Expert Systems with Applications 40 (2013) 1177–1195

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

RetriBlog: An architecture-centered framework for developing blog crawlers

Rafael Ferreira [a,*], Fred Freitas [a,1], Patrick Brito [b,2], Jean Melo [a,1], Rinaldo Lima [a,1], Evandro Costa [b,2]

[a] Federal University of Pernambuco, Av. Prof. Moraes Rego, 1235, Cidade Universitária, Recife, PE 50670-901, Brazil
[b] Federal University of Alagoas, Campus A. C. Simões, Av. Lourival Melo Mota, s/n, Cidade Universitária, Maceió, AL 57072-900, Brazil

ARTICLE INFO

Keywords:
Social web
Blog crawler
Content extraction
Tag recommendation

ABSTRACT

Blogs have become an important social tool. It allows the users to share their tastes, express their opinions, report news, form groups related to some subject, among others. The information obtained from the blogosphere may be used to create several applications in various fields. However, due to the growing number of blogs posted every day, as well as the dynamicity of the blogosphere, the task of extracting relevant information from the blogs has become difficult and time consuming. In this paper, we use information retrieval and extraction techniques to deal with this problem. Furthermore, as blogs have many variation points is required to provide applications that can be easily adapted. Faced with this scenario, the work proposes RetriBlog, an architecture-centered framework for the development of blog crawlers. Finally, it presents an evaluation of the proposed algorithms and three case studies.

1. Introduction

Web 2.0 is changing the way users interact with information (O'Reilly, 2005). It allows them to become an active part of the Web through new tools that enable the creation and content management in a collaborative manner. Among these tools the importance of blogs should be emphasized.

Blogs are proliferating on the Web, particularly due to its simplicity and popularity. Simplicity of blogs use is portrayed, e.g. by Rosenbloom (2004) pointing out that "a blogger needs only a computer, Internet access, and an opinion". On the other hand, regarding the popularity of blogs, Blood (2004) declared that "Weblogs have become so ubiquitous that for many of us the term is synonymous with 'personal Web site' ". Furthermore, according to a Technoratis report,[3] the activity on the blogsphere doubles every two hundred days. This fact also highlights blogs as a very interesting tool which may be used in several areas, e.g., e-commerce and e-learning.

In addition, other companies like Blog Pulse[4] and Blog Scope[5] have interesting statistics. The first have more than 160 million blogs indexed and more than 60 thousand are created every day. The second indexes approximately 53 million blogs and over a trillion posts.

Due to the large volume of information generated on blogs, it becomes unfeasible to extract information manually. Thus, efficient computational approaches are necessary to extract information from blogs and make a profitable use of its relevant content. This information may be: user preferences, which issues are being addressed in the blogosphere, interesting texts to certain domains (Xiong & Wang, 2008). Thus they may be used, for example, to determine if a product is well accepted in the blogosphere (Takashi Yamada, Atsushi Yoshikawa, Takao Terano, Geun Chol Moon, & Go Kikuta, 2010) or to assist in learning environments (educational blogs) (Shiyi Qiao, Xiaohong Chu, & Shuwei Wang, 2009; Yang, 2008). Hence, some research has being carried out to automate the information extraction process from blogs (Fujimura et al., 2006; Joshi, 2006).

For the document acquisition task, the field of information retrieval (IR) stands out (Manning, Raghavan, & Schütze, 2008), since it is mainly concerned with identifying relevant texts for a particular purpose within a huge text collection. In order to enable users to perform their searches, the text and information related to each blog need to be properly indexed and stored. So, the blog crawlers are responsible for performing such task (Arasu, Cho, Garcia-Molia, Paepcke, & Raghavan, 2001).

However, to build a blog crawler, we should consider many aspects related to blogs themselves, such as language used by them, blog indexing service, preprocessing tasks, and indexing techniques. The variability and size of blogosphere pose two main problems to be dealt: (i) the difficulty of conceiving a general approach to deal with such variability; and (ii) the need to assist the developer with tools for extracting the main content of the blog, and the preprocessing of the blog text. In order to address this problem the approach that has been used in other works is the

* Corresponding author. Tel.: +55 (81) 30344984.
E-mail addresses: rflm@cin.ufpe.br (R. Ferreira), rflm@cin.ufpe.br (F. Freitas), patrick@ic.ufal.br (P. Brito), jccm@cin.ufpe.br (J. Melo), rjlima01@gmail.com (R. Lima), ebc.academico@gmail.com (E. Costa).
[1] Tel.: +55 (81) 30344984.
[2] Tel.: +55 (82) 3214 1401.
[3] http://technorati.com/blogging/feature/state-of-the-blogosphere-2008/.
[4] http://www.blogpulse.com/.
[5] http://www.blogscope.net/.

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2012.08.020

creation of a framework (Johnson & Foote, 1988). Two works deal with these problems (Chau, Xu, Cao, Lam, & Shiu, 2009; Ferreira et al., 2010). However, these frameworks do not follow some principles of software engineering, like explicit traceability between software architecture and implementation, which facilitates, for example, the evolution of the framework.

An approach centered on architecture is a solution to solve this problem. Develop a system based on architecture (Garlan & Shaw, 1994) has advantages such as: (i) Control of complexity; (ii) Modularity; (iii) facilitates the development and instantiation of the system. This approach fills the gap left by the systems mentioned above.

Besides this problem, some basic problems like extract the main content of blogs and recommend tags for posts are also a challenge to get interesting information from blogs. One solution for the first problem is to create algorithms that can detect the main content of an HTML page without any prior information. The second deals mainly with the problem of tags alignment. The same blog can be labeled with different tags due to differences in culture, knowledge and experiences of users (Subramanya & Liu, 2008).

On the one hand, to handle the first problem, some algorithms have been implemented to deal with the blogs content extraction, they are: Boilerplate Kohlschütter, Fankhauser, and Nejdl (2010), Document Slope Curve (DSC) Pinto et al. (2002) and Text-to-Tag Ratio (TTR) Weninger and Hsu (2008) and the heuristic Link Quota Filter (LQF) Gottron (2007) which are among the more efficient algorithms for content extraction. Furthermore, the algorithms mentioned have very desirable properties in the context of research, including the independence of the structure. This also allows us to apply them to other types of HTML pages (e.g. news), independent of language, and fast execution time.

On the other hand, to recommend tags, this paper proposes four services. These services are classified into approaches based on tags and based on post. These services deal with this problem differently, allowing users to choose the appropriate type of service according to his needs. For example, if the user has a corpus with little noise, it may use a service based on the post to achieve better results. On the other hand, if he needs a fast implementation, services based on tags are more appropriate.

Therefore, this work proposes an architecture-centered framework for developing blog crawlers. This framework aims to provide services to build applications in the Blogosphere. This work deals with general aspects as ease creation of a blog crawler and specific problems such as a tag recommendation. This framework is carried out both a quantitative and a qualitative evaluation. The quantitative evaluation focuses on tests in the main proposed algorithms. Another one shows the advantages of using architecture-centered development, evaluation. Finally, three case studies are described.

The rest of the paper is organized as follows. Section 2 describes aspects related to blogosphere, social media, information retrieval and software architecture. Section 3 introduces the proposed framework, its architecture and its implementation aspects. Three case studies that validates the proposed approach are described in Section 4. Section 5 present the experiments performed in the main proposed algorithms. Section 6 contextualizes the proposed framework against some related work, making the contribution of the proposed approach more explicit. Finally, some conclusions and discussion of possible future work are presented in Section 7.

2. Background

This section describes the necessary background to understand the proposed framework. The follow subsections describe the Blogosphere and the social media, information retrieval, and some software engineering aspects.

2.1. Social media and blogosphere

Social media represents a broad change on how people interact with one another, allowing them easily participate in and contribute (Cho & Tomkins, 2007). Interesting tools available for interaction in social media include blogs, forums, and wikis. This work is concentrating on the blogs.

Blogosphere is the name given to the whole community of blogs on the Internet, it is part of the infrastructure which the ideas are developed and transmitted. Blogs are essentially only the published text of the thoughts of some author, while the Blogosphere is a social phenomenon.

According to Wikipedia the blog (a contraction of the term "web log") is a type of website whose structure allows quick update and it is usually maintained by an individual with regular entries of commentary, descriptions of events, or other material such as graphics or video. Entries are commonly displayed in reverse-chronological order, focusing on the proposed theme of the blog, it can be written by a variable number of people, according to the policy of the blog.

Blogs are proliferating on the Web, particularly due to its simplicity and popularity. Simplicity of blogs use is portrayed, e.g. by Rosenbloom (2004) pointing out that "a blogger needs only a computer, Internet access, and an opinion", and Blood (2004) declared that "Weblogs have become so ubiquitous that for many of us the term is synonymous with 'personal Web site' ".

On the other hand, regarding the popularity of blogs, according to Technorati,[6] one of the biggest aggregations of blogs on the Web, annually makes a survey on the state of the Blogosphere and its trends. The research of 2008 and 2009 will be used to define interesting points about the current state of blogs. In 2008, the Technorati research concluded interesting positions about the size of the Blogosphere. All studies show that blogs are a global phenomenon that hit the Web. From this research is possible to realize interesting facts as the large number of posts that are written every day, nine thousand, and there are over 133 million blogs indexed by Technorati between years 2002 and 2008. According to the same estimate the activity in blogs doubles every two hundred days.

Two interesting questions were answered in the 2009 survey (Technorati, 2009): (i) Who are the bloggers?, and (ii) The What and Why of Blogging. They will be answered in the sub-sections below.

2.1.1. Who are the bloggers?

The word bloggers refers to a person who write on blogs, according to Technorati research, the bloggers in general, are part of a group of highly educated and rich. Almost half of all bloggers surveyed have a graduate degree, and most have a family income of $ 75,000 or more per year. Besides these data, other interesting features are:

- 60% are 18–44;
- The majority are more affluent and educated than the general population;
- 75% have college degrees;
- 40% have graduate degrees;
- One in three has an annual household income of $75 K+;
- One in four has an annual household income of $100 K+;
- Professional and self-employed bloggers are more affluent: nearly half has an annual household income of $75,000 and one third topped the $100,000 level.

These features indicate that people who are using blogs are qualified, then the published texts provide interesting information.

[6] www.technorati.com.

2.2. Information retrieval

Information retrieval (Manning et al., 2008) consists of finding material (usually documents) from an unstructured nature (usually text) that satisfies some essential information from within large collections (usually stored on computers). In last years, the scientific community and some companies have invested in this area. As an example, several works (Baeza-Yates & Ribeiro-Neto, 1999) has investigated domains like modeling, classification and categorization of information, systems architecture, filtering, and query languages.

Among the well-known data structures to retrieve some information, the inverted index (Hatcher & Gospodnetic, 2004; Manning et al., 2008) is a common structure that enables to build a lookup table in which the words in certain document are adopted as items to the list of documents where each term occurs. The main purpose of this structure is to serve as an index of a book, persisting pages that contain the words. That is, the inverted index assures a better performance than perform a page by page search.

Moreover, it is essential to store the frequency of a term in a particular document. In the end of this process, the inverted index may be called of the dictionary. However, a concern could be easily identified in this process: there are words that have a few expressiveness in the document, for example, articles. Other words may have the same semantics, although with different syntax, like words in masculine and feminine, plural and singular. In order to solve these problems, before indexing the documents, it must be sent to a preprocessing scheme.

There are several techniques that renders the text before indexing them, among them are Hotho, Nnrnberger, and Paa (2005):

- **Lexical analysis**: This technique divides the document in tokens and it makes a word-by-word checking, removing the symbols that belongs the language.
- **Filtering**: In this technique, they are removed words from the dictionary and documents. As a example, one of these methods consists of stopping the filtering of words. That is, it removes words that have some or no content, like articles, conjunctions, and prepositions. Furthermore, words with both high and low frequency of occurrence are considered in this pattern and may be removed.
- **Lemmatization**: These methods make the mapping of verb forms such as the infinite tense and nouns into the singular form. Thus, the form of the word must be known.
- **Stemming**: They are methods that build the basic forms of words, i.e. strip the plural "s" from nouns, the "ing" from verbs, or other affixes. A stem is a natural group of words with equal (or similar) meaning. After the stemming process, every word is represented by its stem. For instance, the verbs traveling and traveled are both transformed into travel;
- **Removing spam**: They are techniques that searches and removes all text in documents that has a irrelevant meaningful to the users.

After the preprocessing and creation of the dictionary, it adopts search engines based on the boolean or statistical model (Manning et al., 2008) that uses the process of the inverted index. With this mechanism, the user may retrieve any document within the dictionary. Fig. 1 shows the flow of a basic process of information retrieval.

Tools such as Lemur (2009) and McCallum (1996) have several features in information retrieval, however Lucene (2001) is the mostly known tool that provides information retrieval in multiple languages.

