A Model for Visualizing Large Answers in WWW Retrieval

Omar Alonso (*) and Ricardo Baeza-Yates (**)
(*) Oracle Corp. and Department of Computer Science, Virginia Tech, VA. oalonso@vt.edu
(**) Department of Computer Science, Universidad de Chile, Chile. rbaeza@dcc.uchile.cl

Abstract

In this paper we present a model for visualizing large collections of documents in World Wide Web retrieval, independently of the retrieval system. Our proposal allows to ease the use of visualization tools which partially solve the problem of data overload on Internet. We present a specific software architecture to separate the user interface from the retrieval system and its current implementation in Java, including the bookpile visual metaphor.

1. Introduction

The typical generic scenario for searching, retrieving, and displaying information is the following. A user has an information need about a certain topic. With a user interface he/she formulates a query to the system (1). The query starts an action in the system (search engine, information retrieval (IR) system, digital library, or other) (2). The system will retrieve (or not) objects and will display them with appropriate messages and layouts in the same graphical user interface (GUI) where the user entered the query (3). Finally, the user decides if the documents are relevant or not. He/she can either exit the system because the information was found or refine the query and start again (4).

The process of searching information in World Wide Web (WWW) can be performed in the following two ways:

- The user types keywords or phrases in search engines (e.g. Altavista, Hotbot, Northern Light, etc.) or browse and also search over predefined classifications (e.g. Yahoo). The technology used here belongs to the domain of information retrieval. The underlying technology of a search engine can be simple inverted files, vector space models, Boolean systems, etc. [5].

- Using an autonomous software agent to perform searches and inform the user of the results. This type of agents belongs to the area of artificial intelligence or agent-oriented programming [10,16]. A very useful form of software agents is as an information retrieval agent, whose main function is to find and gather relevant information [9].

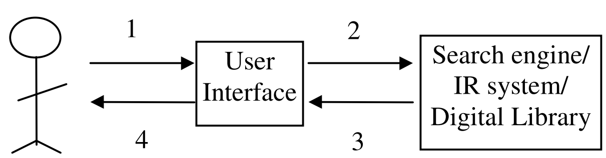

Figure 1. A scenario to search, retrieve, and visualize information.

Search engines interfaces are intuitive and in some cases restricted by the nature of the WWW. There is a limited use of color, no pull-down menus, and limited user interaction. The typical input interface is a simple box where the user fills the terms to search plus two buttons: one to submit the query and the other to cancel it. A framework to develop interfaces for text searching that can be applied to WWW can be found in [2]. We are more interested in the interfaces for visualizations.

The visualization process of the answers can be text only or more rich and complex with the use of a graphical metaphor. In the text only approach, the user gets a list of the top 10 or 20 best documents that potentially contains the information. The list usually contains the title, its URL, size, date, and an abstract of no more than 4 lines of the document. The user opens each document until the desired information is finally located. This is not a problem when the target document is located in the first 20 answers. It becomes a problem when the output of a query is a list of hundreds or thousands of documents. A graphical metaphor presents a rich interface in which the user can browse, filter, process, and reformulate the query.

Hence, visualization is an effective tool to partially solve data overload problems in WWW retrieval when answers contain hundreds of documents. However, nowadays user interfaces are fixed and do not allow to plug in visualization tools. In this paper we address the dependence problem of the user interface with the retrieval software. Our model separates the retrieval part from the user interface. This is particularly useful when the user would like some specific visualization metaphor which may not be available for all retrieval systems. In our model the user can choose one or more visualization stored as Java applets and communicate with one or more retrieval systems. We present our current implementation for the architecture presented here, including a prototype of the bookpile visualization metaphor, an idea presented in [1].

The organization of the paper follows. First, we describe our model. Next, we show its implementation. We finish with some preliminary conclusions and future work.

2. The Model

We want to make the visualization process independent of the searching process in a standard way. By standard we mean that different graphical metaphors can interact with a certain search engine and that different search engines and retrieval agents can produce output for the same visualization. That is, the user interface is independent of the retrieval system and in some sense we can say that is "defined by the user" from a set of available visual interfaces. Our model consists of three main parts: searchers, visualizers, and an interchange markup language as an independent representation.

