A New Information Filtering Method for WebPages

Sergio López, Josep Silva

Universidad Politécnica de Valencia
Camino de Vera s/n E-46022 – Valencia, Spain
{slopez,jsilva}@dsic.upv.es

***Abstract*—Internet is a huge source of information. Search engines have indexed much of this information and are able to extract the relevant webpages that are related to a given query. However, once the search engine retrieves a set of webpages, the user has to read the webpages in order to find the relevant information. This task is a time consuming task because webpages often mix information related to different subjects, and also because they usually contain advertisements and publicity that tries to call to attention the reader using pictures, videos, etc. In this work we define a novel technique for information filtering of webpages. This technique uses the distance of elements in a webpage to approximate semantic relations. The technique is able to work only (with any webpage that has not been pre-processes in advance). We present our implementation and show the usefulness of the technique with examples.**

***Index Terms*—Information Filtering, HTML Filtering, Slicing Websites.**

I. Introduction

INTERNET contains millions of webpages with information of practically all subjects. The efforts of the scientific community in defining efficient techniques for information retrieval have produced good results (be referred to [1] for a survey of this kind of techniques); and current search engines are continuously indexing webpages that can be later retrieved with a query producing high quality results. However, once a collection of webpages is retrieved, the user is forced to read them in order to find the information of interest.

Surprisingly, the task of filtering a webpage in order to remove the useless information has not been automated. There exist very few tools devoted to filter the information of a webpage, and they are often very limited, or need to pre-process the webpages before filtering them. The lack of real-time (online) applications able to automatically filter a webpage gives an idea of the difficulty of the problem. This difficulty is mainly produced by the fact that webpages are usually implemented with plain (X)HTML, and this language is not prepared to qualify the semantic information.

There are some attempts to solve this situation. For instance, in the Semantic web, an ontological model of webpages is constructed and the knowledge is modelled and queried using languages such as RDF [2] or OWL [3]. Another approach tries to introduce semantic labels into webpages, so that explicit semantic relationships between elements exist. These labels are called microformats [4,5,6], and are already used by many webpages that include places, contact info, etc. Unfortunately, 99% of the webpages of Internet were created without taken into account these models.

A recent approach for information retrieval is based on an extension of search engines called answer engines. These tools construct indexes of webpages where the information is labeled with semantic information that allows to extract implicit information. When a query is specified by the user, these tools try to construct an answer with the information of the webpages. The most important tool of this kind 'Wolfram Alpha', but unfortunately, it is quite limited and answers can be only produced for very simple questions. These tools are considered a future option for information retrieval, but they are not currently being used for information filtering.

Another kind of tools that are related to our work are the parental control tools. These tools determine if a webpage contains violence or pornographic content. If so, the whole webpage is bloqued. This contrast with our tool in which the webpage is not bloqued. It is simply filtered skipping the unwanted content. Notable tools of this kind are Webguard [¿?], Anti-Porn [¿?] and Naomi [¿?].

There are a few tools devoted to filter webpages by removing unnecessary contents, but they often need to pre-process the webpages that are going to be filtered. This imposes important restrictions on the webpages that can be processed, and thus the implemented tools are usually offline tools. For instance, Webwatcher [¿?] and Webfilter [¿?] use a proxy to analyze the webpages. Preprocesado preguntar a cual es cual Hector?

Manuscript received on March 10, 2010. This work has been partially supported by the Spanish *Ministerio de Ciencia e Innovación* under grant TIN2008-06622-C03-02, by the *Generalitat Valenciana* under grant ACOMP/2009/017 and by the *Universidad Politécnica de Valencia* (Programs PAID-05-08 and PAID-06-08).