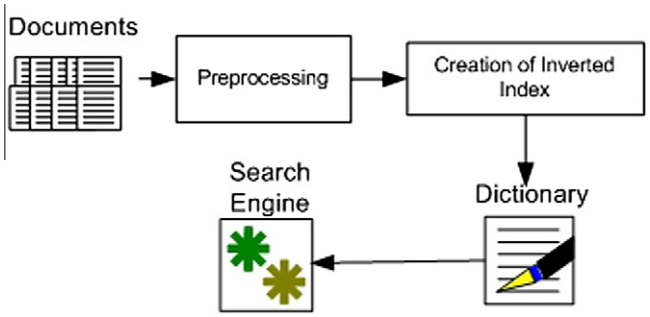

Fig. 1. Flow of a basic process of IR.

2.3. Web crawlers

The search engines are becoming increasingly essential to obtain relevant information from large amount of data that a Web offers. Search engines store huge collections of pages with the help of web crawlers, which are responsible for traveling by Web through links and to collect data that are usually indexed to the user could runs queries efficiently, (Shkapenyuk & Suel, 2002).

According to Shkapenyuk and Suel (2002), Web crawler is a computer program that exploits the graph structure of the Web to move from page to page. As already mentioned, the greatest motivation to create Web crawlers was to recover the Web pages and add them or their representation in a local repository to facilitate the search for the user. In its simplest form a crawler starts of a page and from there use external links to reach the other pages.

If the Web is a static collection of pages, then come a time when the crawlers were no longer needed, since all the pages you would be linked to a repository. However, the Web is a dynamic entity that evolves with different rates and speeds. Hence there is a continual need for crawlers to help applications stay current as new pages are added and old ones are deleted, moved or modified.

Below, we describe details of the flow activities and the basic architecture of a Web crawler.

2.3.1. Flow activities

Fig. 2 shows the flow of a basic sequential crawler, it will be detailed below.

- **Initialize frontier with seed URLs**: The frontier is the to-do list of a crawler that contains the URLs of unvisited pages. For this step can be implemented, for example, a FIFO (First in, First Out) scheduling scheme to sort the URIs that will be accessed;
- **Check for termination**: Checks if the frontier is empty, if not returns the next URI to be analyzed;
- **Fetch page**: Obtain the page, typically through a request made through an HTTP client. Deal with connection problems and determines some characteristics of the pages, as the last update;
- **Parse page**: Creates a parser on the page to extract useful information and possibly guide the next step of the crawler, moreover, seeks to eliminate information not expressive or useless;
- **Add URLs to frontier**: Adds URIs, obtained in the previous step, to the frontier.

2.3.2. Basic architecture

Fig. 3 shows the basic components of crawler architecture.

In this architecture there are two fundamental components that are the application and system crawlers. The system crawler has a responsibility to go to the Web and download pages. Moreover, the application one is responsible for passing a list of pages that must be downloaded and make appropriate operations with the data obtained by the system crawler.

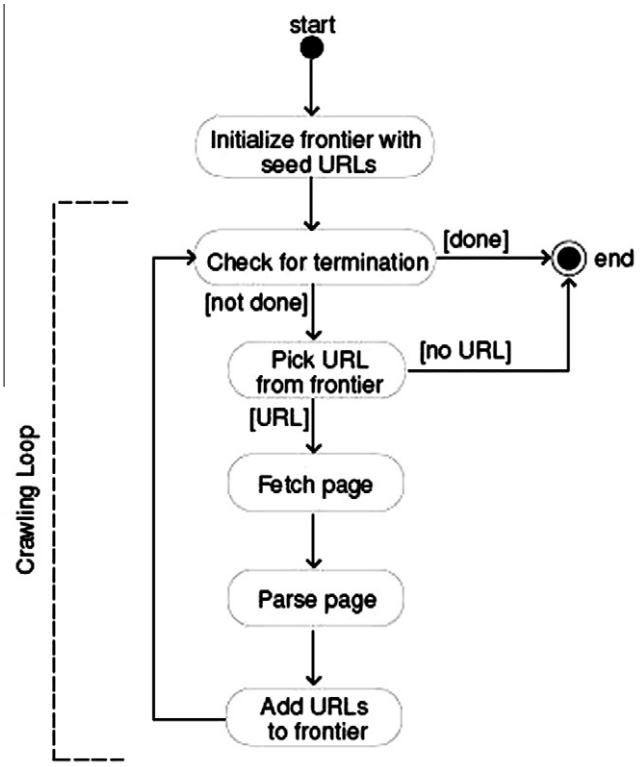

Fig. 2. Flow of a basic sequential crawler (Shkapenyuk & Suel, 2002).

2.3.3. Social crawler

Every day is greater the use of social tools such as wikis, emails, forums, blogs, tools, relationship, among others. These tools contain information that can be valuable if used properly, for example, a networking tool describes some characteristics of users that can be used to profile him and mechanisms for personalization services.

However, this requires mechanisms to obtain information from these tools. Then coming into the social crawlers (Hurst et al., 2009), they are similar to crawlers previously described, but have a fundamental difference, they are designed to gather specific information from tools and not the entire page. For example, a Blog Crawler is a mechanism that obtains information from blog posts, not the entire web page of the blog.

2.4. Framework

There are several definitions of framework, the most famous are those of Johnson (1997) and Johnson and Foote (1988): (i) A framework is a set of classes that include an abstract design for solutions of a problems related family, (ii) a framework is a set of objects that collaborate to perform a set of responsibilities to an application domain, a standard structure of an application that can be customized by the developer. From these definitions we can list some framework's characteristics:

- Reuse the software design and not just the code;
- There is not an application, but rather a framework to build applications in a specific domain;
- It solves problems with similar nature;
- It is incomplete applications.

Besides the features mentioned from the above definitions, others must be emphasized (Fayad, Schmidt, & Johnson, 1999; Sauv, 2000):

- **Reusability**: This is the main purpose of the framework, but to be reusable it needs to be usable, so it must be well documented and easy to use;
- **Extensibility**: It should contain abstract functionality (without implementation) which must be completed;
- **Completeness**: It needs to address the problem domain intended;
- **Inversion of Control**: The inversion of control is a key feature in the framework's architecture. It determines which set of methods for a specific application should be instantiated by framework. the framework is responsible for the flow control.

The framework could be classified in several ways, for example, in regard to where it is used (Sauv, 2000):

- **Support framework**: It provides operating system service such as file access and distributed computing;
- **Application framework**: Also known as horizontal framework. These provide features that are applied to a large variety of problems. They are used in more than one domain. For example a framework for building GUI interface;
- **Domain framework**: Also known as vertical framework. These provide features that are applied in a specific area. For example, framework for building control applications for manufacturing.

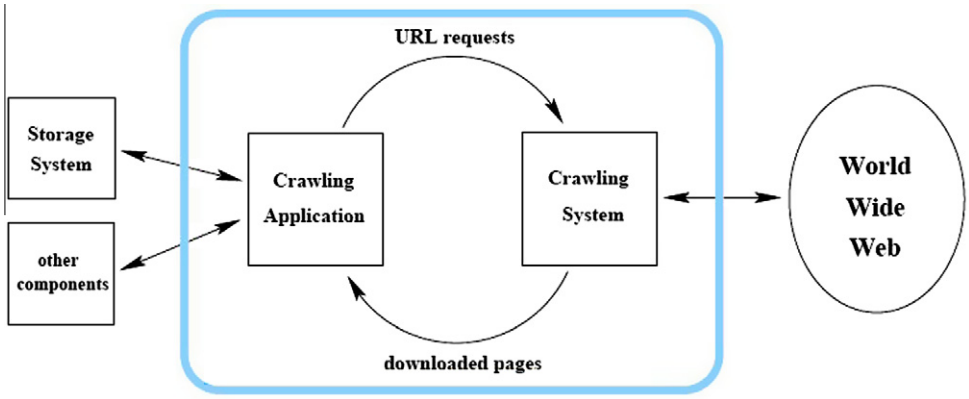

Fig. 3. Basic two components of the crawler (Shkapenyuk & Suel, 2002).

The frameworks could also be classified as Fayad et al. (1999):

- **White box framework**: It is strongly connected to language features. It requires a good understanding of framework to create an application;
- **Black box framework**: It is instantiated from some configuration process, such as XML. These frameworks mainly use composition (classes or components) and delegation to perform customization. It do not requires understanding of internal details to produce an application;
- **Gray box framework**: It is designed to avoid the disadvantages presented by white and black box frameworks. It allows a degree of flexibility and extensibility without exposing internal information unnecessary.

To design a framework is necessary to be aware of the following properties (Clements & Northrop, 2001):

- **Framework's core**: It is a set of classes that collaborate to implement the systems architecture;
- **Extension points**: It admits extension points, typically as abstract classes and interfaces;
- **Flow Control**: The flow is defined by the classes that are at the core, and they are responsible for invoking the classes that extend the extension points;
- **Hot spots**: They are the flexible points of the framework. These points are subject to extension. They are usually implemented through inheritance and abstract methods;
- **Frozen spots**: They are fixed parts of a framework, provides the services already implemented the framework. Usually perform indirect calls to the hot spots.

Some advantages of using the framework are Sauv (2000) and Sommerville (2004):

1. Less code to design and write;
2. Reduced costs and development time;
3. Code more reliable and robust;
4. Reduced maintenance and easier evolution;
5. Better consistency and compatibility between applications;
6. Stabilization Code (fewer errors) due to its use in various applications.

On the other hand we can also mention some disadvantages of working with framework (Sauv, 2000; Sommerville, 2004):

- Requires more effort to build;
- Benefits are realized in a long term;
- It needs to modify the development process and create new incentives;
- It requires documentation, maintenance and support.

The framework's documentation is one of the factors that may define its success. It needs to adapt to different developers, so it must have different levels of abstraction. Four topics are bound to be documented (Markiewicz & Lucena, 2001):

- **Purpose**: It contains a brief description of the framework and the problem domain for which it was developed;
- **How to use the framework**: it ensures the reusability of the framework;
- **Purpose of application**: it shows examples that help better understand the framework;
- **Design**: It must include the classes, their relationships and collaborations.

2.5. Software architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprises software elements, the externally visible properties of those elements (architectural components and connectors), and the relationships among them (architectural configuration) (Bass, Clements, & Kazman, 2003).

Since the software architecture represents the structure of the software system, it is a set of significant decisions about the software organization. Those decisions concerns the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements (Kruchten, Henk Obbink, & Stafford, 2006). Moreover, since software architecture deals with the design and implementation of the high-level structure of the software, its decisions have a decisive impact into the quality attributes of the whole system, such as availability, reliability, and testability.

Fig. 4 presents an example of an architectural configuration involving two components and a connector.

Architectural components (e.g., A and B) are primary computational elements or data stores of a system and are defined through the list of operations that they provide (provided interfaces), as well as the operations necessary for executing their functionalities (required interfaces). To exemplify the concept of provided and required interfaces, component A of Fig. 4 has a provided interface called IA_Provided, and a required interface called IA_Required. Architectural connectors (e.g., conn) are mediators of the communication and coordination activities among components. That is, they define the rules governing component interaction and specify any auxiliary implementation mechanism required. The way that components and connectors are connected together is defined by the architectural configuration.

Besides representing the structure of a software system, the software architecture can also represent its behavior through architectural scenarios, which are relevant at the architectural level because they enhance the structural representation of the system with an abstract representation of its behavior.

Create, maintain and evolve the architecture of a system is a difficult task, which is usually guided by the design decisions of the architect and the development team. Due to the subjective nature of these design decisions, the development team and especially the software architect must exercise their abilities to recognize the best solutions. To assist in this task, the software architect can be based on a list of principles that should always be pursued. We following present the six basic principles of software architecture:

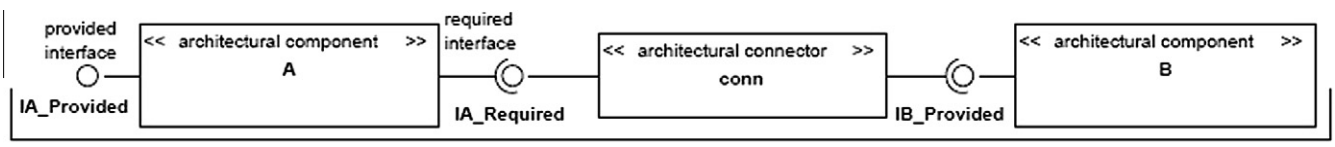

Fig. 4. Example of an architectural configuration in UML.

