VisageWeb: Visualizing WWW Data in Visage

Michael Higgins
MAYA Design Group, Inc.
2100 Wharton Street
Suite #702
Pittsburgh, PA 15203
+1 412 488 2900
higgins@maya.com

Peter Lucas
MAYA Design Group, Inc.
2100 Wharton Street
Suite #702
Pittsburgh, PA 15203
+1 412 488 2900
lucas@maya.com

Jeffrey Senn
MAYA Design Group, Inc.
2100 Wharton Street
Suite #702
Pittsburgh, PA 15203
+1 412 488 2900
senn@maya.com

Abstract

VisageWeb is an information-centric user interface to the World Wide Web built within the Visage data visualization environment. This paper traces the development of the VisageWeb project, using it to motivate an exploration of how an information-centric architecture copes with new visualization challenges. We conclude with a presentation of the VisageWeb prototype itself.

Keywords

World Wide Web, Information Visualization, User Interface

Introduction

Visage is an information-centric environment for exploring and visualizing data [1]. Our goal in the VisageWeb project is to extend the Visage environment to easily accommodate interfaces for browsing, organizing, exploring, and analyzing information on the World Wide Web.

A number of environments exist already to visualize the topographical structure of the Web. Some rely primarily on the explicit hyperlink connections amongst pages, others analyze page content to generate relationships [2,3].

VisageWeb is unique in that while it supports "macroscopic" visualizations of this sort, it also allows an analyst to extract data from within pages. The value of the World Wide Web, of course, is the data contained within it. Usually this data is found within a page (or scattered amongst a few).

How does VisageWeb manage this flexibility? Visage's principled means of representing information allows great flexibility in the display and manipulation of that information. By modeling all information as directly manipulable objects, Visage provides the user a sense of direct-access – of "getting your hands on" the data.

We will briefly re-cap the information-centric architecture of Visage. We will then present the information architecture of the Web, drawing appropriate parallels to Visage and highlighting distinctions. Tracing VisageWeb's development, we will see how we needed to recast the Web to suit Visage's architecture and how we adapted Visage's user interface and capabilities to accommodate the Web. Finally, we will show some of the new prototypes for visualizing the Web now possible within Visage.

The information-centric approach

Visage's architecture, both in the details of its user interface and more generally can best be described as *information-centric*. Application-centric and document-centric environments form a progression in which information-centric environments are the next logical step. Application-centric environments give the file primacy – any manipulation of data must be performed through an application upon a file. This level of granularity can be frustrating for the user because it sharply limits the organization of data to what the file system is capable of and limits manipulation and understanding of data to what an individual application permits.

With the advent of the graphical user interface, direct manipulation of files became more commonplace. After files became concrete, they evolved into documents: files coupled with component-managed behaviors. Extremely document-centric environments (like Web Forager [4] and Workscape [5]) allow visual and spatial operations to be performed on their documents.

Visage simply carries this progression one step further. Visage allows a data element to be directly manipulated at any level of granularity. A numeric table entry, a group of bars on a bar chart, or a complex presentation graphic are all equal candidates for user manipulation. Where document-centric interfaces to the Web, such as Web Forager, provide the user the ability to organize and understand collections of Web pages, VisageWeb allows the manipulation of fine-grained information even within a Web page.

To make this philosophy work in practice, we need an *information architecture* – a method of representing data consistently, so that the same fundamental operations may be performed on them everywhere.

Our information architecture is essentially object-oriented, but with a bent toward data representation rather than code representation. We use uniquely identified, extensible bundles of attribute-value pairs to represent every piece of data in Visage. In a traditional object-oriented system, we might also associate behavior with each of these bundles, enforcing a strict interface to hide the data representation. That approach is excellent for

managing a code system, but we are interested here in creating an information visualization system. So we expose the structure of each bundle and allow behavior to be determined polymorphically and dynamically by "frames," discussed later.

We've come to call these bundles of information *u-forms*. Recognizing the need for a universal language for data representation, Dertouzos introduced *e-forms* [6]. His e-forms are simply bundles of attribute-value pairs. Our u-forms extend this idea by attaching a universally unique identifier (or UUID) to each bundle.

The Visage user-interface architecture

Having explained the philosophy and fundamental architecture of Visage, we need to examine the concrete pieces of the system.

