processamento é necessário apenas estender a classe *Preprocessing* e criar um novo método de *factory* em *preprocessingFactory*. O mesmo serve para os outros *hot spots*. Por fim, uma observação deve ser feita sobre o pacote *textExtraction* (vide Figura 4.10), onde além das relações com *Application* e *BlogCrawler*, ele também possui uma relação de dependência com a classe *BlogPostExtraction*, esta relação acontece pelo fato de que para executar o seu principal método a classe *BlogPostExtraction*, precisa de uma instância de *textExtraction*.

O pacote de utilidades (vide Figura 4.11) disponibiliza serviços básicos que serão utilizados pela maioria do sistema. A princípio este conjunto de classes é um *frozen spot*, onde novos serviços podem ser adicionados a ele, porém é aconselhado que estes sejam implementados em classes já exitentes no pacote e suas novas funcionalidades devem ser adicionadas a interface *Utilities*.

O padrão *Facade* é implementado através da interface *Utilities* e da classe *UtilitiesControler*, a última é uma classe "delegadora", ou seja, simplesmente recebe a requisição e repassa para a classe que realmente implementa o método desejado. Desta forma, o

usuário não precisa se preocupar com o tipo da classe que possui a funcinonalidade que ele deseja executa, pois basta chamar a fachada.

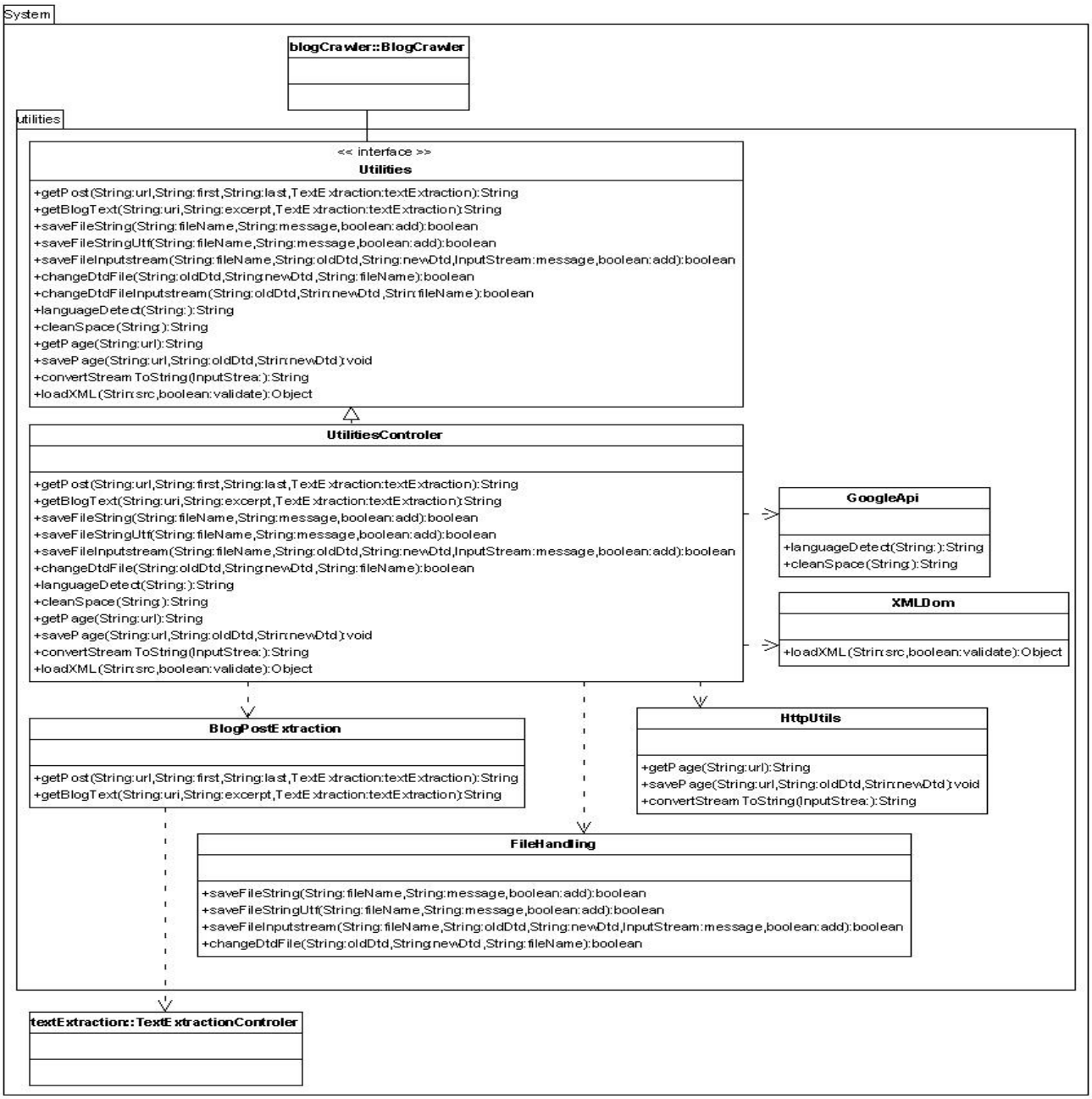

Figura 4.11: Diagrama de Classes de Implementação - *Utilities*

O pacote *blogCrawler* é o mais importante do sistema, pois nele está contido o núcleo do *framework*. Na classe *BlogCrawler* se percebe algumas características inatas de um *framework*, como o controle de fluxo. Como está descrito na Figura 4.12 a classe em questão esta conectada a quase todos os outros pacotes, confirmando assim que ela é o núcleo da aplicação.

