WebVAT: Web Page Visualization and Analysis Tool

Yevgen Borodin, Jalal Mahmud, Asad Ahmed, and I.V. Ramakrishnan

Dept. of Computer Science
Stony Brook University
Stony Brook, NY 11794, USA
{borodin, jmahmud, asada, ram}@cs.sunysb.edu

Abstract. WebVAT is an open-source platform-independent visualization tool designed to facilitate Web page analysis. The tool, built on top of the Mozilla Web browser, exposes Mozilla's internal representation of Web pages, *Frame Tree*, reflecting HTML rendering information. Compared to HTML DOM analyzers, WebVAT provides access to a cleaner, fuller, and more accurate data structure, which contains layout information, CSS, and some types of dynamic content. WebVAT provides a framework for experiments and evaluations of algorithms over the Frame Tree. WebVAT also captures user interaction with the browser and can be used for data collection. WebVAT is a working tool actively used in the HearSay [10] project. This paper describes the architecture, design, and some of the applications of WebVAT.

1 Introduction

The expansion of the Web created a large venue for research. A big niche is taken by Web content analysis, including Web page segmentation, classification, summarization, etc. Like any other research area, Web content analysis requires tools to help with experimentation and evaluation. A number of existing tools allow viewing and editing Web pages, exploring their structure, etc.

Web page analysis often involves examining the internal structure of Web pages, usually represented by HTML DOM trees [2]. DOM trees are widely used for Web information extraction [1]. A number of software tools enable DOM inspection [9, 3]. However, a DOM tree does not specify how exactly to render a Web page, leaving the implementation to Web browsers. HTML DOM trees do not capture the layout information, unless it is explicitly specified in the HTML code. They also do not reflect changes made by Java Script or Cascading Style Sheets (CSS), limiting the information available to Web engineers and researchers.

At the same time, Web browsers (e.g. FireFox, Internet Explorer, etc.), which are perfect for rendering Web pages, do not easily expose the layout of Web pages. Some Web page segmentation algorithms use Web browser API's to obtain page layout (e.g. VIPS [12]), but we are unaware of any open-source tools that make use of visual layout.

In this paper we describe WebVAT, a tool developed specifically for Web page visualization and analysis in the framework of the HearSay [10] project. WebVAT, based on the Mozilla Web browser, provides visualization capabilities and a flexible infrastructure for Web page analysis. WebVAT enables users to analyze the structure and layout of Web pages as they would be rendered by the browser. We are actively using WebVAT for visualization and evaluation of our algorithms, data collection, capturing user interactions - all contributing to the development of the state-of-the-art non-visual Web browser, HearSay. We next present the architecture and design of WebVat in Sections 2 and Section 3, followed by some of the applications of the tool in Section 4. We close the paper with concluding remarks and future work in Section 5.

2 WebVAT Architecture

WebVAT, written in Java, is built on top of Mozilla, an open-source cross-platform Web browser. Thus, WebVAT works on a variety of platforms including Windows, Linux, and OsX. The architecture of WebVAT is shown in Figure 1.

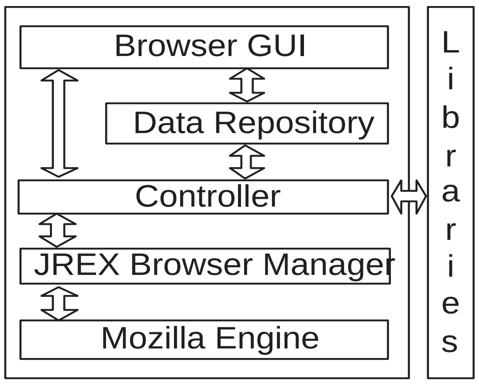


Fig. 1. WebVAT Architecture

Users interact with WebVAT through an event-driven graphical user interface provided by the Mozilla Web browser. Any user interaction with the Browser GUI is captured and can be processed, modified, and recorded by the WebVAT Controller module. The Controller also interfaces with a number of Libraries, containing various Web content analysis modules. The Browser GUI and the Controller share a common Data Repository. Besides using the already implemented functionalities of WebVAT, advanced users can easily extend the Controller and Browser GUI, and test their own algorithms.

WebVAT Controller interacts with Mozilla engine through the JREX Browser Manager [4]. JREX provides Java interface to the Mozilla engine, allowing to call the engine's APIs, define custom event handlers, etc. Mozilla engine supports standard browser functionalities, such as support of cookies, secure connection, etc. We have extended the Mozilla engine and JREX to expose and extract a *Frame Tree*, Mozilla's internal representation of a Web page, *after* the Web page has been rendered by the browser. This way, Mozilla takes care of any dynamic content, cascading style-sheets, malformed HTML, and other rendering problems. This relieves users from having to deal with heavy DOM-tree objects, while giving them fuller and more accurate information about the style and layout of Web pages. While we are using Mozilla for HTML rendering, other browsers will produce a similar data structure after rendering the same HTML pages.

