Web Page Fragmentation and Content Manipulation for Constructing Personalized Portals

Ioannis Misedakis, Vaggelis Kapoulas, and Christos Bouras

Research Academic Computer Technology Institute, Riga Feraiou 61, 26221 Patras, Greece
and
Computer Engineering and Informatics Department, University of Patras, 26500 Rion, Patras, Greece
{misedaki, kapoulas, bouras}@cti.gr

Abstract. This paper presents a web page fragmentation technique, which is utilized for extracting specific parts of web pages and building personalized portals using these fragments. It 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 specific fragments of a web page in different time points. Using this technique a service provider can offer Web users a system for building personalized pages based on the content of their favorite sites. This technique, besides providing a convenient way for browsing, saves time and reduces the cost of browsing in different sites.

1 Introduction

Most web pages have a presentational structure that rarely changes, even if their content changes very often. In this structure there are areas that their content is of common thematic. We call these areas 'Web Components' or 'Web Fragments'. Web users usually show interest for only few of the thematic areas of their favorite web sites. For example, some users visit only the sports section, while others prefer to view news about politics and economy. The technique that is presented in this paper could be used for building 'personal pages' containing specific thematic areas ('Web Components') from the users' favorite sites. A software tool (working centrally as a data source for the web server of the service) analyzes web pages and fragments them in the thematic areas (Web Components) of which they are composed. Web Components (denoted WCs in the rest of the paper) are extracted from a web page by parsing the HTML code of this web page, identifying the part of the code that belongs to the particular WC and retrieving this code as an independent entity.

The concept of 'Web Components' (WC) was introduced in [1]. A 'Web surfing assistant', which utilizes a similar fragmentation technique for splitting a web page in semantic regions, is presented in [2]. The work presented in [3] and [4] investigates fragmentation's impact on Web performance. Fragmentation of web pages and manipulation (transcoding) of the fragments has been applied also in numerous systems that offer WWW services to handheld devices, such as PDAs and mobile phones ([5-7] amongst others).

2 Fragmentation Algorithm

HTML tags inside HTML files are nested, which means that the code of a web page can be represented as a tree (HTML tree or Tag tree). Extracting the Web Components (WCs) of web pages could be done by identifying and extracting some particular nodes of this tree. The fragmentation algorithm, which fragments a web page in WCs, must be able to recognize the nodes of the HTML tree that represent WCs. Most web pages use TABLE tags for their layout presentation, which lead to the decision to use the nested table structure of a web page for its fragmentation. By ignoring all the nodes of the HTML tree except the TABLE nodes, it is reduced significantly in complexity and depth. The algorithm uses this reduced tree to make the calculations for the fragmentation of the page and afterwards it can retrieve the actual content of each component by following links to the original HTML tree. The reduced trees (index trees) for two popular web sites are shown in Figure 1. Each circle marked with bold border represents a table that was chosen as a Web Component.

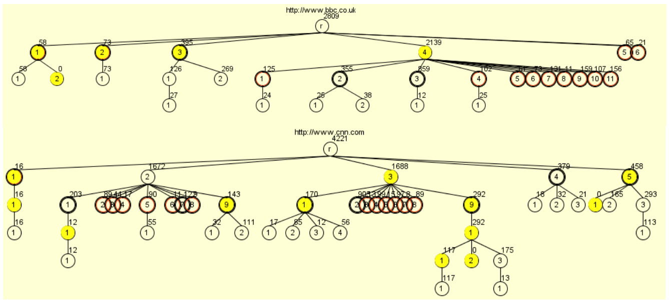

Fig. 1. Index trees for two popular web pages.

The fragmentation algorithm is used for the *web pages' analysis and fragmentation,* which includes two phases: training and update. Both in the *training* phase and the *update* phase, a software mechanism downloads the web page, parses it and fragments it into web components. The steps of the fragmentation algorithm are presented in the procedure below:

Fragmentation Algorithm

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

1) Fetch the latest instance of the web page from its respective URL
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

Note: 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 from the HTML tree and store them.

