Web Page Fragmentation for Personalized Portal Construction

Bouras Christos Kapoulas Vaggelis Misedakis Ioannis

Research Academic Computer Technology Institute, 61 Riga Feraiou Str., 26221 Patras, Greece and Computer Engineering and Informatics Department, University of Patras, 26500 Rion, Patras, Greece

+30-2610-960375 *+30-2610-960355* *+30-2610-996954*

bouras@cti.gr *kapoulas@cti.gr* *misedaki@cti.gr*

Abstract

Web portals offer many services and wealth of content to Web users. However, most users do not find interest in all the content present in these sites. Most of them visit some specific sites and browse in specific thematic areas of them. In this paper, a software technique is presented that allows the viewers of web sites to build their own personalized portals, using specific thematic areas of their preferred sites. This transcoding technique is based on an algorithm, which fragments a web page in discrete fragments using the page's internal structure. A training and update procedure is used for identifying the different instances of thematic areas in different time points.

1. Introduction

Most web sites have a static structure for the presentation of their content. In content-rich web sites this structure comprises of areas of content of common semantic. These areas are called 'Web Components', because the web page can be split entirely in such discrete areas. Most users of the Web visit some specific web sites every time they are engaged in a browsing session and they usually show interest for some specific thematic areas.

In this paper a technique is presented that could assist web users in their browsing sessions. This technique could be used for building a web site that allows its users to construct 'personalized pages' containing content from their favorite sites. A user could have in a single web page, all the thematic areas of the sites he prefers. The presented technique premises the usage of a software tool, working centrally (as a data source for the web server), and which analyzes selected web pages and fragments them in thematic areas.

'Web Components' (denoted WC from now on) was introduced as a concept in [1]. The fragmentation algorithm that is used in the system was presented in [2]. A 'Web surfing assistant' is presented in [3], which utilizes a similar fragmentation technique as the one presented in this paper for splitting a web page in semantic regions. Several transcoding systems have been presented that aim to provide users of small-screen devices, such as PDAs or WAP-phones, an alternative, enhanced way of browsing the Web (see e.g. [4]). Finally, [5] proposes a system, which, like the one presented in this paper, focuses on the problem of identifying a particular part of a web page in different time points, besides fragmenting a page.

This paper continues with a brief presentation of the fragmentation algorithm that has been implemented in section 2. In section 3 the training and update procedure is presented. Section 4 presents an evaluation of the training technique and the paper concludes with some future work thoughts in section 5.

2. Fragmentation Algorithm

A browser renders a web page based on the HTML file that represents the page. The tags inside the HTML file are nested. This means that the code of the page can be represented as a tree (HTML tree). We could extract the parts of the page that represent the different Web Components of the page just by extracting some particular nodes of the HTML tree.

Most web sites use tables for building their layout. This lead to the decision to use the table structure of a web page for fragmenting the page. If we ignore all the nodes of the HTML tree except the TABLE nodes, the HTML tree is reduced significantly in complexity. Based on the amount of content (text) of each node, the algorithm chooses which nodes must be considered as the building components of the web page.

The fragmentation algorithm is used for the *web pages' analysis and fragmentation,* which includes two phases: training and update.

Algorithm 1: Fragmentation Algorithm

Steps 1-4 are used both in the 'Training' and the 'Update' phase.

1) Fetch the latest instance of the web page
2) Parse the web page and construct the HTML tree
3) Analyze the HTML tree and produce the index tree
4) Analyze the index tree and calculate which nodes must be marked as Web components

Steps 5 and 6 are used only in the 'Update' phase.

5) Check if there are differences in the structure of the index tree from the index tree of the 'training' phase or if there are differences in the number of the web components selected. In case there are differences, recalculate the web components.
6) Extract the Web Components and store them.

For the needs of step 2 an HTML parser was created, which builds the HTML tree. Step 3 of the fragmentation algorithm constructs (using the HTML tree) another tree structure ('index tree') that is used in step 4 for recognizing which areas of the HTML file will be extracted as Web Components. The index tree is significantly smaller in size than the HTML tree, since only the TABLE nodes have been kept from it. The ID of each node depends on the position of the node in the index tree. In each node in this new structure, its corresponding node in the HTML tree is linked and also some information for calculations that will be performed later is stored. This information includes the length of the text of this node in the HTML file (with and without the tags), the ID of the node, the number of images that are included under this node and finally the number of links that can be found in the text (content) of the node.

