Efficient Text Content Extraction and Browsing of WWW Documents using the Abstract Text Viewer

Richard H. Fowler[1], Yavuz Tor, David Navarro, and Laura M. Grabowski
Department of Computer Science
University of Texas – Pan American
Edinburg, TX 78539
[1]fowler@panam.edu

Abstract

The Abstract Text Viewer (ATV) is an integrated suite of text reading tools for electronic documents designed to increase efficiency and effectiveness of content extraction. ATV reads a HTML formatted document to create more abstract representations, such as a heading structure for overviews. The system uses both well-known techniques for text representation and novel display and content extraction techniques. Initially, documents are displayed with an overview + detail model. The detail window displays the entire document text, possibly highlighting keywords, and an overview window displays document's headings together with a semantic summary of each section based on keyword extraction. The system provides additional methods to improve readability, such as fish-eye view, zooming, and highlighting.

1. Introduction

Reading electronically presented text has become as ubiquitous as the computer itself in research and the workplace. Electronic documents are easy to distribute by the support of Internet technology and are increasingly widely used to replace paper documents. Electronic documents facilitate management in terms of classification, reference, indexing, and searching. Searching, in particular, can be improved dramatically with appropriate indexing and retrieval method enabling users to reach the information they seek in seconds.

The need for electronic text-reading tools, designed around principles of usability, gains its importance in meeting the electronic reading needs of a variety of readers. Semantics of the document plays a significant role in this regard, since the interface should represent well and be consistent with the semantics of the document.

The standard in navigation when working with two-dimensional data is scroll bars. Having a "thumb" to represent where the view currently is in relation to the entire document. In newer operating systems the thumb itself has also gone to represent the size of view in relation to the complete text. Though this is intuitive, it adds to the overhead of navigating in a large document. This is caused by the users' lose of orientation in the complete document when changing focus to manage the scroll bar. Another problem with this approach is that in working with the scroll bars themselves in very long documents, the thumb becomes almost oversensitive to any movement. For example, moving the thumb down a few pixels might cause a large document to scroll down several pages.

Another approach in navigation that attempts to allow the user to go where he or she needs in a fast and comfortable way is rate-based scrolling. A problem with rate-based scrolling is disorientation of the user. The text blurring at high speeds causes the user to loose track of where they currently are, making it more difficult to get to the desired point in the document.

Hornbæk et. al [4] tested three interfaces, linear, fisheye, and overview+detail, for reading electronic documents. 20 subjects read scientific documents with each interface and were tested after the end of the reading. In their experiment, Hornbæk et al. compared how subjects' reading activity was supported by a linear, fisheye, and an overview+detail window. They analyzed usability differences by the grades of the end-of-reading test, satisfaction and preference data, and by a log of the subjects' interaction with the interfaces. Based on the test results, the overview+detail interface provided better effectiveness and satisfaction scores, while the fisheye view was the most efficient. All subjects chose overview+detail interface as their preference.

Donskoy and Kaptelinin [5] also compared interface type for usability. They studied the role of animation in visualization with scroll bars, zoom, and fisheye view. Fisheye view with animation yielded better performance than without animation, and scrollbars and

zoom view were slower. Subjects preferred the animated versions of these techniques over the "no animation" forms.

Zooming can play an important role in readability of text documents. It widens the information space and provides better access to details of subject of interest. Zooming has been used as the fundamental interaction method in several interfaces ([6],[7]). In their paper, Hornbæk et al [8] list research prototypes of zoomable interfaces that include interfaces for storytelling (Druin et al.[9]), Web browsing (Hightower et al. [10]), and browsing of images (Combs & Bederson [11]).

As Hornbæk et al. note, three benefits of overview+detail interfaces can be observed. First, Navigation is more efficient, since the user can navigate by using the overview window [12]. Second, overview windows help users keep track of where they are in the information space [13], and third, the overview gives the reader a feeling of control [14]. A drawback of overview+detail interface is mentioned in Hornbæk et al. in that "spatially indirect relation between overview and detail windows might strain memory and increase the time used for visual search." For this reason, it is good practice to keep overview and detail windows tightly coupled. In their experiments, Hornbæk et al. observed that about 80% of the subjects preferred overview interface combined with zoomable interface.

