Information Filtering and Information Retrieval with the *Web Filtering Toolbar* [1]

Josep Silva[2]

Computer Science Department
Technical University of Valencia
Camino de Vera s/n, E-46022 Valencia, Spain

Abstract

Nowadays, Internet is the main source of information for millions of people and enterprises. However, the information in Internet has not been classified yet and, consequently, the search for information is one of the most important tasks and processes performed by users and systems. In particular, for *WWW* human users the search for information is the main (time-consuming) task performed. In order to face this problem both the industrial and the academic communities have developed many methods and tools to index and search Web pages. The most extended solution is the use of search engines such as *Google* and *Yahoo*; however, while current search engines can be a suitable solution to find a particular Web page, they are useless to find the relevant information in such a page. Hence, once a Web page is found, the user must search on it in order to verify if the information needed is in there. This is a problem which until now has not been satisfactorily solved. In this paper we present a tool able to automatically extract from a Web page the information (text, images, etc.) related to a filtering criterion without the use of semantic specifications or indexes and without the need of offline parsing or compilation processes. This tool has been published as an extension for the Firefox's Web navigator.

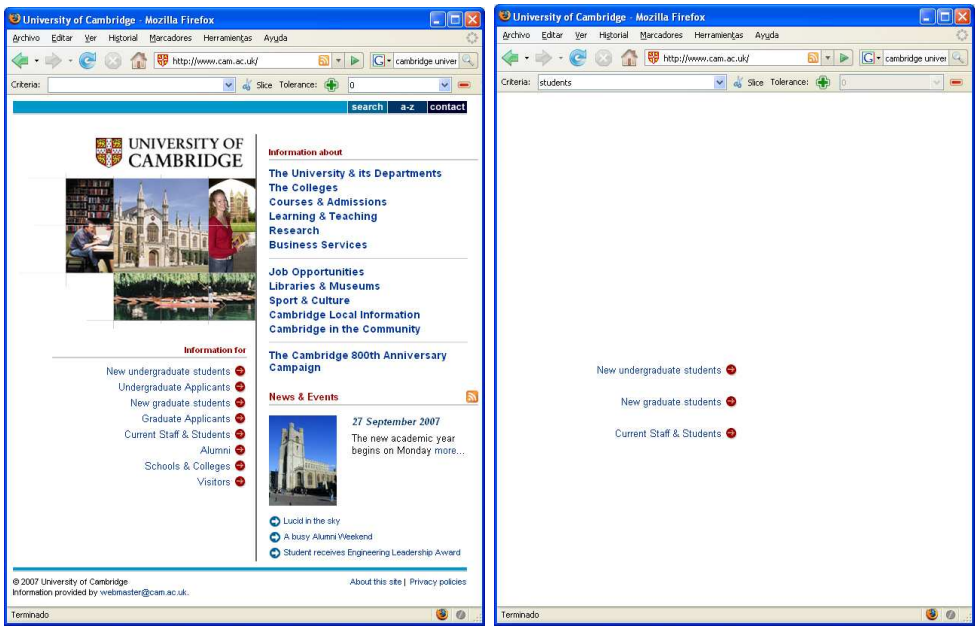
Keywords: Filtering, Web pages, HTML.

1 Introduction

Internet is the main information provider in the world. The estimates vary from 15 to some 30 billion Web pages shared with heterogeneous information about practically all matters. However, the organization of this information has not been centralized, and each independent node is responsible of maintaining its own information. This makes the classification of the information contained in the Web pages a very difficult task, and thus, the search for information is one of the most important problems in Internet nowadays. One of the most extended solutions in order to find information in Internet is the use of search engines. For instance, the two most used search engines are Google and Yahoo with more than one hundred and fifty million searches every day (42.7% and 28.17% of the market) [11]. A search

[1] This work has been partially supported by the EU (FEDER) and the Spanish MEC/MICINN under grants TIN2005-09207-C03-02, TIN2008-06622-C03-02, and *Acción Integrada* HA2006-0008.