Data, as mentioned, are represented as universally unique attribute-value pairs called u-forms. A typical Visage database is a Web of u-forms, each with links to other u-forms, rather than a hierarchy with well-defined leaves and interior nodes.

Graphically, Visage represents u-forms with *visual elements*. A visual element (or simply, element) refers to any atomically manipulable object in Visage. Examples include bars in a bar chart, a data cell in a table, and text labels on plots. Each element corresponds to one u-form in the database. This relationship is one-to-many, a u-form may have many different elements representing it.

Any element can have certain core interface operations performed on it at any time throughout the Visage environment. These operations include:

Drill-down, which allows users to traverse a link from a data object represented by a visual element to related objects in the database.

Drag-and-drop, which allows people to move visual elements among frames.

Coordinated marking, which means that when a person marks an element with a certain color, other elements that share the same underlying u-form are marked that color as well.

Recomposition, which allows people to group objects (retrieved by drill-down) by common values of a specified attribute.

Magnification, which means that every visual element in Visage can be magnified, or scaled, to conserve screen real estate or provide a better view of the visualization.

Frames are grouped collections of other visual elements that function as complex visualizations. This grouping is achieved through Visage's scripting facility. It is important to realize that frames are just visual elements that represent aggregate u-forms together with the members of the aggregate, not a special kind of object in the system. If the frame is a bar chart, it displays its members as bars, together with axes and labels. If the frame is a scatter plot, it maps its members into points. If it's a table, it maps the members into cells and organizes them by row and column. If it's a slide it may be an arrangement of other frames for purposes of a Powerpoint-style presentation.

Through its scripts, each frame defines a user-interface "physics" [7]. Because the u-form object model is pervasive throughout Visage, data can be copied from one frame into another, allowing multiple polymorphic views of the same data. Furthermore, the core interface operations are available in every frame. To put it another way, frames provide a specialization mechanism for viewing data, without sacrificing the generally available operations and representations. A frame or collection of frames can be thought of as a special-purpose "information appliance" [8].

The concept of information appliance is of particular importance. VisageWeb is an attempt to build a Web browser as an information appliance within Visage. Because all such appliances share the u-form data model and the ubiquitous basic user interface of Visage, doing this gives us an immediate entree into other Visage information appliances – plot charts, outliners, and so on. Furthermore, it facilitates developing a direct-manipulation interface to organizing and navigating amongst and within Web pages.

Building the VisageWeb appliance presents two major challenges: extending the Visage environment to handle data from the Web and, more importantly, mapping the information architecture of the Web into that of Visage. It is important to note that Visage's information architecture was designed to support the subsumption of other information architectures, so this is not an unnatural goal.

The information architecture of the World Wide Web

Visage has a strong and principled information architecture. We insist on the representation of all data as a network of u-forms. The computation that presents the data in a particular visualization is the responsibility of a particular frame. This enforces a sharp separation between the representation of data and its presentation, and makes multiple views of the same data commonplace within Visage. (This is reminiscent of Smalltalk's MVC, see [9].)

The Web, however, has its own information architecture. Since it is a highly distributed entity and a morass of de facto standards and individual publishers, it is not so rigorous. Bearing in mind that this summary can't be complete – new "Web technologies" emerge daily – let's examine the information architecture of the Web [10]:

Uniform Resource Locators The topology of the World Wide Web is that of a network, and each node in the network is identified by a URL. They are responsible for making navigation possible within the Web. These URLs, as their name implies, specify data *location* rather than data identity.

HyperText Markup Language The principal language of the Web is HTML. It is a language that has enjoyed unprecedented success, but also evades consistent

definition and interpretation. The original conception of HTML meant to separate the representation of documents from their visual presentation. As the Web has evolved, however, authors have demanded tighter control over the visual rendering of their documents. Responding to this, extensions of HTML designed to deal with presentation in more and less principled ways have emerged, including the TABLE element and Cascading Style Sheets.

Multipurpose Internet Mail Extensions MIME provides an extensible type system for the Web [11]. It informs a browser about the nature of the information that has been downloaded, which in turn hints at what display method might be appropriate. Generally, the end-user has at least some control over what this display method is.

Comparing the architectures

The task at hand is to map these elements of the World Wide Web's information architecture into that of Visage. Let's look at some promising correspondences.