A classe *BlogCrawler* possui uma relação de composição com quase todos os *hot spots*, com exeção do *Application* e *TagParser*, com estes existe uma relação de dependência,

pois para executar seus métodos precisa de um dos tipos acima como entrada. Além disto, ela possiu ligação direta com as classes de utilidades e com as *factories* dos *hot spots*[1].

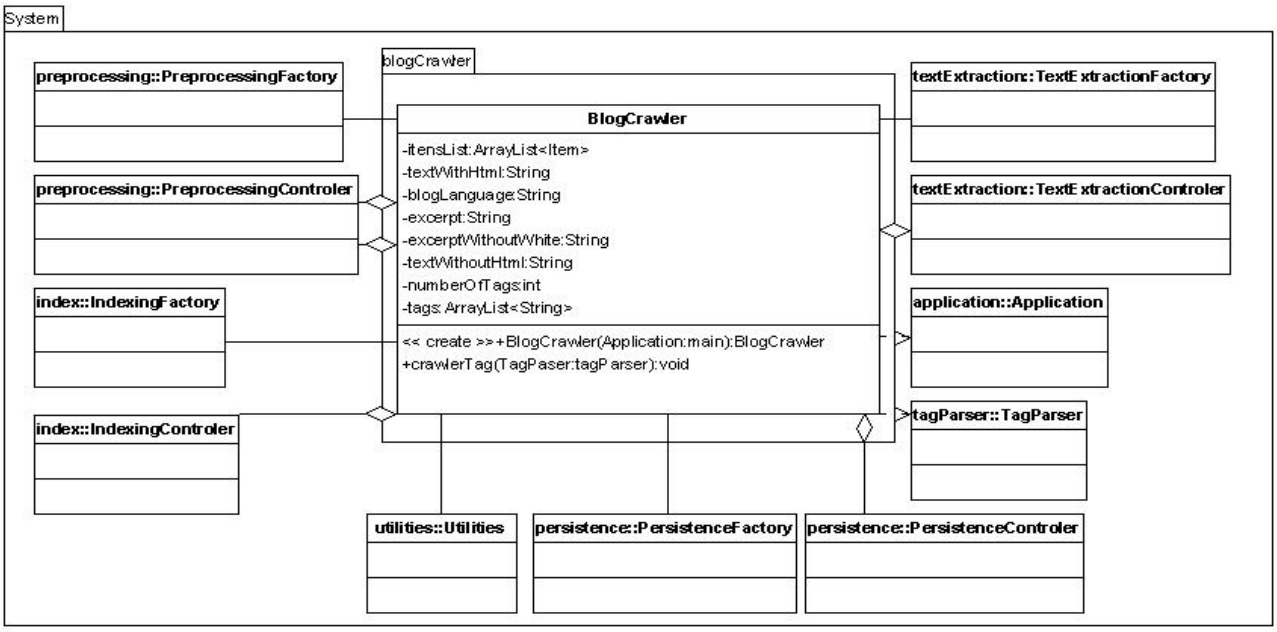

Figura 4.12: Diagrama de Classes de Implementação - *BlogCrawler*

Por fim, o pacote *Data*. Este é o pacote mais simples de todos, possui apenas os tipos de dados usados no sistema. Por enquanto, somente o tipo Item, que contém informações sobre um determinado post do blog, como: resumo do texto, texto completo, data em que foi publicado, entre outros. Mesmo sendo um pacote muito simples, o pacote *Data* foi criado para facilitar a organização e garantir mais extensibilidade do *framework*.

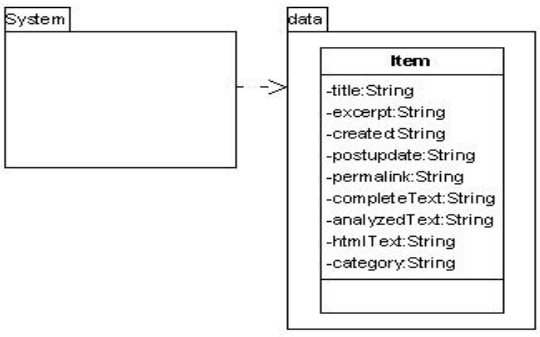

Figura 4.13: Diagrama de Classes de Implementação - *Data*

Uma obesrvação importante é que todas as classes que possuem atributos tem os métodos *get* e *set* dos mesmo, eles não foram colocados na modelagem simplesmente para não poluir o diagrama.

[1]Na fase de análise, diagrama conceitual, ainda a classe *Application* ainda não era considerada uma dependência, possuia apenas uma simples associação. Já na fase de projeto, com o detalhamento dos requisitos e das funcionalidades, foi que ela passou a ser uma dependência.

