Web document analysis based on visual segmentation and page rendering

Cong Kinh Nguyen, Laurence Likforman-Sulem, Jean-Claude Moissinac, Claudie Faure and Jérémy Lardon
Telecom-ParisTech, Paris, France
{ cnguyen, likforman, moissinac, cfaure, lardon} @telecom-paristech.fr

Abstract **– This paper proposes an approach for segmenting a Web page into its semantic parts. Such analysis may be useful for adapting blog or other pages on small devices. In this approach, we take advantage of both dynamic layout after rendering and textual information. Our method segments the page into blocks and then classifies the blocks. A classification in semantic parts is performed thanks to a SVM-based machine learning approach using a set of 30 textual and visual-based features. Evaluation is conducted on a Web blog database. Results are provided for both block classification and blog segmentation into articles.**

Keywords: Internet document, block segmentation, semantic block, Web page segmentation.

1. INTRODUCTION

Web Document Analysis (WDA) plays an important role to the Web ecosystem. Not only is the WDA used to understand Web content for search engines, but also it is for restructuring a new visual representation, especially for adapting a Web page on mobile devices.

Web documents have several representations. They can be analyzed through Document Object Model (DOM) or as images displayed by a browser. Image representation is lacking textual and tag information present in Web documents. On the other hand, methods for WDA based on DOM tree obviously avoid this lack. In addition, we can take advantage of visual information defined in HyperText Markup Language (HTML) or by JavaScript and Cascading Style Sheet (CSS) associated to layout Web elements. The visual information includes positions, colors, font styles, font sizes, and borders of the elements. That's why there is a lot of research work already done concerning WDA by taking care about visual information on DOM tree. As an example, in [1], visual representation on DOM tree is used to extract the Web page's content structure. In contrast, [3] proposes an approach to understand semantic Web page structure based on features such as DOM tree structure feature (path from root to leaf), geometric features, neighboring nodes, HTML attributes (encoding visual information), and linguistic features. [5] takes into account both visual and structural features to segment a Web page. [2] is more specific and uses both content and structure to analyze HTML medical articles. They define a model of HTML medical articles, and then they recognize each part of the article following this model.

In our approach for WDA, not only we take care about visual representation, but we also consider information present within the DOM tree. This combination helps us in segmenting more conveniently the Web page. In this study, we focus on blog Web pages, but parts of our approach could be applied to other text-based Web pages.

Overview of the system is showed in Fig. 1. We first extract information contained in DOM just after rendering a page on a browser. We next construct a *properties tree*. From this tree, we extract *elementary blocks*. Features are extracted from elementary blocks and elementary blocks are semantically labeled using Support Vector Machine (SVM). For the blog case, we finally group elementary blocks into articles based on semantics. The rules and the features involved in the block segmentation and classification are defined according to the principles of web page design that are described in many books and web sites such as in [4]. Designers follow these principles when choosing page layout and typography to make content easy to read and to understand by the users.

The remainder of this paper is the following. In Section 2, we describe the page decomposition process into elementary blocks. In Section 3, elementary blocks features are presented. Such features are used for training a SVM classifier. Section 4 describes how block labels can be corrected through a post-processing and how the final segmentation into blog articles is obtained. In Section 5, we present experiments and show our results. Section 6 concludes the paper.

2. SEGMENTATION OF WEB PAGES BASED ON VISUAL INFORMATION

We first present our architecture which takes advantage of page rendering and image capture while using DOM analysis. Our architecture is easily compatible with Web pages in HTML 5 [9] and CSS 3 [10]. In fact, we do it by directly extracting the DOM information contained in a Web page when rendering the page by the Firefox browser [7]. The extracted information includes position, color, border, font family, font size, node name, and content of HTML element. The attributes extraction is performed by extra scripts in