Similarities

First, the URL on the Web serves some of the same purposes that a UUID does in the Visage repository. Both allow larger aggregate structures of information to be constructed by referencing other structures. URLs work by telling you where to go to get a piece of information – you must assume that whatever is at that location is what you wanted. UUIDs are handles to particular bundles of information. These bundles have persistent "identity" – an identity that is preserved across changes of storage location and even across changes in the content of the object.

Part of what we've discovered from analyzing the Web in these terms is that both of these ideas have merit. A UUID makes tasks like revision control and data replication tractable. A URL provides instructions for retrieval. The Visage architecture defines a class of u-forms that serve as *pointers*, containing a field that has a location reference and one that has a unique identifier, so that a system can find the information and then make sure that it is what was expected.

Visage attaches type information to its u-forms to assist frames in rendering them successfully but dictates no particular behavior with respect to types. The MIME type system is fairly flat by comparison with type systems found in many programming languages. It is designed to represent a document-centric model and does not scale down very well. Information of high granularity is likely to be merely an "octet-stream" in MIME, which is not very informative. Fortunately, the Visage type system can subsume MIME, since it is more expressive, and augment it when necessary.

Differences

To deal with the lack of well-defined uniqueness on the Web, we simply imposed it. Whenever a Web page is brought into Visage, u-forms are created to represent it. They also have attributes that indicate from what URL the u-forms originated and even from what part of the parsed HTML stream they came. In that way we can make an intelligent guess later, if the page is fetched again, about whether the new page and the old are "the same." Many sites on the Web update their content continually. Since the Web has no pervasive notion of identity, information providers can't declare whether new content should be treated as an entirely new u-form or an update of an old u-form. VisageWeb can either create a new u-form or update an existing one, relying on heuristics involving URL and page structure to decide when to do which.

HTML is a fairly straightforward language to represent in the u-form environment since it is essentially hierarchical. Its parsing rules are a bit strange, but well-defined. The only difficulty is that some pieces of an HTML document have no individual identity, not even a URL. There is no well-defined way to refer to one paragraph or another within an HTML document. Visage's information-centric agenda requires us to be able to manipulate much finer granularities, so we parse incoming HTML into a tree of u-forms, once again imposing individual identities for every tag encountered. In principle, every distinguishable parse element in the HTML stream could be represented as a u-form – that means every word. Because the user typically doesn't want to use a drag-and-drop direct-manipulation interface on individual words, we aggregate them according to their parent tags for manipulation in the user interface.

Sometimes it's possible to grant more identity to elements of a Web page – images and forms, for instance, have URLs associated with them. Even non-URLed elements can be identified heuristically – "the third paragraph" is a perfectly sensible heuristic. While this isn't as good as real unique identity, such heuristics can have great practical value in many real-life situations.

MIME types serve a useful but limited classification function. Visage provides composable and extensible types of its own. Typically, in the case of a decomposable MIME type like HTML, we enrich the type structure significantly, adding types for every tag and entity. For less easily parsed types like "image/gif" we simply use the MIME type. VisageWeb's approach to handling types it doesn't understand is the same as most browsers': it tries the best it can with a default visualization. VisageWeb is somewhat more forgiving than other browsers, since Visage is designed to handle "unfamiliar" data gracefully.

It turns out that anything with a URL is fetched via the Hypertext Transport Protocol (HTTP) [12]. HTTP attaches many interesting attributes like "content type," "last modified date," and so forth, to every transmission. For

system administrators and site managers, this meta-information (recorded in the u-forms by VisageWeb) can make very informative visualizations. This means that even information of a MIME type that Visage can't render may have interesting meta-information to visualize.

Building the information-centric Web browser

Visage is as much an experiment in a new user interface as it is a visualization environment. In fact, we believe that the two roles dove-tail nicely, producing an information exploration environment. This differentiates it, we think, from some other experimental Web browsers [13]. One notable browser that exists as part of a larger environment that might well be characterized as an "information exploration environment" is the pad++ "zoomable browser" [14]. It's useful to recount some of the adjustments that had to be made to the user interface to accommodate the visualization challenges afforded by the Web.