4.1.6 Diagrama de sequência

O diagrama de sequência é interessante para mostrar uma característica inerente de um *framework*, a inversão de controle. Existem quatro classes envolvidas neste diagrama, duas são *frozen spot*, *BlogCrawler* e *Utilities*, e as outras são instanciadas pelo usuário quando vai especificar a aplicação, *Application* e *TagParser*.

A Figura 4.14 mostra a grande quantidade de interações entre as classes citadas. O núcleo do *framework* possui a sequência de operaçãos que devem ser feitas, mas grande parte delas são feitas pelo usuário, aumentando assim o nível de variabilidade do sistema. Por exemplo, no início da interação a aplicação requisita que o *crawler* seja criado, como resposta obtém uma requisição para que algumas variáveis sejam inicializadas por ela. É importante perceber que algumas operações (como a 10 do diagrama) são de responsabilidade de quem está instanciando o *framework*, porém elas utilizam serviços do próprio sistema para serem realizados. Ou seja, para pré-processar o texto a aplicação vai simplesmente escolher qual o tipo de pré-processamento desejado e chamar um método do *framework* para executá-lo.

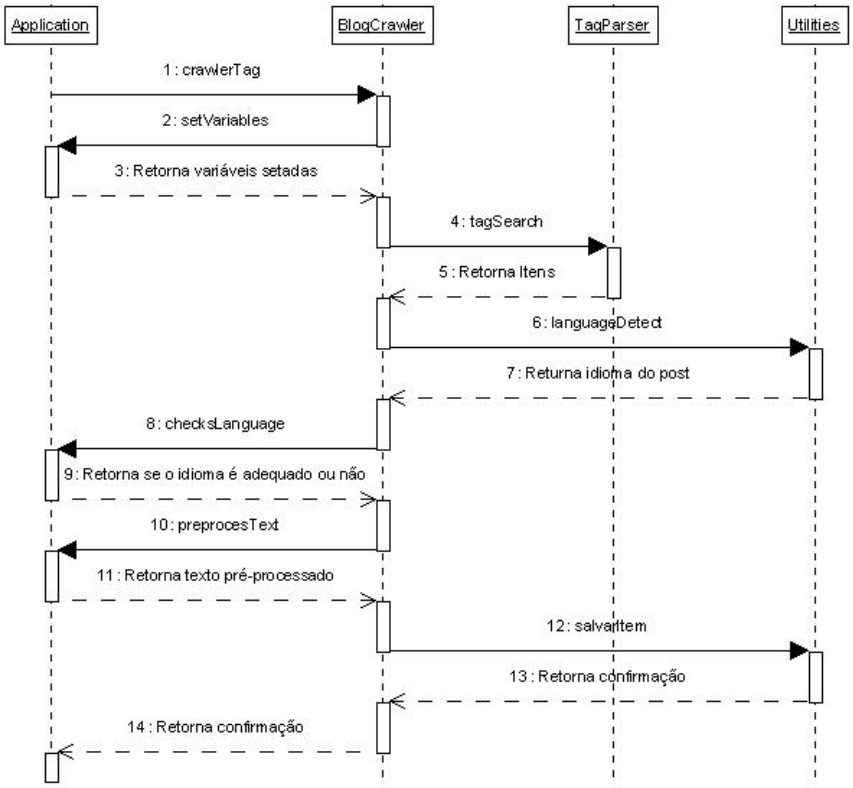

Figura 4.14: Diagrama de Sequência

4.1.7 Instanciação do *framework*

Para criar uma aplicação utilizando o *framework* o usuário deve estender as classes *Application* e *TagParser*, como será mostrado na Seção 4.1.5. A primeira possui dados para configuração da aplicação, já a outra precisa que o usuário retorne de alguma forma pelo menos 3 atributos do blog: o link do post/blog, o resumo do post/blog e a categoria(tag) a que ele pertence. Estender a segunda classe é mais complexo, contudo, existem empresas[2]que trabalham com blogs, como foi citado na introdução, que normalmente disponibilizam serviços de tags. Então o trabalho é basicamente escolher uma dessas empresas e fazer um parser nas informações que ela disponibiliza. A princípio, só é aconselhado utilizar dados de uma mesma fonte, pois a utilização de várias fontes pode causar inconsistências, por exemplo, o mesmo blog pode ter marcações diferentes em empresas diferentes. Mais detalhes são apresentados na Capítulo 5.

4.2 Algoritmo para extração de texto de páginas HTML

Um problema encontrado durante a implementação do *framework* foi que nenhum sistema de busca, para blog, disponibiliza serviços que retornem somente o conteúdo do *post*. Além disto, os blogs não possuem uma estrutura universal, ou seja, cada blog possui estruturação próprias de tags HTML, dificultando a extração do texto utilizando APIs especializadas em HTML. Por estas razões, o trabalho também propõe um algoritomo para extair o texto do *post* de blogs a partir do seu código HTML.

