A Web Clipping Service's Information Extraction Mechanism

Christos Bouras Giorgos Kounenis Ioannis Misedakis Vasilis Poulopoulos
Research Academic Computer Technology Institute, Riga Feraiou 61, 26221, Patras, Greece and
Computer Engineering and Informatics Department, University of Patras, 26500, Rion, Greece
+30-2610-960375 +30-2610-996954 +30-2610-960465 +30-2610-996954
bouras@cti.gr kounenis@ceid.upatras.gr misedaki@cti.gr poulop@ceid.upatras.gr

Abstract

Information overload is one of the most important problems of today's WWW. Users are often lost in a wealth of information when searching about a topic. Although they have specific information needs, using a search engine or regularly browsing in popular news sites for updates can lead them to search in tens or hundreds of possibly relevant pages or loose some updates that were interesting for them. One promising solution for this problem is 'web-clipping' services. A service like this continuously searches the Web and whenever it finds a web page that might interest some users, it informs these users that they should visit this specific page. This paper describes the information extraction mechanism of a web-clipping service that is being designed as part of a larger project for information search and manipulation. This mechanism is used to extract information from web pages and find out which links should be followed.

1. Introduction

Web Clipping services target users who have specific information needs, but lack the time or knowledge to search and browse the Web. Their concept is simple: Subscribers of the service, after registering, can create one or more keyword lists, that characterize articles or other pieces of text they would find useful. Whenever a new article or piece of text is posted in a website that contains these words (or products of these words, synonyms, etc), the user is notified via email, that a new article has been posted and it should be checked. A short piece of this article is given with the notification, allowing the user to decide if he/she is interested in reading it. In addition, the user can visit a 'personalized' page, where he can see the latest 'clips' (snippets) of text that match his/her keyword lists.

This paper describes the information extraction mechanism of a web clipping service. This service is part of a multi-service platform for information search and manipulation. Various software modules contribute in more than one of the offered services. In order the system to be as much flexible and scalable, its design must be quite modular with easily interchangeable modules and well-defined interfaces between them. The Web Information Extraction module of the web clipping service is named 'Page Analyzer' and its purpose is to extract clips from web pages and give hints for the traversal of web sites. The functionality of this particular service is of course supported by various other software modules, which will be described briefly.

The web-clipping service, as well as the whole platform, is server -centric and accessed through the WWW. Although several systems have been proposed that are client based, we choose a server-centric architecture due to the vast amount of pages that a server-based crawler can explore.

Several Web Information Extraction systems have been presented in the past. The goals of these systems vary in general. However, all of them have a mechanism that extracts textual or other kind of information (for instance tabular data) from web pages. Such systems include WebCQ (Liu et al. 2000), THOR (Caverlee et al. 2003) , IEPAD (Cang et al. 2001), etc. We will describe these systems in the respective section of this paper.

This paper continues in section 2 with the related work and in section 3 with a short reference to the platform architecture. A short description of each software module is given there. However we do not emphasize on the internals of the software modules. In section 4 a comprehensive description of the 'Page Analyzer' is given. This module is the main focus of this paper. Section 5 describes two algorithms, which are used for the page analyzer, while in section 6 an experimental evaluation of the main functionality of the Page Analyzer is presented. Finally, in section 7 our future work plans and some conclusions are given.

2. Related Work

This section describes systems that have some similarities with our own. Although lots of systems have been presented in the past, we describe here three of them: WebCQ, THOR and IEPAD.

WebCQ is a server-based change detection and notification prototype system for monitoring changes in arbitrary web pages. Users of the system select which pages they want to monitor and customize the frequency that a stored page is refreshed and the frequency and the way changes in a web page are summarized and sent to the user.

The system consists of four main components: a change detection robot that discovers and detects changes to arbitrary web pages, a proxy cache service that reduces the communication traffics to the original information provider on the remote server, a personalized change presentation tool that highlights changes between the web page last seen and the new version of the page, and a centralized notification service that not only notifies users of changes to the web pages of interest, but also provides a personalized view of how web pages have changed or what is fresh information.