[2] Email: jsilva@dsic.upv.es



(a) University of Cambridge's Web site (b) Filtered version

Fig. 1. Filtering the University of Cambridge's Web site

engine is a system with two main processes: indexing and searching. The interface between both processes is an index. Indexes are built by means of a collection of robots which visit Web pages, process their contents, and index them by words. For instance, given a word, an index returns the set of pages which contain this word ordered by an impact rating. Nevertheless, these indexes treat pages as atoms, thus, they cannot specify which part of the Web page contains the required information.

In consequence, current search engines are useful to find Web pages, but they are useless to find the needed information in a Web page. The information contained in a Web page can be huge and heterogenous; therefore the search for information in a Web page can still be a time-consuming task. When the user is human, she is usually forced to read and scroll a part of the Web page before she knows if this page contains what is being searched.

Surprisingly, the scientific community has put little attention on this problem, and currently there are no tools available to suitably filter the information of a single Web page. In this work, we present a method to extract relevant information from a Web page based on a process of filtering. In our approach the user specifies a filtering criterion, i.e., a text which says what is being searched and then, a new Web page is automatically produced which only contains the information related to the filtering criterion. From the best of our knowledge, this is the first tool able to produce a slice from any Web page. In the following sections we will describe how this approach works and how it has been implemented and distributed.

The rest of the paper has been organized as follows: Firstly, next section shows the usefulness of a Web filtering tool by means of an example. In Section 3 the filtering process is described. In Section 4 the *Web Filtering Toolbar* which is the implementation of our approach is presented. We describe related work in Section 5. Finally, in Section 6 we present the conclusions and we define some future lines of