Como entrada para o algoritimo são necessários o link e o início[3] do post. Um probema encontrado é que o resumo que as ferramentas disponibilizam normalmente são retornados em forma de *String*, ou seja, perdem todas as tags que se encontram nela, por exemplo, as páginas HTML possuem tags específicas para deixar os caracteres em negrito, neste caso essas tags são perdidas, então se pegássemos a *String* com o resumo e comparasse com a *String* do HTML provavelmente não encontraríamos igualdade entre as duas. Por isso, além de utilizar o resumo, ele também é dividido e recomparado até o início do texto ser identificado. Por outro lado, apesar de não ser estruturado, existem algumas tags que são responsáveis por terminar um bloco de uma página HTML, como a tag , elas serão utilizadas para determinar o fim do texto. A seguir será mostrado um pseudocódigo para melhor entendimento, as funções e variáveis usadas são as listadas a seguir:

- achouInicio: Determinar se achou o início do texto(variável booleana);
- resumo: Resumo do post que se deseja extrair (variável de texto);
- HTMLBlog: HTML completa do blog(variável de texto);

[2] *Por exemplo: Technorati, Icerocket, Blogcatalog, Delicious.*
[3] Normalmente os resumos disponibilizados pelas ferramentas de busca são o início do texto.

- indiceInicio: Índice do começo do HTML que foi encontrado o texto(variável numérica);
- achouFim: Determinar se achou o início do texto (variável booleana);
- listaDePossiveisFinais: Lista dos possíveis finais para o texto, pré-definida (lista de texto);
- indiceFim: Índice do fim do HTML que foi encontrado o texto (variável numérica);
- compara(x, y): Compara dois textos e retorna se existe o texto x em y;
- compara(x, y, z): Compara dois textos e retorna se existe o texto x em y, a partir de z;
- indice(x, y): Retorna o indice que achou o texto;
- indice(x, y, z): Retorna o indice que achou o texto, a partir de z;
- divideTexto(x): Retira x caracteres do final do texto;
- tamanho(x): Retorna o tamanha da variável de texto x.

Código 4.1: Pseudocódigo - *SummaryStrategy*

```
1  faca
2    achouInicio <- compara(resumo, HTMLBlog)
3
4    se (AchouInicio = true)
5      indiceInicio <- indice(resumo, HTMLBlog)
6      achouFim <- compara(listaDePossiveisFinais, HTMLBlog,
7      indiceInicio)
8      se (achouFim = true)
9          indiceFim <- indice(listaDePossiveisFinais, HTMLBlog,
10         indiceInicio)
11         retorna textoEntre(indiceInicio, indiceFim)
12     senão
13         retorna "não achou"
14   senão
15     resumo <- divideTexto(10)
16
17 enquanto (tamanho(resumo) > 0)
```

Capítulo 5

Estudo de Caso

Este capítulo descreve a criação de uma aplicação utilizando o *framework* detalhado no capítulo anterior e realiza uma análise do algoritmo proposto para extração do texto do blog/post a partir de uma página HTML. Serão utilizadas métricas como precisão e *recall* para a análise do algoritmo.

5.1 Criando uma aplicação com o *framework*

Antes de apresentar como foi a instanciação da aplicação, serão listados os requisitos necessários para tal atividade. Este processo é bastante simples, bastando apenas extender as classes *Application* e *TagParser*.

Na classe *Application* é preciso realizar as seguintes operações:

- *Inicializar a variável numberOfTags*: Número de tags que vão ser usadas na aplicação. Do tipo inteiro;
- *Inicializar a variável tags*: Array de *strings* com as tags que se deseja extrair;
- *Inicializar a variável applicationLanguage*: Idioma que se deseja trabalhar. A principio só serão trabalhados blogs com o mesmo idioma;
- *Inicializar a variável preprocessing*: Tipo de pré-processamento deseja aplicar ao texto;
- *Inicializar a variável persistence*: Tipo de persistência será usada;
- *Inicializar a variável indexing*: Serviço de indexação será aplicado;
- *Inicializar a variável textExtraction*: Técnica de extração do texto, a partir da página HTML, que será utilizada;

- *Implementar o método checksLanguage*: Deixa o *framework* idependente de idioma, não é responsabilidade do sistema saber qual idioma será utilizado e sim da aplicação que o utiliza;
- *Implementar o método preprocesText*: O *framework* disponibiliza serviços de pré-processamento, mas a aplicação que utiliza eles quando necessário.

Já a classe *TagParser* só possui um método (*tagSearch*) entretanto ele é um pouco mais complexo de ser implementado, por ser o responsável por acessar a fonte de informação para retornar alguns detalhes sobre o post que deseja ser indexado.

5.2 Instanciação do *framework*

Como foi descrito acima para instanciar uma aplicação utilizando o *framework* proposto precisamos extender duas classes. A seguir seram mostradas as especificações da aplicação que foi criada e logo após seram mostrados como foram os requisitos para instanciação citados acima.

5.2.1 Especificações da aplicação

A aplicação foi criada com base nos blogs armazenados na *Technorati* [Technorati 2009b]. Esta empresa disponibiliza um serviço de marcação bastante abrangente, que possui desde tags mais gerais como política até algumas mais específicas, como Barak Obama.