Users may monitor a web page for any change or specific changes in images, links, words, a chosen phrase, a chosen table, or a chosen list in the page. Furthermore, WebCQ allows users to monitor any fragment in the page specified by a regular expression, although transformation of a user-defined regular expression into an efficient one is not supported, therefore decreasing the usefulness of this feature. Another important feature of WebCQ architecture is its server-based proxy cache service. The system marks only changes of raw text, not changes of structure.

To present changes to the user, the system makes the choice between displaying the old and the new document merged together, displaying only the differences, displaying the two documents side by side and combining the above methods considering what changes are to be presented (links, images, text etc).

THOR is a scalable and efficient mining system for discovering and extracting QAPagelets from the Deep Web. QAPagelet is used to refer to the content region in a dynamic page that contains the result to the query performed. The purpose of the system is to separate the useful, dynamic region from the static one, which may be a navigation or an advertisement section.

A unique feature of THOR is its two-phase extraction framework. In the first phase (page clustering), pages from a web site are grouped into distinct clusters of structurally-similar pages. Structural differences would exist for example between a multi-matches page consisting of a list of query matches, a single match page with detailed information on a particular query result and a no match's page.

THOR uses a tag-tree based similarity metric. To weight tree-tag signatures, THOR uses a variation of term-frequency inverse-document-frequency (TFIDF). For measuring the similarity between different pages the cosine similarity metric is used. To cluster the pages the Simple K-Means algorithm is used, mainly for it's efficiency. In the second phase, pages from each page cluster are examined through a sub tree filtering algorithm that exploits the structural and content similarity at sub tree level to identify the QAPagelets.

IEPAD is an information extraction system designed to recognize patterns in Web documents. These patterns are used to extract information based on the structure of the documents. No a priori knowledge of their format is required.

The system uses an automatic approach to find the extraction rules, contrary to other approaches which require various amounts of human intervention or training examples and therefore are not scalable and practical.

The IEPAD system includes three components, an extraction rule generator which accepts an input Web page, a graphical user interface, called pattern viewer, which shows repetitive patterns discovered, and an extractor module which extracts desired information from similar Web pages according to the extraction rule chosen by the user.

The main data structure used to discover repeated patterns is the PAT tree, which is a Patricia tree (Morrison, D. R., 1968). The PAT tree is constructed over the tag token strings of the web document. After some patterns are found the suggested metrics of the patterns are evaluated (regularity, compactness and coverage) to eliminate some patterns. The results acquired can be customized by adjusting the thresholds of those metrics. Occurrence partition is possible when a web page has a particular alignment and order. Further customization is possible by defining which tags are going to be encoded to form the tag token strings and which will be ignored. The algorithm requires linear time.

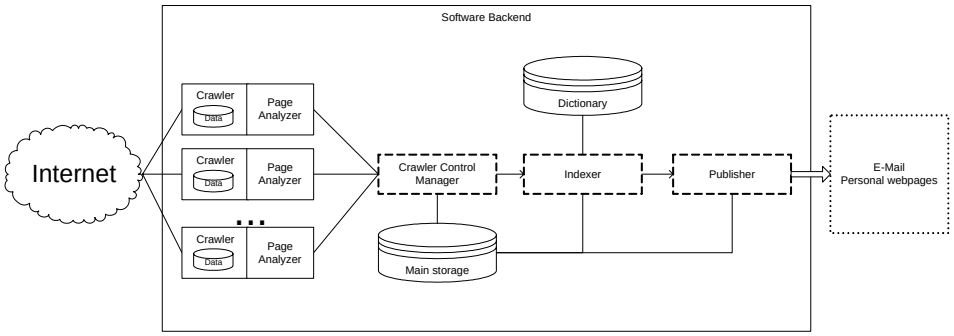

Figure 1: System Backend Architecture

3. Architecture

This section describes the architecture of the platform that hosts the web-clipping service. The most important factor of implementing the web-clipping service (and the whole platform in general) is making the backend software scalable. Thus, effective retrieval and storage of information is a very important factor to consider during the design phase.