Step 1 is fairly simple. The fragmentation algorithm requests the html file of the web page from the respective URL and downloads it locally.

An HTML parser was built for the needs of step 2. It takes as input the text of the HTML file and constructs the HTML tree. In this structure all the necessary information for reconstructing the HTML file in its initial form are stored.

Step 3 of the fragmentation algorithm takes as input the HTML tree and constructs the reduced tree, which is used in step 4 for recognizing the Web Components of the page. It is used also as an index for the HTML tree and therefore it is named ‘index tree’. The algorithm starts from the root of the HTML tree and recursively traverses the whole tree. For every TABLE node that it meets, it adds a new node in the index tree. The ID of each node is the path from the root of the index tree to this node. Each child of a node has a number indicating its position relative to the other children. Starting from the root and combining these numbers we get the path and the ID of each node. Each node in the index tree has a link to its corresponding node in the HTML tree and also stores some additional information about the node, including the size of the text of this node (with and without the tags), the ID of the node, the number of images and finally the number of links contained in it.

In step 4, the fragmentation algorithm uses the index tree in order to decide how the web page will be fragmented. It recursively traverses the index tree trying to find nodes that match some particular criteria. When a node, which meets those criteria, is found, the algorithm stops traversing its children and the node is marked as a WC. This means that the whole sub-tree beneath this node is considered as a single entity that can be used by the users of the service for building their site. The children of this node are part of the WC and cannot be used standalone, since the algorithm has decided that their content is of minor importance (or too small) in order to be used as a WC. We have to note here that a node in the index tree that has been selected as a WC cannot be used directly to get the actual content of the WC. In order to achieve this, a link must be followed from the node in the index tree to its respective node in the HTML tree and from there the HTML code of the WC can be acquired in text format.

The criteria that are used for deciding if a node of the index tree (i.e. a part of the HTML file) is suitable for being used as a WC are related to the size of the content of this node and its internal structure (i.e. the number of children and descendants of this node). In its current form, the algorithm calculates the ‘size of the content’ of a node by calculating the length of the *pure text* (i.e. without the tags) that is found inside the node (future plans include to use the area occupied by the component in the web page instead of the length of the text). If node *p* meets the following criterion then it is marked as a Web Component without even examining its internal structure:

$$AverageRatio \leq Ratio_p \leq 2 * AverageRatio \tag{1}$$

$$1 \leq Ratio_p * (Number\ of\ Content\ Nodes) \leq 2 \tag{2}$$

$$Ratio_p = \frac{Pure\ Text\ Length\ in\ the\ node\ p}{Pure\ Text\ Length\ of\ the\ root\ node} \tag{3}$$

$$AverageRatio = \frac{1}{Number\ of\ Content\ Nodes} \tag{4}$$

Relation 1 (or its equivalent relation 2) expresses the intuitive criterion that a Web Component must be ‘medium’-sized, in comparison with the whole page size. $Ratio_p$

is calculated by dividing the pure text length included in node p by the text length of the entire page, giving the percentage of the node's content to the content of the whole page. This expresses the relative size of the Web component (regarding the size of the whole page). AverageRatio is the percentage of the text of the whole page that a node would have if all nodes that contain content were equally sized. This metric is used as a base for an approximation of a 'medium-sized' component. It is calculated as the inverse of the number of the content nodes of the index tree. If the size of the content (text) of a node is greater than the average size (i.e. the AverageRatio) or smaller than the double of the average size, then the node is considered 'medium-sized' and is selected as a WC.