Além disto, foram utilizados apenas blogs no idioma inglês, pois para aplicar as técnicas de pré-processamento seria mais viável trabalhar apenas com uma língua. Então, mesmo a *Technorati* disponibilizando um arcervo que conta com blogs/posts em vários idiomas, serão utilizados somente os que estiverem em inglês.

Poderia ainda ter sido utilizadas várias tags, mas para o experimento ficar mais controlado, e para melhor avaliar o algoritmo proposto, foram escolhidas cinco categorias, sendo elas: educação, econômia, entretedimento, esportes e filmes. Como os blogs são em inglês, elas ficam, respectivamente: *education*, *economy*, *entertainment*, *sports* e *movies*.

Então, em suma, a aplicação irá recuperar blogs da *Technorati* que estejam em inglês e pertencem às categorias supracitadas.

5.2.2 Extensão das Classes

O cógido a seguir mostra a implementação da classe *Application*. Nele podemos notar que foram criados os atributos e métodos necessários para gerar a aplicação.

Código 5.1: Implementação da classe *Application*

```
1
2  public class TechnoratiApplication extends Application{
3
4      private int numberOfTags;
5      private ArrayList<String> tags;
6      private String language;
7      private Preprocessing preprocessing;
8      private Persistence persistence;
9      private Indexing indexing;
10     private TextExtraction textExtraction;
11
12     public TechnoratiApplication (){
13
14     TechnoratiApplication
15     numberOfTags = 5;
16
17     tags = new ArrayList<String>()
18     tags.add("education");
19     tags.add("economy");
20     tags.add("entertainment");
21     tags.add("sports");
22     tags.add("movies");
23
24     applicationLanguage = "en";
25
26     preprocessing = preprocessingFactory.getEnglishStemming();
27
28     persistence =  persistenceFactory.getMySqlPersistence();
29
30     indexing = indexingFactory.getLuceneIndexing();
31
32     textExtraction = textExtractionFactory.getSummaryStrategy();
33
34     }
35
36     public boolean checksLanguage(String blogLanguage){
37         return blogLanguage.equals(this.getLanguage());
38     }
```

```
39
40      public String preprocesText(String text){
41          try {
42              return this.getPreprocessing().analyzeString(text);
43          } catch (IOException e) {
44              e.printStackTrace();
45              return null;
46          }
47      }
48  }
```

As linhas 26-32 do código acima mostra a instância do modelo de decisões que está sendo usada. É nesta parte que o usuário define quais *hot spots* serão utilizados, vide seção 4.1.3.

Por fim, a implementação da classe *TagPaser*. A seguir segue a explicação do código:

1. Inicialização de variáveis básicas(Linhas 5-7);
2. Recuperação da página da *Tecnhorati* que contém os blogs de uma determinada tag (Linhas 9-10);
3. Inicialização de variáveis auxiliares(Linhas 12-16)
4. Parser para obtenção dos atributos dos blogs contidos na página da *Tecnhorati* (Linhas 20-41);
5. Criação dos atributos do item para ser adicionado ao array de retorno (Linhas 43-46);

Código 5.2: Implementação da classe *TagParser*

```
1
2   public class TechnoratiTagParser extends TagPaser{
3
4       public ArrayList<Item> tagSearch(String tag) {
5           HttpUtils httpUtils = new  HttpUtils();
6           ArrayList<Item> technoratiItemList =
7           new ArrayList<Item>();
8
9           String page = httpUtils.getPage("http://technorati.com"
10          + "/tag/" + tag);
11
```

```
12          int firstIndexPermalink, lastIndexPermalink,
13          firstIndexNamePost, lastIndexNamePost,
14          firstIndexExcerpt, lastIndexPermaExcerpt = 0;
15          String permalink, namePost, excerpt;
16          int links = 0;
17
18          while (links < 20){
19              Item item = new Item();
20              firstIndexPermalink = page.indexOf("<h3><a
21 ␣␣␣␣␣␣␣␣␣␣␣␣class=\"offsite\"", lastIndexPermaExcerpt) + 29;
22              lastIndexPermalink = page.indexOf("\">",
23              firstIndexPermalink);
24
25              permalink = page.substring(firstIndexPermalink,
26              lastIndexPermalink);
27
28              firstIndexNamePost = lastIndexPermalink + 2;
29              lastIndexNamePost = page.indexOf("</a>",
30              firstIndexNamePost);
31
32              namePost = page.substring(firstIndexNamePost,
33              lastIndexNamePost);
34
35              firstIndexExcerpt = page.indexOf("<br␣/>",
36              lastIndexNamePost + 16) + 6;
37              lastIndexPermaExcerpt = page.indexOf("<div>",
38              firstIndexExcerpt);
39
40              excerpt = page.substring(firstIndexExcerpt,
41              lastIndexPermaExcerpt);
42
43              item.setTitle(namePost);
44              item.setPermalink(permalink);
45              item.setExcerpt(excerpt);
46              technoratiItemList.add(item);
47              links++;
48          }
49
50          return technoratiItemList;
```

```
51      }
52  }
```

Com essas duas classes implmentadas, é preciso criar um *main* bastante simples para rodar o sistema:

Código 5.3: *Main*

```
1
2       public static void main(String[] args) {
3
4           ApplicationControler applicationControler = new
5           ApplicationControler(ApplicationFactory.
6           getTechnoratiApplication());
7
8           TagParserControler tagParserControler = new
9           TagParserControler(TapParserFactory.
10          getTechnoratiTagParser());
11
12          BlogCrawler blogCrawler = new
13          BlogCrawler(applicationControler);
14
15          blogCrawler.crawlerTag(tagParserControler);
16      }
```

5.3 Análise do algoritomo proposto

Esta seção irá analizar os resultados da extração dos textos utilizando o algoritomo proposto no trabalho. Serão utilizadas duas métricas que são usadas em sistemas de recuperação de informação, são elas: precisão e *recall* [Baeza-Yates and Ribeiro-Neto 1999].