2. Abstract Text Viewer

The *Abstract Text Viewer (ATV)* provides functionalities addressing problems that users of electronic documents might have in reading and managing text. ATV's combination of text viewing and abstraction tools is unique. The following tools (or functionalities) are provided in ATV.

Overview + Detail, addresses both tracking and focusing problems readers might face. This is provided, as usual, in two windows, one for overview and one for detail. However, there are slight changes how this method is applied. First, the overview window is not provided in a linear structure, but instead the context is given as headings and paragraphs, as well as structured in a tree view. Coordinated with the overview window, the detail window provides a fish-eye view (Furnas[2]) applied such that the parts that are visible in the overview window are bigger in size than the ones are not. Manual operations are also allowed for convenience, since the reader may want to keep uninteresting parts smaller.

A second system coordinates hyperlinks to maintain easy navigation through the documents. This lets readers see the referenced document quickly. However, readers may also need some mechanism to go back and forth between those two documents. For this function ATV provides two tools, Go-Forward and Go-Backward (as in a browser), to increase the efficiency of the navigation.

For most cases readers are searching for specific information. This could be the total information contained by the document or information that is covered in a small part of the document. For the second case, readers might need some tools to direct them to the part that contains the information they are looking for. ATV provides a highlighting mechanism for this purpose in which the reader enters the words that might be a part of the information search. ATV, then, highlights those words that are entered in the detail window.

Zooming also gains importance for some readers that have preferences of the character sizes, for example some readers might like bigger fonts, while others prefer smaller. Zooming tools (in and out) are provided by ATV, to enable readers to change the font size.

A simple mechanism of showing paragraphs that are most likely representative document content is provided. While the document is being initially parsed, a term vector for the document is constructed, later used to facilitate retrieval of specific information in the document. Additionally, a term vector for each paragraph is constructed. Using the paragraph level information, a coloring scheme is applied on the detail window based on the relevancies of those paragraphs to the complete document content.

2.1 ATV's Semantic and Visual Structures

The system's structure can be separated into two parts: semantic and visual. Semantic structure is formed by parsing the document and building a term-vector representation of the document and each of its paragraphs. Each of the systems specific parsers is used to parse specific MIME types, and the parser is chosen at run-time, based on the MIME type of the requested document.

The document structure, constructed by the parser, consists of an array of words that occur in the document, and a list of paragraphs and links based on the index positions of words in this word array. Having this structure constructed, ATV can build the visual views. There are two views present in ATV: overview and detail. Overview detail constructs a tree structure

based on the heading types of the *Paragraph*s, listed in the *Document*. Detail window is more complicated, since most of the provided tools are aimed to work in this view. *ParagraphView* is another class constructed by ATV that encapsulates the viewing information for the *Paragraph*.

2.2. ATV Architecture

Initially, ATV loads s the font preferences that were saved from the last execution. Then a document is selected to open, and the document's MIME type is determined and an appropriate parser is invoked. The parser component then loads the document and parses it based on its MIME type..

ATV sends messages to both overview and detail views display the document based on the parse structure, as detailed below. A snapshot of ATV on work, is given in Figure 1. For the fisheye view in the detail view, each of the paragraphs, whose appearance is controlled by *ParagraphView*, has two states of vision: open or closed. In open state, the paragraph is painted in normal fonts to enhance the reader's efficiency, while the closed state is just painted with small character sizes, to provide reader a bird's eye view of the uninteresting part. All these states of the paragraphs are controlled by the overview window that appears on the left. Overview window shows the titles and headings, and normal paragraphs in a tree structure, so the reader can easily see the structure of the document, while keeping track of his current position in the document. All the titles or paragraphs that are visible in the overview window are in open state in the detail window. This provides the reader a better control on the document and enables him to decide which parts are really interesting to him. The reader can also open a closed paragraph by just clicking on it in the detail window, however it does not affect the overview window, to give some freedom to the reader.