When the user enters an address or navigates a link, the Controller extracts the Frame Tree of the Web page from the Mozilla engine. Figure 2 (b) shows a frame tree corresponding to the Amazon.com Web page. A $Frame\ Tree$ as a tree-like data structure that contains Web page content, along with its 2-D coordinates and formatting information, that specifies how the Web page has to be rendered on the screen. Frame coordinates refer to the *upper-left corners* of the corresponding Web page segments displayed in the browser, independent of the screen resolution or the size of the browser window.

A frame tree is composed of nested $frames$[1], so that the entire page is a root frame, containing other nested frames down to the smallest individual objects on the page. The browser window in Figure 2 (a) shows the Amazon.com Web page with some of the frames highlighted. The corresponding frame tree nodes are selected in windows (b) and (c). The frame-trees are partially expanded to demonstrate the types of frames. We distinguish between the following classes of frames: text, links, images, image-links, form-elements, XHTML, and non-leaf frames. We next describe the design of WebVAT.

3 WebVAT Design

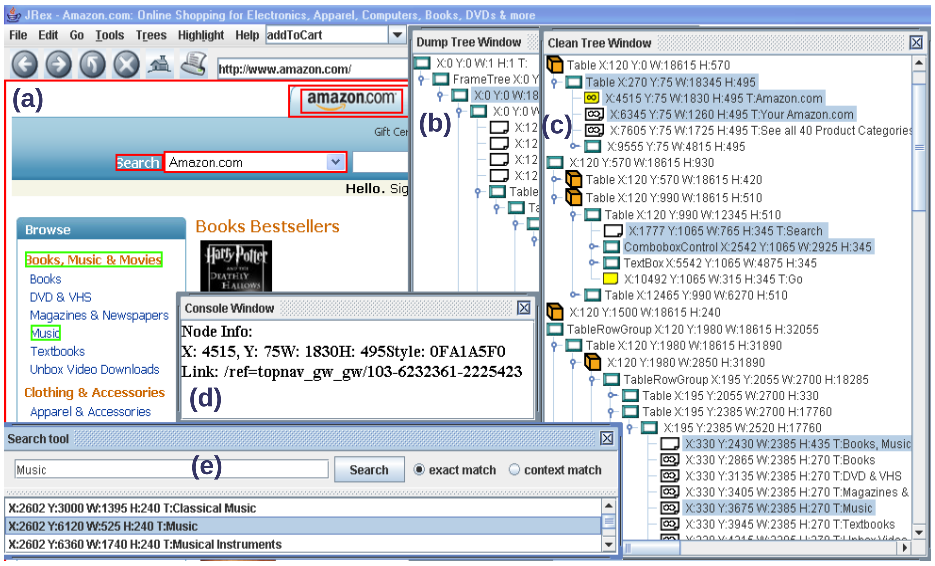
WebVAT is designed around the Mozilla Web browser interface, which displays the browser window with a standard menu extended with *Tools*, *Trees*, and *Highlight* (see Figure 2).

The *Tree* menu contains the list of all frame-tree windows that can be displayed on the screen. Different frame-tree windows can be used to visualize the results of experimental algorithms. For example, Figure 2 (b) shows the original frame tree produced by the Mozilla engine, while window (c) shows a frame tree that was processed and segmented into blocks (3-D icons) by our geometrical clustering algorithm [6].

With minimal code changes, WebVAT can support any reasonable number of frame-tree windows, synchronized by the observer handler (part of Browser GUI module). Selecting any node in any tree also selects the corresponding nodes in all other active frame-tree windows, and highlights the corresponding frame in the browser window, as can be seen in Figure 2 (a), (b), and (c). The *Highlight* menu items give additional control over highlighting functionality by allowing to clear highlighting, use different colors to highlight frames, etc. The experimental algorithms executed by WebVAT can activate frame-tree windows and highlight frames to visualize the results.

The *Tools* menu contains a growing number of useful tools. Among them there are: search, Figure 2 (e), which allows to find frames by the contained text, or can use other experimental search algorithms; console window that can