A searcher is a software that performs searches over the WWW. A visualizer is a graphical (or text only) implementation of a metaphor. A searcher outputs data about the information retrieved in a markup format. A visualizer takes the output from the searcher and displays the information according to its metaphor. Examples of searchers are: search engines and software agents. Examples of visualizers are horizontal bookpile [1], Scatter/Gather [3], TileBars [4], Vibe [7], Lyberworld [8], etc.

A set of visualizations provides options to the users. The visualization of quantitative information consist of principles to help achieve the main goal: communicate complex ideas with clarity, precision, and efficiency. These principles generate design options and should not be applied in a formal way [7]. There is also a technology issue here. For a text only visualization the user only needs a browser that can parse HTML. For a higher degree of sophistication in the metaphor, more technology is needed for its implementation (JavaScript, Java, VRML, etc.). Design options also provides technology options. Altavista provides alternatives for text only, detail information, and graph view. This means that for certain preferences the user needs a JavaScript and Java capable browser.

Figure 2 shows visualizers and searches interacting in our model. In the left part we have a set of different graphical metaphors (visualizers). Each metaphor is interacting with a searcher that can be a search engine or a software agent. The first search engine outputs answers for two different graphical metaphors and the second metaphor is using as input the output of a search engine and a software agent for its visualization process.

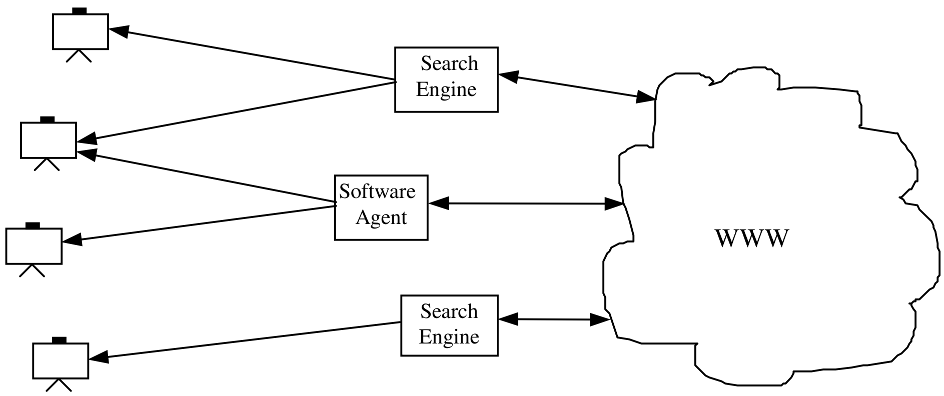

Figure 2. Searchers and visualizers.

As intermediate language we use XML (eXtensible Markup Language), a simplified subset of SGML (Standard Generalized Markup Language) that maintains extensibility, structure, and validation [12,13], [15]. The DTD (Document Type Definition) specifies the set of required and optional elements for documents to conform to that type. The DTD will be used by searchers and visualizers to exchange data over the network. Searchers outputs and visualizers inputs an instance of this DTD that consists of data about the documents found (e.g. type of object, URL, size, language, type of query, etc.). For example, in a query for finding information about "Compiler design" the output of a search engine or software agent using arbitrary attribute-value pairs would be like:

```
<!DOCTYPE seans
‟http://sarvis.cs.vt.edu/~oalonso/xml/seans”>
<?xml default seans
     searcher = ” ‟
     query = ” ‟
     answer = ” ‟
     object = ” ‟
     wc = ” ‟
     rank = ” ‟
     url = ” ‟
     title = ” ‟
     size = ” ‟
     date = ” ‟
     language = ” ‟
?>
<seans>
     searcher = ”Sengine‟
     query = ”Compiler <or> design‟
     answer = ”docs=287 wtop=78 wbot=1‟
     object = ” type=HTML‟
     wc = ”26, 14‟
     rank = ” order=1 weight=78 ‟
     url = ”
http://www.cs.uni.edu/cs5152/notes.html ‟
     title = ” Introduction to Compilers ‟
     size = ” 22 Kb ‟
     date = ” 12 Feb 97  ‟
     language = ”English‟
</seans>
```