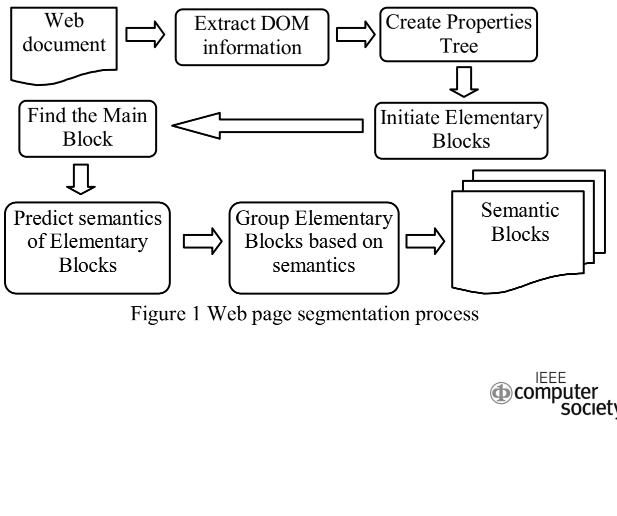

Web document
Extract DOM information
Create Properties Tree
Initiate Elementary Blocks
Find the Main Block
Predict semantics of Elementary Blocks
Group Elementary Blocks based on semantics
Semantic Blocks

Figure 1 Web page segmentation process

JavaScript automatically added into the Web page. An example about calculating the left edge coordinate of an HTML element is shown in Fig. 2. After rendering the Web page, we calculate these attributes for every node on DOM tree. Then, we construct the Properties Tree (PT) in which each node corresponds with one of DOM tree, but also contains information related to their rendering. Thus, each node of the PT includes both tag and rendering information. Such information will be used for classifying blocks into semantic labels (cf. Section 3). In the rest of this paper, if we mention a node, it means one on the PT, except we explicitly indicate one on DOM tree.

2.1. Elementary blocks of a Web page

We define an Elementary Block (EB) as a node of the PT whose content is as structurally autonomous as possible. We mainly rely on tag information. Thus, we recursively do a depth-first traversal of the PT (from the root to its children) and create an Elementary Block corresponding to a node if the node is one of the following nodes: heading node (<h>), anchor node (<a>), paragraph node (<p>), user list node (<ul>). Leaf nodes[1] and border nodes[2] are also associated to EBs. If a node is used to create an EB, we resume the traversal to its siblings. Examples of EBs are shown in Fig. 3.

2.2. Macro-elements extraction in the blog case

The objective of blog segmentation is to separate the different blog's articles and to classify the different blocks within each article. Blog's articles can be found between a header and footer and on the left or right of side bars. Thus blog's articles belong to a so-called *main content* block. The process consisting of segmenting the blog page into these macro elements (header/footer/side bar/main content) is called the principal segmentation. It can be useful for Web pages other than blog pages. EBs within footer and header and side bars are discarded. Only the EBs found in the main block will be classified by the SVM classifier.

Principal segmentation

In a blog page, its overview often can be defined as in Fig. 4: header and footer elements can be extracted by a horizontal cut considering blocks' positions saved on the PT. The others macro-elements including the main content block and sidebar(s) almost spreading through the page's height can be extracted by a vertical cut. Hence, the principal segmentation is performed when there is a change of cut direction from horizontal cut to vertical cut, and the main content block is the biggest sub-block found in the principal segmentation. In reality, it is essential to take a threshold to filter misrecognition of main block because sometimes, some elements (e.g. image for statistic[3]) might cause more than a change of cut direction.

[1] Leaf node is a node which has no child and whose node name is not "hr".

[2] Border node is a node which has a border and node's size in terms of pixel is smaller than a threshold.

[3] Wordpress [8] has a small image to calculate the statistic view for the blog.