A precisão mede a capacidade do sistema em detectar textos que realmente deveriam ser retornados, ou seja, dentre os textos, quais o sistema detectou, quais realmente estavam na especificação, detecta os falsos positivos. No nosso caso, quais estavam em inglês e continham realmente o texto. Em outras palavras, dos textos retornados, todos atendem as especificações?

O *recall* mede a habilidade do sistema em detectar os textos, levando em consideração todos os textos presente no conjunto, ou seja, dentre o universo de todos os textos pertencentes ao conjunto, quais o sistema realmente detectou corretamente, detecta os falsos negativos. Em outras palavras, dos textos que deveriam ser retornados, quais foram?

Foram escolhidos vinte blogs de cada categoria citada acima, onde os resultados foram analizados manualmente. Ou seja, o algoritmo foi rodado para cada categoria com blogs previamente conhecidos para medir as métricas. A Tabela 5.1 mostra os resultados para cada categoria.

Tabela 5.1: *Resultados por categoria*

	precisão	*recall*
Educação	61,5%	65%
Econômia	83,3%	66,6%
Entretedimento	60%	60%
Esportes	87,5%	80%
Cinema	75%	88%

Como se pode perceber algumas categorias obtiveram resultados melhores que as outras, por exemplo, a categoria esporte (precisão = 87,5% e *recall* = 80%)obteve melhor resultado do que a categoria entretedimento (precisão = 60% e *recall* = 60%). Isto se deve ao fato de que os blogs de esportes normalmente estão ligados a empresas (televisão, rádio, notícias, clubes de futebol...), desta forma o texto do blog é mais formal e com uma linguagem com poucas gírias. Por outro lado os textos sobre entretedimento normalmente é postado por pessoas comuns, com isso a linguagem informal é dominante, dificultando a extração do texto.

Obtendo a precisão e o *recall* médio, fazendo a média aritmética entre todas as categorias, temos: precisão média = 73,46% e *recall* médio = 71,92%. Estes resultados são considerados bastante satisfatório para o trabalho, visto que na literatura os limeares destas métricas estão pouco maiores que 60% [Raghavan et al. 1989].

Capítulo 6

Considerações Finais

O presente trabalho teve como objetivos principais propor um *framework* para criação de blog *crawlers* baseado em contexto e um algoritmo para extração de texto a partir de arquivos HTML. O primeiro foi detalhado sob vários aspectos, como utilização de diagramas e explicação sobre os requisitos, para facilitar o entendimento e a sua extensão. Isto ficou evidenciado na seção estudo de caso, onde foi apresentado exemplo de aplicação sendo criada a partir do *framework*. Por ser um *framework*, o programa desenvolvido apresenta várias características inerentes da engenharia de software baseada em reúso, como: i) maior confiabilidade; ii) redução de riscos; iii) desenvolvimento acelerado; iv) facilidade de manutenção e evolução.

Segundo, o algoritmo proposto foi apresentado sob forma de pseudocódigo, onde o mesmo utiliza o resumo inicial do texto e o link da página como entradas e, com isso, a informação desejada é extraída. Além disso, foram apresentados resultados utilizando duas métricas bastante comuns na literatura: precisão e *recall*, onde os números obtidos foram bastante satisfatórios.

Como principais trabalhos futuros, em relação ao framework, temos:

- Criar serviços de busca;
- Criar outras implementações para os *hot spots*, principalmente os que possuem apenas uma implementação, por exemplo, o *Indexing*;
- Criar um classificador para alinhar os resultados de empresas diferentes e marcar possíveis blogs sem tags.

Para o algotirmo de extração:

- Pesquisar/implementar outros algotirmos para extração de texto a partir de arquivos HTML para comparar os resultados;
- Utilizá-lo para obter textos, de páginas HTML, que não sejam blogs.

Referências Bibliográficas

[Abney 1991] Abney, S. (1991). *Principle-Based Parsing: Computation and Psycholinguistics*. Kluwer Academic Publishers, Norwell, MA, USA.

[Apache 2005] Apache (2005). Http client. http://hc.apache.org/. Último acesso em Dezembro de 2009.

[Arasu et al. 2001] Arasu, A., Cho, J., Garcia-Molia, H., Paepcke, A., and Raghavan, S. (2001). Searching the web. *ACM TRANSACTIONS ON INTERNET TECHNOLOGY*, 1(1):2–43.