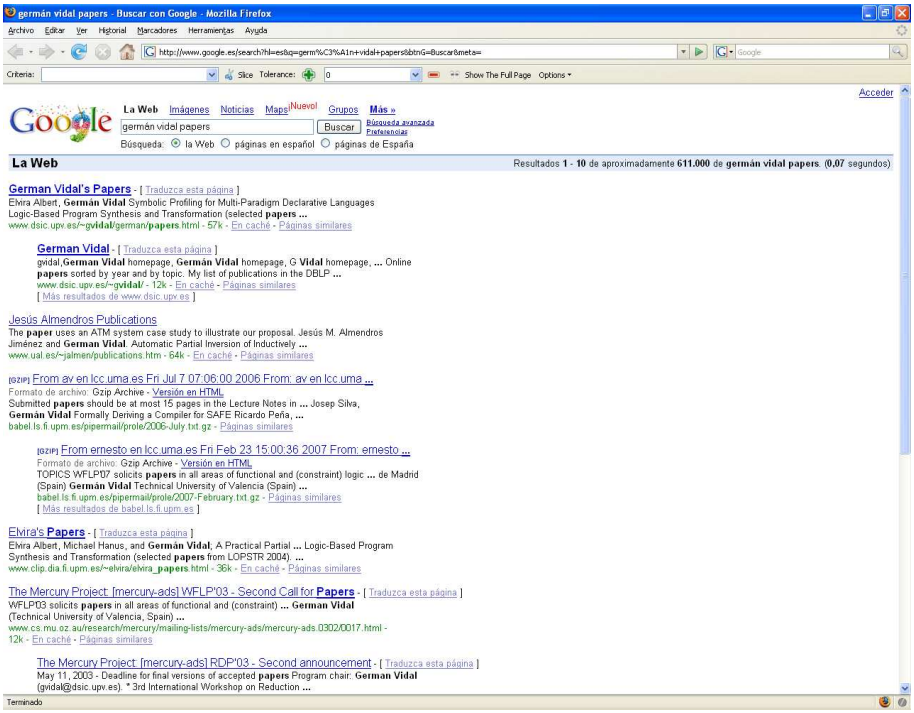


Fig. 2. Searching in Google for Germán Vidal's papers

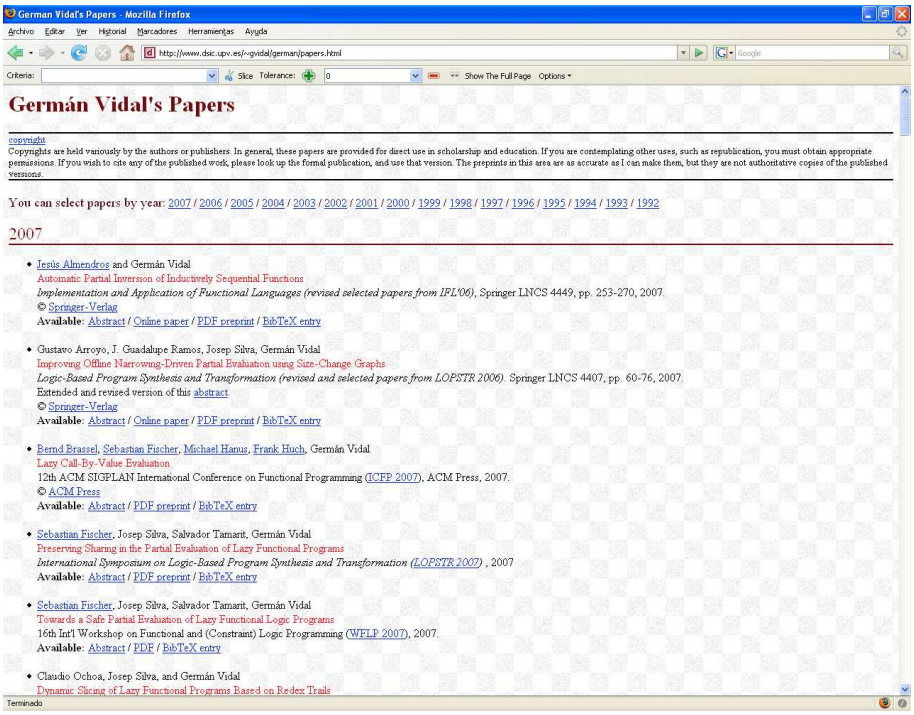
research.

2 Motivation: Filtering Web Pages

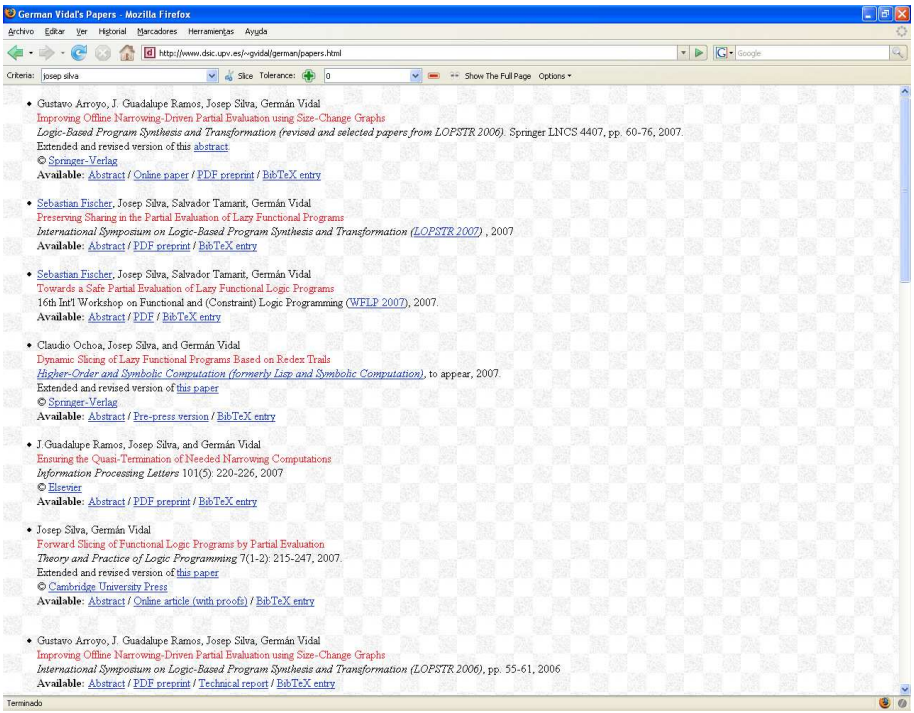
The main objective of our technique is Web filtering. The usefulness of this technique can be easily shown with an example: The most simple (and common) functionality of our technique is related to information filtering (removing all the information which is of no interest). A small example of that is the Web site produced by filtering the University of Cambridge's Web site with respect to the filtering criterion "*students*". This is shown in Figure 1 where all the information (including pictures, links, etc.) which is not related to students has been removed.