Project Requirements and Rationale

- Users should be able to directly manipulate Web pages at different levels of granularity. For instance, a user should be able to tear an HTML table, paragraph, or image off a Web page and drop it in another frame (even another Web page!). Users should be able to rearrange the contents of a page and add other Visage elements to a page. Supporting all these levels of granularity is important to allow visualizations to be constructed from information wherever it is found.
- Users should be able to directly manipulate entire Web pages, analyzing each Web page as a collection of HTML u-forms, image u-forms, and other content u-forms. This is necessary for VisageWeb to be consistent with the rest of Visage.
- Users should be able to directly manipulate collections of Web pages, visualizing these collections in other frames. This allows macroscopic visualizations.
- Asynchronous operations should be available. Users expect a threaded, non-blocking browsing environment.
- "Web operations" such as traversing hyperlinks and parsing HTML should be available from the high level Visage scripting language. As much as possible should be implemented as script to facilitate user change and user incorporation into larger appliances. It is a tenet of Visage that the system designer cannot anticipate all possible visualizations, so the analyst must be able to extend the environment.
- All the normal Visage operations should be available. Web pages, and pieces of Web pages, need to be navigable (in the drill-down sense), copyable, draggable, and scalable. Again, this is necessary for consistency with Visage as a whole.
- Normal Web operations should be available in the user interface. The user should have point-and-click hyperlink traversal, forward and backward navigation, a refresh operation, and organization of favorite links and pages. To the extent that this falls out of standard visualizations within Visage, so much the better.

Early Attempts

Our initial prototypes were unsatisfactory in almost all of these respects. Visage is implemented over a platform-independent C++ kernel, which provides an interpreter for the high-level scripting language, an interface to data repositories containing u-forms, and various event-handling and drawing primitives. Our first attempt at providing a browser appliance was simply to add a set of new primitives to the kernel.

We added a hard-coded frame for rendering HTML, various URL and HTTP routines for fetching Web pages, and an HTML parser in C++. All of this was much too low-level. It lacked flexibility and made implementing the user-interface nearly impossible. It also complicated moving data from the "HTML frame" to other frames and back. We realized we had failed to follow our own architectural guidelines.

So we threw all of it away. We carefully segmented the problem into pieces that truly required a more systems-oriented language and those that could be handled in the high-level Visage script. We chose Java rather than C++ for our systems work, making asynchronous operations a lot easier.

Our final design placed fetching URLs and parsing HTML in systems-level Java code, accessible through a typical function call API from the scripting language. Rendering Web pages – the definition of the "HTML frame" – became a high-level chore. Thus we could bring to bear the native power of the Visage environment to render HTML in a hierarchical "outline" form using the standard Visage outliner frame, or in any one of the other Visage frames available.

The design of the browser frame presented a challenge, however. As we've described, a Visage frame visualizes a collection of data. But a Web page is a hierarchy – a collection of collections of collections. (This falls out of the hierarchical tag-structure of HTML.) It follows that the natural design for a Visage frame that renders a Web page should be a frame containing nested subframes, which themselves contain frames, and so on. This is a compelling idea, because it means that when the user grabs the part of a Web page containing a paragraph, he will get that frame which represents the paragraph, and it will quite naturally be manipulable within Visage.

Unfortunately, our visual and user-interface design style for frames was too heavy. We had certainly conceived of nested frames before but never confronted dozens or hundreds of nested frames. Nor had we considered that frame decorations like scrollbars, titlebars, and buttons

might be out of place when nested. Obviously, a Web page cannot be faithful to its author's design if every paragraph comes with a scrollbar and a title!

As we were running into these problems, we had also begun to be very frustrated with the fact that users encountering Visage for the first time confused it with a standard Window-Icon-Menu-Pointer (WIMP) environment. Part of the problem was that frames, with their attendant scrollbars and buttons and titles, looked too much like windows. Thinking of frames as windows has proven to steer beginning Visage users away from some of Visage's most compelling capabilities.

Rethinking the User Interface

We decided to dispense with the extra baggage. Frames are simply areas on which elements are arranged. Scrollbars appear when and if necessary. Titlebars are present if a frame designer decides one is called for. Modifier buttons are simply absent.