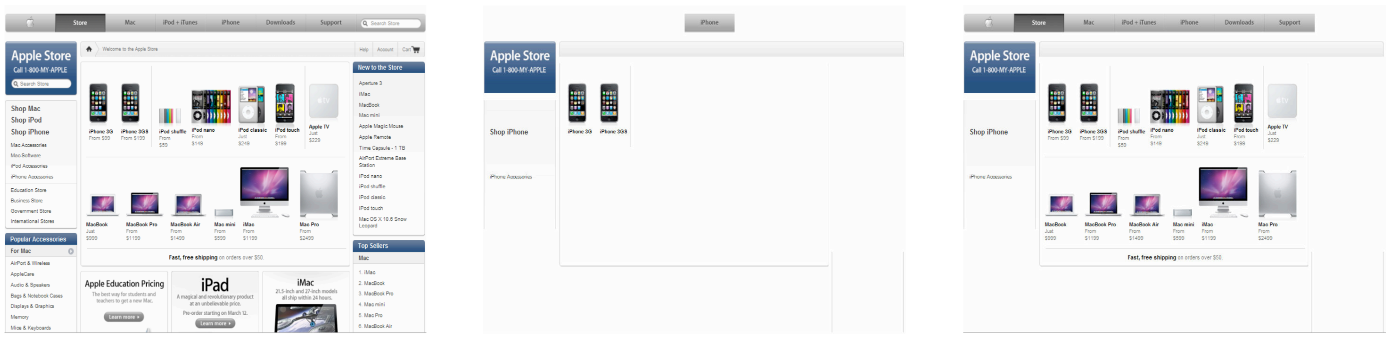


Fig. 1. A la izquierda se encuentra la página sin filtrar y a continuación se encuentra la página filtrada con tolerancia 0 y 1.

From the best of our knowledge, our tool is the first approach that can filter webpages online without any pre-compilation or pre-parsing phases.

The rest of the article has been organized as follows: In Section II we present the technique from a practical point of view by showing examples of use. Section III presents the technique and introduces an algorithm to automatically filter a webpage for a given query. In Section IV we describe our implementation and, finally, Section V concludes and gives some directions for future work.

II. Motivation

This section presents real examples of information filtering using the technique presented in this paper. In our first example, we consider a user that is browsing on the Apple Store in order to buy an iPhone. When we open the main webpage of the Apple Store (shown in Figure 1 in the left), we see that there is much information (including images and menus) not related to iPhones. Therefore, the user is forced to read unnecessary information in order to find what she is looking for.

Now, consider that we have a tool available able to filter all the information not related to iPhones. Our algorithm is able to filter a webpage and only show the relevant information according to the given filtering criterion. For instance, the algorithm would produce the new filtered webpage shown in Figure 1 (center). Observe that even images and the main horizontal menu have been filtered.

If the user considers that this information is not enough, or has been filtered too much, she can augment the information shown. In this case, the webpage is automatically reprocessed to include more information. The result is shown in Figure 1 (right).

Now let us consider another scenario where a user has loaded the Facebook's page of its creator Mark Zuckerberg. Observe that Facebook uses the left area of the pages for a publicity area with annoying auto-changing advertisements. In this case, we can use the filtering tool in the opposite way than before. We can delete all the information related to a given word. For instance, we can specify that we want to delete all the "ads". Observe that advertisements have been filtered out. This tool does not need to use proxies [7] or to pre-process [8] the filtered webpages. It can work online with any webpage.

III. Filtering Information from Webpages

In this section we formalize our technique for information filtering and visualization.

A. *Slicing the Relevant Information*

The Document Object Model (DOM) [9] is an API that provides programmers with a standard set of objects for the representation of HTML and XML documents. Our technique is based on the use of DOM as the model for representing webpages. For the sake of concreteness, in the following we will assume that a DOM tree is a data structure that represents each single element of a webpage with a node labelled with a text. This simplification assumes that all nodes have a single attribute, and it allows us to avoid in our formalization and algorithms low-level details such as the distinction between different kinds of HTML elements' attributes. For instance, in our implementation we have to distinguish and query different properties depending on the element that we are analyzing, e.g., image nodes are queried by using their *alt*, *longdesc* and *src* attributes.

Definition 1 (DOM tree): A *DOM tree* $t=(V,E)$ is a tree whose vertices V are labelled nodes connected by a set of edges E.

In the following, we will refer to the label of a DOM node n with $l(n)$; and we will refer to the root of a DOM tree t with $root(t)$. We also use the notation $n -^{x} n'$ to denote that there exists a path of size less or equal to x between nodes n and n'.