3. ELEMENTARY BLOCK CLASSIFICATION

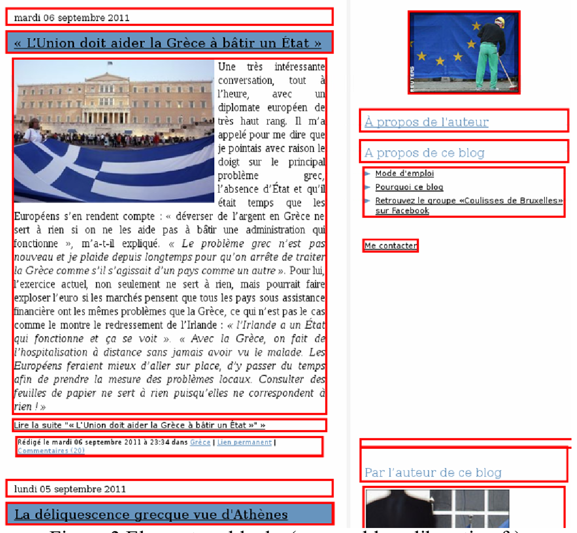
In this section, we describe how elementary blocks are labeled into their semantic function. Elementary blocks resulting from the main content extraction (cf. Section 2.1) are labeled. We consider eight labels: *title*, date, *author*, *category*, *content item*, *rest of article*, *comment*, and *other*. These labels correspond to semantic functions of elementary blocks. To classify elementary blocks according to labels, a feature vector of length 30 is extracted on each elementary block. Amongst them, we distinguish contextual features which capture information from neighboring EBs from the others.

3.1. Non-contextual features

Author feature

This feature indicates whether an EB possibly includes a name entity. We use predefined keywords commonly

```
function calcLeft(element) {
  var el = element;
  var left = 0;
  while(true) {
    var parent = el.offsetParent;
    left += el.offsetLeft;
    if (parent != null) {
      el = parent;
    } else {
      break;
    }
  }
  return left;
}
```
Figure 2 Calculation of the left edge coordinate of an HTML element


mardi 06 septembre 2011

« L'Union doit aider la Grèce à bâtir un État »

Une très intéressante conversation, tout à l'heure, avec un diplomate européen de très haut rang. Il m'a appelé pour me dire que je pointais avec raison le doigt sur le principal problème grec, l'absence d'État et qu'il était temps que les Européens s'en rendent compte : « déverser de l'argent en Grèce ne sert à rien si on ne les aide pas à bâtir une administration qui fonctionne », m'a-t-il expliqué. *« Le problème grec n'est pas nouveau et je plaide depuis longtemps pour qu'on arrête de traiter la Grèce comme s'il s'agissait d'un pays comme un autre »*. Pour lui, l'exercice actuel, non seulement ne sert à rien, mais pourrait faire exploser l'euro si les marchés pensent que tous les pays sous assistance financière ont les mêmes problèmes que la Grèce, ce qui n'est pas le cas comme le montre le redressement de l'Irlande : *« l'Irlande a un État qui fonctionne et ça se voit ». « Avec la Grèce, on fait de l'hospitalisation à distance sans jamais avoir vu le malade. Les Européens feraient mieux d'aller sur place, d'y passer du temps afin de prendre la mesure des problèmes locaux. Consulter des feuilles de papier ne sert à rien puisqu'elles ne correspondent à rien ! »*

Lire la suite "« L'Union doit aider la Grèce à bâtir un État »" »

Rédigé le mardi 06 septembre 2011 à 23:34 dans Grèce | Lien permanent | Commentaires (20)

lundi 05 septembre 2011

La déliquescence grecque vue d'Athènes

À propos de l'auteur

A propos de ce blog

- Mode d'emploi
- Pourquoi ce blog
- Retrouvez le groupe «Coulisses de Bruxelles» sur Facebook

Me contacter

Par l'auteur de ce blog

Figure 3 Elementary blocks (source: blogs.liberation.fr)