1. **Encapsulation**: The architecture is organized into relatively independent parts, in order to be an abstract representation of a software solution, in which the implementation details are omitted.
2. **Low coupling**: Due to the encapsulation already discussed, the communication between the architectural components must occur through well-defined communication protocols.
3. **High granularity**: One of the main advantages of specifying the software architecture is the ease for representing complex structures. The comprehensive understanding of the system is a result of abstraction afforded by the architecture. Thus, the components that compose an architecture typically represent a role or category, which may be detailed through the definition of an internal architecture.
4. **High cohesion**: Cohesion represents the degree of closeness between groups of features. The high cohesion of architectural components is a consequence of the definition of clear and easily understood roles.

2.6. Component-based development

Component-based software development (CBSD) or component-based software engineering (CBSE) is concerned with the assembly of pre-existing software components into larger pieces of software. Underlying this process is the notion that software components are written in such a way that they provide functions common to many different systems. The goal of CBSD is to allow parts (components) of a software system to be replaced by newer, functionally equivalent, components.

The concept of software components had been initially suggested by McIlroy (1968) as a way of tackling the software crisis, yet only in the last decade or so has the idea of component-based software development taken off. Nowadays there is an increasing market place for Commercial Off-The-Shelf (COTS) components, embodying a "buy, dont build" (Brooks, 1987) approach to software development. The promise of CBSD is a reduction in development costs: component systems are flexible and easy to maintain due to the intended plug-and-play nature of components.

According to Szyperski (2002), a software component is a unit of composition with contractually specified interfaces and explicit context dependencies. An interface is a set of named operations that can be invoked by clients. Context dependencies are specifications of what services are required by the software components in order to be successfully executed. Such dependencies are usually specified through required interfaces (D'Souza & Wills, 1999).

2.6.1. COSMOS* model

The COSMOS* model (Leonel Aguilar Gayard, Cecília Mary Fischer Rubira, & Paulo Astério de Castro Guerra, 2008) uses programming language features and well-known design patterns to represent software architectures explicitly in the program code. The COSMOS* model defines the specification and implementation of COSMOS* components, COSMOS* connectors, COSMOS* architectural configurations, and COSMOS* composite components.

The COSMOS* model is divided in a specification model, which a component exposes its specification; an implementation model, which guides the internal implementation of a component; a connector model, which specifies the connection between components using connectors; and a composition model, which new components or entire systems are built using existing components.

Therefore, the main goals the COSMOS* model are: First, provide an accompanying from software architecture to program code. Second, help maintenance of the implementation. This, in its turn, is regarded as a model which reduces the coupling between architecture's modules and still offers software reuse, so we decided to use it for developing the framework proposal. Although COSMOS* has four sub-models (specification, implementation, connector and composition), this work only uses two which are the specification model and implementation model because the connectors are simple classes. It is important quote that all framework's components were implemented using COSMOS*.

2.7. Feature model

A feature is a system property that is relevant to some stakeholder and is used to capture similarities and variabilities between systems (Czarnecki, Helsen, & Eisenecker, 2005). The feature modeling is the activity of identifying similarities and variability of the products of a product line, or frameworks applications, in order to organize them in a model. A feature model represents a hierarchical decomposition of resources, usually including two types of relationships: aggregation and generalization. The aggregation relationship is used if a resource can be decomposed into a set of sub-functions, and the relationship of generalization is used when a resource can be specialized with more specific information (Lee & Kang, 2004).

Relationships between a parent feature and its child features (or subfeatures) are categorized as:

- Mandatory: child feature is required;
- Alternative: one of the sub-features must be selected;
- Optional: child feature is optional, in other words, one or more children can be instantiated or no child as well.

For instance, in Fig. 5 (Lobo & Rubira, 2007), to instantiate the *ATM system* is necessary to choose instances of *User identification* and *Balance*(mandatory relationship). On the other side in order to have the *User identification* is required to choose only one subtype between *Touch screen* e *Card Reader*(alternative relationship). Finally, to instantiate *Balance* it is necessary one instance of *Display*(mandatory relationship) and the instance of *Printer* is optional (optional relationship).

3. RetriBlog: an architecture-centered framework for creating blog crawlers

The RetriBlog architecture consists in a gray-box framework (Fayad et al., 1999), which allows the fast development of blog crawlers. In other words, it provides services that developers can easy modify to create new crawlers, they instantiate the framework using only high level interfaces. It was developed following an architecture-centered process and implemented in Java according to the COSMOS* component implementation model (Aguilar Gayard et al., 2008). In addition, the framework provides services to solve specific problems found in several web applications, such as content extraction, and tag recommendation.

An architecture overview is shown on Fig. 6. It is divided into five modules. We briefly describe each module (a complete description is presented in the following section).

- **Crawler Specification**: This module contains the information to instantiate the framework. In order to create an application, the user has to provide the require specifications;
- **General Services**: It implements the some basic services. Such services are the frozen spots of the framework;
- **Application Services**: The main application services. These services constitute the hot spots of the framework;
- **Toolkit**: This module contains a set of tools which are used to perform the services of previous modules;
- **Persistence**: It is responsible for the data storage. Currently it supports MySQL database.

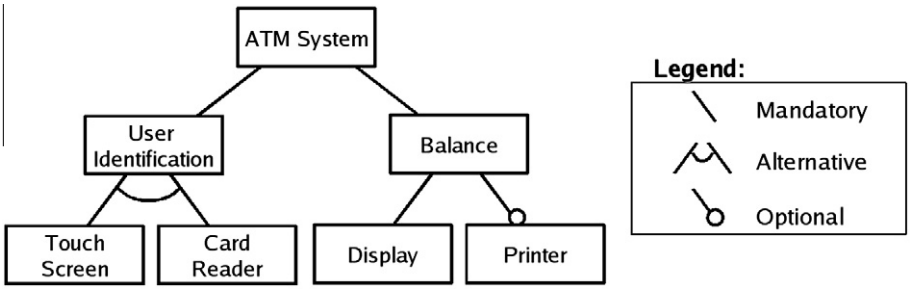

Fig. 5. Feature model.

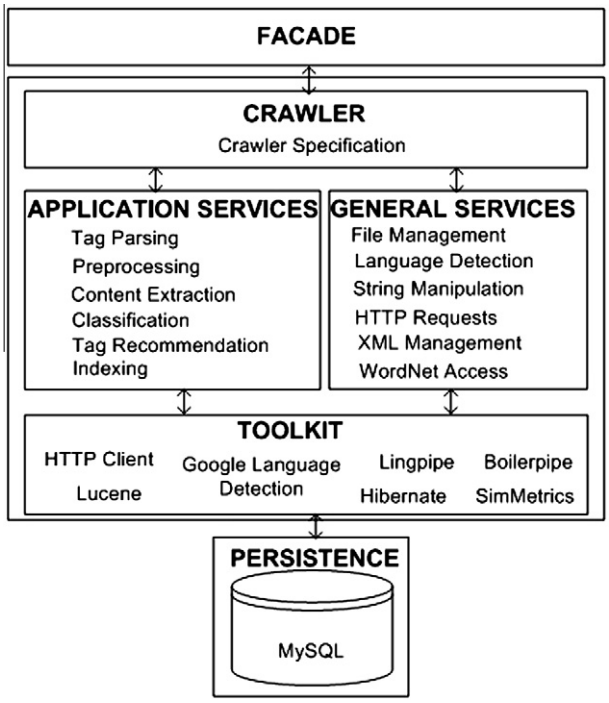

Fig. 6. RetriBlog architecture.

Fig. 7 shows how each module is locate on standard architecture for web crawlers described in Section 2.3.2.

3.1. Architectural design

Before explaining the framework architecture is important to highlight some design decisions and their consequences.

- **Framework**: We decided to create a framework to due with the variability points aforementioned;
- **Gray-box framework**: The advantage of developing a gray-box framework is that the user does not need to know all the details about the source code to create an application (such as white-box) and at the same time is not "stuck" to what is already defined (such as black box);
- **Architecture-centered development**: We use this approach to create the framework to improve modularization, instantiation cost and evolution of the system. Moreover, the code becomes easy to understand;
- **Component-based development** (CBD): The use of CBD brings two main advantages: (i) the increase in productivity, it occurs due to the reuse of existing components in the construction of new systems, (ii) improve quality, due to the fact that the components used have already been employed and tested in other contexts;
- **COSMOS***: The main advantage of using COSMOS* is that the all specification is defined using design patterns. Therefore any developer who knows design patterns easily understand the RetriBLog code. In COSMOS* the concept of required and provided interfaces and connectors are explicitly represented.

Fig. 8 presents the feature model. This model uses the notation proposed in Gomaa (2004), which represents hot and frozen spots in an explicit way.

Some elements in the Fig. 8 are explained bellow: (i) The *Content extraction*, *Indexing*, and *Tag parsing* features are mandatory, they have to be instantiated; (ii) The *Classification* and *Tag recommendation* features are optional; (iii) The subtypes of all features, except the *Preprocessing*, are all alternate, i.e., the developer can choose one of them for each main feature; (iv) The *Preprocessing* feature has two mandatory and four optional sub-features.

The collection of alternative and optional features represent the hot spots of the framework. The interaction among all these

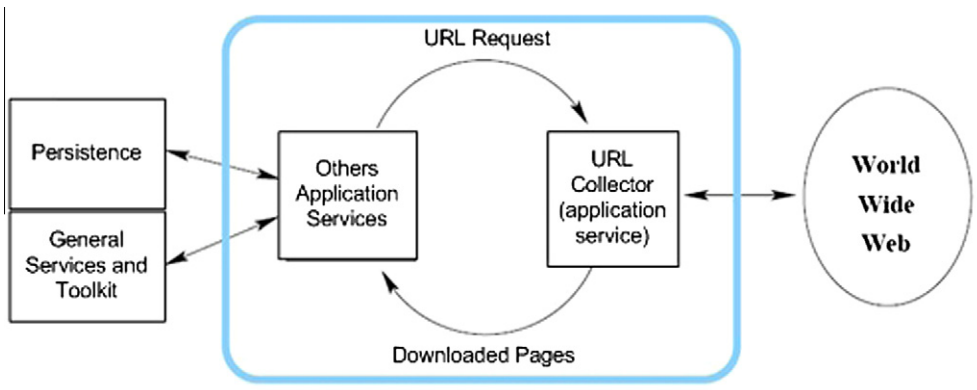

Fig. 7. RetriBlog architecture – based on standard architecture for web crawlers.

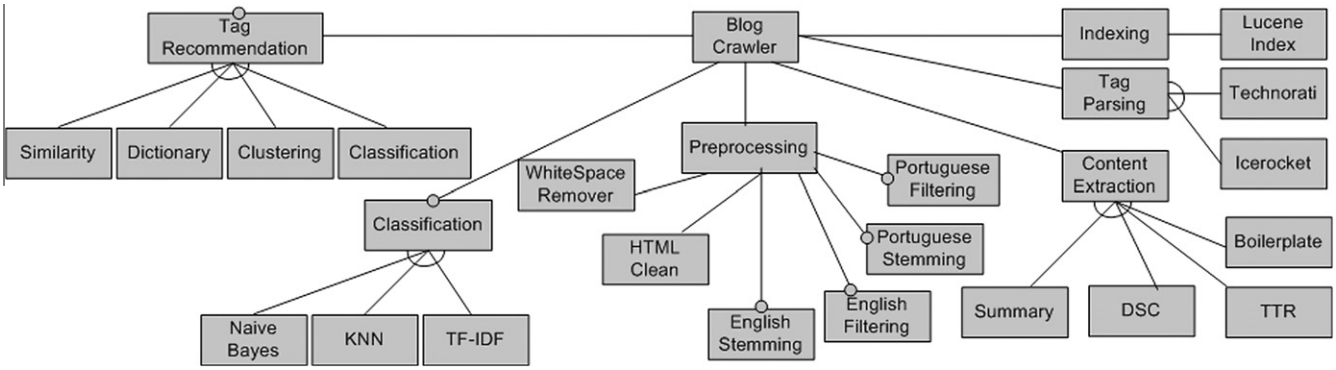

Fig. 8. Feature model.

features defines the decision model. In the following we present a brief description of each hot spot:

- **Preprocessing**: This hot spot removes the information considered irrelevant to the blog analysis. The current version of the framework provides six types of preprocessing tasks: *HTML cleanning*, which is responsible for cleaning all HTML tags; White Space Remover, it removes extra white spaces; *English and Portuguese filtering*, which removes English and Portuguese stop words respectively; *English and Portuguese stemming* (Hotho et al., 2005), which performs a stemming technique applied to English and Portuguese words respectively.
- **Content extraction**: This service extracts the relevant textual content of blog posts. We consider as relevant textual content, the main blog content without banners, links, advertisements, etc. Three services were implemented: (i) *Document slope curve* (DSC) Pinto et al. (2002), which treats a web page as a sequence of tokens. These tokens can be html tags or text. In DSC algorithm, when there is a section of the web page with a low number of HTML tags, the algorithm considers this section as relevant textual content and extracts it. (ii) *Text-to-tag ratio*(TTR) Weninger and Hsu (2008), which uses the ratio of the amount of non-tag characters by the number of HTML tags per text line to extract content; and (iii) *Boilerplate* (Kohlschütter et al., 2010), which consists of an algorithm based on decision tree to examine shallow features of the text. The decision tree algorithm classifies the textual content of the blog either as relevant or irrelevant.
- **Indexing**: This service indexes texts to improve the search process. It was implemented using Lucene[7] tool.
- **Classification**: This hot spot provides text classification services based on statistical techniques. Three services are available: (i) *Naive Bayes* (Mitchell, 1997), which provides a trainable Naive Bayes text classifier using token features; (ii) *KNN* (Baoli & Qin, 2003), which implements a k-nearest-neighbor classifier based on Euclidian distance as similarity measure; and (iii) *TF-IDF* (Salton & Buckley, 1988), which provides a service for training classifiers based on the weighting of term frequency inverse document frequency taken as token features.
- **Tag recommendation**: It provides four different services for the alignment of blog tags. (i) *Similarity*, which is based on the Levenshtein Distance metric (Miller, Vandome, & McBrewster, 2009). (ii) *Dictionary*, which addresses words with similar semantics using WordNet (Miller, 1995). (iii) *Clustering*, which performs the alignment of blog tags using a classical version of the KNN cluster algorithm (Tan, 2006). (iv) *Classification*, which is based on a bag of words classifier. This classifier uses the words from blog posts and their respective frequency to train and generate a Naive Bayes classifier (Jiang, Wang, Cai, & Yan, 2007).
- **Tag parsing**: This hot spot performs a search and returns a list of blog URL's matching to a given tag. It is currently available for searching using the Technorati [8] and Icerocket.[9]

Fig. 9 shows the framework architecture. It follows a heterogeneous architectural style based on the layered and independent components architectural styles. Four high-level layers have been defined: (i) *The application layer* which contains the general process created with the framework (crawler process); (ii) *The system layer* which contains the main components of the framework (system kernel); (iii) *The business layer* which controls the framework persistence; and (iv) *The infrastructure layer* which represents the database connection and it is realized by the JDBC framework.

It is important to highlight that some internal components of the System layer have been omitted. For example, although the Preprocessing component was represented as a single component, it only represents a facade component which propagates the request to other (omitted) components that actually implement the preprocessing strategies presented in Fig. 8, such as the Clean HTML and the White Space Remover. The same occurs to some other components in Fig. 9. The architectural connectors have been omitted in the figure as well.

The following sections detail each modules of the framework architecture and its components.

3.2. Crawler module

The crawler module consists of the part of the framework that developers have to modify. First, they have to create a Specification Class in which they set up their preferences. This class encapsulates the parameters that will be used by the tag parsering, content extraction, indexing, and preprocessing algorithms. Once these parameters entered, the framework starts the blog crawler, which demonstrates how easy the creation of an application using the RetriBlog framework is.

3.3. General services module

The general services module implements basic services used in the crawling process. This module does not have variability points. New components can be easily integrated because all components

[7] http://lucene.apache.org.

[8] http://technorati.com.

[9] http://www.icerocket.com.

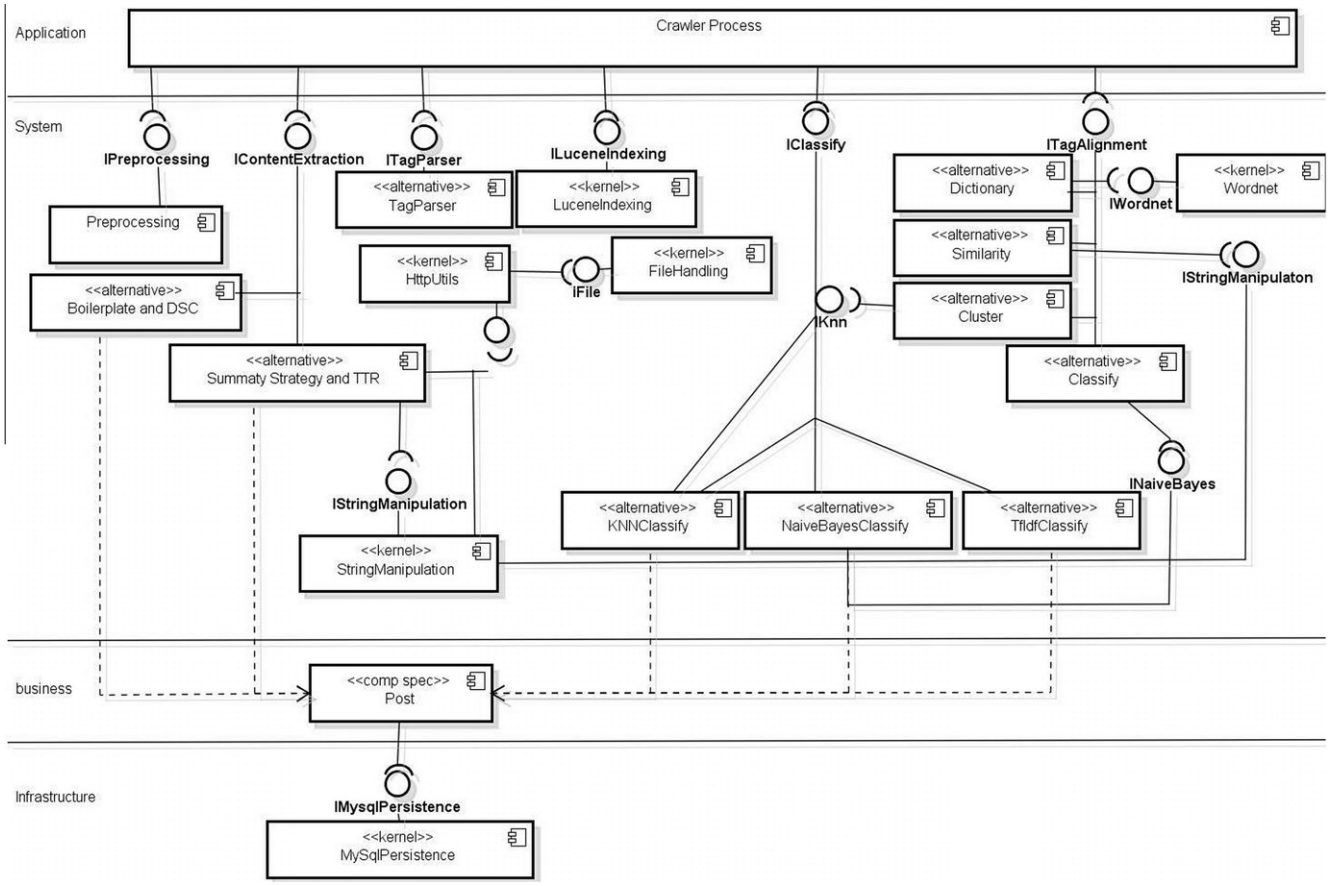

Fig. 9. Framework architecture.

of this module are isolated from each other. The services provided by this module are: (i) **File management**: This service provides basic file operations, including saving, changing, loading, etc.; (ii) **Language detection**: It gives a text snippet in natural language, this service outputs the language in which the text was written; (iii) **String manipulation**: Some typical operations on strings are provided by this service, such as counting the number of words, searching for a particular word in a string, calculating the similarity measure between two strings (using Levenshtein distance metric); (iv) **HTTP request**: This service provides access to HTML pages. For instance, page requesting and page saving operations are available by this service; (v) **XML management**: It provides basic operations on XML files, for example: loading, saving, searching; (vi) **Wordnet access**: It uses an English word as input and returns all synonyms of it suggested by the WordNet lexical database.

3.4. Application services module

This module contains the main services of the framework. They were implemented using a component-based approach, which increases reusability. As already stated, the application services are the hot spots of RetriBlog, i.e., they are the variable part of the framework. It follows a detailed description of each of the components available in this module.

3.4.1. Tag parsing component

This service is responsible for blog crawling, i.e., this component gathers the blogs that will be analyzed. In fact, the Tag Parsering component carries out the following subtasks in the crawling process: (i) access a blog search engine; (ii) capture blogs links; (iii) returns a list of blog url's matching some tag. The Retriblog framework provides two default blog search engines (Technorati[10] and Icerocket[11]).

3.4.2. Preprocessing component

One problem identified in text information retrieval systems is that some words have little representative value for the document. For example, words containing the same semantics as the words in the masculine and feminine, plural and singular. To solve these problems, before indexing the documents should be submitted to a preprocessing (Hotho et al., 2005).

The current version of the RetriBlog can perform six types of preprocessing: (i) *CleanHTML*, it is responsible for cleaning all HTML tags; (ii) *EnglishFiltering* (Frakes & Baeza-Yates, 1992), which removes English stop words; (iii) *PortugueseFiltering*, it removes Portuguese stop words; (iv) *EnglishStemming*, performing a classical stemming technique (Porter, 1997) to English, reducing a word to its lemma; (v) *PortugueseStemming*, performing a classical stemming technique to Portuguese Porter (1997); and (vi) WhiteSpace, which removes extra white spaces.

3.4.3. Content extraction component

This component detects and extracts relevant textual content from blogs posts. It ignores navigational menus, advertisements banners, and structural information presented on them. In order to achieve this, the RetriBLOG provides three state-of-the-art CE algorithms: (i) *Document Slope Curve* (DSC); (ii) *Text-to-tag ratio*

[10] http://technorati.com.
[11] http://www.icerocket.com.

(TTR); (iii) *Link quota filter* (LQF). These three algorithms make use of ratios and proportions to detect the main textual content of the document.

The DSC, as shown by Pinto et al. (2002), treats a web page as a sequence of tokens. These tokens can be html tags or text. When there is a section of the webpage with a low number of HTML tags, the algorithm considers this section as relevant textual content and extracts it. Menus and structural information supposedly contain a high number of tags and a small amount of text; therefore they are ignored by the algorithm.

The TTR algorithm, as presented by the authors in Weninger and Hsu (2008), considers the ratio of the count of non-HTML-tag characters to the count of HTML-tags per line. When there is a line with a high TTR ratio, i.e., there are a lot of text characters and just a few HTML-tags in one line, the algorithm considers the line to be important content; otherwise, the line is ignored by the algorithm.

The LQF was implemented as a complement to the others two algorithms in order to reduce the number of link lists, advertisements and structural information. In LQF, each block level node of the DOM-tree is analyzed. The algorithm looks for the ratio of text which is inside link nodes to text outside link nodes (Gottron, 2007). When this ratio is above a defined threshold the entire block is discarded.

In addition to those algorithms, the *Boilerplate detection* (Kohlschütter et al., 2010) was integrated to the framework. The Boilerplate Detection algorithm uses text features, structural information and densitometric features to find the content sections of a website. First, the web pages are segmented into atomic text blocks. These blocks are then annotated with the aforementioned features. Then, decision trees and linear support vector machines are used to classify the sections into content or boilerplate.

Since all the above mentioned algorithms are based on ratios, text features and structural information, they are language independent and do not require any training corpus. Furthermore, they are very efficient for processing large datasets.

3.4.4. Classification component

The Text Classification (Text Categorization) consists in classifying a set of documents into a fixed number of predefined categories based on its contents. Each document may be assigned in one, multiple or no category at all (Sebastiani & Delle Ricerche, 2002).

Supervised text classification is an important task in this work, because it can arrange the pages gathered by our framework into groups. As a result, common topics presented in a corpus can be found out, also providing the user an easy way to access information according to his/her topics of interest.

RetriBlog relies on three different algorithms for supervised text classification: Naive Bayes, KNN (k-Nearest Neighbor), and TF/IDF (Term-Frequency/ Inverse Document Frequency). These classifiers were implemented using LingPipe[12] which is a toolkit for text processing through computational linguistics techniques. It provides a large number of approaches for text classification.

The Naive Bayes algorithm is a probabilistic classifier, based on Bayes Theorem Mitchell (1997). It assumes that all attributes of the data training are independent of each other given the context of the class, ignoring thus, possible dependencies.

The KNN algorithm (Baoli & Qin, 2003) finds k (in general, k is a small positive integer) nearest neighbors among the training documents and the new input, assigning the category of the majority of neighbors to the latter.

The TD/IDF algorithm is based on term-frequency and inverse document frequency weights. Such weights consist of a statistical measure used to evaluate how important a word is to a document in a collection of documents. The term-frequency is the ratio of the number of occurrences of a term in the document to the number of terms in the document. The inverse document frequency represents the importance of the term in the corpus; it can be obtained by taking the logarithm of the ratio of the number of documents to the number of documents that contains the given term.