Definition 2 (Webpage): A *webpage* is a pair (u,t) where u is an URL and t is a DOM tree.

We allow the user to specify complex queries that contain multiple words and metadata such as *""* for exact search, and boolean operators (*and*, *or*, *not*) to produce combinations of texts that force a particular order of words, or force the existence or inexistence of a particular (sub)text. The syntax of the queries allowed in our implementation is given by the following grammar:

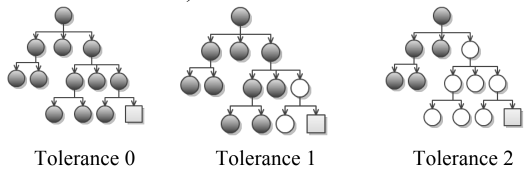
```
Query = Word
Query = Word Operator Query
Query = EmptyWord "Word" Query
```

```
Operator = and | or
Word = character EmptyWord
EmptyWord = character EmptyWord
EmptyWord = ϵ
```

Where terminals are in bold and **character** refers to any alphanumerical character. However, for simplicity, in our formalization we will assume that queries are composed of a single word. The extension of the technique for multiple words is trivial and it only requires the iteration of the method over all the words of the query. This has been already done in our implementation, and thus, the interested reader is referred to its (open) source code for implementation details.

Together with the text, the user must specify other parameters that configure the query. These parameters are the following:

- *Keep Structure:* It is a boolean value used to specify if the structure of the webpage should be maintained. For instance, if a DOM element that contains the specified text is inside a table which is inside a layer, keep structure decides whether these container elements should be filtered out or not.
  For instance, consider the Figure 2. We can assume that the only nodes that contain the word specified by the user are nodes 6 and 12 (we refer to these nodes as key nodes, and we draw them in the following with a square). If keep structure is activated, then the light nodes are selected by the algorithm and the dark nodes are filtered out.
  Therefore, keep structure decides whether the paths between the root node and the key nodes should be filtered out or not.
- *Tolerance:* It is an integer number that is related to the amount of information that the user wants to see in the filtered webpage. The tolerance is used to decide what elements of the DOM tree are related to the user specified word. With a tolerance of 0, only elements that contain the specified word should be retrieved. With a tolerance of 1, only elements that contain the word and those that are in a distance of 1 to them should be retrieved, and so on.



- *Inverse Filtering:* It is used to decide whether we filter out the elements not related to the specified text (normal filtering), or we filter out the elements related to the specified text (inverse filtering).

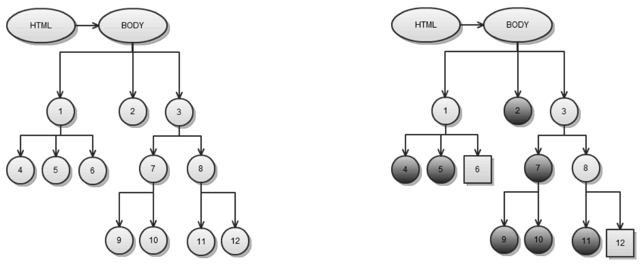


Fig. 2. A la izquierda se encuentra la estructura de una página y a su derecha el filtrado con las opciones Keep Structure and Tree activa.

Definition 3 (Query): A *query* is a tuple *(w,t,k,i)* where *w* is a word that is associated to the information which is relevant for the user; *t* is an integer that represents the tolerance required in the search, *k* is a boolean value that specifies whether the structure should be maintained, and *i* is a boolean value that specifies whether the filtering is inverse.

We are now in a position to present our algorithm for information filtering:

Algorithm 1: Information retrieval from single webpages

```
Input: A webpage $P=(u,t)$ and a query $q=(w,t,k,i)$
Output: A webpage $P'=(u,t')$

Initialization: $t=(v,e)$, $t'=(\emptyset,\emptyset)$

(1)  key_nodes = $\{n \in v \mid l(n)=w\}$
(2)  relevant_nodes = $\{n \in v \mid n \overset{t}{-} n' \wedge n' \in$ key_nodes$\}$
(3)  if ($k=True$)
(4)      ancestors = $\{n \in v \mid n_0 \rightarrow^* n \rightarrow^* n_1 \wedge n_0=root(t)$
                      $\wedge n_1 \in$ key_nodes $\cup$ relevant_nodes$\}$
(5)  else
(6)      ancestors = $\emptyset$
(7)  successors = $\{n \in v \mid n_0 \rightarrow^* n \wedge n_0 \in$ relevant_nodes $\}$
(8)  if ($i=True$)
(9)      final_nodes = $\{n \in v \mid n \notin$ (successors $\cup$ ancestors)$\}$
(10) else
(11)     final_nodes = successors $\cup$ ancestors
(12) edges = $\{(n,n') \in e \mid n,n' \in$ final_nodes $\}$

return $P'=(u,$(final_nodes, edges))
```

Roughly, Algorithm 1 proceeds by (i) finding the *key_nodes* that are those whose label is equal[1] to the text specified by the user (*w*). (ii) From these nodes, the *relevant_nodes* are computed, which are those whose syntax distance to the relevant nodes is lower than the tolerance specified by the user (*t*). The idea of using the tolerance (that is a measure of syntax distance) as a measure of semantic relation is in important contribution of this technique. Compared to other techniques that use semantic labels, ontologies, semantic hierarchies or

[1] The restriction of being equal is taken for simplicity of presentation. In the implementation, a lexicon could be activated to also consider synonyms. Of course, standard semantic distances for word similarity can also be used.

indexes, this idea seams to be too simple. However, several experiments using our implementation with real examples together with massive use of anonymous users demonstrate the practical utility of this syntax distance. (iii) if keep structure is activated, then the *ancestors* of the relevant nodes are collected. (iv) Next, the *successors* of the relevant nodes are also collected. Finally, (v) if inverse filtering is not activated, then the *final_nodes* of the filtered DOM tree are the successors and the ancestors. Otherwise, the final nodes are the rest of nodes. The edges of the filtered DOM tree are the edges that connect the final nodes. Therefore, Algorithm 1 has a cost linear with the size of the DOM tree.

Example 1: Assuming that the only nodes that contain the word *w* are 6 and 12, the light nodes of Figure 2 correspond to the slice produced by the query *(w,0,True,False)*.

Observe that the final filtered webpage (that we refer to as *slice*) is always a subset of the original webpage, and this subset keeps the original structure of information because the paths between retrieved elements are maintained.

It should be clear that Algorithm 1 is able to work with complex webpages composed of other webpages by means of frames or iframes. The reason is that the DOM trees of frames and iframes are a subtree of the webpage that contains them. The DOM API offers methods to navigate into the DOM tree of a frame so that they can be treated separately or as a part of their parent webpage.

Another interesting property of the technique is that filtering can be done incrementally. Since the result of the filtering process is a webpage whose DOM tree is valid, it is possible to slice the own slice produced by the technique. This means that it is possible to combine positive and negative filtering. For instance it is possible to filter a webpage and show all the information related to the tennis player Federer, but excluding the information related to the tennis player Nadal. This can be achieved by first filtering the webpage with the word 'Federer', and then filtering the result with the word 'Nadal' and activating the inverse filtering option. (BORRAR EL PARRAFO??)

B. Visualizing the Slice.

The slice computed by Algorithm 1 can be visualized in different ways, and depending on the results expected by the user only some of them are appropriate.

Keep structure plays an important role in the way in which elements are visualized. If keep structure is activated, the path of nodes from the root of the slice to the key nodes belong to the slice. This path often contains DOM nodes of container objects such as tables or layers. These nodes have a size and thus they fill an area of the screen. Therefore, if we delete these nodes, then the position of the other nodes changes in the screen.

From an implementation point of view, the path cannot be always deleted, because some nodes of the DOM model need to have a particular parent to be correct. For instance, it is not correct to delete the parent of a *LI* HTML element because it represents an element of a list and needs the list to be correct (in particular, it needs a parent of type *OL* or *UL*). Similarly, an element of type row (*TR*) or column (*TD*) needs a *TABLE* element as its parent. Lo que haremos será cambiar el puntero al nodo padre mímino necesario para representarse.