2.2.1. HTML Parser. HTML parser is the only parser implemented that has the *CParser* interface. All HTML documents opened with this parser have HTML-structured code for a document. Not all tags are parsed by this parser, but only the ones that have useful structural information about the document, and also the ones that convey link information to other documents.

Structural parts that are considered to be useful are the header tags (*<h1>*, *<h2>*, …, *<h6>*) and the paragraph tag (*<p>*). Link information is captured by analyzing *<a>* tag and all the link information that the document has are preserved. This enables on-text linkage in the detail window. In addition to those tags, image tag (*<img>*) is also handled and all images are replaced by "<img>" automatically to give user an idea about how the original document has been organized. This also lets the user take links on-text that does not have any word, but only image(s).

2.2.2. Overview Window. The Overview window consists of a tab control that is used to change the view between "*Structure*" and "*Links*" tabs. The structure window is designed to show a structural summary of the document. nce current version of ATV works with HTML, the structural parts that are considered are header tags (h1, h2, …, h6) and the paragraph tag (p). These tags are assumed to be the structural parts that convey information about how the document is organized. All these parts are inserted into a tree view in the following way:

For all headers, the priority order is h1, h2, …, h6.

Paragraphs are the least-priority units.

For any structural unit,

If the previous unit has a higher priority than this, it is inserted into the tree as the previous one's child.

If the previous unit has the same priority as this, new one is inserted into the tree as a sibling of the previous one.

If a previous unit has a lower priority, then the hierarchy is traversed bottom-up to the root until a node with higher priority is found. The new node is inserted as the child of this.

After the tree is built by this method, only the first-level nodes of the tree are visible. Thus, the detail window is updated such that only the visible structural units are in open state. When reader opens a tree node to see its children, the detail window is updated again to reflect the changes. This provides better control on the document.

Attached to each node is an image that expresses what structural unit that is. This allows user to know the real organization of the document thoroughly. Figure 1 shows a screenshot that shows how this is all organized in the overview window.

2.2.3. Links. This window gives information about the document's linkage information. All links that exist in the document are listed in alphabetical order of their URL. If the reader wants to see the links only, this is the place to look at. Redundant links to the same