We can also embed tags around content to give more meaning. The DTD is not embedded in the document. By binding a meaning to the XML tag `<seans>`, we understand what is contained in that element: the start tag, the end tag, and the contents between them.

```
<html><title>Output</title>
<body>
<seans>
<searcher> Sengine
<query> Compiler <or> design </query>
<answer docs=287 wtop=78 wbot=1>
<object type=HTML>
   <wc> 26, 14 </wc>
   <rank order=1 weight=78>
   <url> http://www.cs.uni.edu/cs5152/notes.html
   </url>
   <title> Introduction to Compilers </title>
   <size> 22 Kb </size>
   <date> 12 Feb 97 </date>
   <language> English </language>
</object>
<object type=HTML>
   <wc> 24, 9 </wc>
   <rank order=2 weight=56>
   <url> http://www.cs.com/c++/proj.html </url>
   <title> Course Project on Compiler Design
   </title>
   <size> 250 Kb </size>
   <date> 10 Mar 97 </date>
    <language> English </language>
</object>
<object type=ASCII>
    <wc> 4, 10 </wc>
    <rank val=3 weight=29>
    <url> http://www.cs.com/c++/notes.txt </url>
    <title> Course Notes for CS456 </title>
    <size> 50 Kb </size>
    <date> 19 Mar 97 </date>
    <language> English </language>
</object>
  ...
</answer>
</searcher>
</seans></body></html>
```

The searcher can give additional information about itself. Then follows the query and its answer. The answer is a collection of objects (documents). In the example we show some typical information that can be given and that is useful for the visualizer.

A visualizer takes the file markup and displays the information text only or uses a graphical metaphor. For a text only view the computation is straightforward. In a graphical metaphor each tag should be map to an internal representation and then display it. One advantage of this intermediate format is that a searcher can include different attributes of each document that can be used by some visualizers and on the other hand, some visualizers can profit optionally from searchers providing more information. Also by separating structure and content from presentation, the same XML document can be written once, then displayed in a variety of ways (a computer monitor, within a cellular-phone display, etc.)

3. Implementation

We assume a search engine or software agent that outputs the answers of a query as a XML document that is valid against the model's DTD. There are several systems and tools for implementing searchers and we are not going to deal with them. For our implementation purposes we assume that a searcher outputs a XML document like the one described in the previous section. With that markup our graphical metaphor implementation as a Java applet will display the information.

We implement our model with a three-tiered architecture: a server, a middle-tier server, and the client where the data is displayed. An alternative architecture could be a layered client-server architecture that consists of databases, mediators, and users [17].

Figure 3 shows the three-tier architecture that implements our model. In the server side we have, as an example, different types of servers: a RDBMS, an IR system, a software agent, and a search engine. Each of them produce XML documents (in a purist SGML/XML terminology, a document is data plus markup) that are rendered by a visualization metaphor. The middle tier server delivers the XML between a searcher (a server) and a visualizer (client).

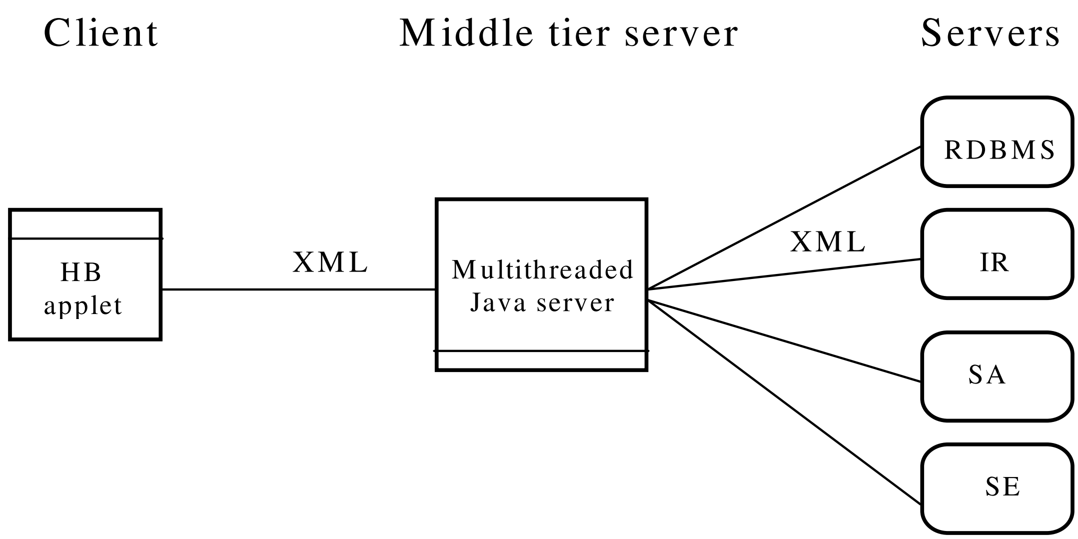

Figure 3. A three-tier architecture that implements the model.