In the case that the nodes are kept, it is still possible to visualize the webpage in two different ways. The first possibility is to hide the elements with the property:

Style.visibility="hidden"

This is equivalent to make the element invisible. But it still fills an area. The other possibility is to remove the element from the page thus compressing the information at the top part of the webpage. This can be done with the property:

Style.display="none"

In presence of frames (and iframes), the visualization of the slice is a bit more complex. The reason is that in a frame of size *x*, it is possible to include elements whose total size is greater than *x*. It is possible thanks to the use of scrolls. However, after slicing, many elements in the frame are filtered out, and thus the size of the slice usually fits in the space assigned for the frame. In order to ensure that these elements are visualized, we transform the frame into a layer with auto scrolls. This ensures that scrolls are only used when the information of the sliced frame does not fit in the layer.

Another important task related to visualization is the fact that the user can be working with different webpages at the same time. Therefore, each webpage must be in a different window or tag, and all of them can be filtered. In order to correctly visualize the slice of each window or tag, the tool must record the properties of all nodes of each slice. Therefore, each window or tag has an associated data structure including the visualization properties of each node in this webpage.

IV. Implementation

Our implementation of the technique is an official addon of the Firefox web browser. Therefore, it has been implemented following the standard Firefox design. The implementation languages are XUL (for the graphical use interface) and Javascript. We do massive use of the DOM API to access and manipulate properties of the nodes.

The implementation keeps for each slice (e.g., in different tags) two arrays of DOM nodes: one for visible nodes, and one for hidden nodes. For efficiency reasons, it is not needed to check all the nodes of the DOM tree when filtering. The property *innerHTML* allows us to know if the HTML of a node contains a text. If it does, the descendants are explored to find key nodes. It is does not, the node is discarded.

Es importante destacar que si el criterio es de dos o más palabras, un nodo puede necesitar de dos o más varios nodos hijos para coincidir con el criterio mientras que ninguno de sus nodos hijos es capaz de coincidir con todos los parámetros.

In the implementation, there is an option that can be activated by the user called "*keep tree*". When it is activated, the paths from the root to the key nodes are part of the slice. Otherwise, they are filtered out. Básicamente, todos los enlaces del nodo BODY con sus hijos se rompen y posteriormente colgamos todos los nodos del vector KeyNodes del nodo BODY.

We have implemented an additional functionality in the tool that allows the user to perform continuous filtering while she is surfing in Internet. This means that it is possible to activate this option and the bar acts as an agent that automatically helps the user during the search. In particular, if the user is looking for information related to Mercedes Benz, then the bar will automatically filter all the webpages that are loaded only showing the information related to Mercedes Benz. From the point of view of the user, the navigation is the same, but the webpages that she reads are already filtered when she loads them. Of course, she can always augment or reduce the tolerance to change the amount of information shown. It is also possible to press the button *Show All* to see the original page. Our current implementation has been released as version 1.5 of the Firefox's *Web Filtering Toolbar*. This tool is an official extension of the Firefox web browser that has been tested and approved by the expert evaluators of the Firefox's developers area, and that has more than 10.000 downloads at the time of writing these lines. The most stable version of the tool can be downloaded from Firefox's addons area or Author Webpage:

https://addons.mozilla.org/es-ES/firefox/addon/5823
http://users.dsic.upv.es/~jsilva/webfiltering/

V. ARQUITECTURA

El plugin utiliza la versión orientada a objetos de Javascript, lenguaje interpretado para la Web. Contiene principalmente un vector de una clase principal llamada 'webfiltering' que contienen todos los métodos y atributos necesarios para realizar un filtro sobre una página web, de esta manera cada filtrado sobre una página web es independiente del resto de filtrados. La clase consta principalmente de 3 vectores que contendrá todos los nodos divididos en grupos según el criterio, también consta de otras variables para guardar información sobre el filtrado o sobre la página en cuestión como la URL o el contenido. Los métodos de la clase estarían principalmente divididos en métodos públicos que se llaman desde la interfaz XUL del plugin y métodos privados necesarios para completar la funcionalidad o para compartir funcionalidad en común (Figura 3). Los eventos generados por la interfaz XUL son gestionados por un método central que se encarga de gestionar y llamar a su correspondiente método.