The actual decisions about how a web page will be fragmented are taken in step 4. The fragmentation algorithm recursively parses the index tree trying to find nodes that match some particular criteria. When such a node is found, the algorithm stops traversing its children and the node is marked as a Web Component. When the fragmentation algorithm finishes the traversal of the index tree, it makes some last refinements of the Web Components selections.

After this point, the algorithm includes two more steps, which are used only in the update phase and will be described in the respective section of the paper.

The fragmentation algorithm has been presented in [2].

3. Transcoding Technique

In this section the methodology for constructing personalized web pages based on Web Components is presented. There are three phases: *Web pages' analysis and fragmentation*, *components selection (by the user)* and *personalized page synthesis*.

3.1. Web pages' analysis and fragmentation

Web pages analysis and fragmentation involves a software tool whose role is to continuously analyze selected web pages and update the information that is stored about them and the HTML code of the components. This tool is not installed on the users' personal computers, but functions centrally, as a data source for the web server of the service provider. WCs are not created dynamically upon users' requests, but are extracted and stored by this software mechanism. The building code of each Web component is updated in frequent time intervals.

Web pages' *analysis* and *fragmentation* is a multi-step procedure. This procedure includes two main phases. The first one is the *training phase.* Each web page is examined for a given period of time, areas that can be treated as WCs are detected and signatures are created for them. The *update* procedure fragments the web page and updates the latest instances of the WCs that are stored in the system.

In the following sections the training and the update phases are examined more thoroughly.

3.1.1. Training Phase. During the training phase, a web page is analyzed many times. In the end of the training phase, the training algorithm's output is the number of the WCs of the examined page, and a *unique identifier* for each one of these components. This unique identifier can be used for identifying a WC in a web page instance that has changes in the page structure or changes in the number of the WCs. This training phase would not be required if changes never happened in the web page' structure and the relative size of its content areas. The training phase allows the system to have pre-built knowledge about how to handle these changes. One basic assumption for its correct functioning is that changes like these *do not happen during the training procedure*. We use the simplest solution when this situation occurs, which is to reject the samples that have changes.

A web component can be characterized by many factors: Its position inside the index-tree, its relative position to the other web components, its ID inside the index tree, its content (in terms of text or images), its content size (in terms of text length or number of images) and others. However, it is quite difficult to find a criterion that can be used to *uniquely* identify a WC from the others that are contained inside a Web page. We have to note here that this is necessary for the proper functioning of the system, since the users must be able to select which components they wish to see in their personalized page and the system must be able to recognize them from the list of the Web Components extracted from the fragmentation algorithm.

The training phase can be split in four sub-phases:

1. Data gathering phase
2. Comparison of *Content Vectors* of instances of the same WC and extraction of a single *Constant Content Vector (CCV)* for each Web Component
3. Comparison of the CCVs of all the WCs of the web page and extraction of the *Identifier Content Vector* (*ICV*) of each Web Component.
4. Assignment of *signatures*.

In the data gathering phase, during fixed intervals of time the fragmentation algorithm is activated and the index tree for the specific page instance is stored. The goal is to have enough specimens of the index tree for a time interval in which all the content changes that happen regularly in

the web page have taken place. When this predefined monitoring time period has passed, k specimens of the index tree have been collected, where $k = \left\lfloor \frac{\text{Monitoring Period}}{\text{Sampling Interval}} \right\rfloor$.

In the second phase, some calculations take place that aim to recognize the content of each Web Component that stays *constant* during the monitoring period. For each WC of the index trees, the fragmentation algorithm constructs a data structure that contains its content, i.e. every word of the text inside the WC and the filenames of the images contained in the component. This data structure is named *'Content Vector' (CV)* and is a characteristic of each *WC instance* (This means that the CV can be different for different instances of the same WC). The CV is a pair of two vectors, one containing the text terms inside a WC instance and one containing the filenames of the images. These are symbolized as T_p and I_p respectively.

$$T_p = \{w \mid w \text{ is a word inside the pure text of WC } p\}$$
$$I_p = \{z \mid z \text{ is an image contained in the code of WC } p\}$$
$$CV_p = (T_p, I_p)$$