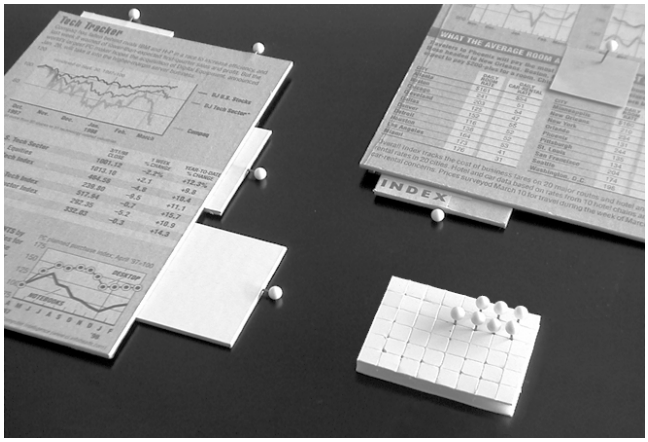

Figure 1: Photograph of early physical mockups of attachments.

This gives a pleasing spartan design. Under the new design, attention is focused on the visualization within the frame – distracting visual elements are present only when necessary. It leaves us with a thorny problem, though. When we need buttons and other extra controls, where should we put them?

Our solution is the "attachment." Attachments are lightweight containers of UI devices. They can be attached to a frame in any one of a number of positions and can be exposed in a variety of fashions:

Tack The "minimized" exposure; all that is visible is a tiny pushpin to indicate that an attachment is present.

Icon An icon or set of icons representing the attachment's functions is visible. Usually the icons serve as buttons for the functions.

Exposed The full attachment is exposed.

The attachments may be torn off the frame when not needed or moved into any one of their exposure states. This neatly solved the problem of a place for the controls. Here was somewhere where a maximize button could reside, or a refresh button, or a status bar. Dynamic query sliders, always lightweight objects in Visage, are also now attachments.

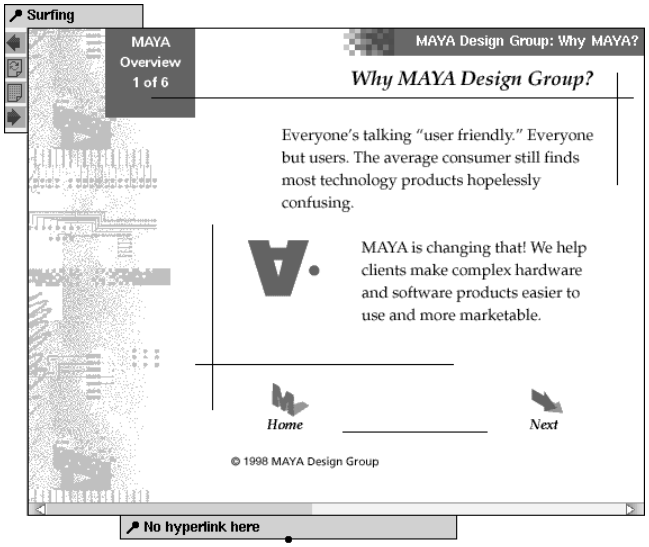

Figure 2: A Web page with an attachment that handles Web navigation and one that displays URLs when the mouse rolls over a hyperlink.

A smorgasbord of visualizations

The novelty of VisageWeb lies in its ability to easily generate a variety of visualizations. We'll illustrate and describe a number of them.

To begin, let's look at a visualization of the finest granularity of information that VisageWeb processes: HTML tags within a Web page.

Outliner	content
http://www.maya.com/	
By: children	
HTML	
By: children	
HEAD	
By: children	
TITLE	
By: children	
Text Content	MAYA
Text Content	Design
Text Content	Group
BODY	
By: children	
TABLE	
By: attributes	
Attributes for BODY	

Figure 3: A Web page, left, with an alternate visualization of it in the outliner frame.

Figure 3 shows a scaled down Web page next to an outliner frame. Both the Web page frame and the outliner frame are visualizing the same data, to wit, the MAYA Design Group's home page. The Web page frame shows it as a Web browser traditionally would: a rendered document. The outliner, by contrast, shows the Web page as an aggregate with subordinate aggregates illustrating the hierarchical internal structure of a Web page.

It is important to emphasize that the outliner frame is one of the standard visualization tools available in Visage.

It knows nothing about Web pages and requires no special adaptations to work with VisageWeb. This is a consequence of their shared information architecture.