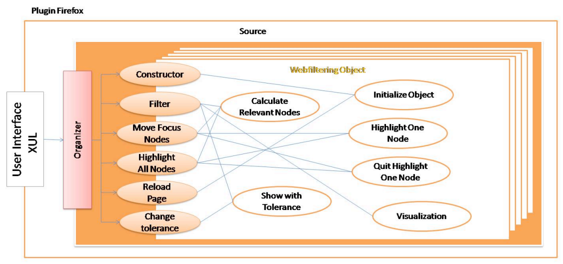


Fig. 3. Estructura del plugin.

VI. CONCLUSIONS AND FUTURE WORK

This work introduces a novel technique for information filtering that uses syntax distances to approximate semantic relations. The technique is able to work online and extract information from websites without any pre-compilation, labeling, or indexing of the webpages to be analyzed.

Currently, we give the same semantic value to all the kinds of nodes in a DOM tree. Future work will include the analysis of what nodes provide information that can be considered as more reliable. For instance, meta-tags and microformats are good candidates.

We would like our tool to be able to work in any browser. Therefore, we plan to implement a version of the bar that is not integrated into Firefox. This is possible because the implementation language is Javascript and thus standard browsers can interpret it. We have already implemented a version of the tool as a webpage with a top frame that contains the bar.

In [10] a technique for information retrieval from single webpages was proposed, but a formalization of the algorithm was not defined. In this section we introduce this algorithm and adapt it to be later used for our algorithm for information retrieval from multiple webpages. (CAMBIAR. Para reducirlo habría que poner que es possible integrar en una pa´gina y punto)

REFERENCES

[1] J.M. Gómez Hidalgo, F. Carrero García, E. Puertas Sanz. *Named Entity Recognition for Web Content Filtering* International Conference on Applications of Natural Language, NLDB2005, pages 286-297, 2005

[2] W3C Consortium, Resource Description Framework (RDF). www.w3.org/RDF

[3] W3C Consortium, Web Ontology Language (OWL). www.w3.oeg/2001/sw/wiki/OWL

[4] Microformats.org. The Official Microformats Site. http://microformats.org/, 2009.

[5] R. Khare, T. Çelik Microformats: a Pragmatic Path to the Semantic Web. Proceedings of the 15h International Conference on World Wide Web. International World Wide Web Conference. Poster Sessions pages 865-866, 2006.

[6] R. Khare. Microformats: The Next (Small) Thing on the Semantic Web? *IEEE Internet Computing*, 10(1):68–75, 2006.

[7] Suhit Gupta, Gail E. Kaiser et al. *Automating Content Extraction of HTML Documents*. World Wide Archive vol.8 issue.2, pages 179-224, 2005.

[8] Po-Ching Li, Mind-Dao Liu, Ying-Dar Lin, Yuang-Cheng Lai *Accelerating Web Content Filtering by the Early Decision Algorithm*. IEICE – Transactions on Information and Systems vol. E91-D, pages 251-257, 2008.

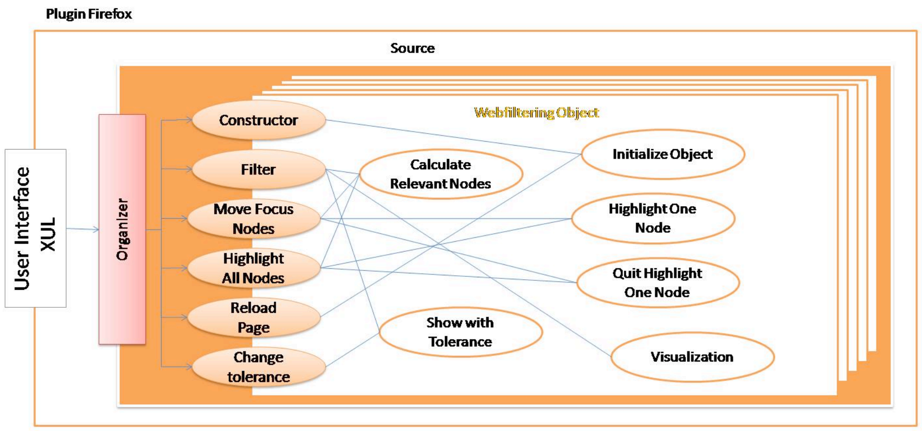
[9] W3C Consortium, Document Object Model (DOM). www.w3.org/DOM