The architecture of the net-clipping service is quite similar to that of a search engine (Arasu et al. 2001), (Cho et al. 'Synchronizing a database to improve freshness', 2000), (Huang et al.), (Page et al., 1998). Actually, this service is a form of a search engine. The users do not submit queries, but they have one or more constant queries, expressed by their keyword lists.

In this section we describe the architecture of the platform and the design decisions that were followed. The most important software modules of the system backend are shown in figure 1. These software modules are described in the next paragraphs.

The crawler is the software module that does the actual work of fetching remote resources from the WWW. We have chosen a distributed architecture for the crawler of our platform. The reason is simple: Distributing the task of fetching resources from the web to a set or collaborating crawlers, increases the number of resources that can be downloaded simultaneously and the number of sites that can be monitored by our platform. This way the collective throughput of the distributed crawler system is highly increased.

The crawlers' role is simple. They receive orders from the 'Crawler Control Manager' and they fetch the resources they have been asked for. However, they are not just 'dummy' network interfaces. They have their own local storage facility and they have the ability to decide on some issues regarding the resources they will fetch. These features significantly decrease the load on the central crawler control manager and increase the scalability of the system. In addition, a distributed crawling system is more fault-tolerant, since failures in one crawler does not affect the stability of the whole distribute crawler system.

Each crawler maintains a list of seed urls. These urls are the initial search pages. The crawler uses the links inside them to traverse the pages of web sites or sets of interconnected web sites. A copy of the text included in these web pages is kept in the local storage of each crawler. Each crawler incrementally updates the database of local copies of web sites. This means that it searches for a) new web pages added in these web sites, which have not been fetched previously and b) web pages that have changed since their previous version was stored. When a page of the second category is found, a new copy (instance) of it is stored, instead of a replacement of the old one. This is not useful for the net-clipping service, but contributes to some other services of the system.

The crawler control manager is the program that synchronizes the procedure or fetching resources (web pages) from the WWW. It is the interface of the whole system to the WWW, since the distributed nature of the crawler system is transparent to the rest modules of the software backend.

The Crawler Control Manager has knowledge of what specific sites each crawler is monitoring. In addition, it can request resources from every crawler and it can send instructions to them. However, it does not keep a replicated copy of every page the crawlers have fetched. These resources are kept in the local storage of the respective crawlers. The Crawler Control Manager keeps (in the main storage) only information that is used for synchronizing the services of the platform and information that must be kept centrally in order to be used by all the crawlers.

The Page Analyzer is the software module, which is responsible for analyzing each page that is downloaded from the web. All the decisions regarding how a web clip will be extracted are made in this module. The Page Analyzer mechanism is the main focus of this paper and it will be described thoroughly in the next sections.

The indexer creates a list of the keywords contained inside the analyzed page and the locations inside the page. Afterwards, it updates the main index, which is stored in the main storage. A copy of each page is stored in the local storage of the crawler that fetched it.
The dictionary is a lexical database, which serves many services of the whole platform. It is used for stemming and for finding synonyms of words. The use of a dictionary helps to increase the precision of the results. Describing the dictionary of the platform is out of the scope of this paper. Since we are still in the design phase of the platform we have not actually concluded in a specific implementation. However, we have decided not to create a new dictionary, since this is a very expensive task, in terms of money and time. We are searching some open source implementations (e.g. ISPELL, ASPELL) and we will use the one that best serves our needs. Although we need a dictionary with support for the Greek language, it would be useful if the mechanism that we will finally use had support for English also.
The Publisher is the software module that is responsible for notifying registered users about newly discovered and extracted web clips, which are of interest to them. This notification will occur through email and through a summarization web page. When a clip is discovered that contains keywords which match the given keywords of a specific user, the user will be notified via email about these clips. The clip (short text) and a title of the clip will be sent to the user via email. In cases where a user has selected lots of keywords and receives a lot of emails, it would be possible (upon user selection) to notify the user sending an email for a set of clips and not for each one of them. In addition to the email notification, the users will be able to see all the clips that match their keywords, using a personal 'summarization' web page.
The Main Storage is a database management system containing all the information that must be kept centrally. For the needs of the web clipping service, this information includes the text of the clips (title and short text) and information about the interests of the users. The Main Storage is out of the scope of this paper and will not be analyzed further. More information on the design of a high-performance storage system can be found in (Hirai et al., 2000).