Let's examine the Web page frame more closely. The outliner visualization makes it clear that the entire page is not the finest level of information available. The seasoned Visage user would therefore expect to be able to interact with those interior structures in a drag-and-drop fashion. In fact, this is the case.

Figure 4: A Web page that is being edited using direct manipulation techniques.

Figure 4 shows that a user can edit the Web page by dragging pieces of it out. In the figure, a GIF image has been copied out of the Web page and a paragraph of text is in the process of being copied (the stylized arrow has been added to clarify the operation). It is also possible to *remove* structures from the Web page, rather than copying them. (Among other things, this allows one to easily remove flashing banner ads.) If this is done, the Web page presentation reorganizes itself appropriately.

Being able to easily remove content from a Web page inspires the question: is it possible to add content? The answer is yes, and in just the way one would expect. Any Visage element may be dropped into a Web page. It will be incorporated into the Web page's flow of information as naturally as is possible.

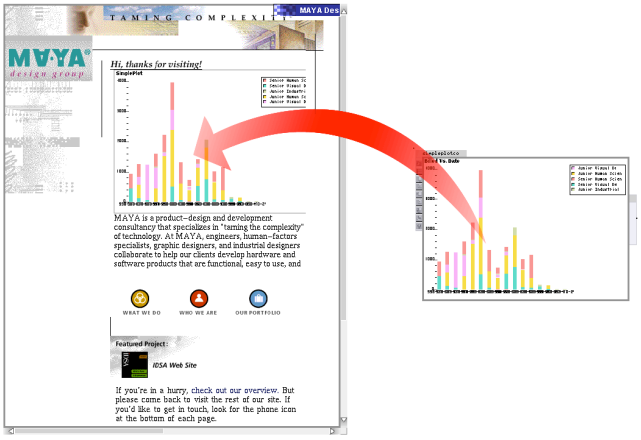

Figure 5: A standard Visage visualization being added to a Web page frame.

Moreover, it is then possible to publish the Web frame. This provides a very natural mechanism for publishing visualizations constructed in Visage to an audience: simply create the visualization using any Visage frames desired, then drop them into a Web page, and (using an attachment created for this purpose) publish the page. "Publishing" really does two things. It generates GIF images of the elements involved, together with an HTML wrapper. It also generates a serialized version of the Visage elements which is written into a comment in the HTML. It then writes the whole thing to a location accessible by a Web server (usually via a distributed filesystem).

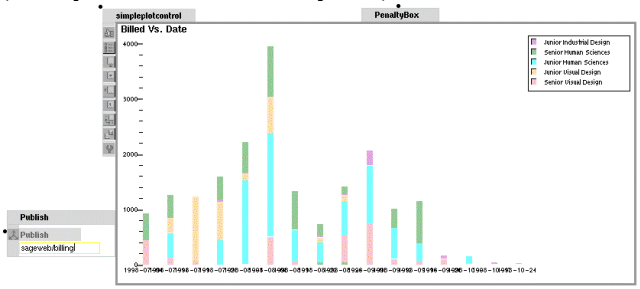

Figure 6: A Visage visualization being published directly to the Web.

The consequence of this is that users using traditional Web browsers can see the visualizations generated in Visage, but VisageWeb users can not only see the visualizations, but recover the frames published as live objects, since VisageWeb can interpret the hidden serialization.

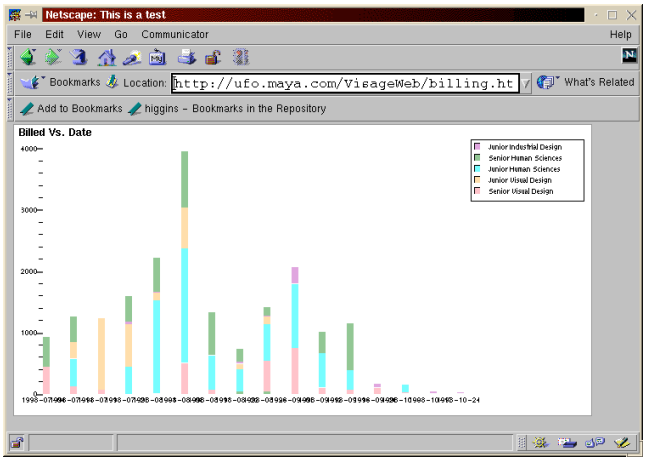

Figure 7: The Visage visualization from Figure 6 in Netscape.