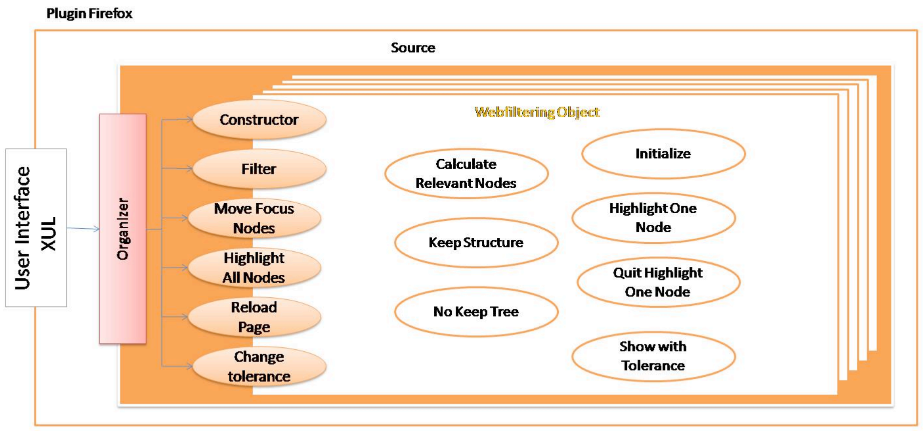
[10] J. Silva, *Information Filtering and Information Retrieval with the Web Filtering Toolbar*. Electronic Notes in Theoretical Computer Science, vol. 235, pages 125-136, 2008.

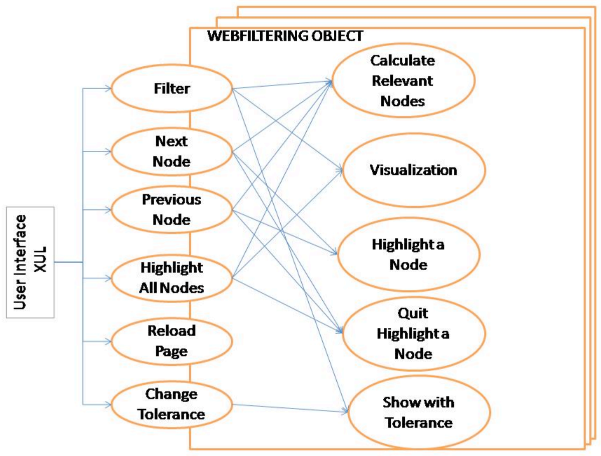
[11] R. Baeza-Yates, C. Castillo, *Crawling the Infinite Web: Five levels are enough*. WAW, Lecture Notes in Computer Science, vol.3243, pages 156-167. Ed. Springer 2004.

[12] A. Micarelli, F. Gasparetti, *Adaptative Focused Crawling*. The Adaptative Web, pages 231-262, 2007.

[13] Jakob Nielsen. *"Designing Web Usability: The Practice of Simplicity*"; New Riders Publishing, Indianapolis ISBN 1-56205-810-X; 2010.


Plugin Firefox
Source
User Interface XUL
Organizer
Constructor
Filter
Move Focus Nodes
Highlight All Nodes
Reload Page
Change tolerance
Webfiltering Object
Calculate Relevant Nodes
Show with Tolerance
Initialize Object
Highlight One Node
Quit Highlight One Node
Visualization


Plugin Firefox
Source
User Interface XUL
Organizer
Constructor
Filter
Move Focus Nodes
Highlight All Nodes
Reload Page
Change tolerance
Webfiltering Object
Calculate Relevant Nodes
Keep Structure
No Keep Tree
Initialize
Highlight One Node
Quit Highlight One Node
Show with Tolerance


WEBFILTERING OBJECT
User Interface XUL
Filter
Next Node
Previous Node
Highlight All Nodes
Reload Page
Change Tolerance
Calculate Relevant Nodes
Visualization
Highlight a Node
Quit Highlight a Node
Show with Tolerance