4. Page Analyzer

The 'Page Analyzer' is the software module, which analyzes each page that is fetched by the distributed crawler system. This paper emphasizes on this specific module.

4.1 Description

In addition to the extraction of the keywords and indexing, which is a standard step of every search engine, the web-clipping service has some additional requirements: Each extracted 'clip' must have a title and a short text that contains the keyword (or a synonym, etc). In addition, the users should be notified about clips that have a considerable amount of text and are worth reading and not just about every simple instance of a keyword. This means that when a link (containing a keyword) points to an article, the user should be notified about the article and not about the page that contains the link or both. The required actions for extracting this additional information are performed by the page analyzer. Some heuristics will be used for detecting if a page is an 'information page', worth of extracting clips, and not a 'hub' page, containing just links to other pages. These heuristics will consider the relative ratio of text versus links inside the examined page, as well as the absolute amount of text inside the page, in comparison to the average amount of text inside 'hub' and 'information' pages. If a page is classified as an 'information' page, then the page analyzer will continue with the extraction of clips from it, otherwise it will just update the index of the main storage.
If a page is classified as an 'information' page then a title must be assigned to it and a short text must be extracted to accompany each keyword. However, in order to reduce the overhead of this step, a comparison is performed between the list of the extracted keywords and the list of the keywords that the users have submitted for notification about related articles. The clip extraction process will be followed only for those keywords that the users are interested in.
When the list of the keywords that the clip extraction process will be applied has been prepared, the surrounding text of the keyword will be extracted (one sentence before the keyword and one sentence after). The title for the clip will be extracted by using again some heuristics. The most obvious of them, is by analyzing semantically the HTML tag tree of the specific 'information' page. More specifically the page analyzer searches for some semantic 'hint', which denotes that a small piece of text (not more than 10 words) is a title. This 'hint' must be 'near' the text of the

extracted clip. There are some HTML tags, which are usually used for titles, such as H1, H2, etc ('heading' tags). These are the first category of 'hints' to search for. However, nowadays, usually CSS styles are applied to the titles of articles. In the absence of 'heading' tags, the page analyzer will search for CSS styles that are not used widely inside the page body. If CSS styles cannot be found, then the first sentence of the article, where the clip belongs in, will be extracted as the title of the clip. This procedure is followed until all the clips have been extracted.

4.2 Definitions and Metrics

Every web page is composed by a set of files: HTML code, CSS external definitions, images, etc. The main file that contains the content of a web page is the HTML file. HTML tags are nested, which means that the content of the HTML file can be represented as a tag tree. This is depicted in **Error! Reference source not found.**. The 'Tag tree' is referred in some publications as the DOM tree.

HTML tags vary in their role inside the HTML tree. Some of them have *structural* role, i.e. they define how the HTML page's content is structured inside the page. The most frequent of them are table-related tags, such as TABLE, TR, TD, TH, etc. Some other tags have *presentational* role, i.e. they define the style of the page's content (how the page's content appears inside the page). Examples of these tags include B, STRONG, FONT, etc. These tags are used nowadays rarely, since cascading style-sheets achieve the same results in a more structured manner. There is a third group of HTML tags that contains special-purpose tags, such as A (Anchor tag). The placement of HTML tags in a nested order and their semantic meaning allows the extraction of useful hints for the Page Analyzer mechanism. Both the Structural and the Presentational role of tags will be exploited in the procedure of properly extracting clips (snippets) from web pages.