3.4.5. Tag Recommendation component

Social tagging recently emerged as a mean to ease blog searching by making use of the tags provided by the user community. The technology behind social tagging allows users to tag blog posts and share their own tags with the web community. Search engines may take advantage of these tags to provide better search results. However, the same blog post may be tagged differently due to differences in users' culture, background knowledge, training and experiences. Moreover, according to Golder and Huberman (2006), three types of problems may occur: (i) *synonym*: when there are multiple words with the same meaning; (ii) *polysemy*: the same word might refer to different concepts; (iii) *basic level variation*: the variation of tag granularity. One example of the latter type of problem arises when bloggers post about their pet, they may assign general tags like "animal" or "domestic animal" or specific ones, like "dog" or "cat".

With the aim of addressing the above mentioned problems, RetriBlog provides four tag suggestion services, which are categorized into tag-based and post-based approaches. Currently there are four recommendation services implemented: (i) Similarity, (ii) Dictionary, (iii) Classification and (iv) Clustering.

The Similarity-based service is focused on the spelling of words. These services deal with plural, verb conjugation, gender difference, and words with the same stem. It is based on the Levenshtein Distance (Miller et al., 2009). This distance metric is given by the minimum number of operations needed to transform one string into another, with the allowable edit operations being insertion, deletion, or substitution of a single character. The Levenshtien metric widely used for applications that need to determine how similar two strings are.

The dictionary-based service treats synonyms of words. This service addresses words with similar semantics. A definition of synonymous is a word having the same or nearly the same meaning as another word or other words in a language. This method uses a synonyms dictionary (thesaurus) which is a simple index that links a word to their synonyms.

The classification-based service is focused on a bag of words classifier which uses the words of blog posts and their respective frequency to generate a Naive Bayes classifier. The Naive Bayes classifier is a simple probabilistic classifier based on Bayes Theorem with independence assumptions. This classification algorithm works as follows: first, in a learning phase, it generates a list of words with their frequencies from the input corpus. Each word in this generated list is labeled with the class it belongs to; resulting in a sort of "dictionary" for each class. Then, a tree is constructed from this class dictionary whose leaves are classes and intermediate nodes indicate probabilities calculated according to the Bayes theorem. Finally, when a new text is submitted to the classifier, it traverses the generated tree until it finds a leaf that indicates the class that best match the words in the input text. Our implementation of the clustering-based service consists of a classical version of the KNN (k-nearest neighbor) cluster algorithm. In the KNN algorithm, an object is classified by a majority vote of its neighbors, with an object being assigned to the most common class among its k nearest neighbors.

The main contribution of these services are to handle the aforementioned problems differently, allowing developers to choose the appropriate kind of service according to their requirements. For

[12] http://alias-i.com/lingpipe/.

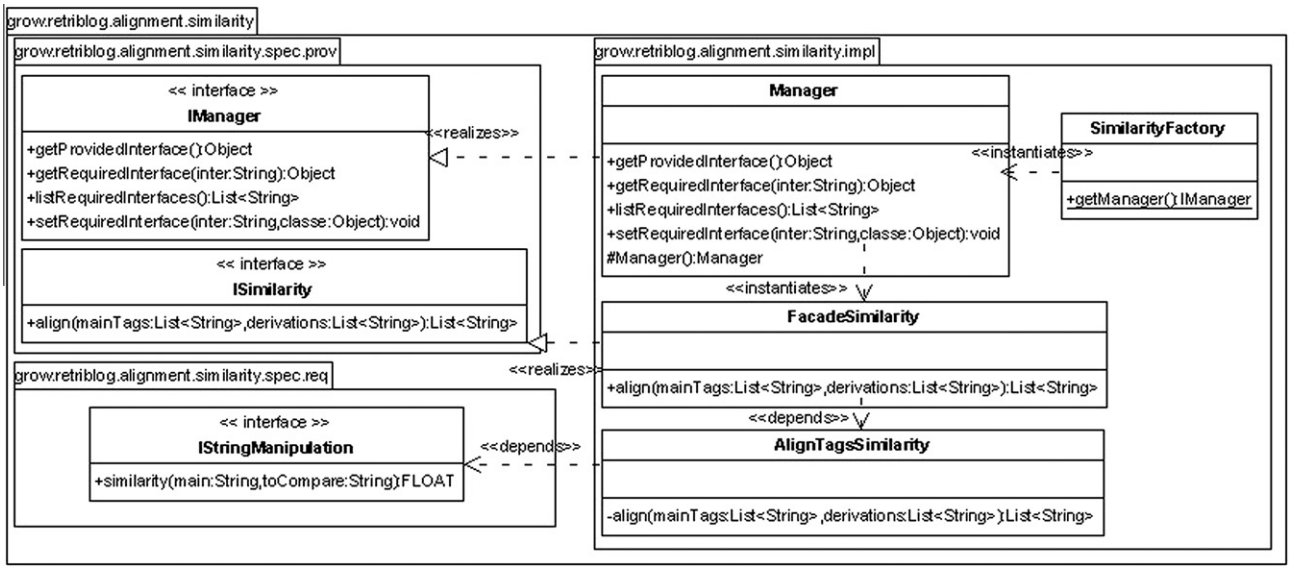

Fig. 10. Similarity component.

example, if users have a corpus with low level of noise, they may use a post-based service to achieve better results. On the other hand, if they need a fast implementation, tag-based services are more appropriate to the task at hand. Another example is if the user want to deal specifically with the synonym problem. He/she must use the Similarity-based service.

3.4.6. Indexing component

In order to allow fast searching for relevant documents in a large collection of documents, our framework uses inverted indexes (Hatcher & Gospodnetic, 2004). An inverted index consists of an index data structure in charge of storing a mapping from words to its locations in a document or a set of documents.

The RetriBlog framework provides an indexing service that uses the Lucene[13] to perform textual indexing of blogs. This indexing component creates an inverted index that can be easily accessed.

One of the features of Lucene is that it works with any type of file regardless of layout or even the language. This means that Lucene can be used to index and search data stored in various file formats, including .html pages (local or remote), .txt, * .doc, .pdf, or any other formats which has textual information in it.

3.5. Toolkit and persistence modules

The Toolkit Module has several interfaces to the following set of APIs and tools: (i) Lucene and Lingpipe, for extraction and text retrieval; (ii) Hibernate and Elmo, which handle data persistence transparently; (iii) HttpClient, for HTTP page retrieval; (iv) Google Language Detection, to detect the language of the blogs textual content; (v) Boilerpipe, to extract the main content of the page; (vi) SimMetrics, it compares similarity between strings. The Toolkit module makes transparent the use of each one of these tools providing an effortless access to them, which decreases the learning time. For instance, users do not to go deeper in details about Lucene API in order to create an application that actually uses it. Instead, they could delegate to the Lucene's functions of the framework.

The Persistence Module is responsible for the storage. It supports MySQL databases. The system has a class that represents the blog post, with the following attributes: (i) Id, Instance identifier, automatically generated by the database; (ii) Title, post title; (iii) Author, post author; (iv) Excerpt, post summary; (v) Created, the date of posting; (vi) Permalink, post link; (vii) HTMLText, post text with HTML; (viii) CompleteText, post text without HTML; (ix) AnalyzedText, post text after preprocessing; (x) Charset, page codification; (xi) Category, post category; (xii) Search Engine Indexing, the search engine that maintains a blog indexing service.

3.6. Component-based implementation

The component-based implementation of the framework is based on the COSMOS* implementation model (Aguilar Gayard et al., 2008). This model contributes to improving the traceability between the software architecture and the source code. COSMOS* uses programming language features, such as interfaces, classes and packages, and a set of design patterns to represent software architectures explicitly in the program code. Furthermore, the guidelines defined by the COSMOS* model also improves the system modularity and the internal evolution of the software components. The framework's source code is available for download.[14]

In order to illustrate the implementation, Fig. 10 shows the class diagram of the Similarity component, which implements a technique of tag recommendation.

According to the COSMOS* model, users only have access to the *spec* package and the *SimilarityFactory* class. The *IManager* interface is defined by COSMOS* and provides the component basic operations. For example, operations for getting instances for the provided interfaces of the component (*getProvidedInterface*), and for binding an object that implements a required interface (*setRequiredInterface*). The *ISimilarity* interface consists of a provided interface that specifies the operations to the tag recommendation service.

4. Case study

This section presents a three case studies created using RetriBlog. The case studies emphasize the advantages of using the components-based and architecture-centered approaches. They serve mainly to show the flexibility and reusability of the framework.

[13] http://lucene.apache.org.

[14] https://sourceforge.net/projects/retriblog/.

4.1. Case study 1: agent based system to retrieve blogs from Tecnhorati

This application crawls blogs from Technorati. The main motivation of this choice is that the company stores many blogs in distinct languages and it is constantly researching about the state of the Blogosphere. The application is working with blogs in English due to the fact that techniques for preprocessing are particular to each language. The following sections explain the architecture of agents, and configuration of the crawler.

4.1.1. Architecture of agents

The proposed architecture is comprised of agents as well as the configuration and persistence modules. Fig. 11 presents the details of this architecture.

The architecture of agents defines specific agents, besides that, it provides some extension points to add new functionalities to the framework. These agents are built using the Jade framework (Bellifemine, Caire, & Greenwood, 2007), and some extensions have been done to increase the flexibility of the framework. The features in the architecture are described as follow:

- **Manager agent (MA)**: This agent manages the agents of the system. It is responsible for adding and removing them of the system. It provides the following behaviors: (i) Management – it verifies the information contained in the configuration layer in order to add a new searcher agent; (ii) Addition – it inserts new agents in the environment; (iii) Removal – it removes agents from the environment.
- **Searcher agent (SA)**: This agent is responsible for crawl the Blogosphere. In other words, it retrieves information from Blogs, stores the information in a MySQL database and creates an inverted index. It is classified by categories and performs a flexible search behavior. Moreover, it uses information contained in the configuration module to make its implementation.
- **Configuration**: The configuration module describes the necessary information for the implementation of SAs, such as indexing company and blog language. It adopts an ontology to describe the searcher agents.
- **Persistence**: This module uses the indexing and access to the MySQL database services provided by the framework.

4.1.2. Configuration of RetriBlog

The components selected to create the crawler were:

- Technorati, the searcher agent accesses the link of Technorati in its tag to make the parser to obtain information of blogs such as posts, links, and blog summary;
- It extracts content using summary strategy;
- It uses English Filtering service to preprocess step;
- It uses 2 schemes of indexing proceeding from tool Lucene, Keyword and Text, and saves the collected data in the MySQL Database.

To load these components is necessary to create the following specification code (Fig. 12):

This code follows the model COSMOS*:

- Instance interfaces and their managers (lines 3–12);
- Set the required interfaces (lines 16–35);
- Receives the provided interfaces (lines 37–47).

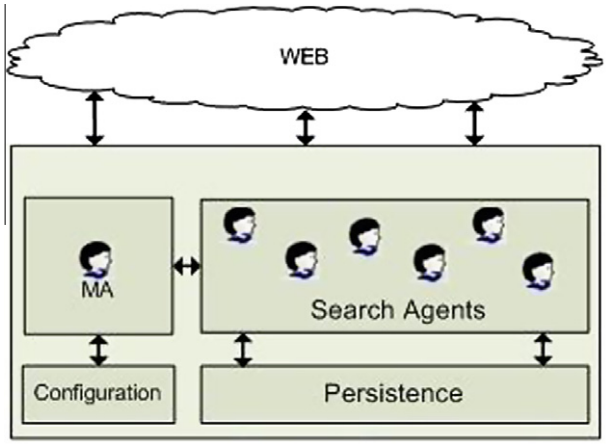

Fig. 11. Architecture of agents.

```
1  public class Case1Specification {
2
3    ITechnoratiUrlCollector  urlCollector;
4    ISummaryStrategy         contentExtraction;
5    IEnglishFiltering        preprocessing;
6    ILuceneIndexing          index;
7    String                   language;
8
9    IManager managerUrlCollector;
10   IManager managerContentExtraction;
11   IManager managerPreprocessing;
12   IManager managerIndex;
13
14   public Case1Specification(){
15
16     managerUrlCollector =
           TechnoratiUrlCollectorFactory
17     .getManager();
18     managerContentExtraction =
           SummaryStrategyFactory
19     .getManager();
20     managerPreprocessing = EnglishFilteringFactory.
           getManager();
21     managerIndex = LuceneIndexingFactory.getManager
           ();
22
23     managerUrlCollector.setRequiredInterface("
           IHttpUtils",
24     HttpUtilsFactory.getManager().
           getProvidedInterface());
25
26     managerContentExtraction.setRequiredInterface("
           IHttpUtils",
27     HttpUtilsFactory.getManager().
           getProvidedInterface());
28
29     managerContentExtraction.setRequiredInterface
30     ("IWhiteSpaceRemover", WhiteSpaceRemoverFactory.
           getManager()
31     .getProvidedInterface());
32
33     managerContentExtraction.setRequiredInterface
34     ("IStringManipulation",
           StringManipulationFactory
35     .getManager().getProvidedInterface());
36
37     urlCollector = (ITechnoratiUrlCollector)
38     managerUrlCollector.getProvidedInterface();
39
40     contentExtraction = (ISummaryStrategy)
41     managerContentExtraction.getProvidedInterface();
42
43     preprocessing = (IEnglishFiltering)
44     managerPreprocessing.getProvidedInterface();
45
46     index = (ILuceneIndexing)
47     managerIndex.getProvidedInterface();
48
49     language = "en";
50 }
```

Fig. 12. Specification class – case study 1.