We assume that for the k specimens of the index trees the number of the WCs that have been selected in each fragmentation and the index tree's structure remain the same. Using the ID of each WC, the training algorithm acquires this WC's instances and its CVs from the collection of the index trees. Following this it compares the k CVs of each WC and keeps only the content that exist in all the CVs. In the end of this procedure the algorithm has constructed a data structure that keeps the content of each Web Component that remained constant during the whole training procedure. This structure is named *'Constant Content Vector' (CCV)* and is a characteristic of a WC independently of its instances in different time points.

$$CCV_p = \bigcap_{t=1}^{k} CV_{p,t}$$

($CV_{p,t}$ is the Content Vector of the t^{th} instance of WC p).

The CCV of a WC is derived taking into account only the content of this specific WC. But the goal of the training procedure is to produce identifiers for all the WCs of a Web page, which could uniquely identify all of them in the web page. Therefore, in step 3 of the training procedure the CCVs of all the WCs are compared mutually and the text or images that exist in all the CCVs are removed. These reduced CCVs are named *'Identifier Content Vectors' (ICVs)*. In addition, if the content of an ICV is contained completely inside the content of another ICV, then the first WC and its ICV are marked as *weak*. This means that its ICV cannot uniquely identify it. In the end of step 3, each WC has an ICV that uniquely identifies it in the Web Page, with the exception of WCs that are marked as weak.

$$ICV_p = CCV_p - \bigcap_{i=1}^{p_{\max}} CCV_i$$
$$WC_p \text{ is weak if } \exists\, WC_q : ICV_p \subseteq ICV_q$$

In step 4 the ICVs are set as the *signatures* of their respective WCs. However, it is possible that some components have an empty or weak ICV (an empty ICV is also weak, since it is a subset of all other ICVs). This ICV cannot be used as a signature. Therefore, in step 4 the training algorithm detects the WC with weak ICVs and assigns another kind of data structure, which is based in their relative position in the web page and in the content size, as a signature for them. The way this signature is constructed is explained with an example:

Figure 1 shows the index tree graph of www.in.gr. While almost all of its Web Components (which have been marked with bold in the index graph) have an ICV as a signature, the leaf nodes, which are descendants of node with the ID 3-6, have been chosen as Web Components and all of them have empty ICVs. Therefore, the algorithm must assign a separate signature to every one of them. The Web Components with IDs 3-5-4 and 3-7-1-1 are the first components before and after the series of the components with empty ICVs. These two components are included in the signature data structure. In addition, the position of each one of these components with empty ICVs inside their list is stored in the signature data structure. These three fields denote the relative position of the components regarding the other components in the index tree. Additionally, two more fields are used to store the text size and the number of images included in each component. These are used in order to identify a component in cases where the relative position is not enough. Therefore, the identifier data structure for components with weak ICVs has the structure shown in Figure 2. In this identifier the previous and the next component (1^{st} and 2^{nd} field) are marked with their *sequence number* in the *page index*, which is the final output of the training procedure. The page index is a matrix, which contains the ID and the signature of every Web Component of the page.

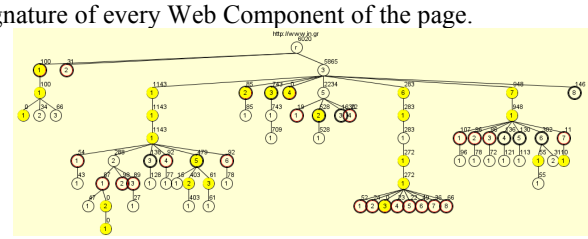

Figure 1: Index tree for www.in.gr

Figure 2: Identifier for weak components

3.1.2. Update Phase. Following the training phase, knowledge has been acquired about what output to expect from every fetching and fragmentation of the web page.

The full structure of the web page is available (shown in anyone of the k index trees collected during the training phase) and also the page index, which stores all the web components signatures.

The role of the update phase is to update the stored data with the latest instances of the Web Components (HTML code, images, etc). The time interval between each fetching and update of a web page depends on the frequency of change of its content.