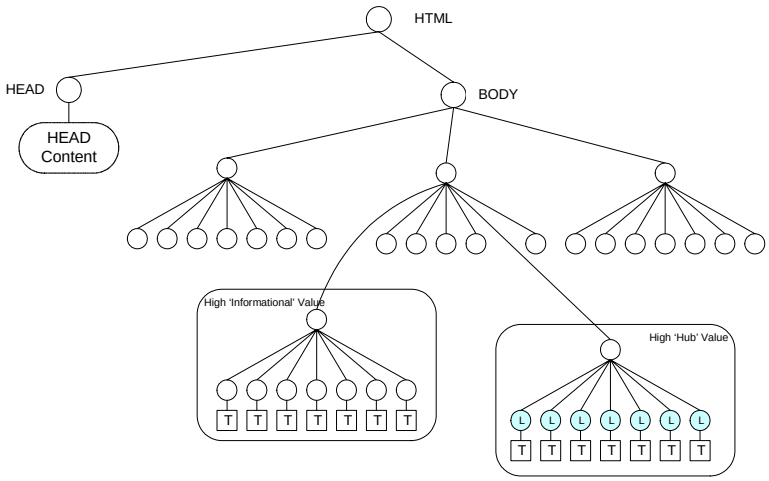

Figure 2: HTML TAG tree

In the following paragraphs we will give a more formal definition of some notions and structures that will be mentioned in the rest of this paper:

A *Tag Tree* is a tree structure that shows the structure and content of an HTML page. The text of the page is contained in the leaves of the tree. The leaves contain either text or tags that do not require a 'Closing Tag' (such as IMG). The nodes contain tags that have a Closing Tag.

Tag Tree: TT= (V,E), where V is the set of nodes and E is the set of edges

A *node* inside the tag tree represents every HTML element inside the HTML code of the web page. The text is contained in the *leaf nodes.* The *edges* of the tree tag represent the nested relationship of HTML tags inside the page. This relationship offers valuable information for the Information Extraction process. Edges are directed and denote a parent-child relationship. Every node can have a lot of children and only one parent. The root node (which represents always the HTML tag) is the only node, which does not have a parent. The leaf nodes do not have any children.

The set of *children* of node (tag) *p* are represented in the rest of this paper as C_p, while the set of *descendants* are represented as D_p.

$$C_p=\{q, \text{node } q \text{ is child of node } p\}$$

$$D_p=\{q, \text{node } q \text{ is descendant of node } p\}$$

A *tag object* (also called *simple tag object* or *simple object*) is a node, which contains only leaf values. If an object does not contain any tag with no respective Closing Tag, then the only child of this object is a text node. Simple objects are the basis for the computations that take place in Page Analyzer. A *complex object (CO)* may contain both simple objects and other complex objects. A whole web page is a complex object. The mechanism of the Page Analyzer aims to partition a web page in Complex Objects and locate those Complex Objects that a) contain interesting information and b) can lead to other pages which contain interesting complex objects.

One basic notion for the information extraction algorithm is the distinction of pages and areas inside pages as 'informational' and 'hub'. Informational areas (or pages) are those, which contain text (articles, news, etc). The clips must be extracted from such areas and pages. On the other hand 'hub' areas and pages contain links to other pages, which could contain interesting clips. The information extraction mechanism must be able to separate the 'information' areas from 'hub' areas, extract clips from information areas and follow links from the proper hub areas.

This distinction can be seen more clearly in some real web sites. An example can be seen in figure 3 (from www.flash.gr). This page contains some text for 4 main articles and after them there are many links for other articles. Although the 4 main articles at the top of the page contain links to pages with the full versions of the articles, the text contained in the main page is useful in the clip extraction process, since it could serve as a summary for the main article. Therefore the information extraction mechanism should use this text as well.

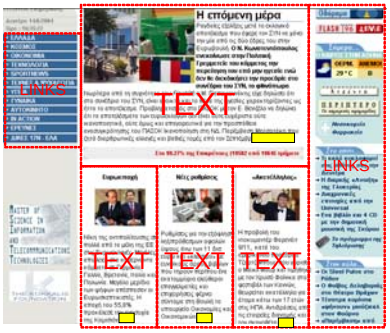

Figure 3: Page with high 'informational' and 'hub' value

In order to separate 'informational' from 'hub' areas inside a web page, the information extraction mechanism uses the tag tree of the web page and performs some calculations. Their purpose is to assign an 'informational' and a 'hub' value to the *complex objects* inside the web page.