However, our technique also allows to perform more complex processes allowing information retrieval processes (extracting the relevant information and reconfiguring it in a convenient and readable fashion). For instance, let us assume [3] that we are looking in Internet for a list of papers published in common by the researchers Germán Vidal and Josep Silva. A first solution could be to find Germán Vidal's Web page and look for his list of publications. To do so we can use the Google search engine and specify a search criterion; for instance, we can type "Germán Vidal papers". The output of Google is shown in Figure 2. Unfortunately, the result produced by Google—and by the rest of current Web search engines—is only a list of Web pages. Therefore, once a Web page is selected, we have to manually review the information contained in this Web page in order to find the information searched.

[3] This assumption is a real example performed on November 28th, 2007.



(a) German Vidal's papers Web page



(b) German Vidal's filtered Web page

Fig. 3. Filtering Germán Vidal and Josep Silva's papers in common

As it can be seen in Figure 3 (a), this process is time-consuming because there are more than 90 publications in this Web page, and only 26 are in common with Josep Silva. Therefore, the user must review all the publications to know which of them are Josep Silva's publications.

Now, let us assume that we have available a tool which allows us to filter a Web page according to a filtering criterion. Since we are looking for Josep Silva's publications, we can type the filtering criterion "Josep Silva". Then, we press the filtering button and a new page is automatically generated from the previous one by extracting all the information related to Josep Silva and putting it together. The result is the list of publications in common by Germán Vidal and Josep Silva (See Figure 3 (b)). In this case, we have found the desired information with a single click. Note that, in this case, the result produced is not limited to information removal, but a reconfiguration of the relevant information has been done (look at the scroll of the Web pages).

This simple example describes the normal behavior of our tool. Although this is the main functionality, it can be parameterized to alter its standard behavior so that different kinds of filters can be used. The next section describes how the filtering process works; we will explain some additional options in the implementation section.

3 The Filtering Technique

The main advantage of our approach to Web pages filtering is its simplicity. The basic idea is to see a Web page as a set of components and delete (or hide) all those components which are not relevant with respect to a filtering criterion.

A Web page can be seen as a tree of labeled nodes. This tree is internally represented in the *Document Object Model* (DOM) with a data structure called '*document*' whose root node is labeled with "$HTML$" and that usually has a child labeled with "*body*". We use this model to see Web pages as a hierarchically organized collection of nodes. With the API of DOM we can explore the nodes of a Web page and query their properties.

Our filtering algorithm first constructs a tree-like data structure which contains all the information of the Web page hierarchically organized in document order. This data structure is really an object with provides methods to query the tree. Hence, after this phase, we have a Web page represented as a data structure, and a mechanism to explore such a data structure.

Now, we need to define the notion of filtering criterion and determine what should be the result produced for it. In our setting, a filtering criterion is a pair $(words,\ flags)$ where $words$ is a text and $flags$ is a vector with flags for the filtering tool. In general, the user only needs to specify $words$, because $flags$ contains options that can be left unchanged with their default values [4]. $words$ specifies the information that the user is looking for; therefore, the filtering technique has to produce a new Web page which only contains those nodes containing information related to this text. This can be easily checked by comparing $words$ with the

[4] See Section 4 for a complete description of the current $flags$.

information contained in the attributes of each node.