Let's turn now to a more realistic visualization problem. Since we can access and visualize data within a Web page, it seems natural to try to exploit data found in a normal HTML table. We'll use a real-world example to motivate our discussion.

The MAYA Design Group, like many companies, uses an online timesheet system with a Web-based interface. Every employee enters time, billed to various projects, into the system every week. Project managers can call up HTML pages that, having interrogated a database, display these billed hours.

Unfortunately, even for rather small projects a tabular visualization is difficult to understand. When the projects become moderate sized (involving, say, more than ten people over dozens of weeks) the tables are almost worthless.

An analyst using VisageWeb can simply drag the table into a more useful frame, and generate a more useful visualization.

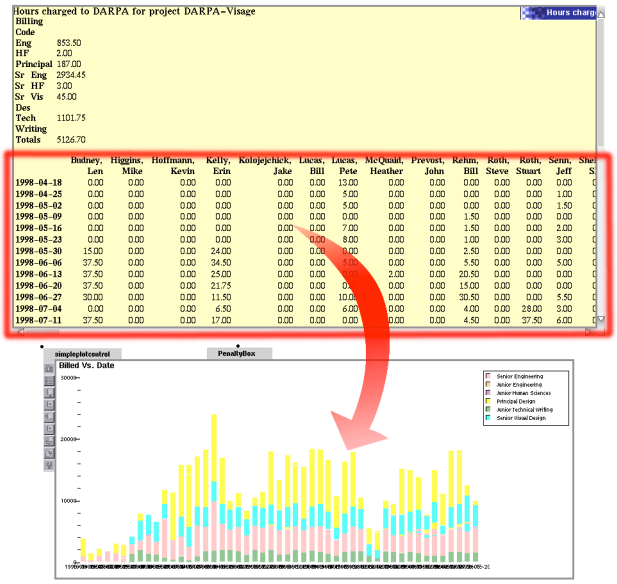

Figure 8: An HTML table of financial billing data is dragged to a plot chart.

Now we have a plot chart of billing, color-coded by department within MAYA, displayed across time. This makes spending trends easy to spot. Further analysis reveals the particular employess responsible for a spike in spending.

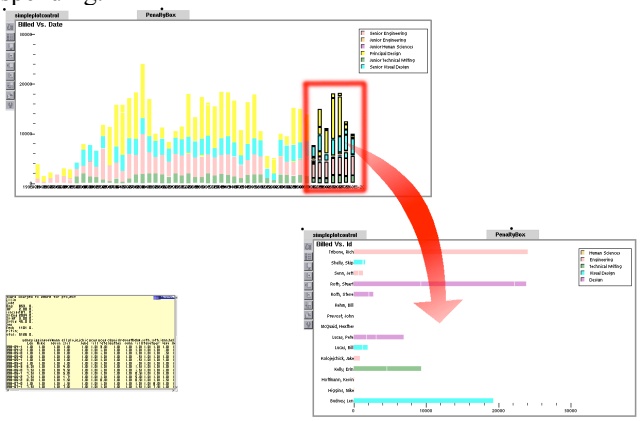

Figure 9: Plot chart of billing data from Figure 8 is further analyzed. Source Web page is shown scaled down in lower left.

The catch, of course, is that most tables found on the Web are not well-formed [15]. VisageWeb, by default, naively assumes that columns define attributes of rows. This works fairly well much of the time, but the analyst must often add script code to interpret the tables. (In the example cited, alternate rows of the table were used by the HTML author for visual formatting and are irrelevant to the data analysis task, so script had to be added to remove them.) A mechanism is provided to facilitate doing this in an ad-hoc manner.

Let's now move to tasks involving the higher, "page-level," granularity. A common method of finding information on the Web is to use a search engine. Typically, the result is a big list of hyperlinks. Visage's scripting interface makes it a matter of a few minutes work to design a wrapper for the search engines (similar to the "Net Search" button on Netscape's browser) that fetches the top ten entire pages, representing them as thumbnails.

Figure 10: The results of a Web search.