4.2. Case study 2: A blog recommendation system for helping students in educational forum interactions

This case study describes a blog recommendation system created using the proposed framework. This case study is divided into two parts: (i) blog crawler, and (ii) blog recommendation. These applications are designed for helping students in MASSAYO environment.

MASSAYO (Bittencourt, Costa, Silva, & Soares, 2006) aims to build an educational platform based on Semantic Web technologies, offering the following advantages: (i) extensibility, (ii) participatory knowledge creation; and (iii) user adaptability.

In face of this context, the applications proposed in this case study deal with these aspects in the forum tool. First, it is extensible because the developer is allowed to create new applications by changing few lines of code. Then, it uses texts from blogosphere ensuring universal and participatory knowledge creation. Finally, the recommendation system tracks each student and recommends specific blogs for each one.

Next section presents the blog crawler and blog recommendation applications.

4.3. Blog crawler

The application crawlers IceRocket's blogs from categories related to software engineering, and stores it in a MySQL database. As a result, an index with blog's information is created. The recommendation system, detailed in next section, use this information to suggest blogs to students.

In order to create this application is necessary to choose the alternative components and possible optional components. Firstly, we choose Lucene Index (Index), IceRocket (tag parser), TTR (content extraction) as alternative components, and English Filtering as optional. In addition, the application uses Clean HTML, White Space Remover, Lower Case (preprocesing) as mandatory components, i.e. every time when the framework is instantiated they are loaded.

These services were chosen because the application requirement was performance. Therefore, the TTR and English Filtering components are more suitable in this case because they use simple algorithms to perform these operation.

Code used to load these components (Fig. 13):

This specification is similar to the first case study:

- Instance interfaces and their managers (lines 3–12);
- Set the required interfaces (lines 16–34);
- Receives the provided interfaces (lines 39–46).

```
1  public class Case2Specification {
2
3    IIceRocketUrlCollector    urlCollector;
4    ITTR                      contentExtraction;
5    IEnglishFiltering         preprocessing;
6    ILuceneIndexing           index;
7    String                    language;
8
9    IManager managerUrlCollector;
10   IManager managerContentExtraction;
11   IManager managerPreprocessing;
12   IManager managerIndex;
13
14   public Case2Specification(){
15
16     managerUrlCollector =
           IceRocketUrlCollectorFactory
17     .getManager();
18     managerContentExtraction = TTR Factory.
           getManager();
19     managerPreprocessing = EnglishFilteringFactory.
           getManager();
20     managerIndex = LuceneIndexingFactory.getManager
           ();
21
22     managerUrlCollector.setRequiredInterface("
           IHttpUtils",
23     HttpUtilsFactory.getManager().
           getProvidedInterface());
24
25     managerContentExtraction.setRequiredInterface("
           IHttpUtils",
26     HttpUtilsFactory.getManager().
           getProvidedInterface());
27
28     managerContentExtraction.setRequiredInterface
29     ("IWhiteSpaceRemover", WhiteSpaceRemoverFactory.
           getManager()
30     .getProvidedInterface());
31
32     managerContentExtraction.setRequiredInterface
33     ("IStringManipulation",
           StringManipulationFactory
34     .getManager().getProvidedInterface());
35
36     urlCollector = (IIceRocketUrlCollector)
37     managerTagParser.getProvidedInterface();
38
39     contentExtraction = (ITTR)
40     managerContentExtraction.getProvidedInterface();
41
42     preprocessing = (IEnglishFiltering)
43     managerPreprocessing.getProvidedInterface();
44
45     index = (ILuceneIndexing)
46     managerIndex.getProvidedInterface();
47
48     language = "en";
49 }
```

Fig. 13. Specification class – case study 2.

4.4. Recommendation system

This system recommends blogs for helping students in MASSAYO educational forum. In summary, the system monitors the forum discussion among the students and, in case of identifying students questions, the system recommends blogs for them in order to overcome doubts. The recommended blogs were crawled using the application described in the previous section.

The main objective of this system is to highlight the facility provided by our framework when reusing components. It uses the Naive Bayes component for classification and English filtering in its preprocessing step.

In the Recommendation system, posts may be classified as (i) *Doubt*, student questions appeared in forums; (ii) *Explanation*, when questions issues are explained; and (iii) *Neutral comment*, comments unrelated to any of the above classes. The system only recommends blogs if a post is classified as a doubt. Fig. 14 shows how the overall recommendation process works.

- **User interaction**: The user interacts with the forum on the environment, and every post is sent to the recommendation system;
- **Post classification**: The recommendation system classifies (via the Naive Bayes component) a user's post as a doubt, an explanation or a neutral comment;
- **Post processing**: Using the English Filtering component, the recommendation system filters out stop words prior to text processing;

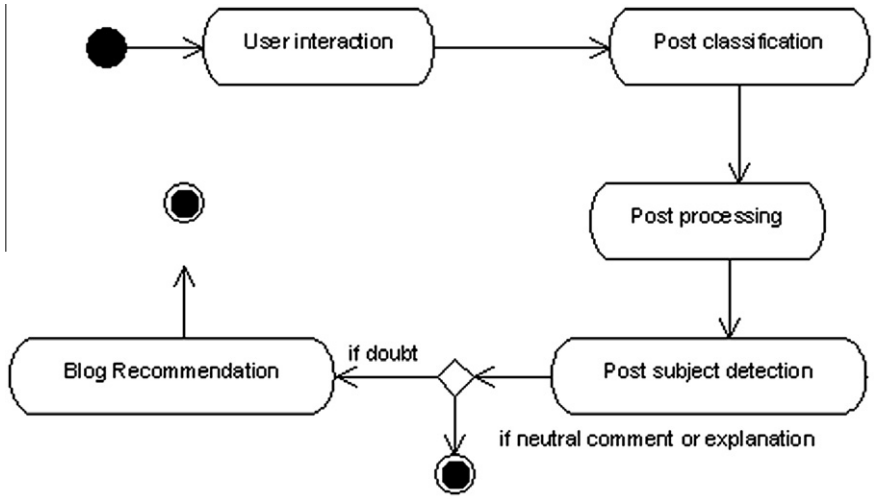

Fig. 14. Activity diagram.

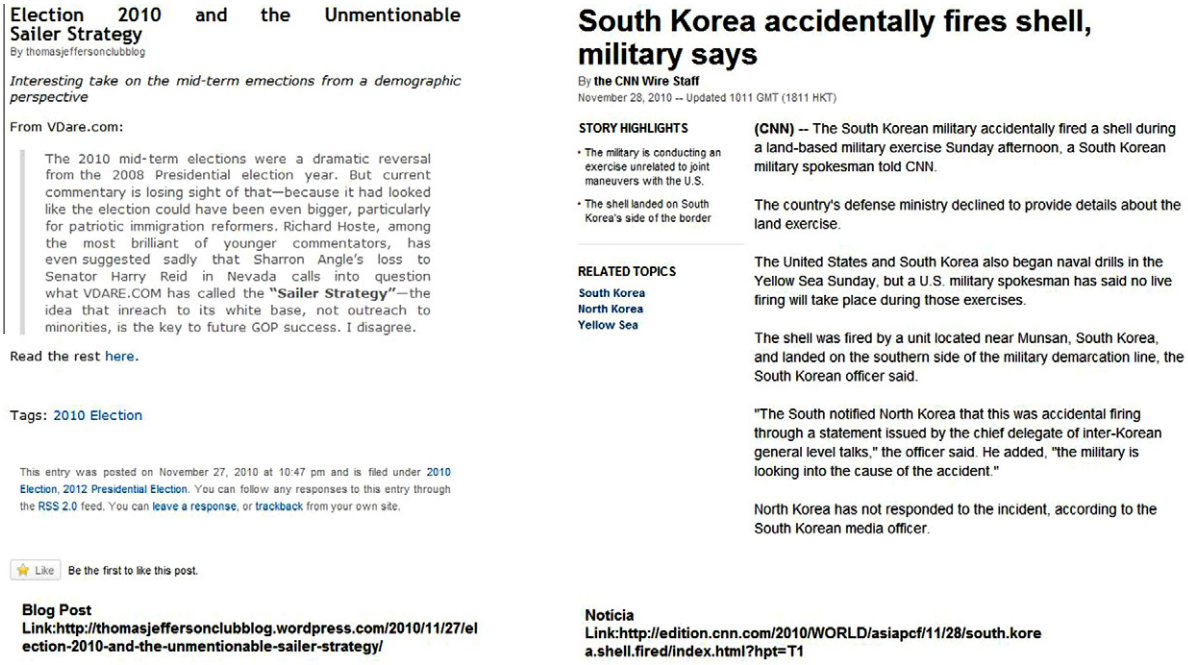

Fig. 15. Similarity between blog and news.

- **Post subject detection**: In order to determine the subject of the post, the recommendation system uses course keywords provided by the MASSAYO environment. Course keywords are created when a course starts. Next, the system matches these keywords with those from user's post with the aim of find out its subject;
- **Blog Recommendation**: The recommendation system tests if a post concerns user's doubts. If that is the case, then the system suggests a blog to the user; it does nothing, otherwise.

This system uses two services provided by the proposed framework. It shows that services can be easily reused by other applications even if they are not blog crawler.

4.5. Case study 3: News crawler for clipping

The term clipping defines the process of selecting news from media such as newspapers, magazines and websites, to result in a set of clippings on a specific subjects (Baeza-Yates & Ribeiro-Neto, 1999). Journalists use it in various purposes including select similar news aiming to define the accuracy of this news.

However, before doing the clipping is necessary to crawl the news. With this purpose we create a news crawler. The objective of this case study is to show that RetriBlog could also be used to create news crawlers easily. It happens because the structural similarity between news and blogs, Fig. 15 shows this similarity.

This application retrieves news from three sites of Pernambuco, they are: *FolhaPE*, *NE10*, *Journal of Pernambuco*.[15] As the crawler collects URLs from three different sites, one process is create for each site, i.e., three components were used collector URLs that were created specifically for this application. The components chosen for the application:

[15] http://www.folhape.com.br/ www.ne10.uol.com.br,http://www.diariodepernambuco.com.br/.

- **URL collector**: FolhaPE, NE10 and Journal of Pernambuco;
- **Content extraction**: DSC;
- **Preprocessing**: Portuguese Filtering;
- **Indexing**: Lucene Index.

```
1  public class Case3Specification {
2
3    IFolhaUrlCollector      urlCollector1;
4    INE10UrlCollector       urlCollector2;
5    IDiarioUrlCollector     urlCollector3;
6    IDSC                    contentExtraction;
7    IPortugueseFiltering    preprocessing;
8    ILuceneIndexing         index;
9    String                  language;
10
11   IManager managerUrlCollector1;
12   IManager managerUrlCollector2;
13   IManager managerUrlCollector3;
14   IManager managerContentExtraction;
15   IManager managerPreprocessing;
16   IManager managerIndex;
17
18   public Case3Specification(){
19
20     managerUrlCollector1 = FolhaUrlCollectorFactory.
           getManager();
21     managerUrlCollector2 = NE10UrlCollectorFactory.
           getManager();
22     managerUrlCollector3 = DiarioUrlCollector.
           getManager();
23     managerContentExtraction = DSCFactory.getManager
           ();
24     managerPreprocessing =
           PortugueseFilteringFactory
25     .getManager();
26     managerIndex = LuceneIndexingFactory.getManager
           ();
27
28     managerUrlCollector1.setRequiredInterface("
           IHttpUtils",
29     HttpUtilsFactory.getManager().
           getProvidedInterface());
30
31     managerUrlCollector2.setRequiredInterface("
           IHttpUtils",
32     HttpUtilsFactory.getManager().
           getProvidedInterface());
33
34     managerUrlCollector3.setRequiredInterface("
           IHttpUtils",
35     HttpUtilsFactory.getManager().
           getProvidedInterface());
36
37     urlCollector1 = (IFolhaUrlCollector)
38     managerTagParser.getProvidedInterface();
39
40     urlCollector2 = (INE10UrlCollector)
41     managerTagParser.getProvidedInterface();
42
43     urlCollector3 = (IDiarioUrlCollector)
44     managerTagParser.getProvidedInterface();
45
46     contentExtraction = (ITTR)
47     managerContentExtraction.getProvidedInterface();
48
49     preprocessing = (IPortugueseFiltering)
50     managerPreprocessing.getProvidedInterface();
51
52     index = (ILuceneIndexing)
53     managerIndex.getProvidedInterface();
54
55     language = "pt";
56 }
```

Fig. 16. Specification class – case study 3.