Relation 2 could be rewritten in a more abstract form as:

$$l \leq Ratio_p * (Number\ of\ Content\ Nodes) \leq u \qquad (5)$$

where $0 \leq l \leq u \leq u_{max} = (Number\ of\ Content\ Nodes)$

The values of l and u express the lower and upper bound for the length of a node's text in order to be considered 'medium-sized' (see figure 2). Relation 5 means that if a node's text length is greater or equal than $\frac{l}{u_{\max}}$ and smaller or equal than $\frac{u}{u_{\max}}$ of the whole page text length, then this node is considered 'medium-sized' and is selected as a Web Component. By substituting l=1 and u=2 in (5) we get the criterion expressed in (2). The values of $l=1$ and $u=2$ were arbitrarily chosen, since they resulted in good fragmentation of web pages. In case we had set a value for l that was smaller than 1, then the algorithm would select nodes with text length smaller than the text length of the average node, which is already small. We chose u=2 after experimenting with several web sites and examining the fragmentation's results. However, future work plans include further testing with more web sites in order to find the 'ideal' values for the constants l and u. It has to be noticed here, that the ideal values will not be the same for each web site, since they solely depend on the web pages' structure and content.

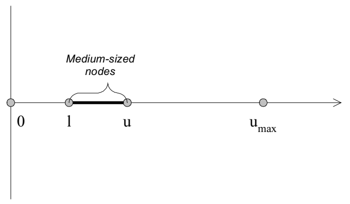

Fig. 2. Definition of 'medium-sized' nodes.

The other major criterion for fragmenting a web page is based only on the structure of the index tree. The areas that intuitively are perceived as Web Components are usually composed of more then one TABLE tags, one of which contains the main body of the Component's content, while the others are layout tags or tags with insignificant amount of content. So, when the fragmentation algorithm finds out one node of the index tree that contains *less than four children* and *less than five (in total) descendants* (not including layout nodes) it selects this node as a Web Component. This criterion helps 'refine' the results of the criterion that is based on the content's size.

Finally, if the fragmentation algorithm reaches a leaf node, it selects it as a WC, because it cannot be furthered analyzed. This can lead to selection of nodes as WCs that have very small content. But it was included, because we do not want to have content loss. Future work plans include the improvement of the fragmentation algorithm by adding the possibility to create WCs by merging neighboring nodes.

When the fragmentation algorithm finishes the traversal of the index tree, it makes some last refinements of the Web Components selections. More specifically, if it finds a Web component that is the single child of its father it selects the father as a component. This is done because probably the father of the previously selected component is a layout table tag or contains some content that is related to its child node (such as the title or author of an article).

The algorithm includes two more steps, which are used only in the update phase and will be described in its respective section.

3 Construction of Personal Pages Based on Web Components

The technique for constructing 'personal pages' based on WCs is presented in this section. Three steps are required: *Web pages' analysis and fragmentation*, *WCs selection (by the user)* and *personalized page synthesis for presentation to the user.*

3.1 Web Pages' Analysis and Fragmentation

Web pages' *analysis* and *fragmentation* is performed in two phases. The first is the *training phase*, where each web page is analyzed for a period of time. The training algorithm parses the web page many times, fragments it and stores some data from every parsing. When enough data have been gathered, the algorithm analyses them and calculates which areas of the page will be extracted as Web Components and also assigns a *unique identifier (signature)* for each one of them. The *update* phase begins when the training has been completed. This procedure fragments the web page and updates the latest instances of the Web Components that have been stored. The training and update procedures are described more thoroughly in [8].

3.1.1 Training Phase

The goal of the training phase is to provide knowledge about how to fragment a web page in Web Components. During the training phase a web page is parsed many times and its various instances are analyzed. Its outcome is a unique identifier for the WCs that are contained in the page. These identifiers (signatures) are based on the *content* and the *relative position* of the WCs and are stored in the *page index*. The signatures of WCs are utilized in the update phase, in cases where the fragmentation algorithm detects different number of WCs or the page structure has differences than the usual. We assume that changes like that do not happen during the training period.

Many factors could be used for distinguishing a WC from the rest: Its ID inside the index tree, its relative position to the other WCs, its content, its content size and others. However, finding a criterion to *uniquely* identify a WC from the others in all the page instances is a difficult task. This is done by the training procedure.