3.1 The horizontal bookpile: a visualizer

Our graphical metaphor called horizontal bookpile is based in [1]. We use a "library" or "bookpile" metaphor to visualize documents depending if the tool is used horizontally or vertically. The "pile" metaphor has been used before, but as an icon for documents. Each document (seen like a book) is represented as a rectangle with a particular color, height, width and position into the set. Each one of these graphical attributes, including the order of the list, can be mapped to a document attribute (rank, number of hits, size, date, etc.). These user selected mappings allow to study different relations between the attributes on the document set, helping the user to select the desired documents. A select button allows to chose a document subset by using the mouse. Every time the user clicks in one of the boxes, the document is fetched and displayed either as is or could be interpreted by another visualizer for single documents. Figure 4 shows an example where order is mapped to document rank, color to the number of hits, width to document size and height to document age. This can be easily changed by clicking in the pull down menu besides each graphical attribute. The visualization has been reduced to 80% of its natural size and colors have been mapped to gray levels.

There are two major components in this Java client: the XML parser and the visualization metaphor. The parser extracts the original data and the visualization metaphor renders it.

3.2 The middle-tier server

This multithreaded server delivers XML documents from the searchers (servers) to the visualizers (clients). Because of security issues in Java and Netscape or other browsers, the applet can only open and retrieve URLs that are in the same machine where the applet resides. This Java server fetches and opens any type of URL for the applet. The server is a multithreaded server such that different machines can load the same visualization applet and open any given URL.

3.3 The searchers

There are several choices for implementing the searchers. We will describe a file system approach and a RDBMS approach.

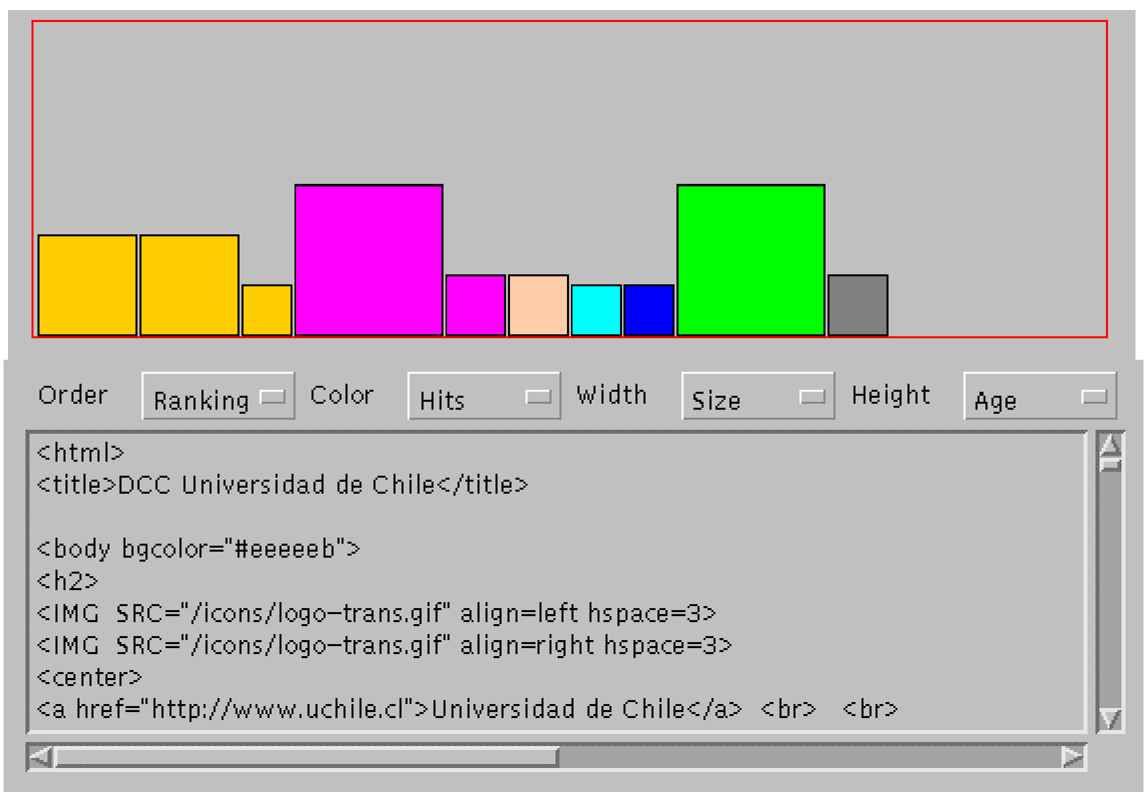

Figure 4. The horizontal bookpile.