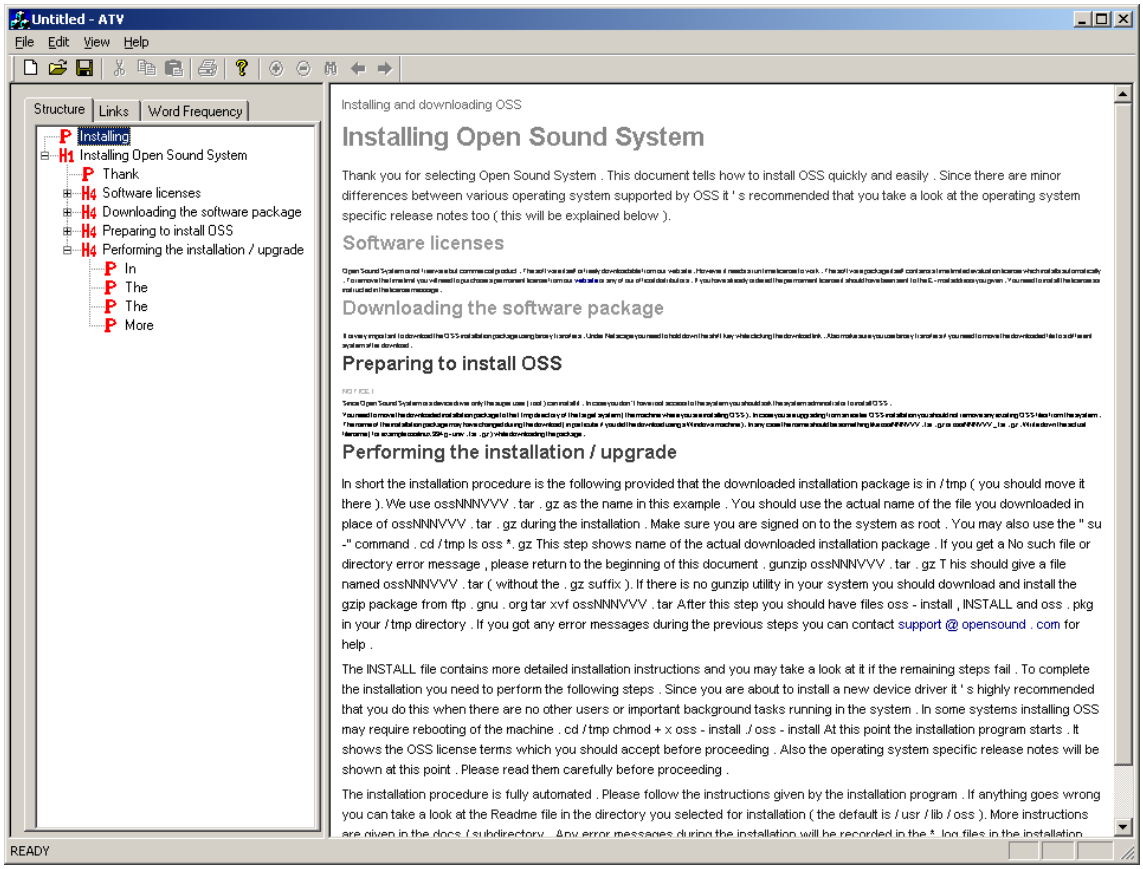

Figure 1: Structure of the document is shown in the tree view in the overview window.

document can be easily identified here, since they would appear adjacent to each other in the list. This window, also, controls the detail window by allowing the reader open a document that is in the list by just double-clicking on it.

The procedure to open a new document is the same for all instances. First, MIME type of the new document is captured. If the current parser cannot parse the document of this MIME type, then an appropriate parser is constructed. At this point in the research, only implemented parser is HTML, so if a document cannot be opened with HTML parser, then a blank page is the result. Reader can *go back* to see the last document. Second, the location of the document is determined. There are two access options; Internet access or local access. For Internet access, a request is sent to the server and the data sent by the server, as response, is copied to the buffer. If the document is to be accessed locally, then the file is opened and contents are copied to the parser's buffer. Then, parser starts to parse the document that resists in its buffer. For both access types, the current document's location is used as context, so that relative links are enabled in documents.

2.2.4. WordFreq. The wordfreq (short for "word frequency") is part of the ATV tool program to aid the user in getting an "insight" of a Web domain before reading it. A small java applet will crawl a domain, parse and stem [15] each word and add them to a simple Microsoft Access Database, the database keeps count on the number of times that word (stemmed version of it) appeared. Then wordfreq uses the text in the current page compares the number of times each word appeared in the domain vs. users thresholds. (Word freq will discard all "stop words"). Also to ensure efficiency the words are stored in a CStringList class and the alphabetized afterwards redundant words are removed. This way only one connection used to the Access Database using OBDC then the information is sent to the detail window.

Word Freq also can show the list of words and frequency on the left form view that was within the threshold requirements of the whole domain. And it shows the most used word in the UTPA Computer Science domain is "comput". The stem of Computers, computer, Computing, etc. This also aids if in case the