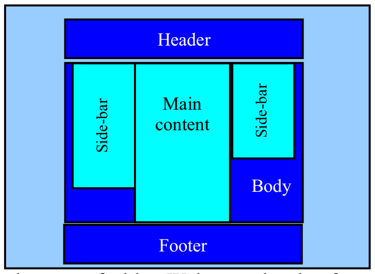


Figure 4 Macro-elements of a blog Web page: header, footer, side bars and main content

used in name patterns, e.g. *posted by*, *written by*, *created by*, *by*. These keywords are defined in a resource file; we can add a new one and remove an existed one from the resource file. Keywords belong to different languages, e.g. in French: *publié par*, *créé par*, etc. If the EB's text contains one of these patterns, the feature's value is accredited to 1, otherwise to 0.

Date feature

The date feature examines whether an EB probably contains a date by which the article was published. Hence, if the EB's text involves a date, the feature's value is assigned to 1, otherwise 0. In reality, there are blog sites displaying the published date with abbreviation forms or long formats, e.g. "*10 jan*", "*Thursday, December 15th, 2010*", etc. Therefore, we use regular expression to match a published date displayed on the browser and the one understood by machine by using two resource files containing month's abbreviation and date patterns.

Readmore feature

Long articles in a blog may be displayed through several pages. Readmore feature aims at, if an EB has a link indicating that we can pursue the link to continue reading an article. Firstly, the readmore's keywords (e.g. *continue reading*, *read the rest*, *read more*, etc.) predefined by a resource file are looked up in the EB's text. The feature's value is accredited to 1 if a keyword is found in the text, otherwise to 0.

Heading feature

The article's title is very often described by a heading node. In addition, some blog sites also use a heading node to place a published date. Heading feature examines whether an EB contains heading nodes (possibly multiple ones when they are wrapped in a paragraph node or a user list node). This feature's value is assigned to the percentage of the length of text being contained in the heading nodes and the one of all the EB's text. The percentage is used because for a real title, its value is often 1; otherwise in some cases, e.g. an EB contains a paragraph node (<p>) and this node contains some child-nodes (one of them is a heading node), its value does not equal to 1.

Link feature

The article's title, comment, rest and category almost contain an anchor node (<a>). Link feature examines if an EB contains anchor nodes (multiple ones when they are wrapped in a paragraph node, a user list node, etc.). This feature's value is set to 1 if there is an anchor node found in the EB, otherwise to 0.

Other feature

EBs that do not contain useful information should be assigned to an "other" label. The "other" feature's value is accredited to 1 if the EB's text equals to one of the keywords predefined in a resource file such as "on", "tagged", "categories:", so on. Otherwise, the feature's value is assigned to 0.

Attribute features

Eight attribute features aim at characterizing blocks which include descriptive information related to 8 functions: date, author, title, category, content, remainder, comment and other. We exploit such descriptive information embedded in CSS and HTML element's attributes. These attributes are *id='date'*, *rel='category tag'*, *class='entry'* and so on. Thus, attribute features examine whether an EB has the descriptive information. The attribute features' value is assigned to 1 if a keyword predefined in resource files is found in HTML element's attributes such as *id*, *class*, *rel*, *title*, *href*, otherwise to 0.

3.2. Contextual features

Classifying an EB may be dependent of its context. The following features include information on EBs themselves and also from their neighborhood. For each EB, there are at most four neighbors: at its left, right, above or under. Hence, there are four features for color, font family, font size, and parent level. In total, we have sixteen contextual features.

Color features

Color features indicate if the neighbors' color is identical to the EB's. For lacking neighbor(s), we assign a value corresponding with it (them) to -1. Otherwise, the value 0 is assigned if the color of neighbor is different, and 1 if it is identical. Therefore, the color features include four values for each neighbor.

Font Family features

Font family features estimate the identity between the EB's font name and its neighbors'. The font family features' value is calculated as the color features' one.

Font Size features

Font size features examine the comparison of font size between an EB and its neighbors. For lacking neighbor(s), we accredit -1 to its (their) feature(s)'s value. Otherwise, the font size features' value is respectively accredited to -0.5, or 0, or 1 for smaller, equal, or greater font size of neighbors compared to the one of the EB.