Before continuing, one important fact must be mentioned about the content of the WCs. They can be classified in three categories, based on the *changes of their content*: There are some components for which the content never changes (for example an area with links to categories of news), there are some others for which the entire content changes (for example an area with news headlines) and there is a third category of components for which some part of the content changes, while another part remains constant (for example an area with news headlines that has in the top a 'NEWS HEADLINES' title). The training phase uses the constant part of the components content for the first and third categories in order to assign a unique identifier for them, while it uses the relative position and the content size of the components for the second category.

The training phase for a web page can be split in four sub-phases:

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

During the data gathering phase, the fragmentation algorithm is activated in fixed intervals of time and the index tree for the specific page instance in that point of time is stored. The page that is analyzed is monitored for a given period during which all the updates that usually happen in it have been performed. For news portals this period is usually 24 hours. In this monitoring period k specimens of the index tree are collected, where $k = \left\lfloor \frac{Monitoring\ Period}{Sampling\ Interval} \right\rfloor$.

When the monitoring period has been completed, the content (text and images) that stayed constant is detected. The content of each WC, i.e. every word of the text inside it and the filenames of the images contained in it, is stored in a data structure named *'Content Vector' (CV)*. This structure is a characteristic of each *Web Component instance* (This means that the CVs can be different for different instances of the same WC). 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 exists in ALL the CVs. This is stored in a vector named *'Constant Content Vector' (CCV)* and is a characteristic of a WC independently of its instances in different time points.

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

Step 4 is the final phase of the training procedure. The ICVs of the WCs are assigned as their signatures in this step. The WCs that have an empty or weak ICV are assigned another kind of data structure as a signature, which is based in their relative position in the web page and their content size. The training phase produces at the end the *page index*, which is a matrix containing the ID and the signature of each WC of the page.

3.1.2 Update Phase

The update phase has many similarities with the training phase. It continuously fetches the web page, parses it and calculates (using the fragmentation algorithm) the web components of the web page. Following this, it stores the latest instances of the web components in the Web Server of the system, in order to be used by the users for their personalized portals' creation.

The fragmentation algorithm produces in step 4 the index tree of the web page instance that was fetched and marks some nodes as WCs. We assumed that during the training phase no changes happen in the page structure or in the number of the detected WCs. But generally, changes may appear during the update phase. In this case the fragmentation algorithm has to by-pass the problems caused by the changes. Step 5 of the fragmentation algorithm uses the index tree and the page index for detecting if changes appeared. This check is done by checking for differences in the ID fields 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' CV in the placeholder of the signature, since the signature is a characteristic of all the instances of a WC and a CV is a characteristic of each single instance). If no changes appear, the fragmentation algorithm continues with step 6. Otherwise, a special 'fragmentation correction algorithm' is triggered, which is presented below. It is the most complex algorithm implemented for the functioning of the system, since many situations may lead to triggering it.

Fragmentation Correction Algorithm

```
(1)If ($WC_{count}$ in the page index from training== $WC_{count}$ in the instance
page index){
  Compare the signatures contained in the page index with the Content
  Vectors contained in the instance page index*/
  (2)If (signatures match) {
       Mark for extraction the Web components based on their signatures
    }
  (2) else {
       Extract all the Web components that their CVs match with signa-
       tures 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 (or mark for extraction) the WCs based on their IDs
  }(3)
  (3) else {
       Counter++;
       (4)If (Counter<4){
           (5)If ($WC_{count}$ in the instance > $WC_{count}$ from training){
                Increase the u parameter of the fragmentation algorithm
                and recalculate the index tree and the Web Components.
            } (5)
            (5) else{
                Decrease the u parameter of the fragmentation algorithm
                and recalculate the index tree and the Web Components.
            }(5)
        }(4)
        (4)else {
            Get the initial fragmentation (with the default value of the
            u parameter). Extract all the WCs that can be extracted based
            on their CVs. Extract all the remaining WCs based on their
            order of appearance and their content size (closest match).
         }(4)
        }(3)
    }(1)
```

When the fragmentation correction algorithm finishes, the WCs are extracted, some changes are performed in their HTML code and they are stored in the Web Server of the service provider (step 6 of the fragmentation algorithm).