[Baeza-Yates and Ribeiro-Neto 1999] Baeza-Yates, R. A. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Benevenuto et al. 2009] Benevenuto, F., Rodrigues, T., Cha, M., and Almeida, V. (2009). Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM Internet Measurement Conference (IMC)*, pages 49–62, New York, NY, USA. ACM.

[Blogcatalog 2009] Blogcatalog (2009). Blogcatalog. http://www.blogcatalog.com/. Último acesso em Dezembro de 2009.

[Booch et al. 2005] Booch, G., Rumbaugh, J., and Jacobson, I. (2005). *Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Object Technology Series)*. Addison-Wesley Professional.

[Brin and Page 1998] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117.

[Clements and Northrop 2001] Clements, P. and Northrop, L. (2001). Software product lines: practices and patterns. 0201703327.

[Cockburn 2000] Cockburn, A. (2000). *Writing Effective Use Cases*. Addison-Wesley Professional.

[Company 2009] Company, Y. (2009). Delicious. http://delicious.com/. Último acesso em Dezembro de 2009.

[Czarnecki et al. 2005] Czarnecki, K., Helsen, S., and Eisenecker, U. (2005). Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169.

[Dawkins 1990] Dawkins, R. (1990). *The Selfish Gene*. Oxford University Press.

[Eckert 2008] Eckert, K. (2008). Blogalyzer. http://wifo1-54.bwl.uni-mannheim.de:8080/cb/project/65. Último acesso em Dezembro de 2009.

[Fayad et al. 1999] Fayad, M. E., Schmidt, D. C., and Johnson, R. E. (1999). *Building application frameworks: object-oriented foundations of framework design*. John Wiley & Sons, Inc., New York, NY, USA.

[Frakes and Baeza-Yates 1992] Frakes, W. B. and Baeza-Yates, R. (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice Hall PTR.

[Gaizauskas and Wilks 1998] Gaizauskas, R. and Wilks, Y. (1998). Information extraction: Beyond document retrieval.

[Gamma et al. 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns. elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

[Glance et al. 2004] Glance, N. S., Hurst, M., and Tomokiyo, T. (2004). Blogpulse: Automated trend discovery for weblogs.

[Golder and Huberman 2005] Golder, S. A. and Huberman, B. A. (2005). The structure of collaborative tagging systems. Technical report, Information Dynamics Lab, HP Labs.

[Google 2009] Google (2009). Api ajax de idioma. http://code.google.com/intl/pt-BR/apis/ajaxlanguage/. Último acesso em Dezembro de 2009.

[Hatcher and Gospodnetic 2004] Hatcher, E. and Gospodnetic, O. (2004). *Lucene in Action (In Action series)*. Manning Publications.

[Hotho et al. 2005] Hotho, A., Nurnberger, A., and Paass, G. (2005). A brief survey of text mining.

[Hurst and Maykov 2009] Hurst, M. and Maykov, A. (2009). Social streams blog crawler. pages 1615–1618.

[Icerocket 2009] Icerocket (2009). Icerocket. http://blogs.icerocket.com/. Último acesso em Dezembro de 2009.

[Johnson 1997] Johnson, R. E. (1997). Components, frameworks, patterns. *COMMUNICATIONS OF THE ACM*, 40:10–17.

[Johnson and Foote 1988] Johnson, R. E. and Foote, B. (1988). Designing reusable classes. *Journal of Object-Oriented Programming*, 1(2):22–35.

[Kleinberg 1999] Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46:668–677.

[Lee and Kang 2004] Lee, K. and Kang, K. C. (2004). Feature dependency analysis for product line component design. pages 69–85.

[Lemur 2009] Lemur (2009). The lemur toolkit. http://www.lemurproject.org/. Último acesso em Novembro de 2009.

[Lobo et al. 2007] Lobo, A. E., Brito, P. H. S., and Rubira, C. M. F. (2007). A systematic approach for architectural design of component-based product lines. Technical report, Instituto da Computação - Universidade Estadual de Campinas.

[Lucene 2001] Lucene (2001). Lucene. http://lucene.apache.org/java/docs/. Último acesso em Novembro de 2009.

[Manning et al. 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, 1 edition.

[Markiewicz and Lucena 2001] Markiewicz, M. E. and Lucena, C. J. (2001). Object oriented framework development. http://www.acm.org/crossroads/xrds7-4/frameworks.html. Último acesso em Dezembro de 2009.

[Mathes 2004] Mathes, A. (2004). Folksonomies - cooperative classification and communication through shared metadata. *Computer Mediated Communication*.

[McCallum 1996] McCallum, A. K. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/ mccallum/bow.

[Middleware 2009] Middleware, R. H. (2009). Hibernate. https://www.hibernate.org/. Último acesso em Dezembro de 2009.

[Motoyama and Varghese 2009] Motoyama, M. and Varghese, G. (2009). I seek you: searching and matching individuals in social networks. In *WIDM '09: Proceeding of the eleventh international workshop on Web information and data management*, pages 67–75, New York, NY, USA. ACM.

[Pant et al. 2004] Pant, G., Srinivasan, P., and Menczer, F. (2004). Crawling the web. pages 153–178.