In the current implementation, we consider that a node in a Web page is *relevant* with respect to a filtering criterion ($words$, $flags$) if and only if any of its attributes contain the text $words$.

Since the nodes in the tree have a set of different attributes depending on the considered node, we have to check different properties in different kinds of nodes. For instance, an image in a Web page is represented with a node with the property $tagName == \text{“}IMG\text{”}$. Images have a special attribute called '*alt*' which contains an alternative text to display if the image can not be loaded. Thus, we compare this property when the node is an image.

Once the relevant nodes have been found, we proceed by extracting those nodes which are related to them. Therefore, we do not restrict ourselves to exact text comparison. Instead, we use a proximity function which relays on the following assumption: "*semantically related objects are syntactically close*". After our experiments and intensive use of the *Web Filtering Toolbar*, this assumption has proven to be very precise in practice because, often, Web pages structure the information with tables, menues, and other similar components which join together the related information. Moreover, a slicing criterion can match different parts of a document, and thus one slicing criterion usually points to a set of nodes of the tree structure. Then, for each relevant node we extract the associated information by using the proximity function.

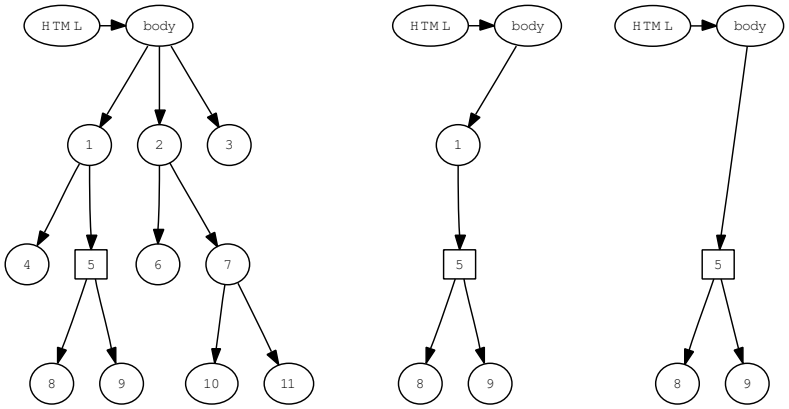
Despite our tool is prepared to work with a lexicon to find related information to a given query (e.g., synonyms), and it could also use metainformation associated to Web page's nodes; the current implementation does not exploit these features. The current proximity function is a configurable (see Section 4 for details) distance measure which allows to get nodes related to the relevant ones.

Example 3.1 Consider the Web page at the left of Figure 4 whose relevant node is 5. There are different possibilities to produce a new (filtered) Web page with respect to node 5. For instance, we could delete all the nodes except 5; or all the nodes except 5 and its descendants (see Figure 4 (c)); or except 5, its descendants and its ancestors (see Figure 4 (b)) depending on how much information related to 5 we want to filter.

However, not all the solutions of the previous example produce the expected result: Some of them would destroy the structure of the Web page because the filtered nodes would appear at the top of the new Web page instead of being at their original position. In some cases, this is the expected result, for instance, this is the case of Figure 3 (c). However, in general, the user does not want to destroy the Web page structure, and thus non-relevant nodes cannot be destroyed. For instance, in Figure 1 (b) the structure of the original Web page is kept. Fortunately, there is an easy solution: we can hide the nodes not related to 5 instead of deleting them. All these possibilities can be configured by means of the $flags$ component of the filtering criterion described in the next section.

To summarize, we filter a given Web page with respect to a filtering criterion by

(i) Building a data structure containing all the elements in the Web page,



(a) Web page (b) *Keep tree* on (c) *Keep tree* off

Fig. 4. Tree representation of a Web page

(ii) Traversing the data structure to find the relevant nodes with respect to the filtering criterion, and

(iii) Deleting (or hiding) all the nodes from the Web page except the relevant nodes according to the filtering criterion's flags.