The code of this application is similar to code shown in the other two case studies. However, it presents a fundamental difference, three collectors are instantiated. Fig. 16 shows the specification.

The explanation of the code is:

- Instance interfaces and their managers (lines 3–16);
- Set the required interfaces (lines 20–35);
- Receives the provided interfaces (lines 37–53).

4.6. Discussion about the case studies

The case studies had two main goals: (i) show the system flexibility by instantiating the framework in different contexts, and (ii) show the advantage of using an architecture-centered approach. For this reason we used the maximum of different components in the implementation of case studies.

The first point has been satisfied since we use different services of the framework in various contexts. With blogs and news, different algorithms, applied to education, among others.

The second point is very evident in the explanations of the codes of the specification classes. In every case the explanation was the same, changing only the lines of code which it referred. This shows that we only choose the components and create the configuration class, which is basically the same every time, to generate an implementation of the framework.

Other points that are evidenced in these case studies are:

- The system presented is a gray-box framework, since the user only needs to know the interfaces to instantiate the system, he does not need to know the details of each class;
- It is a domain and application framework, because their services were made to deal with the crawler blogs (domain), but can be used for other purposes such as news crawler (application).

5. Preliminary experiments

This section describes the experimental setup conducted in our research work and presents the preliminary results. Experiments were performed in three services: (i) Content Extraction, as a result we have a comparison between the algorithms and the results were as expected; (ii) Classification, as a result it was proved that using content extraction and preprocessing improves the classification; and (iii) Tag Recommendation, as a result we have the comparison between the algorithms and some rules generated. In addition, a qualitative evaluation about using COSMOS* is presented.

5.1. Content extraction

For the experiments concerning to the CE evaluation, we used the news dataset described in Kohlschütter et al. (2010). This dataset consists of 621 manually collected news articles from 408 news websites. Each document was annotated identifying news headlines and its main content. These features were used for testing the CE algorithms.

The performance evaluation is based on the classical measures used in IR systems, i.e., precision P, recall R, and F1-measure (Baeza-Yates & Ribeiro-Neto, 1999). Precision is the ratio of extracted relevant items to all extracted items. Recall is defined as the ratio of the extracted relevant items to all relevant items. Based on those two concepts, the F1-measure provides a suitable measure defined as the harmonic means between precision and recall. Table 1 shows the results for each CE algorithm/heuristic.

As detailed in Table 1, the LQF heuristics was able to get rid of some link sections and advertising, considerably improving the

overall performance of the DSC algorithm, which has obtained the best performance.

On the other hand, in the CE experiments, we concentrated our analysis on a threshold proportional to the amount of text of the news page. Therefore, CE algorithms might make mistakes when processing news pages with many or longer comments, because the threshold for extraction will be higher than usual. Another problem that we have faced with all the CE algorithms, was that they sometimes failed to extract small text elements, which are separated from the main content, e.g., publication date. However, we expect that this problem can be solved by using simple heuristics based on the attributes of these elements.

Regarding CE issues, both DSC and TTR algorithms presented some problems to detect textual content in news web pages which contains an elevated number of comments. Comment sections in news pages may bias the algorithm due to its variable ratio of HTML tags to text. On the other hand, the LQF heuristics was able to get rid of some link sections and advertising, considerably improving the overall performance of the DSC algorithm.

5.2. Classification

We selected 270 articles from the news dataset described above. These articles were manually labeled, i.e., we assigned them to one of the four categories (Sport, Entertainment, Technology, and World) as shown in Table 2. This second dataset was used for training and testing the classification algorithms.

Similarly, we adopted the classical measures in IR field (P, R, and F1) to evaluate the classification results. The tests were performed using 10-fold cross validation.

For each classification algorithm discussed here, we aimed to analyze the influence of CE and preprocessing on the classification results. For that, we performed the following evaluations: (i) page classification with no preprocessing; (ii) page classification after content extraction (CE); and (iii) page classification after CE and stop words removal.

Considering the classification results, Table 3 shows that the best F-measure (over than 90%) was achieved by the TF-IDF weighting. Furthermore, our initial hypothesis was confirmed: all classification algorithms obtained better performance after the CE preprocessing phase. We argue that CE preprocessing should be carefully considered as a crucial component in news pages retrieval architectures based on classification.

5.3. Tag recommendation

The blog dataset used in all experiments of this section was gathered by the proposed framework first presented in Ferreira et al. (2010). This framework has been instantiated to collect blog posts from Technorati (http://technorati.com) and Icerocket (www.icerocket.com), collecting a total of 623 posts written in English language, and stored in MySQL database tables. For our experiment purposes, we are interested in the following elements of the blog post: title, post link, summary, text in HTML, text without HTML tags, and tag. Table 4 shows the tag distribution for this dataset. The blog tags were randomly selected among the most used tags according to the indexing services used in the experiments. After that, the blog posts related to these tags were crawled.

The results are expressed in terms of the classical metrics from IR: precision, recall, and F-measure. In this case, precision P measures how many tags were correctly aligned, whereas recall R indicates how many tags were aligned, considering the total set of relevant tags in the overall dataset.

After finishing the experiments, P and R values were obtained by human evaluation, i.e., the generated rules by the services were analyzed by four human evaluators. The final values were determined in agreement reached among them. Fig. 17 shows a comparison among the results achieved by the Similarity, Classifier and Clustering-based services.

The performance values of similarity-based service were obtained when the degree of similarity was between 65% and 70%, but the results above 55% are considered acceptable. Considering the classifier and clustering-based results, we can see that education and sports tags had better performance than entertainment tag. One reason that could explain that is the use of different lan-

Table 1
Content extraction results.

Algorithm	Precision (%)	Recall (%)	F-measure (%)
TTR	71.09	87.38	78.39
Boilerplate	93.19	91.69	92.44
DSC	**94.33**	88.29	91.21
DSC + LQF	92.37	**98.28**	**95.23**

Table 2
Category frequency distribution.

Category	Frequency
Sport	70
Entertainment	70
Technology	60
World	70

Table 3
Classification results.

Algorithm	Precision (%)	Recall (%)	F-measure (%)
Naive Bayes	60	44	50.7
Naive Bayes (CE)	75	50	60
Naive Bayes (CE + P)	89.2	78.5	83.5
KNN	68.4	55	60.9
KNN (CE)	71.4	68.5	69.9
KNN (CE + P)	80	73.2	76.4
TF-IDF	65.4	60	62.5
TF-IDF (CE)	80.2	75	77.5
TF-IDF (CE + P)	91.3	89.2	90.2

Table 4
Tag distribution for the blog dataset.

Tag name	Freq	Tag name	Freq
Advertising	10	Government	5
Amusement	9	Hollywood	9
Athletics	10	Humor	4
Book	33	Jazz	8
Booking	9	Learning	8
Books	10	Literature	6
Business	33	Manner	8
Bussinesslike	5	Mp3	5
Community	8	Music	38
Cooking	10	Nba	5
Corruption	3	News	8
Didactics	9	Nlf	10
Drama	4	Obama	8
Ebooks	7	Oscars	9
Economics	10	Pedagogy	10
Economy	19	Photography	5
Education	39	Politics	37
Engineering	10	Sport	35
Entertainment	39	Style	6
Fashion	38	Teaching	6
Fashionable	9	Technology	36
Fashions	7	Travel	7
Film	3	Writing	7
Football	9		

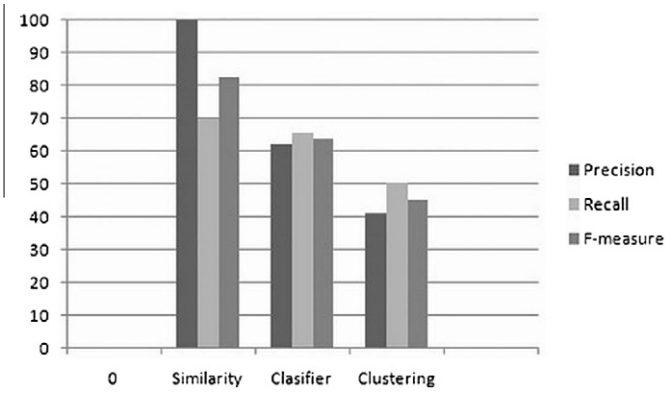

Fig. 17. Evaluation results of tag recommendation services.

guage styles (formal, informal, colloquial, etc.) adopted in blog writing. Since they have connections with organizations rather than individuals, educational and sports blog posts tend to be more formal and better structured. Thus, we had an improvement in the classification results, because the algorithms deal with more "correct" words. On the other hand, blog posts about entertainment are mainly written by public, and they tend to contain colloquial sentences and slang language which might mislead the classifier. Finally, all the synonyms suggested by the WordNet lexical database for a word are considered relevant, for this reason, the dictionary-based service results have not been included in the Fig. 17.

5.4. COSMOS* evaluation

This section discusses some lessons learned and benefits of designing a software framework using an architecture-centered approach and COSMOS*. The main properties identified are:

- **Low instantiation cost**: Since there is a clear definition of the framework's variability, the software developer is able to easily identify the framework's hot spots and then, instantiate it according to the preferences of the desired application. Moreover, our framework provides some example implementation of possible concrete classes and services that realizes the main hot spots. The software developer can also develop other variants, if necessary;
- **Understandability**: The high-level representation provided by the architecture-driven approach allows even low programmers to understand both the details of the framework execution, and the structural aspects for extending it. The uniform decomposition strategy, which was used in the architectural design phase, has made this characteristic more explicit;
- **Modularity and evolution**: A benefit of adopting an architecture-centered strategy for developing the framework is that we can have well defined modules. These modules have specific roles and they are able to communicate with each other via explicit interfaces. In other words, besides reducing the coupling between the components of the system, which allows changes to be made locally at specific points in software modules, it also eases the task of adding and removing a new component;
- **Manageability**: A hybrid architecture, which is composed of layered and independent components, underpins the proposed framework. This architecture can promote parallelism in the development process. Additionally, as already mentioned, the components are self-contained and the developers only need to know their interfaces. Therefore, the management of the components becomes much easier to retain;
- **Monitoring**: Due to the benefits of modular programming, the development time would be shortened because separate groups could work on each component with little need for communication. As a result, it is easier to monitor separate components (subsystems) than the entire system;
- **Reusability**: One obvious consequence of a low coupling and modularity is that each component is more independent and self-contained, which improves the framework reusability. Therefore, it is possible to make changes to a component without change the other ones. Moreover, the components can be used separately in other applications, such as in the recommendation system.

Moreover, the disadvantages are:

- **Internal evolution**: In order to add new features or new points of variation in the framework, the developer needs to know the COSMOS* model and understand the system code. In other words it must have a higher learning curve;
- **Scalability**: Use explicit connectors increases the flexibility and improves outcome. However, this greatly increases the number of indirections which may harm some quality requirements such as performance and memory consumption. However, with this type of system is easier to implement distribution, a possible solution to this problem;

6. Related work

This section describes related work to Retriblog and a comparison among them is performed.

The main systems which provide services to create application with blogs could be divided into two blocks. The first is composed by Blogscope Nitin Agarwal, Shamanth Kumar, Huan Liu, and Mark Woodward (2009), Bansal and Koudas (2007) and Fujimura et al. (2006), they provide services to obtain information from blogs, but they are not designed considering variability.

The second block is frameworks as well as RetriBlog. In this block there are the systems two systems: Chau et al. (2009) and Joshi (2006). Following each of these systems are more detailed.

Blogscope Bansal and Koudas (2007) consists of a system for online analysis of high volume temporally ordered text sources, currently applied to the analysis of the Blogsphere. This system warehouses and indexes the Blogosphere, extracting information in order to perform interactive analysis and information discovery. The main services provided by this tool are keyword detection, support for online OLAP analysis, and summary extraction.

Blogranger Fujimura et al. (2006) is a multi-faceted blog search engine that offers four specialized interfaces for the user: two for topic search, and two others for blogger and reputation search. A graphical user interface provides a filter that classifies the search results based on the intent of the search. In addition, this tool provides services for blog sentiment analysis, and tag detection.

BlogTrackers (Agarwal et al., 2009) is a Java-based desktop application providing a unified platform for crawling and analyzing blog data, which is stored in a relational database. This application grants the user the freedom to choose data of interest and helps in effectively analyzing it. Moreover, it provides services such as indexing, tag recommendation, classification, and content extraction by means of regular expressions.

In Chau et al. (2009) the authors proposed a blog mining framework that comprises spiders, parsers, content and network analyzers, and visualizers. Some services for information extraction, classification, indexing, and blog trend visualization are contemplated by this tool.