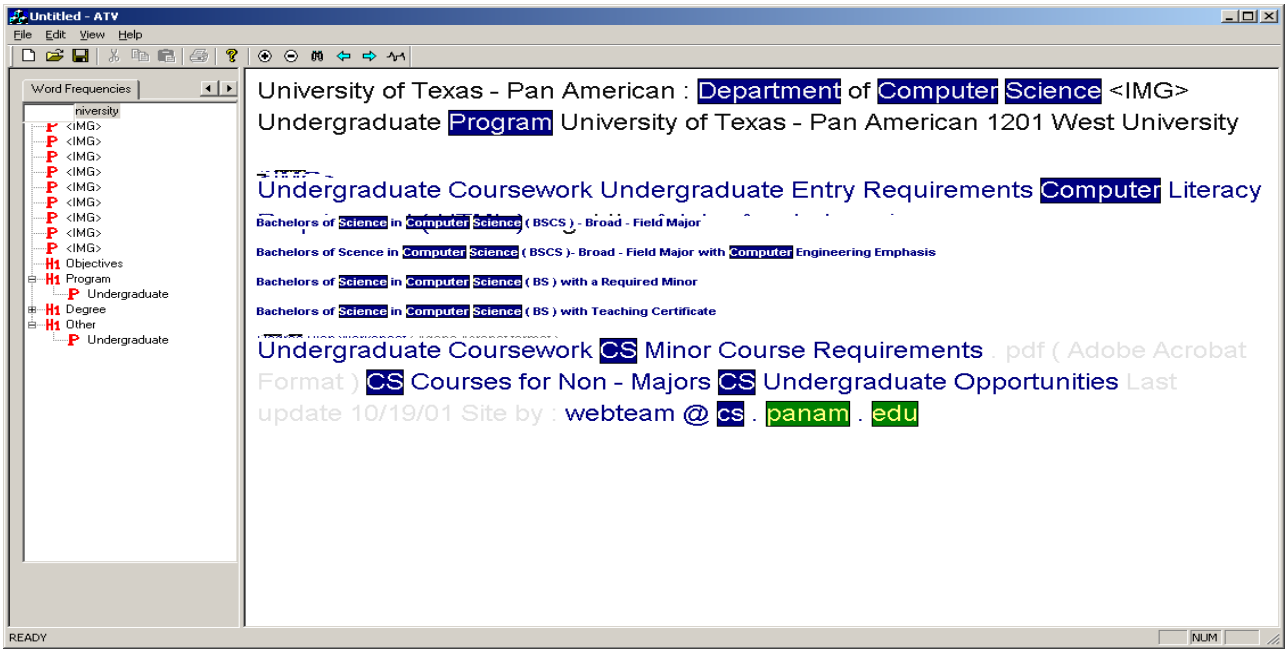

Figure 2: Two thresholds applied for a search. Color, here shade, represents relevance.

page you are looking at does not have by random chance the most repeated used words. Currently, Word Freq supports two user-defined thresholds. Threshold 1 will first configure the view to show the text in a negative form, with a dark blue background. Threshold 2 will show the text in a negative form too but with a dark green background. This dual-threshold mechanism is depicted in Figure 2.

2.2.5. Detail Window. This is the window in which the user actually reads the document. Therefore, this is the main field that efforts had been focused on. A fisheye view, coordinated with the overview window's structure tab, is applied to this window. So, the reader can control the parts that he wants to read. This window basically consists of *ParagraphView*s and preferences that apply to them. Each *ParagraphView* gets its internal data from corresponding *Paragraph*, and paints the text on the window based on this data and applied preferences. Main characteristics can be listed as follows:

The more related to document, the darker. The darkness of the text color is specified by the relevancy of the corresponding paragraph to the whole document. The darker text color gains more attention than the lighter one. This knowledge is used to gain reader's attention more to the relevant paragraphs, so he can find more related information without searching for line by line. This is decided by the following function:

$$D(p) = \frac{R(p,d) - \frac{\sum_{p \in d} R(p,d)}{n}}{\sqrt{\sum_{p \in d} \left[R(p,d) - \frac{\sum_{p \in d} R(p,d)}{n} \right]^2}}$$

where R(p,d) is the cosine similarity of term vectors of paragraph p and the document d, and *n* is the number of paragraphs in the document. D(p') analyzes the number of steps that the relevance value R(p', d) is away from the mean relevance value. Each step is equal to the standard deviation of the relevancies of paragraphs to the containing document. This is a naive statistical approach to define the relevance of the paragraph to its document, and not intended to provide the best solution. Based on D(p) value the color of the text is specified.

If the value is negative,

Color(p) = 90 – 15 * D(p)

If positive,

Color(p) = 64 – 10 * D(p)

This equation is used to calculate the color value. The actual color is obtained by assigning the same color value to each of three components of a color (red, green, and blue). This makes more relevant paragraphs darker and less relevant paragraphs lighter.