4 Implementation

After a period of evaluation and testing in the sandbox of the Firefox's developers area, the Firefox's experts community has made public the tool and now it is distributed as an official Firefox's addon. Therefore, there is an available version which can be freely downloaded from the Firefox's Web page. In this section we show how to download, install and use the Firefox's *Web Filtering Toolbar* (see Figure 5).

Fig. 5. Web Filtering Toolbar

4.1 Installing Web Filtering Toolbar

The *Web Filtering Toolbar* is distributed as an *xpi* [5] file. An *xpi* package is basically a ZIP file that, when opened by the browser utility, installs a browser extension. Currently, this extension applies to both Mozilla and Firefox browsers. Since an *xpi* package contains all the necessary information to install an extension, the user only has to drag the *xpi* and drop it over the browser window. The extension will be automatically installed.

The user interface of the *Web Filtering Toolbar*—as the Firefox's interface itself—has been implemented with XUL and Javascript. XUL is an XML implementation which provides the interface components (such as buttons, menus, etc.); and Javascript is used to control and execute the user actions. We have opened the source code; therefore, all the implementation (both source code and executable) of the *Web Filtering Toolbar* is publicly available. To download the tool and some other materials visit:

http://www.dsic.upv.es/ ~ jsilva/webfiltering.

and

http://www.firefox.com/addons.

4.2 Using Web Filtering Toolbar

In the normal case, using the filter is as easy as to type a text t inside a text box and press the button "*Filter*". Then, the tool produces a slice of the current Web page with respect to the filtering criterion (t, f) where f are flags which represent the default options of the tool. However, these options can be easily changed by the user when needed.

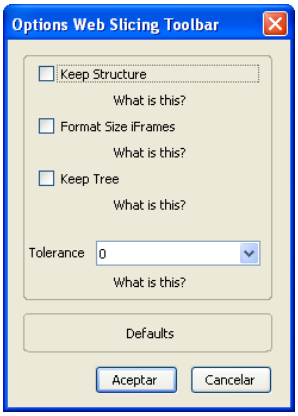


Fig. 6. *Web Filtering Toolbar*'s options

In its current version, our tool can be parameterized with four flags which determine the shape of the final slice (see Figure 6):

Keep Structure. When activated, "keep structure" ensures that the components of the slice keep the same position as in the original Web page; i.e., the Web page' structure is kept and, thus, the final slice will contain blank areas (see Figure 1). If the structure is not kept, all the data in the final slice is reorganized so that no empty spaces exist (see Figure 3).

Format Size iFrames. Iframes allow us to embed a Web page inside another Web page. If "format size iframes" is activated, the size of the iframes of the original Web page is adapted to the slice. Otherwise, the original size is kept. Usually, the Web page of an iframe is bigger than the area reserved for the iframe; hence, the iframe uses scrolls. Often, the slice extracted from an iframe is small, and thus, reformatting the size of the iframes avoids unnecessary empty areas produced by the scroll.

Keep Tree. When a node in the tree representing a document belongs to the slice, it is possible to include also all the nodes belonging to the path between this node and the root (the ancestors). To do this, *keep tree* must be activated. For instance, in the slice of Figure 4 (b) *keep tree* was activated, while in the slice of Figure 4 (c) it was not.

Tolerance. The default tolerance of the slicer is 0. With a tolerance of 0, only the

relevant nodes and their descendants are included in the slice. If the tolerance is incremented, then those nodes which are close to the relevant nodes are set as relevant. Concretely, if the tolerance is incremented by one, the parents (and their descendants) of relevant nodes are set as relevant. For instance, in Figure 4 (a), with a tolerance of 0 and *keep tree* deactivated, the slice produced is shown in Figure 4 (c). With a tolerance of 1, nodes 1—the parent of 5—and 4—the descendant of 1—would be included in the slice. With a tolerance of 2, all the nodes would be included in the slice.