3.2 Personal Page Creation (by the User)

Using a web interface, the user is called to select one of the pages that have been analyzed by the system. Upon making a choice, the user is transferred to a page where all the WCs of the selected page are shown. In this page the user selects the WCs that will be used as the building blocks of his personal page. Having finished with the WCs of the particular page, the user can select another site from the initial page.

3.3 Personal Page Synthesis

A script in the web server of the service provider performs the ‘personal page synthesis’. This is done every time the user requests to see his/her personal page. This script checks the database for the user’s record and retrieves the list of the selected WCs. Then it retrieves from the filesystem of the web server the source code of each selected web component and uses it for constructing the user’s personal page.

4 Evaluation of the Technique

In order to demonstrate the amount of avoided data transfers to the users’ personal computers by using this technique, an experiment was performed. Three popular sites were selected (CNN, BBC and Yahoo) and the fragmentation technique was applied to them. They were split in their respective Web Components and the size of each component was recorded. Assuming that a user selects some web components of each site and rejects all the others, only the data of these components will be transferred to the user’s personal computer.

Table 1. Fragmentation of 3 popular web pages.

CNN		BBC		YAHOO	
Web Component ID	**Size**	**Web Component ID**	**Size**	**Web Component ID**	**Size**
'1	20084	'1	2820	'1	540
'2-1	13408	'2	1201	'2	1203
'2-2	1473	'3	14520	'3	71
'2-3	18752	'4-1	1472	'4-1-1	35
'2-4	533	'4-2	4293	'4-1-2	114
'2-5	10565	'4-3	9593	'4-1-3	35
'2-6	582	'4-4	1448	'4-1-4	114
'2-7	2642	'4-5	565	'4-2	3776
'2-8	1240	'4-6	1046	'4-3	1871
'2-9	2749	'4-7	1855	'4-4	228
'3-1	8656	'4-8	994	'4-5	114
'3-2	12787	'4-9	504	'4-6	2713
'3-3	824	'4-10	289	'4-7	631
'3-4	3456	'4-11	1436	'4-8	365
'3-5	864	'5	300	'4-9	911
'3-6	2656	'6	326	'4-10	1551
'3-7	857	*Total*	42662	'4-11	1956
'3-8	1491			'4-12	590
'3-9	4660			'4-13	3014
'4	6373			'4-14	591
'5	6338			'4-15	1637
Total	120990			'5-1	85
				'5-2	1625
				'5-3	83
				'5-4	1566
				'5-5	35
				'5-6	1024
				'5-7	35
				Total	26513