Parent Level features

Parent Level features measure the structural differences in terms of number of parents from the node used to create an EB to the root node[1] between the EB and its neighbors. The parent level features' value is calculated like the font size features'.

3.3. SVM classification

Each EB in the main content block is classified thanks to an SVM machine learning approach. We consider 20 different labels. There are 8 *principal* labels which are the labels mentioned earlier (see beginning of Section 3) and which correspond to single semantic functions. There are 12 *combined* labels which correspond to multi-function EBs. Indeed, we have noticed that EBs extracted from HTML documents may include several types of

1 The root node is the node with the tag name "HTML".

information. For instance, there are EBs which can be assigned to both "author" and "date" labels.

To classify an EB, we take LibSVM [6] as the core of prediction.

4. POST-PROCESSING AND BLOCK GROUPING INTO ARTICLES

After using SVM which predicts labels of EBs in the main content block, we launch a post-processing to correct residual errors. Block context is of high interest for logical labeling of document images. This still holds for Web documents. Thus Bayesian network or CRF (Conditional Random Field) approaches have been applied to document logical labeling, e.g. in [11] and [12]. We use a simpler ruled-based approach and consider heuristics based on sequences of consecutive EBs, and descriptive information patterns given in HTML element's attributes such as *class*, *id*, *rel*, *href*, *title*. The rules we apply are described as follows:

- For three sequential EBs, if the first and last EBs are assigned to label "content item", the middle EB which is not assigned to label "title" will be modified into label "content item".
- An EB assigned to label "rest of article" must be at the end of consecutive EBs labeled "content item". If the prediction for an EB breaks this law, the EB will be changed to label "other".
- In a blog page, if there is at least one EB assigned to "title" and having its Title Attribute value equal to 1 (cf. section 3.1), all the other assigned titles should conform to this criterion. Otherwise the "title" label is modified into a "content item" label. This rule is also applied for other labels such as "date", "author", "category" and "comment".

After post-processing for all the EBs found in the main block, we group consecutive EBs to compose an article. Apparently, a blog always has patterns to display its articles. Sometimes, there are items such as *date*, *author*, *category*, *comment*, *rest of article*, *other*, or even *content item* are missing. However, an article in order to make sense should have a title. Therefore, we group consecutive EBs based on the titles predicted by SVM. Before starting, we define a *container block* as a block which can contain an EB or consecutive EBs. First, we begin from the list of all EBs contained by the main block. Each EB initializes a *container block*. The following process groups container blocks, two by two, iteratively. Hence, container blocks include one or several labeled elementary blocks. At the end of the grouping process, each container block should include a single blog's article.

Throughout grouping, container blocks should respect the following conditions: *i)* there is at most one EB assigned to the "title" category within a container block, *ii)* all EBs assigned to the "content item" category in the container block should be vertically positioned under a "title" EB. Another condition guiding the grouping process is that the number of container blocks including a "title" EB should not decrease. A container block including a "title" EB cannot thus fuse with another block also including a "title" EB.

The grouping process follows the above constraints and starts by grouping container blocks based on the ascending distance in pixels between two blocks. At the end of the process, a number of container blocks in an article may be ungrouped with the others due to imperfect distribution of distance among parts of an article and articles (e.g. the distance between two blocks is greater than the distance between two articles). Differently speaking, there are container blocks which don't probably contain a "title" EB. Thus, to solve such problem, we group single container blocks to the nearest container block (including a title) based on top-down position of blocks.