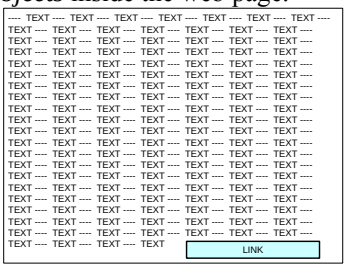

Figure 4: Page with high 'hub' value

Figures 4 and 5 show the most common cases of Complex Objects' (CO) content. The first case (figure 4) shows a CO that contains mainly text (or images) and a link. This CO represents an informational area. The existence of a link probably denotes that by following it the reader will be transferred to a page with more relevant information. This CO has high Informational Value and low Hub Value. However, although the Hub Value of this CO is low, the link should be followed for the Clip Extraction since it probably leads to a web page with more relevant information.

Figure 5 shows a Complex Object, which serves as a hub area. The links inside this area probably lead to articles or news pages, whose thematic area (title) is described by the text of the link. However, this is the simplest form of a hub area. For instance the link text could be smaller and the context of the link could be described by the text before

and after the link. In addition in some other cases the text of the link could be very small (one or two words) or even the link could be an image. Usually such links serve for navigational purposes.

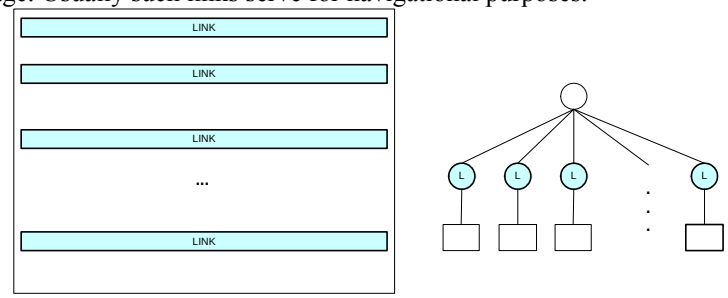

Figure 5: Area with high 'hub' value

Before proceeding with the definition of Information Value and Hub Value we give some other metrics that are also used:

- **NumLinks:** Number of links from node p: L(p)
- **Text:** Amount of text below node p: T(p)
- **Link Density Metrics**

  Ltext=Link Text (in bytes)
  Ctext = Clean Text (in bytes).
  Ltext shows the size of text that is contained inside links, while Ctext shows the size of text that is contained inside a node, without taking into consideration the text that is contained in links.

  Link to Text Ratio: LTR= $\frac{CText}{Ltext}$

  Link Average Ratio: LAR = $\frac{L(p)}{Ltext}$

- **Informational Value (IV):**

$$I_p = WeightIV_p \bullet \sum_{\forall q \in C_p} I_q \text{ , if } C_p \neq \emptyset$$

$I_p = T(p)$, if $C_p = \emptyset$

$WeightIV_p$ is a term that aims to insert the 'importance' of a tag in the calculations for IV_p. In the current implementation $WeightIV_p$ equals 2 for all table-related and list-related tags (such as TABLE, TR, TD, UL, LI, etc) and 1 for all other tags. In future versions of the page analyzer mechanism this might change, based on experimentation.

- **Hub Value (HV):**

$$H_p = \sum_{\forall q \in C_p} H_q \text{ , if } C_p \neq \emptyset$$

$H_p = 1$, if $C_p = \emptyset$ and p.TYPE=LINK

Figure 6 shows the outline of the Page Analyzer Mechanism. The Informational Value is used for finding the informational areas of a page, while the Link Density Metrics, as well as the Hub Value are used for discovering the Hub Areas.

1) Get a page from the respective URL and build the tag tree
2) Make Global Calculations
3) Partition the index tree in Complex Objects and calculate *Informational* And *Hub values*
   a. Partition the tag tree in Complex Objects.
   b. Calculate IV and HV
4) Handle the results
   a. Extract the Clips from CO that have Informational Values
   b. Follow links from areas with high Hub Values

Figure 6: Page Analyzer Mechanism

5. Algorithms Of Page Analyzer

This section describes the algorithms used in the Page Analyzer. There are two main algorithms used: The "Page And Node Characterization" algorithm and the "Clip Extraction" Algorithm. The first algorithm is trying to detect which parts of Web Pages contain information that could be used to extract clips, while the second one is the algorithm that takes an area of a web page (Complex Object) and extracts the information that we need.