5 Related Work

The most extended tool used to search information in Internet is Google [4]. However, Google is a search engine to find Web pages, and thus it is useless for finding information inside Web pages. Google should be complemented with other tools in order to find the desired information inside the reported documents. It is possible to create a search engine and parameterize it according to our needs. For instance, one of the main novel approaches for searching information in Web pages is Nutch [6]. Nutch is built on top of Lucene [8], which is an API for text indexing and searching. Nutch divides naturally into two pieces: the crawler and the searcher. The crawler fetches pages and turns them into an inverted index, which the searcher uses to answer users' search queries. The interface between the two pieces is the index. This approach allows the user to find pages related to a specified criterion; however the level of granularity is the Web page, and thus, it does not allow to find information in a single Web page. Moreover, Nutch needs an index to work, hence, it can not be used with any Internet page as our tool does.

There is however, a kind of tools which can be applicable to any Web page: the Web content filtering tools [2,7]. These tools are responsible to analyze the content of a Web page and decide if it contains a particular content in order to forbid the access to this Web page. The main application of such tools is preventing children from accessing pornographic or racist Web sites, e.g., from school computers. This kind of tools are in some sense related to our tool because they are applicable to any Web page and they must analyze their content. Notable examples are WebGuard [7] and the system developed by Marinilli et al. [10]. The main difference is that while these tools only say if a Web page contains a kind of information, our approach says where in the page is such information.

There are other tools that really filter single Web pages, For instance WebWatcher [9] and WebFilter [12]. Both of them rely on the use of an external proxy server that acts as an intermediate layer between the user and the visited Web site. The proxy filters the pages before they get to the user. Unfortunately, they are quite limited. For instance, in WebFilter, the user must provide (in advance!) a script for each URL she wants to filter. This can be useful for a frequently visited Web page with advertisements, but it cannot be used with unknown Web pages as our tool does. Another filtering advanced system based on scripts is Digestor [3]. On the other hand, WebWatcher is an assistant which helps the user to search through different pages by selecting which links are more probable to reach the desired information. The selection is based on a probabilistic model which uses a

user profile and previous (training) searches.

Despite the big effort put on search engines for the Web, there are very few works devoted to find information inside a single Web page. Most of them, e.g., FilterGus, cannot work with online pages; or need the use of semantic specifications or cannot directly work with HTML code. This is the case for instance of the Phil system [1], a tool to filter XML documents. Phil allows one to extract relevant data as well as to exclude useless and misleading contents from an XML document by matching patterns against it. Unfortunately, to filter a document, it is needed a semantic specification—which usually implies knowing the document structure—; and, moreover, Phil cannot handle HTML pages directly. Another similar approach is [14] which is able to extract a slice of an XML document w.r.t a slicing criterion. This work also argues that XML slices can be used to produce Web page slices in the case that the Web page has been generated from an XML document via XSLT. Unfortunately, the number of pages in Internet generated via XSLT is very small, and moreover, in this approach the filter is done at the level of the XML document which usually is not public. Therefore, neither this tool can be used to filter Internet Web pages in general. Finally, there is another work [13] in which a part of a Web application can be extracted by using the well-known program slicing technique [15]. Therefore, this technique uses the control and data dependences of a Web application (which often implies several pages) to extract a slice; hence, it is useless to extract a slice from a single static HTML Web page.

From the best of our knowledge, the *Web Filtering Toolbar* is the first tool that can filter any Web page, it works online, and it is autonomous (it does not need profiles, scripts, previously generated indexes or semantic specifications about the Web page that is going to be filtered).

6 Conclusions and Future Work

Nowadays, Internet contains billions of Web pages which are constantly being browsed, queried and modified by internet users. Often, the amount of information retrieved by a human user while searching cannot be absorbed in a pleasant and/or understandable fashion. To solve this situation, the scientific community has invested a lot of research effort producing several indexing and search systems which reduce the inherent complexity of such a massive amount of data. However, most of the current search engines consider Web pages as the basic unit of information which should be provided to the user. From our point of view, this is frequently an error which makes the user to load, read and scroll a lot of useless information. In this work, we have introduced a new approach to Web filtering which allows the user to automatically filter from a Web page irrelevant information by specifying a filtering criterion. This approach has been implemented and integrated into the Firefox browser producing a very practical tool that has demonstrated its usefulness after an intensive phase of testing by real users.