5. EVALUATION

In this section, we evaluate the two preceding tasks, i.e. the classification of elementary blocks and the segmentation of the Web page into articles. Evaluation measure is based on precision and recall for the 8 principal categories. For classifying EBs using SVM, we take a training dataset of 46 blog pages including 7952 EBs. The test dataset consists of 49 blog pages and includes 9354 EBs. 9719 labels are assigned to these EBs since more than one label can be assigned to one EB. We show the results of EB classification in Tables 1 and 2. Table 1 shows that category "title" is the best predicted category and there are only few misclassifications. This result is important since our grouping process of EBs into articles mostly relies on the SVM classification accuracy of "title" EBs (cf. Section 4). We also compare the SVM EB labeling without and with post-processing in Table 2; we improve recall and precision by less than 1% in absolute value. But the post-processing phase benefits more to the "title" category (as seen in the Table 1, precision is increased by 4%) and thus segmenting the Web page into articles is more reliable.

For article segmentation, we take all previous test 49 blog sites and group their EBs into distinct articles. As previously, we compute precision and recall values. We compare automatically segmented with manually labeled articles without considering labels assigned to EBs therein. Results for article segmentation with and without preprocessing are shown in Table 3. These results obviously show that the article segmentation task benefits from post-processing as mentioned earlier. However, results for article segmentation are lower than for EB classification since article segmentation is a task of higher level. We have noticed that errors mostly occur for the first and last article of a Web page due to extra EBs such as navigation information.

In Fig. 5 and Fig. 6, we show two automatically segmented articles along with the EB labels predicted by SVM. One of these EB is assigned to multi-function "category + comment".

Table 1 Experimental results for classifying EBs into 8 categories

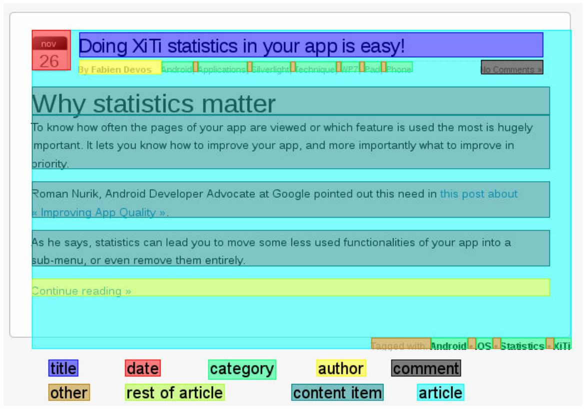
	Title	Date	Author	Category	Comment	Content Item	Rest of article	Other
# manual labels	522	501	399	895	465	5023	77	1837
Precision	95.6	91.5	76	98.5	93.7	87.1	80.9	88.9
Recall	99.6	81.2	54.1	91.1	93.3	94.4	93.5	69.4
Precision (with post.)	99.8	98	82.2	98.7	93.7	86.7	80.9	88.9
Recall (with post.)	99.6	78.2	54.6	90.9	93.1	95.5	93.5	69.4

Table 2 Average results for classifying EBs

#EBs	#manual labels	Precision	Recall	Precision (post.)	Recall (post.)
9354	9719	89	87.4	89.5	87.7

Table 3 Experimental results for segmenting blog pages into articles

# articles	Precision	Recall	Precision (post.)	Recall (post.)
522	79.8	77.5	86.4	79.5


nov 26

Doing XiTi statistics in your app is easy!

By Fabien Devos Android | Applications | Silverlight | Technique | WP7 | iPad | iPhone No Comments »

Why statistics matter

To know how often the pages of your app are viewed or which feature is used the most is hugely important. It lets you know how to improve your app, and more importantly what to improve in priority.