5.1 Algorithm 1: Page and Node Characterization

This algorithm is used for finding areas inside web pages that have high informational and hub values. The first category of areas (complex objects) are used for extracting clips, while the second category is used to find which links the crawler should traverse next.

```
TraverseTagTree(node p){
    Int IV, HV;
    Foreach q (child of p) {
        TraverseTagTree(q)
        IV=IV+q.IV;
        HV=HV+q.HV;
    }
    IV=IV+T(p)
    If (p.TYPE==LINK) {
        HV=HV+1;
    }

    //STOP CRITERIA
    If (IV > ThrIV) {
        CLIP_DETECTED(p);
        STOP();
    }
    else if (HV>ThrHV) STOP(); //Follow proper links
}
```

Algorithm 1: Page and node characterization

5.2 Algorithm 2: Clip Extraction

The clip extraction algorithm is used when a Complex Object with high Informational Value has been detected. This means that the area that is represented by this CO contains information that should be extracted. Three are the important factors of this procedure:

- If any link exists inside the specific area, then it should be followed, since probably the page pointed by the link contains relevant information
- Extract the important keywords from this area. These keywords must much keywords from the users' profiles
- Find a Title and a Summary for the extracted clips.

While 'merging' the relevant areas and extracting the keywords is a straightforward procedure, creating a Title and Summary for the clips requires the use of some heuristics.

```
ExtractClips(node p){
    //It follows the page and node characterization algorithm.
    If  (p.links.size>0) {
        FollowLinks(p);
    }
    else {
        //The only area to extract the clips is the one denoted by p
        Collection kw=ExtractKeywords(p.TEXT);
        Collection clips;
        Foreach (keyword) in kw {
            // A clip contains the keyword, summary and title
            Clip clip=new Clip(keyword);
            clip.Summary=GetTextNear(kewyword.Position, p.TEXT);
            clip.Title=p.HasTitle?p.TITLE:GetFirstSentence(p.TEXT);
        }
```

```
    }
}
```

Algorithm 2: Clip extraction

6. Experimental Evaluation

This section presents an experimental evaluation of this proposed technique. We measured the amount of informational and hub areas that the Page Analyzer mechanism discovered in the homepages of various portals. We wanted to evaluate how the page analyzer performs on homepages of real web sites. The test pages are from international and Greek sites. The results are presented in Table 1 and Figure 7.

Table 1: Partitioning of homepages

	URL	Inf. Areas	Hub. Areas
1	http://www.cnn.com	6	14
2	http://www.msn.com	2	21
3	http://www.cbsnews.com	5	14
4	http://www.slashdot.org	17	10
5	http://www.yahoo.com	2	17
6	http://www.in.gr	3	20
7	http://www.flash.gr	7	14
8	http://www.contra.gr	7	6
9	http://www.e-go.gr	1	12
10	http://www.microsoft.com	1	9
11	http://www.e-one.gr	1	12
12	http://www.netscape.com	3	10
13	http://www.builder.com	10	9
14	http://www.developer.com	5	10
15	http://www.linux-	8	13
16	http://www.linux.com	7	11

As it was expected, the detected hub areas inside the homepages in web sites are much more than the information areas inside these sites. This happens because most homepages serve just as a 'hub' page, which leads users to other pages. However, some homepages, such as the ones of www.contra.gr, www.slashdot.org and www.builder.com, have more informational areas than hub areas

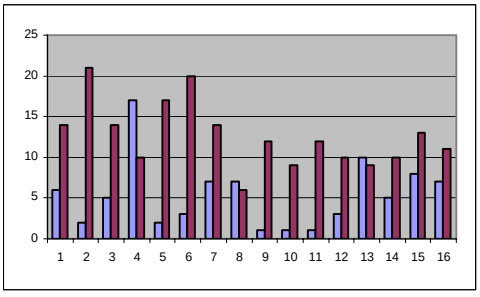

Figure 7: Partitioning of homepages

This is because these sites have a lot of information is their homepage also. Internal pages of these sites, which contain more information, have less hub areas and more informational areas (or bigger in size).