There are some directions for future work that are still open. Regarding the implementation, the current release cannot handle frames because it implies that a Web page can display several HTML documents at the same time. We are currently developing a new version of the *Web Filtering Toolbar* which includes a special

treatment for Web pages containing frames. Another implementation extension we plan to overcome is the use of a lexicon during the filtering phase. This will allow to filter Web pages by using alternative filtering criteria semantically related to the one specified by the user. For instance, if the user is searching for information related to "cars", the filter could also include in the slice all the information related to "autos" and "automobiles". Finally, we plan to extend our tool in order to be able to filter XML documents. Some preliminary research in this direction has shown that our technique is directly applicable to XML documents. We would like to take advantage of the fact that with XML documents we have available a DTD which provides helpful information about the structure of the document.

7 Acknowledgements

The author wish to thank Mercedes García for her help in the implementation of the first version of the *Web Filtering Toolbar*.

References

[1] M. Baggi and D. Ballis. A lazy implementation of a language for approximate filtering of xml documents. Technical report, University of Udine, 2007.

[2] E. Bertino, E. Ferrari, and A. Perego. Content-based filtering of web documents: the maX system and the EUFORBIA project. *International Journal of Information Security*, 2(1):45–58, 2003.

[3] T.W. Bickmore, A. Girgensohn, and J.W. Sullivan. Web page filtering and re-authoring for mobile users. *The Computer Journal*, 42(6):534–546, 1999.

[4] N. Blachman. Google guide: Making searching even easier. Accessed on October 10th 2007., URL: http://www.googleguide.com.

[5] Mozilla Developer Center. Cross-platform installer module (xpi). Accessed on October 10th 2007., URL: http://developer.mozilla.org/en/docs/XPI.

[6] The Apache Software Foundation. Nutch version 0.7 tutorial. Accessed on October 10th 2007., URL: http://lucene.apache.org/nutch/tutorial.pdf.

[7] M. Hammami, Y. Chahir, and L. Chen. Webguard: A web filtering engine combining textual, structural, and visual content-based analysis. *IEEE Trans. Knowl. Data Eng*, 18(2):272–284, 2006.

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

[9] T. Joachims, D. Freitag, and T. M. Mitchell. WebWatcher: a tour guide for the word wide web. In Martha E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 770–775, Nagoya, JP, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[10] M. Marinilli, A. Micarelli, and F. Sciarrone. A hybrid case-based architecture for information filtering on the web. In Sascha Schmitt and Ivo Vollrath, editors, *Challenges for Case-Based Reasoning - Proceedings of the ICCBR'99 Workshops, Seeon Monastery, Germany, July 27-30, 1999*, pages 23–32. University of Kaiserslautern, Computer Science, 1999.

[11] NetRatings. Nielsen netratings: Internet data & ratings. Accessed on October 10th 2007., URL: http://www.netratings.com.

[12] J. Pereira, F. Fabret, H.A. Jacobsen, F. Llirbat, and D. Shasha. Webfilter: A high-throughput XML-based publish and subscribe system. In *Proceedings of the 27th International Conference on Very Large Data Bases(VLDB '01)*, pages 723–725, Orlando, September 2001. Morgan Kaufmann.

[13] Filippo Ricca and Paolo Tonella. Web application slicing. In *Proc. of International Conference on Software Maintenance (ICSM'01)*, pages 148–157. IEEE Computer Society, 2001.

[14] Josep Silva. Slicing XML documents. *Electr. Notes Theor. Comput. Sci*, 157(2):187–192, 2006.

[15] M.D. Weiser. Program Slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.