[1] Note, this is different from HTML frames.



```
X:0 Y:0 W:1 H:1 T:
FrameTree X:0 Y
  X:0 Y:0 W:18
    X:0 Y:0 W
      X:12
      X:12
      X:12
      X:12
      Table
        Ta
```

```
Table X:120 Y:0 W:18615 H:570
  Table X:270 Y:75 W:18345 H:495
    X:4515 Y:75 W:1830 H:495 T:Amazon.com
    X:6345 Y:75 W:1260 H:495 T:Your Amazon.com
    X:7605 Y:75 W:1725 H:495 T:See all 40 Product Categories
  X:9555 Y:75 W:4815 H:495
X:120 Y:570 W:18615 H:930
  Table X:120 Y:570 W:18615 H:420
  Table X:120 Y:990 W:18615 H:510
    Table X:120 Y:990 W:12345 H:510
      X:1777 Y:1065 W:765 H:345 T:Search
      ComboboxControl X:2542 Y:1065 W:2925 H:345
      TextBox X:5542 Y:1065 W:4875 H:345
      X:10492 Y:1065 W:315 H:345 T:Go
    Table X:12465 Y:990 W:6270 H:510
X:120 Y:1500 W:18615 H:240
TableRowGroup X:120 Y:1980 W:18615 H:32055
  Table X:120 Y:1980 W:18615 H:31890
    X:120 Y:1980 W:2850 H:31890
      TableRowGroup X:195 Y:2055 W:2700 H:18285
        Table X:195 Y:2055 W:2700 H:330
        Table X:195 Y:2385 W:2700 H:17760
          X:195 Y:2385 W:2520 H:17760
            X:330 Y:2430 W:2385 H:435 T:Books, Music
            X:330 Y:2865 W:2385 H:270 T:Books
            X:330 Y:3135 W:2385 H:270 T:DVD & VHS
            X:330 Y:3405 W:2385 H:270 T:Magazines &
            X:330 Y:3675 W:2385 H:270 T:Music
            X:330 Y:3945 W:2385 H:270 T:Textbooks
```

```
Node Info:
X: 4515, Y: 75W: 1830H: 495Style: 0FA1A5F0
Link: /ref=topnav_gw_gw/103-6232361-2225423
```

```
X:2602 Y:3000 W:1395 H:240 T:Classical Music
X:2602 Y:6120 W:525 H:240 T:Music
X:2602 Y:6360 W:1740 H:240 T:Musical Instruments
```

Fig. 2. WebVAT in action

be used for any output, Figure 2 (d); evaluation tool to record questionnaire answers, user-evaluation results, etc.; and, a data collection tool that allows to save HTML pages and the corresponding XML frame trees with selected frames. The data collection tool can also save sequence of pages, recording followed links, and the action labels (e.g.: *addToCart*), which can be selected from the combo box in the menu panel of the main window in Figure 2 (a).

4 WebVAT Application: The HearSay Experience

WebVAT can put to use in a number of applications. In this section, we describe its role in the research and development of the HearSay non-visual Web browser [10].

WebVAT was used to verify the correctness of the Frame Trees while we were modifying the Mozilla engine code. The HearSay browser is also based on Mozilla; HearSay uses a number of algorithms and techniques to clean the frame trees, analyze their content, and convert them into audible dialogs. WebVAT helped verify all of the algorithms used in HearSay.

The frame-tree window in Figure 2 (c) displays the results of our Web page segmentation algorithm [6], which identifies geometrically aligned blocks as semantic clusters of information (marked as 3-D blocks). We are currently working on expanding our partitioning algorithm to find repeating patterns within the blocks [8].

We also used WebVAT as a *data collection* tool. The participants were asked to identify and select some links and the information pertaining to the same topic around them in a number of Web pages. They were also asked to identify the information relevant to the links on the pages, to which the links were pointing. The data was, then, used to test our context collection algorithm. The same data was used in training an SVM-based statistical model to identify relevant information in Web pages while following links from one page to another [5, 6]. WebVAT helped us visualize and evaluate the results of the algorithms. We are now using WebVAT to construct a process model for Web transactions [11].

5 Conclusion and Future Work

In this paper we described the architecture, design, and applications of our Web content visualization and analysis tool, WebVAT. An open-source version of WebVAT will be soon publicly released with the HearSay Web browser (www.cs.sunysb.edu/∼hearsay). We identify several directions to further enhance this tool.

We used WebVAT to collect the data for off-line training of statistical models. It may be possible to integrate different machine learning modules with WebVAT to train statistical models online, while using the tool. For example, we plan to use WebVAT to collect data to train Bayesian models for transactional concept detection such taxonomy, search result etc. WebVAT can also be enhanced to create ontologies – the knowledge underlying the semantic Web. Users will be able to highlight sections of Web pages, or nodes of the frame tree, and specify the corresponding concept name. This will facilitate learning ontologies from examples.

References

1. S. Chakrabarti. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *In Proceedings of WWW 2001*, 2001.
2. `http://www.w3.org/DOM/DOMTR`.
3. `http://www.dubbeldam.com/DOMSpy.html`.
4. `http://jrex.mozdev.org/`.
5. J. Mahmud, Y. Borodin, D. Das, and I. Ramakrishnan. Combating information overload in non-visual web access using context. In *IUI*, 2007. Short paper.
6. J. Mahmud, Y. Borodin, and I. Ramakrishnan. Csurf: A context-driven non-visual web-browser. In *Proceedings of WWW (To Appear)*, 2007.
7. S. Mukherjee, G. Yang, and I. Ramakrishnan. Automatic annotation of content-rich html documents: Structural and semantic analysis. In *Intl. Semantic Web Conf. (ISWC)*, 2003.
8. `http://www.sharewareconnection.com/pagespy.htm`.
9. I. Ramakrishnan, A. Stent, and G. Yang. Hearsay: Enabling audio browsing on hypertext content. In *WWW*, 2004.
10. Z. Sun, J. Mahmud, S. Mukherjee, and I. V. Ramakrishnan. Model-directed web transactions under constrained modalities. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 447–456, 2006.
11. S. Yu, D. Cai, J.-R. Wen, and W.-Y. Ma. Improving pseudo-relevance feedback in web information retrieval using web page segnmentation. In *WWW*, 2003.