The fragmentation algorithm produces in step 4 the index tree of the web page instance that was fetched and marks some nodes as Web Components. Step 5 takes the index tree and the page index and checks if there are differences in the structure of the page or in the number of the calculated web components. Although it is possible this check to be implemented directly on the index trees, it is done by checking for differences in the ID field between the page index of the latest page instance and the page index that was produced in the training procedure (the page index of page *instances* contains the Web Components' Content Vector in the placeholder of the signature). If no changes show up, then the fragmentation algorithm continues with step 6. Otherwise, a fragmentation correction algorithm is utilized, which aims at fixing the problems. Although it is impossible to create an algorithm that could function with very complex situations, the algorithm presented below shows a very good behavior for the most common cases.

Algorithm 2: Fragmentation Correction Algorithm

```
(1) If (WC_count in the page index from training== WC_count in the instance page index){
      Check the signatures contained in the page index with the Content Vectors
      contained in the instance page index
      (2) If (signatures match) {
            Extract (mark for extraction) the Web components based on their signatures
          }
          (2) else {
            Extract all the Web components that their CVs match with signatures in
            the page index. Extract all the rest WCs based on their order of appearance
            in the page index.
          } (2)
} (1)
(1) else {
      (3)If (index tree structure from training matches with the instance index tree) {
            Extract Web components based on their IDs
      } (3)
      (3)} else {
            Counter++;
      (4) If (Counter<4){
        (5) If (WC_count in the instance > WC_count from training){
            Run the fragmentation algorithm with its parameters set to produce larger
            (and less) Web Components
          (5)} else{
            Run the fragmentation algorithm with its parameters set to produce
            smaller (and more) Web Components
          }(5)
        (4)} else {
          Get the initial fragmentation (with the default value of the u parameter).
          Extract all the Web Components that can be extracted based on their Content
          Vectors. Extract all the remaining Web Components based on their order of
          appearance and their content size (closest match).
      }(4)
    } (3)
  }(1)
```

We have to note here that the algorithm presented above uses the CVs of the WC instances for the comparisons with the signatures of the selected WCs from the training phase. However, it is more complicated if the page contains components with weak ICVs, because these WCs have a different kind of signature. Whenever such a situation occurs, the algorithm compares all the components that have an ICV as a signature and after this comparison has been performed it tries to calculate the rest based on their relative position and their content size.

Another issue is how to make the comparison between the CVs of WCs instances and the ICVs of WCs contained in the page index. The ICV is a subset of the CCV of a Web Component instance. It also uniquely identifies a Web Component, i.e. there are no two Web components with Content Vectors that are supersets of the same ICV. The fragmentation correction algorithm uses this property of the ICVs for the comparisons it has to make. Specifically, it checks an ICV against all the CVs of the WC instances. The first matching instance, is the WC it tries to detect.

When the fragmentation correction algorithm finishes, all the WC instances have been marked for extraction. Then, in step 6 they are extracted and materialized (with some changes in their HTML code) in the Web Server.

3.2. Personalized Portal Creation

Web page analysis and fragmentation aims at having always the latest instances of the WCs that comprise the web pages that are offered to the user for their personalized page creation. Personalized Portal Creation is targeted to the user. It aims at creating a list of the WCs that a user wants to include in his/hers personalized page or altering this list by adding or removing components. Using a special page of the web interface of the system, the user is called to select one of the sites that have been analyzed by the system. When the user makes a choice, he is transferred to a page where all the web components of the selected site are shown. From this page the user can select his preferred WCs. When a user finishes with the selected web page, he/she can be transferred back in the first page where he/she is asked again to select one of the available sites.

3.3. Personalized Page Synthesis

Personalized Page Synthesis is performed by a script in the web server of the service provider. It constructs the user's personalized page using the HTML code of the selected WCs.

It has to be noted that during the personalized page synthesis, a special procedure must be followed for web components that originate from pages using CSS and Javascript or VBscript. In this procedure, the names of styles or functions are slightly changed in order to avoid naming conflicts.

An example of a personal page is seen in figure 3. In this page there are 3 WCs, one from www.e-go.gr and two from www.abcnews.com.