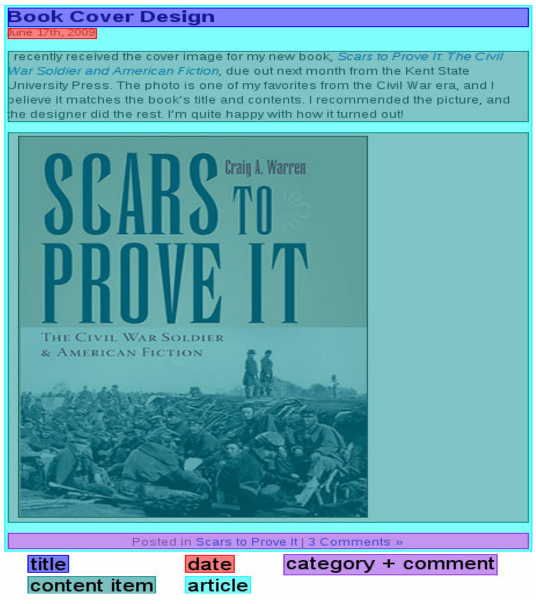
Roman Nurik, Android Developer Advocate at Google pointed out this need in this post about « Improving App Quality ».

As he says, statistics can lead you to move some less used functionalities of your app into a sub-menu, or even remove them entirely.

Continue reading »

Tagged with: Android • iOS • Statistics • XiTi

Figure 5 Results of EB classification and article segmentation on a sample blog page (source: www.expertisemobile.com)


Book Cover Design

June 17th, 2009

I recently received the cover image for my new book, *Scars to Prove It: The Civil War Soldier and American Fiction*, due out next month from the Kent State University Press. The photo is one of my favorites from the Civil War era, and I believe it matches the book's title and contents. I recommended the picture, and the designer did the rest. I'm quite happy with how it turned out!

Posted in Scars to Prove It | 3 Comments »

Figure 6 Results of EB classification (multi-function) and article segmentation on a sample blog page (source: www.1861-1865.org)

6. CONCLUSION

In this paper, we propose an approach for segmenting blog pages into articles using both visual and textual information. Our approach can take advantage of not only the up-to-date development of Web pages in terms of using HTML 5 [9] and CSS 3 [10], but also considering descriptive information wrapped in the DOM tree. However, our current dataset of core prediction could be enlarged in order to get more accurate predictions by SVM. Moreover, it is maybe possible to improve blog segmentation into articles by using article models to define special cases in which consecutive EBs will be grouped into an article if they conform to the models.

For further work, we should try to classify EBs into labels by using other methods such as neural network, hidden Markov model, or genetic model and then make a comparison among them. In addition, there is an opportunity to expand our work to automatically adapt blog pages or even general Web pages on mobile devices.

Acknowledgment: this work was performed thanks to FUI 10 project Rome Mobile.

REFERENCES

[1] D. Cai, S. Yu, J-R Wen, W-Y Ma, Extracting content structure for Web pages based on visual representation, APWeb'03, p. 406-417.

[2] J. Zou, D. Le, G. R. Thoma, Structure and content analysis for html medical articles: a hidden markov model approach, DocEng '07, ACM, p. 199-20.

[3] J. Feng, P. Haffner, M. Gilbert, A Learning Approach to Discovering Web Page Semantic Structures, ICDAR'05, pp.1055-1059.

[4] http://webstyleguide.com/wsg3/index.html

[5] A. Zhang, J. Jing, L. Kang, L. Zhang, Precise Web page segmentation based on semantic block headers detection, IDC10, p.63-68

[6] C.-C. Chang and C.-J. Lin, "LIBSVM : a library for support vector machines", ACM Transactions on Intelligent Systems and Technology, Vol. 2, Iss. 3, 2011, p. 27:1-27:27.

[7] Firefox, http://www.mozilla.org/firefox/

[8] Wordpress, http://wordpress.com/

[9] HTML 5, http://www.w3.org/TR/html5/

[10] CSS, http://www.w3.org/Style/CSS/

[11] S. Souafi-Bensafi, M. Parizeau, F. Lebourgeois, H. Emptoz Logical Labeling Using Bayesian Networks",(ICDAR 2001), p. 832-836.

[12] F. Montreuil, S. Nicolas, E. Grosicki, L. Heutte A New Hierarchical Handwritten Document Layout Extraction Based on Conditional Random Field Modeling, ICFHR 2010, p. 31-36.