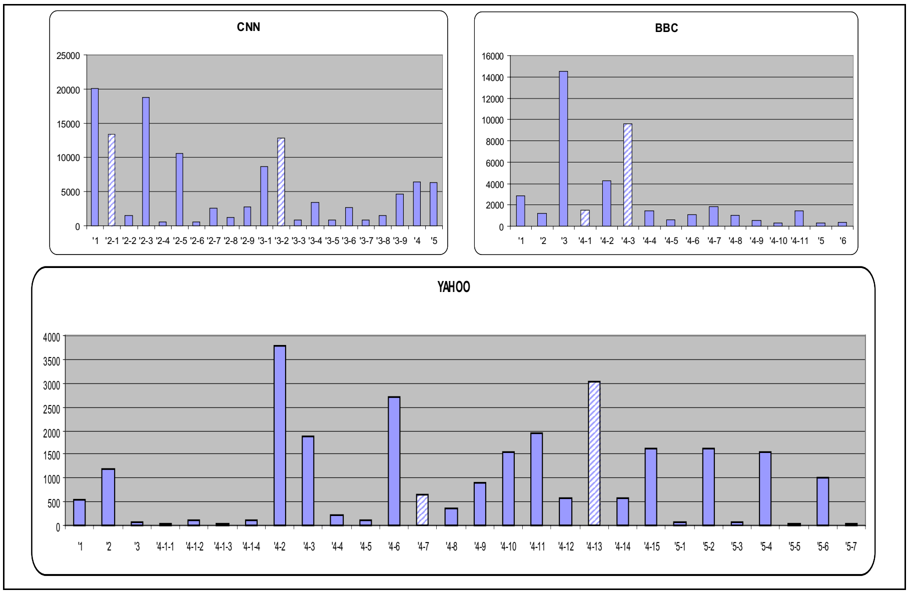

Fig. 3. Fragmentation of 3 popular web pages.

The percentage of *downloaded data (D)* and *avoided data (A)* over the whole page data size, which denote the gain from the technique, can be calculated by the formulas:

$D = \frac{\sum S_{p,k}}{\sum Total_k}$ and $A = 1 - \frac{\sum S_{p,k}}{\sum Total_k}$, where $S_{p,k}$ denotes the size of the pth component of the kth page and $Total_k$ denotes the size of the whole page.

As an example of the gain from using the fragmentation technique, let's assume that a user selects to see in a personal page only the news headlines and the 'general' links from the three sites presented above. These are included in the following Components: 2-1 and 3-2 for CNN, 4-1 and 4-3 for BBC and 4-7 and 4-13 for Yahoo (they are marked with different colour in figure 3). Substituting the respective values in the formulas presented above we get a 78% gain for the user!

Concluding, the fragmentation technique, besides the convenience of presenting in a single page all the desired information for a user, can also help towards the reduction of the data transfers to the users' PCs and increase the perceived 'speed' of the internet connection during the browsing sessions.

5 Future Work – Conclusions

The technique that is presented in this paper can be further improved. There are some cases where the fragmentation algorithm selects small leaf nodes as WCs or some areas of content are not included in any of the WCs of a page. This is a result of using only the TABLE tags for defining the page structure. In future versions of the fragmentation technique the index tree will be constructed using other tags also (TR, TD) and it will be possible to include nodes that are not children of a common ancestor in a single WC. The whole procedure could be also enhanced by merging the training phase with the update phase and by utilizing them for providing 'hints' to the fragmentation algorithm.

Concluding, in this paper we presented the concept of 'WCs' and its application in designing and implementing a software technique that can assist Web users in their browsing sessions, by presenting to them in a single web page only the parts of pages that they are interested in. Usage of this technique enhances the browsing experience, since all the information a user usually accesses in a single browsing session is gathered in the user's personal page.

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 2001, Vol 2, pp.238-244.
2. E. Hwang and Sieun Lee, "Web Surfing Assistant for Improving Web Accessibility", International Conference on Internet Computing (IC'03), Las Vegas, Nevada, USA, June 2003.
3. J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. "A publishing system for efficiently creating dynamic web content", Proceedings of the IEEE Conference on Computer Communications (INFOCOM'00), March 2000.
4. Craig E. Wills and Mikhail Mikhailov, "Studying the impact of more complete server information on Web caching", 5th International Web caching and Content delivery Workshop, Lisbon, Portugal, May 2000.
5. Masahiro Hori, Goh Kondoh, Kohichi Ono, Shin-ichi Hirose, and Sandeep Singhal. 'Annotation-based Web Content Transcoding'. In Proceedings of the 9th International World Wide Web Conference, Amsterdam, Netherlands , May 2000.
6. 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.

7. 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.
8. C. Bouras, V. Kapoulas, I. Misedakis, “Web Page Fragmentation for Personalized Portal Construction”, Proceedings of International Conference on Information Technology (ITCC 2004), Las Vegas, Nevada, USA, April 2004.