[Porter 1980] Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

[Raghavan et al. 1989] Raghavan, V. V., Jung, G. S., and Bollmann, P. (1989). A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems*, 7:205–229.

[Rumbaugh et al. 2004] Rumbaugh, J., Jacobson, I., and Booch, G. (2004). *The Unified Modeling Language Reference Manual (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional.

[Sauvé 2000] Sauvé, J. P. (2000). Projeto de software orientado a objeto. http://www.dsc.ufcg.edu.br/ jacques/cursos/map/html/frame/oque.htm. Último acesso em Dezembro de 2009.

[Seiji Isotani 2009] Seiji Isotani, R. Mizoguchi, I. I. B. e. E. d. B. C. (2009). Estado da arte em web semântica e web 2.0: Potencialidades e tendências da nova geração de ambientes de ensino na internet. *Revista Brasileira de Informática na Educação*, 17:30–42.

[Shirky 2005] Shirky, C. (2005). Ontology is overrated: Categories, links, and tags.

[Shkapenyuk and Suel 2002] Shkapenyuk, V. and Suel, T. (2002). Design and implementation of a high-performance distributed web crawler. pages 357–368.

[Sommerville 2004] Sommerville, I. (2004). *Software Engineering (7th Edition) (International Computer Science Series)*. Addison Wesley.

[Stuckman and Purtilo 2009] Stuckman, J. and Purtilo, J. (2009). Measuring the wikisphere. In *WikiSym '09: Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, pages 1–8, New York, NY, USA. ACM.

[Sun 1995] Sun (1995). Mysql. http://www.mysql.com/. Último acesso em Dezembro de 2009.

[Tanenbaum 2007] Tanenbaum, A. S. (2007). *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA.

[Technorati 2008] Technorati (2008). State of the blogosphere 2008. http://technorati.com/blogging/feature/state-of-the-blogosphere-2008/. Último acesso em Novembro de 2009.

[Technorati 2009a] Technorati (2009a). State of the blogosphere 2009. http://technorati.com/blogging/feature/state-of-the-blogosphere-2009/. Último acesso em Novembro de 2009.

[Technorati 2009b] Technorati (2009b). Technorati. http://technorati.com/. Último acesso em Dezembro de 2009.

[Wikipédia 2009a] Wikipédia (2009a). Blog. http://pt.wikipedia.org/wiki/Blog. Último acesso em Novembro de 2009.

[Wikipédia 2009b] Wikipédia (2009b). Redes de relacionamentos. http://pt.wikipedia.org/wiki/Redes_de_relacionamentos. Último acesso em Dezembro de 2009.

[Wikipédia 2009c] Wikipédia (2009c). Wiki. http://pt.wikipedia.org/wiki/Wiki. Último acesso em Dezembro de 2009.

[Ying Ding and Yan 2008] Ying Ding, Ioan Toma, S. J. K. M. F. and Yan, Z. (2008). Data mediation and interoperation in social web: Modeling, crawling and integrating social tagging data. *Workshop on Social Web Search and Mining*.

Apêndice A

Padrões de projetos utilizados

Esta apêndice irá listar os padrões de projetos utilizados no desenvolvimento do sistema. Como foi dito no capítulo de fundamentação teórica, utilizar padrões é uma boa prática de programação, principalmente quando está se desenvolvendo um software que preza pelo reúso. Com *frameworks* não é diferente. A seguir serão mostrados os padrões, os motivos e os locais do código onde eles foram utilizados:

A.1 *Factory Method*

Utilizado em todos os *hot spots*, para facilitar a instanciação das sub-classes deles. Por exemplo, no pacote *preprocessing* (vide Figura 4.9) existe uma classe *factory* que contém quatro métodos estáticos para instanciar os sub-tipos deste pacote. Utilizando este padrão temos ganhos como aumento da flexibilidade e paralelização da hierarquia de classes.

A.2 *Command*

Assim como o *Factory Method*, este padrão é utilizado em todos os *hot spots*. Mais uma vez utilizando o exemplo do *preprocessing* (vide Figura 4.9). Nele existe uma classe responsável por fazer todas as chamadas para os métodos das sub-classes, *PreprocessinControler*. Com este padrão ganhos como recução do acoplamento e facilidade na inclusão de novas operações ou novos comandos foram observados.

A.3 *Singleton*

O padrão *singleton* é usado para obter somente uma instância do objeto de acesso ao banco de dados, desta forma mantendo a consistência dos dados. A implementação deste padrão encontra-se na classe *HibernateUtility* do pacote *utilities*. As consequências da

utilização deste padrão no projeto foram: criação de apenas um acesso ao banco de dados e o fato de não poluir o código com variáveis globais.

A.4 *Facade*

Este padão é utilizado no pacote *utilities*, como podemos perceber no diagramas de classes (vide Figura 4.11), este pacote possui uma interface que é implementada pela classe *UtilitiesControler*, esta simplesmente vai receber a requisição e repassar para a classe que realmente implementa tal funcionalidade. Ou seja, esta é uma classe "delegadora". As principais vantagens de utilizar este padrão são: isolamento de componentes internos do sistema e baixo acoplamento entre o subsistema e os clientes.