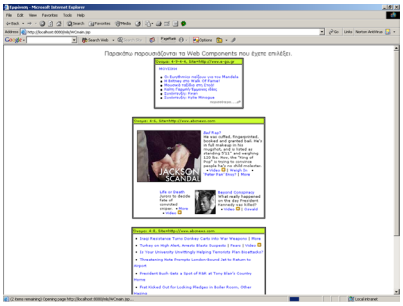

Figure 3: Personal page

4. Evaluation

In order to evaluate the training/update procedure we executed the algorithms in the night of 20 November 2003 with three news websites (CNN, ABCNEWS and CBSNEWS). The time interval between each parsing was 50 minutes and 10 parsings were performed. The results are shown in figures 4 and 5, from which some interesting conclusions are drawn:

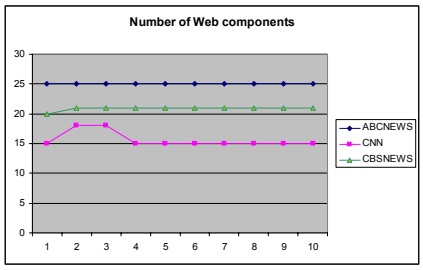

Figure 4: Number of Web Components

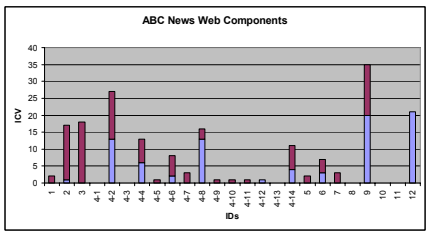

Figure 5: Web components of ABCNEWS

It is clear that the assumption that no changes happen in the number of the WCs during the training procedure does not hold true always. This is shown in the graphs (in figure 4) for CBS News and CNN. However, there are only 2 and 1 fetches respectively (out of 10) that differ from the majority. Therefore, rejecting these samples does not cause significant problems and loss of information.

Figure 5 shows the size of the ICVs of the 25 web components of ABC News. 9 WCs out of the 25 have weak ICVs, while 5 out of the 9 WCs with weak ICVs are also empty. However, almost all of them are weak not due to changes in their content, but due to small size. This shows that the whole technique could be enhanced by not allowing the fragmentation algorithm to select small Web Components (it could merge them with others or just ignore them). Figure 5 shows also that some sites (such as ABCNEWS) utilize a lot of images for building their layout and these images contribute a lot to the content of the Web Components. This leads to the thought that the images should be also considered in the heuristics of the fragmentation algorithm.

5. Future Work - Conclusions

A first prototype of the proposed technique has been implemented and experiments with some sites have been performed. Based on the results of these experiments, we have concluded on several possible improvements. The fragmentation algorithm can be enhanced with more advanced heuristics. Some times it can produce unevenly sized Web Components, as a result of the layout decisions of the author of a web page. This situation could be resolved by using more layout tags for building the index tree or by including the possibility to combine sibling nodes in a WC. Also, the semantics of several tags could be utilized in the process of fragmenting a web page. In addition, the training and update procedures could be enhanced by merging them in a single process of continuously updating the code of the WCs and re-training the system with the latest information about the WCs. Finally, although this was not the initial goal during the design of this technique, we will examine ways of utilizing it (with modifications) towards implementing a fully-automatic transcoding system for enhanced PDA browsing.

6. References

[1] C. Bouras and A. Konidaris, "Web Components: A Concept for Improving Personalization and Reducing User Perceived Latency on the World Wide Web", Proceedings of the 2nd International Conference on Internet Computing (IC2001), Las Vegas, Nevada, USA, June 25th - 28th 2001, Vol 2, pp.238-244.

[2] C. Bouras, V. Kapoulas and I. Misedakis, "A Web-page Fragmentation Technique for Personalized Browsing" (Poster Absract), ACM SAC 2004 (IDM track), March 14-17, 2004

[3] E. Hwang and Sieun Lee, "Web Surfing Assistant for Improving Web Accessibility", International Conference on Internet Computing (IC'03), Las Vegas, Nevada, USA, June 23-26, 2003.

[4] Buyukkokten, H. Garcia-Molina, A, Paepcke, 'Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones', In Proceedings of the Conference on Human Factors in Computing Systems, CHI'01, 2001.

[5] Juliana Freire, Bharat Kumar, Daniel Lieuwen, 'WebViews: Accessing Personalized Web Content and Services', Proceedings of the 10th international conference on World Wide Web, Hong Kong, 2001