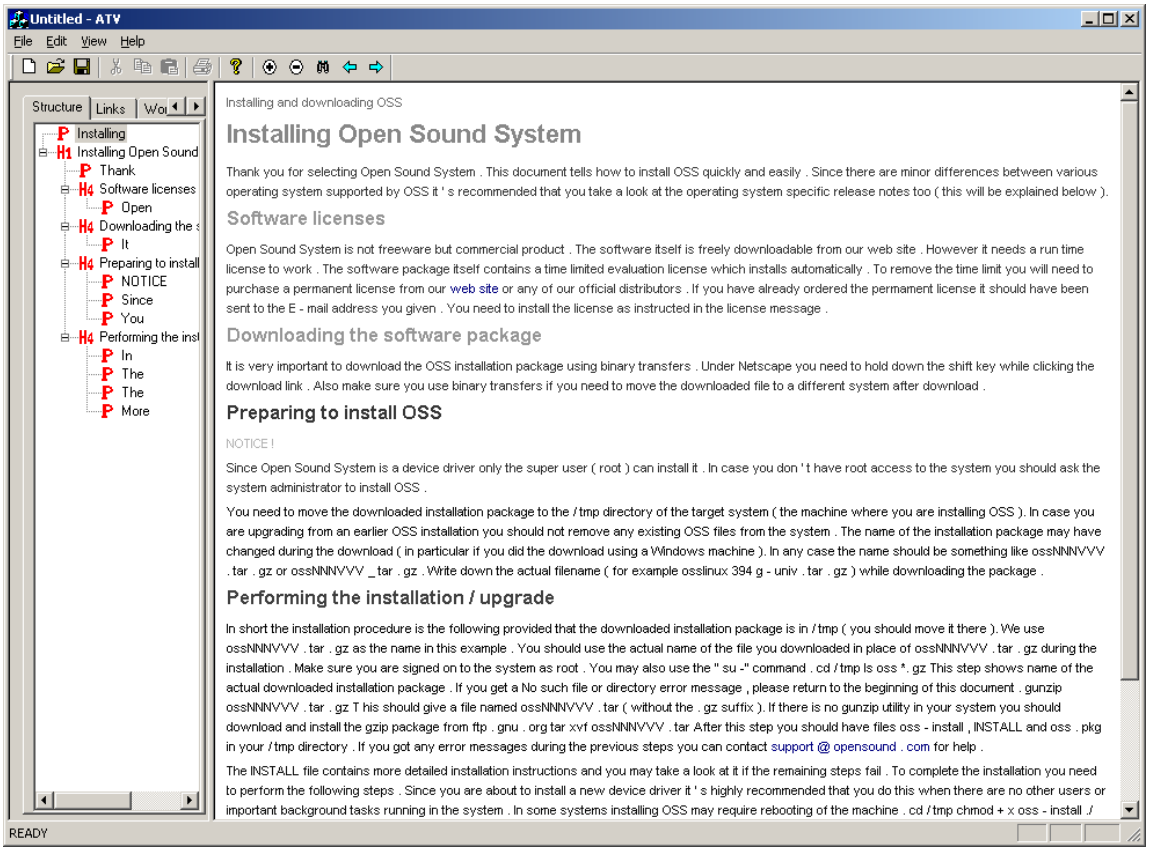

Figure 3: The more relevant paragraphs to the document are painted in darker color. Since this is an installation manual document, the first paragraphs, which explain how to download the file and the copyright information, are less relevant then the last ones, which actually explains the steps for installation.

2.2.6. Searching for occurrences of words. The detail window provides a highlighting mechanism to search for specified words. This enables the reader to find specific information he is looking for, easily. It also conveys some information about the relevance of the document to information need of reader. After user clicks OK button in the dialog, first thing is to parse words in the input string. Then, each word is stemmed by an algorithm that implements Porter's suffix stripping method [15]. The last thing is to send these stems to each of the *ParagraphView*s, where all occurrences are highlighted.

2.2.7. On-context linkage. The linkage structure of the document is preserved in a way that anchors of the HTML references and references themselves are tightly coupled together. Anchors are painted in blue text-color, so that the reader can easily see and take the links. When the cursor points to an anchor word, the cursor is changed to a hand cursor, as usual in all browsers, and the location of the reference is displayed in the status bar. If the user clicks on an anchor word, the location displayed in the status bar is opened. If the click is on a non-anchor word, then the paragraph's state is changed. If it is open, closes; or if it is closed, it opens.