In the file system approach the information retrieval systems works on files that are stored and indexed within the file system. In this case the XML files are stored in a separate directory specifying the data. Following a similar approach to Altavista, a crawler will pull data from the WWW and store it in the system. Frequent indexing on the system enables to have retrieve updated information fast. The vizualizer must load the data from a directory for further rendering.

In the RDMS approach like Oracle's ConText Data Cartridge [14] all the data, indexes, and results are stored in the database with options to be also stored in the file system and the WWW. The visualizer must query ConText using a mechanism like JDBC to render the data to the client.

We have experimented with the bookpile implementation without having any performance problems. One initial concern was if Java was appropriate for fast changes on graphical views that could be potentially very complex. As the server is written in Java, we have installed it in Unix, Windows and MacIntosh platforms, and used the visualization from different Internet locations. Such experiments simulate the answers of the retrieval system passing them to our server.

4 Conclusions

We have presented a model for searching and visualizing large answers in WWW retrieval with emphasis in the visualization process. With so many systems and tools for WWW retrieval, our model defines a way to share the output of those so that the visualization is more independent and therefore an option of several to the user.

Work in progress includes having a DTD for the searchers. Future work includes other document visualizations (either a collection or a single document) and more experiments, as well as developing drivers that will transform HTML output pages of well known systems to our intermediate format. Another line of research includes retrieval systems that allow to query not only content but also structure (e.g. HTML structure) of documents [11].

Acknowledgements

This work was partially funded by CYTED project VII.13 AMYRI and FONDEF project D961064 (CONICYT Chile).

References

[1] R. Baeza-Yates. "Visualization of Large Answers in Text Databases". *AVI'96*, Giubbo, Italy (1996).

[2] B. Schneiderman, D. Byrd, and W. Bruce Croft. "Clarifying Search: A User-Interface Framework for Text Searches". *Dlib Magazine* (January 1997).

[3] M. Hearst. "Interfaces for Searching the Web". *Scientific American* (March 1997).

[4] M. Hearst. "TileBars: Visualization of term Distribution Information in Full Text Information Access". *Proceedings of CHI'95*, Denver, CO (May 1995).

[5] W. Frakes and R. Baeza-Yates. *Information Retrieval. Data Structures and Algorithms*. Prentice-Hall (1992).

[6] E. Tufte. *The Visual Display of Quantitative Information*. Graphics Press (1987).

[7] Olsen, K., Korfhage, R., Sochats, K., Spring, M., and Williams, J.; "Visualization of a document collection: The VIBE system"; *Information Processing and Management* 29, 1 (1993), 69-81.

[8] Hemmje, M., Kunkel, C., and Willet, A. ; "Lyberworld - a visualization user interface supporting text retrieval"; *17th ACM SIGIR*; (Dublin, Jul 1994).

[9] N. Jacobs and R. Shea, "The Role of Java in InfoSleuth: Agent-based Exploitation of Heterogeneous Information Resources". *MCC Technical Report* MCC-INSL-018-96 (1996).

[10] *IEEE Internet Computing*. Special Issue on Software Agents, (July/August 1997).

[11] Navarro, G., and Baeza-Yates, R; "Proximal nodes: A model to query document databases on structure and content"; *ACM Transactions of Information Systems*, (October 1997).

[12] Khare, R., and Rifkin, A.; "XML: A Door to Automated Web Applications"; *IEEE Internet Computing*, (July-August 1997).

[13] W3C "Extensible Markup Language (XML)" http://www.w3.org/TR/1998/REC-xml-19980210

[14] Oracle 8 ConText Cartridge. *Application Developer's Guide*. Oracle Corp. (1997).

[15] C. Goldfarb and P. Prescod. *The XML Handbook*. Prentice-Hall (1998).

[16] J. Bradshaw (Ed.) *Software Agents*. The MIT Press (1997).

[17] M. Shaw and D. Garlan. *Software Architecture*. Prentice-Hall (1996).