BlogHarvest Joshi (2006) consists of a blog mining and search framework that extracts the interests of the blogger, finding and

recommending blogs with similar topics. Some services, such as indexing, sentiment analysis, tag recommendation, and content extraction by using (by using regular expressions) are also provided.

The main difference between RetriBlog for these works is the use of software engineering techniques in this system implementation. Among all the papers presented only BlogHarvest somehow presents a system that provides variation points. All the others have components coupled. As stated in Section 3.1 the RetriBlog has several points of variation and was implemented using components which results in benefits described in Section 5.4.

Another main difference is that none of the work uses preprocessing services. Moreover, the services of content extraction presented are based on regular expression rules or manually produced. Thus they are strongly coupled one type of blog, i.e., for each new type of blog you must create a new rule extraction.

The common services were indexing, tag recommendation/detection and classification. On the other hand, services of sentiment analysis, OLAP analysis and extraction of abstract services are provided by these tools does not include the RetriBlog.

Finally, some of the works presented have interfaces and RetriBlog do not have it. This is due to the fact that the system proposed in this paper was created to assist the developer and not for the end user. Therefore, the interface module was not part of the project.

7. Conclusions and future work

The work presented RetriBlog, a framework to create a blog crawlers using an architecture-centered approach and following COSMOS* deployment model.

The main contributions of this work were: (i) Construction of mechanisms for information retrieval in the blogosphere (blog crawlers); (ii) Implementation of algorithms for content extraction in HTML pages; (iii) Creation of tags recommending services for blogs; (iv) Evaluation about the influence of the use of content extraction and preprocessing algorithms on pages classification performance; (v) Use COSMOS* model to build the framework; (vi) Creation of three case studies to assess the possible benefits of creating a architecture-centric software following the COSMOS* model.

The adoption of development focused on architecture enables a better developing complexity control. The use of software components, in particular COSMOS* model, improves tracking between the software architecture and code, which facilitates error identification and identification of possible evolution points, which is a potential reduction in cost maintenance.

The proposed framework differs from related work by the presence of a preprocessing step, which improves the quality of the final result by cleaning the data before extracting the information and provides services for content extraction, tag recommendation and classification.

As future work are suggested the following extensions and improvements:

- Use of a larger dataset to evaluate the content extraction, tags recommendation, classification algorithms;
- Calculate metrics such as execution time and computational cost of each algorithms to help users choose algorithms that best fits to his application requirements and goals;
- Creating new case studies in other areas such as e-commerce and e-government;
- Create an application directed for the end user, with a simple interface.

References

Nitin Agarwal, Shamanth Kumar, Huan Liu, & Mark Woodward (2009). Blogtrackers: A tool for sociologists to track and analyze blogosphere. In: *ICWSM*.

Leonel Aguilar Gayard, Cecília Mary Fischer Rubira, & Paulo Astério de Castro Guerra (2008). COSMOS*: A component system model for software architectures. Technical report IC-08-04, Institute of computing, University of Campinas.

Arasu, Arvind, Cho, Jungho, Garcia-Molia, Hector, Paepcke, Andreas, & Raghavan, Sriram (2001). Searching the web. *Acm Transactions On Internet Technology, 1*(1), 2–43.

Baeza-Yates, Ricardo, & Ribeiro-Neto, Berthier (1999). *Modern information retrieval* (1st ed.). Addison Wesley.

Nilesh Bansal, & Nick Koudas (2007). Searching the blogosphere.

Yu Shiwen Li Baoli, & Lu Qin (2003). An improved k-nearest neighbor algorithm for text categorization. In: *Proceedings of the 20th international conference on computer processing of oriental languages*.

Bass, Len, Clements, Paul, & Kazman, Rick (2003). *Software architecture in practices*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Bellifemine, Fabio Luigi, Caire, Giovanni, & Greenwood, Dominic (2007). *Developing multi-agent systems with JADE*. Wiley.

Bittencourt, Ig Ibert, Costa, Evandro de Barros, Silva, Marlos, & Soares, Elvys (2006). A computational model for developing semantic web-based educational systems. *Knowledge Based Systems, 22*(4), 302–315.

Blood, Rebecca (2004). How blogging software reshapes the online community. *Communications of the ACM, 47*(12), 53–55.

Brooks, Frederick P. Jr., (1987). No silver bullet essence and accidents of software engineering. *Computer, 20*(4), 10–19.

Chau, Michael, Xu, Jennifer, Cao, Jinwei, Lam, Porsche, & Shiu, Boby (2009). A blog mining framework. *IT Professional, 11*, 36–41.

Cho, Junghoo, & Tomkins, Andrew (2007). Guest editors' introduction: Social media and search. *Internet Computing, IEEE, 11*(6), 13–15.

Paul Clements, & Linda Northrop (2001). Software product lines: Practices and patterns (pp. 0201703327).

Czarnecki, Krzysztof, Helsen, Simon, & Eisenecker, Ulrich (2005). Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice, 10*(2), 143–169.

D'Souza, Desmond F., & Wills, Alan Cameron (1999). *Objects, components, and frameworks with UML: The catalysis approach*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Fayad, Mohamed E., Schmidt, Douglas C., & Johnson, Ralph E. (1999). *Building application frameworks: Object-oriented foundations of framework design*. New York, NY, USA: John Wiley & Sons, Inc.

Rafael Ferreira, Rinaldo J. Lima, Ig Ibert Bittencourt, Dimas Melo Filho, Olavo Holanda, Evandro Costa, Fred Freitas, & Lídia Melo (2010). A framework for developing context-based blog crawlers. In: *Proceedings of the IADIS international conference on WWW/Internet* (pp. 120–126).

Frakes, William B., & Baeza-Yates, Ricardo (1992). *Information retrieval: Data structures and algorithms*. Prentice Hall PTR.

Ko Fujimura, Hiroyuki Toda, Takafumi Inoue, Nobuaki Hiroshima, Ryoji Kataoka, & Masayuki Sugizaki (2006). Blogranger – a multi-faceted blog search engine. In: *Proceedings of 3rd annual WWE*.

David Garlan, & Mary Shaw (1994). An introduction to software architecture. Technical report, Pittsburgh, PA, USA.

Golder, Scott A., & Huberman, Bernardo A. (2006). Usage patterns of collaborative tagging systems. *Journal of Information Science, 32*, 198–208.

Gomaa, Hassan (2004). *Designing software product lines with UML: From use cases to pattern-based software architectures*. Addison Wesley.

Gottron, Thomas (2007). Evaluating content extraction on html documents. *ITA, 23*, 123–132.

Hatcher, Erik, & Gospodnetic, Otis (2004). *Lucene in action (In action series)*. Manning Publications.

Hotho, Andreas, Nnrnberger, Andreas, & Paa, Gerhard (2005). A brief survey of text mining. *LDV Forum – GLDV Journal for Computational Linguistics and Language Technology, 20*(1), 19–62.

Hurst, Matthew, & Maykov, Alexey (2009). Social streams blog crawler. In *ICDE '09: Proceedings of the 2009 IEEE international conference on data engineering* (pp. 1615–1618). Washington, DC, USA: IEEE Computer Society.

Jiang, Liangxiao, Wang, Dianhong, Cai, Zhihua, & Yan, Xuesong (2007). Survey of improving Naive Bayes for classification. In *Proceedings of the 3rd international conference on advanced data mining and applications, ADMA '07* (pp. 134–145). Berlin, Heidelberg: Springer-Verlag.

Johnson, Ralph E. (1997). Components, frameworks, patterns. *Communications of the ACM, 40*, 10–17.

Johnson, Ralph E., & Foote, Brian (1988). Designing reusable classes. *Journal of Object-Oriented Programming, 1*(2), 22–35.

Mukul Joshi (2006). Blogharvest: Blog mining and search framework. In: *Proceedings of the international conference on management of data COMAD*.

Kohlschütter, Christian, Fankhauser, Peter, & Nejdl, Wolfgang (2010). Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web search and data mining, WSDM '10* (pp. 441–450). New York, NY, USA: ACM.

Kruchten, Philippe, Henk Obbink, J., & Stafford, Judith A. (2006). The past, present, and future for software architecture. *IEEE Software, 23*(2), 22–30.

Kwanwoo Lee, & Kang, Kyo C. (2004). Feature dependency analysis for product line component design (pp. 69–85).

Lemur (2009), The lemur toolkit. <http://www.lemurproject.org/>. Accessed on December 2011.

Patrick Brito Ana Elisa Lobo, & Rubira, Cecflia M. F. (2007). A systematic approach for architectural design of component-based product lines. Technical report, Instituto da Computatpo – Universidade Estadual de Campinas.

Lucene, & Lucene (2001). <http://lucene.apache.org/java/docs/>. Accessed on December 2011.

Manning, Christopher D., Raghavan, Prabhakar, & Schütze, Hinrich (2008). *Introduction to information retrieval* (1st ed.). Cambridge University Press.

Marcus Eduardo Markiewicz, & Lucena, Carlos J. P. (2001). Object oriented framework development. <http://www.acm.org/crossroads/xrds7-4/frameworks.html>. Último acesso em Dezembro de 2011.

Andrew Kachites McCallum (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. <http://www.cs.cmu.edu/mccallum/bow>.

McIlroy, M. D. (1968). Mass-produced software components. In: *Proceedings of NATO conference on software engineering*, Garmisch, Germany.

Miller, George A. (1995). Wordnet: A lexical database for english. *Communications of the ACM, 38*, 39–41.

Miller, Frederic P., Vandome, Agnes F., & McBrewster, John (2009). *Levenshtein distance: Information theory, computer science, string (computer science), string metric, damerau? Levenshtein distance, spell checker, hamming distance*. Alpha Press.

Mitchell, T. (1997). *Machine learning (Mcgraw-Hill international edit)* (1st ed.). McGraw-Hill Education (ISE Editions).

Tim O'Reilly. What is web 2.0. design patterns and business models for the next generation of software. <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.h>.

Pinto, David, Branstein, Michael, Coleman, Ryan, Bruce Croft, W., King, Matthew, Li, Wei, et al. (2002). Quasm: a system for question answering using semi-structured data. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries, JCDL '02* (pp. 46–55). New York, NY, USA: ACM.

Porter, M. F. (1997). *An algorithm for suffix stripping*, 313–316.

Shiyi Qiao, Xiaohong Chu, & Shuwei Wang (2009). Study on application strategies of blog in information-based teaching. International joint conference on artificial intelligence (pp. 526–528).

Rosenbloom, Andrew (2004). The blogosphere – introduction. *Communications of the ACM, 47*(12), 30–33.

Salton, Gerard, & Buckley, Christopher (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management, 24*, 513–523.

Jacques Philippe Sauv (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 2011.

Sebastiani, Fabrizio, & Delle Ricerche, Consiglio N. (2002). Machine learning in automated text categorization. *ACM Computing Surveys, 34*, 1–47.

Vladislav Shkapenyuk, & Torsten Suel (2002). Design and implementation of a high-performance distributed web crawler (pp. 357–368).

Sommerville, Ian (2004). *Software engineering* (7th ed.). Addison Wesley. International Computer Science Series.

Subramanya, Shankara B., & Liu, Huan (2008). Socialtagger – collaborative tagging for blogs in the long tail. In *Proceeding of the 2008 ACM workshop on search in social media, SSM '08* (pp. 19–26). New York, NY, USA: ACM.

Szyperski, Clemens (2002). *Component software: Beyond object-oriented programming* (2nd ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Tan, Songbo (2006). An effective refinement strategy for knn text classifier. *Expert Systems with Applications, 30*, 290–298.

Technorati (2009). State of the blogosphere 2009. <http://technorati.com/blogging/feature/state-of-the-blogosphere-2009/>. Último acesso em Dezembro de 2011.

Weninger, Tim, & Hsu, William H. (2008). Text extraction from the web via text-to-tag ratio. In *Proceedings of the 2008 19th international conference on database and expert systems application* (pp. 23–28). Washington, DC, USA: IEEE Computer Society.

Xiong, Huixiang, & Wang, Xuedong (2008). The information filtering under the web 2.0 environment. *Proceedings of the 2008 international conference on information management, innovation management and industrial engineering* (Vol. 01, pp. 141–146). Washington, DC, USA: IEEE Computer Society.

Takashi Yamada, Atsushi Yoshikawa, Takao Terano, Geun Chol Moon, Go Kikuta (2010). Blog information considered useful for book sales prediction. In: *7th International conference on service systems and service management (ICSSSM)* (Vol. 1) (pp. 1–5).

Yang, Xiaohui (2008). Improving teachers' knowledge management with blog platform. *Proceedings of the 2008 international workshop on education technology and training & 2008 international workshop on geoscience and remote sensing* (Vol. 01, pp. 73–76). Washington, DC, USA: IEEE Computer Society.