These results are quite encouraging for the proposed mechanism. The partitioning of the homepages was correct in most pages. However, in some cases, navigational areas were included in the hub areas. These areas in most cases do not lead to pages with high informational values and should be dropped. A filtering 'step' will be included in the next version of the mechanism in order to drop such areas out of the final results.

7. Future Work and Conclusion

This paper presented the Web Information Extraction mechanism of a web clipping service. This mechanism is the most crucial module for the precision and relevance of the extracted clips, in accordance to users' interests. Although we are still at the design phase of the information extraction mechanism and the web clipping service, we tested our proposed mechanism and the results were quite encouraging. We believe that separating areas of web pages based on their informational and hub value can help enhance the results of an information extraction process, since the IE mechanism focuses on pages intended to provide information. However, this mechanism can be furthered enhanced in various aspects. Our future work plans include testing this mechanism against several other approaches for Web Information Extraction proposed in the bibliography and adopting their best features if they are applicable to our mechanism. In addition, the rest of the modules of the whole system will be designed and implemented in parallel, leading to a working first prototype of the whole system. This will allow further testing of the technique, based on real users' usage and satisfaction. We believe the final product will be a very handy solution for handling the vast amount of information that is daily published in the WWW.

REFERENCES

Arasu A., Cho J., Garcia-Molina H., Raghavan S. (2001), Searching the web. *ACM Transactions on Internet Technologies*

Caverlee J., Buttler D., Liu L. (2003), Discovering Objects in Dynamically Generated Web Pages. *Technical report, Georgia Institute of Technology*

Chakrabarti S., Van Den Berg M., Dom B. (1999), Focused crawling: a new approach to topic-specific Web resource discovery. *In 8th International WWW Conference*, Toronto

Chang, C-H, Lui, S-L. (2001), IEPAD: Information Extraction based on pattern discovery. *WWW-10*

Cho J., Garcia-Molina H. (2000a), "Synchronizing a database to improve freshness", *Proceedings of the ACM SIGMOD International Conference on Management of Data*

Cho, J. Garcia-Molina H. (2000b), "The evolution of the web and implications for an incremental crawler", In Proc. of 26th Int. Conf. on Very Large Data Bases, 117-128

Cho J., Garcia-Molina H. (2003), Estimating frequency of change", *ACM Transactions on Internet Technology (TOIT)*, Volume 3, Issue 3, 256 – 290

Cho J., Garcia-Molina H. and Page L. (1998), Efficient crawling through URL ordering. *7th International WWW Conference*

Coffman E. G., Liu Z. and Weber R. R. (1997), Optimal robot scheduling for Web search engines. *Technical Report 3317*, INRIA

Hirai J., Raghavan S., Garcia-Molina H., Paepcke A. (2000), WebBase: A Repository of web pages. *Proceedings of the 9th WWW Conference*

Open Source Dictionary: Ispell and Aspell. Retrieved at September 24, 2004, from http://ispell.source.gr/how_aspell.html

Huang L. (2000), A Survey On Web Information Retrieval Technologies. *Technical report*, ECSL

Liu L., Pu C., and Tang W. (2000). WebCQ: Detecting and delivering information changes on the web. *International Conference on Information and Knowledge Management*

Menczer F., Pant G., and Srinivasan P.(2002), Topic-driven crawlers: Machine learning issues, *ACM* TOIT

Page L., Brin S. (1998), The anatomy of a large-scale hypertextual web search engine. *Computers networks and ISDN systems*, 30:107-117

Shkapenyuk V. and Torsten S. (2002), Design and Implementation of a High-Performance Distributed Web Crawler. *In Proceedings of the International Conference on Data Engineering*

Zeinalipour-Yazti D., Dikaiakos M. (2001), High-Performance Crawling and Filtering in Java, *Proceedings of the 8th Panhellenic Conference on Informatics*, Nicosia, Cyprus

Morrison, D. R. (1968), *Journal of ACM*, 15, 514-534