This feature comes with the necessity of providing backward and forward links. If the reader opens a document accidentally and wants to go back, then he can use File → Go Bacward and File → Go Forward menu buttons, or arrows in the toolbar.

3. Future Work

The abstract Text viewer currently lacks the ability to save the view in original format. If the MIME is html we wish to add the capability to save abstract text in html and to have the ability to view it in a web browser. For wordfreq and add a sub-category for the user to include synonyms when deciding on thresholds.

The current domain crawler only uses one word at a time, configuring it to use two words would increase the size of the database dramatically with the tradeoff being that the user could see related words ranked higher. Other extensions and more detail of the current version are available [16].

4. Acknowledgements

This work was supported by the University of Texas – Pan American Computing and Information Technology Center and NASA Grant NAG 9-1169 to the first author.

5. References

[1] Muter, P. & Maurutto, P. (1991). *Reading and skimming from computer screens: The paperless office revisited*. Behavior and Information Technology, 10(4), 257-266

[2] Furnas, G. W. (1981/1999). The FISHEYE view:A new look at structured files. Readings in Information Visualization, ed. Card, S. K., Mackinlay, J. D. &Schneiderman, San Diego, CA: Academic Press, 311-330.

[3] Igarashi, Takeo, & Hinckley, Ken. *Speed-dependent Automatic Zooming for Browsing Large Documents*.CHI Letters 2,2 P.139-148

[4] Hornbæk, K. & Frokjær, E. *Reading of Electronic Documents: The Usability of Linear, Fisheye, and Overview+Detail Interfaces*. CHI 2001 P. 293 – 300

[5] Donskoy, M. & Kaptelinin, V. *Window navigation with and without animation: a comparison of scroll bars, zoom, and fisheye view*. CHI 97

[6] Bederson, B. & Hollan, J. D. (1994/1999). *Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics*. Readings In Information Visualization. Card, S. K., Mackinlay, J. D. &Schneiderman, San Diego, CA: Academic Press (1999)

[7] Bederson, B., Meyer, J. & Good, L. (2000). *Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java*. Proceedings of the 13th annual ACM symposium on User Interface Software and Technology, 2(2), 171-180.

[8] Hornbæk, K., Bederson, B. B., & Plaisant, C. 2002. *Navigation patterns and usability of zoomable user interfaces with and without an overview*. ACM Transactions on Computer-Human Interaction. Volume 9, Issue 4. (2002), 362-389.

[9] Druin, A., Stewart, J., Proft, D., Bederson, B., AND Hollan, J. D. 1997. *KidPad: A design Collaboration between children, technologists, and educators.* CHI 97, Atlanta. S. Pemperton, Ed. ACM Press, New York, N.Y., 463–470.

[10] Hightower, R. R., Ring, L. T., Helfman, J. I., Bederson, B. B., & Hollan, J. D. 1998. *Graphical multiscale Web histories: A study of PadPrints*. In Proceedings of the Ninth ACM Conference on Hypertext. ACM Press, New York, N.Y., 58-65.

[11] Combs, T. & Bederson, B. B. 1999. *Does zooming improve image browsing?* In Proceedings of the ACM Conference on Digital Libraries. ACM Press, New York, N.Y., 130-137.

[12] Beard, D.B. & Walker, J. Q. 1990. *Navigational techniques to improve the display of large two-dimensional spaces*. Behav. Inform. Techn. 9, 6, 451-466

[13] Plaisant, C., Carr, D., & Schneiderman, B. 1995. *Image browsers: Taxonomy, guidelines, and informal specifications*. IEEE Softw. 12,2, 21-32.

[14] Schneiderman, B. 1998. *Designing the User Interface*. Addison-Wesley, Reading, Mass.

[15] Porter, M.F., 1980, *An algorithm for suffix stripping,* Program, 14(3) :130-137.

[16] Fowler, R.H., Tor, Y., Navarro, D., & Grabowski, L.M. 2003. *The Abstract Text Viewer: A tool suite for content extraction and organization of Web documents*. Technical Report CS-03-28, Department of Computer Science, University of Texas – Pan American. http://cs.panam.edu/TR/cs-tr