Now the user can use Visage analysis tools on the collection of pages. One easy thing to do is plot the Web pages by geographic point of origin. There are databases on the Web (*http://cello.cs.uiuc.edu/cgi-bin/slamm/ip211/*) that map an IP address to latitude and longitude. A Visage user

armed with a script that marks up the u-forms for his or her Web pages with such information can plot the pages on a world map. (In the current prototype such a script is active by default.) More powerful scripted frames could do arbitrary sorts of analysis. In any case, the user has the full power of Visage's graphical data analysis environment to evaluate the results of the search.

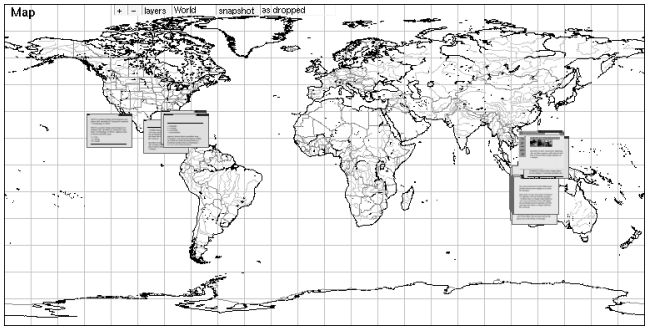

Figure 11: Web pages scattered on a map of the world.

It is also possible to do visualizations that don't involve "Web pages" per se at all. It is an easy matter to write a script that queries a variety of search engines with same search terms, collecting results from each. It then might generate scores for each result based on the relative rank of the discovered URL within each engine's catalog.

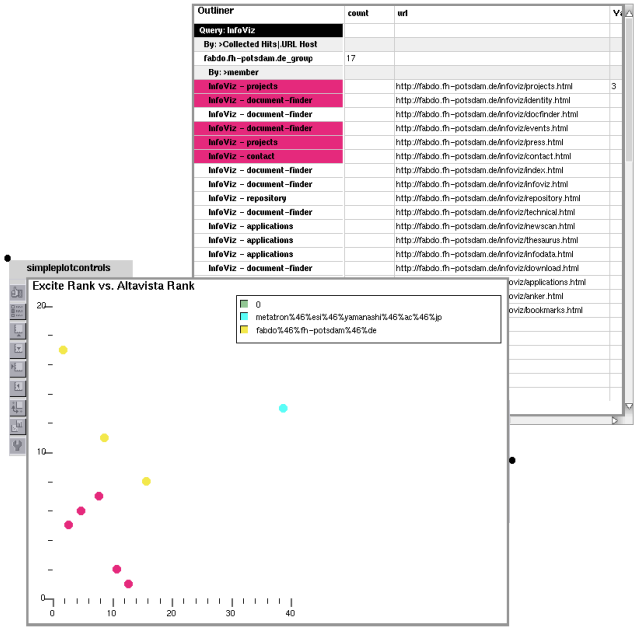

Figure 12: Lower left: Search engine results are plotted with their rank on the Excite engine displayed along one axis and their rank in the Altavista engine along the other. Upper right: Search engine results recomposed by host site. Saturated magenta coloring indicates coordinated selection, less saturated coloring encodes host site.

Interestingly, there is comparatively little overlap amongst the popular search engines' catalogs.

Conclusions

We've adapted the World Wide Web to Visage's information-centric environment, developing an information appliance within Visage for extracting information from the Web.

Creating this appliance has led to new insights about the Visage information architecture and user-interface paradigm. It has also generated an exciting and unique Web browser, fully integrated with the data-manipulation and visualization tools of Visage.

Acknowledgments

This work was funded by DARPA contract #DAA01-97-C-R045. Many of the ideas presented here were developed through interaction with our colleagues at the MAYA Design Group. Particularly valuable engineering assistance in developing prototypes was provided by Phil Stroffolino and Kevin Hoffmann. The "attachments" design is largely the result of work by David Bishop and Phil Oye.

The authors would like to thank Erin Kelly, Heather McQuaid, Susan Salis, Anukul Kapoor, and Jeremiah Blatz for editorial assistance. We would also like to thank Mr. Blatz for help in formatting our screenshots and photographs, and for video production help. Finally, we would like to thank Dr. Steven Roth for many useful insights about the paper in general.

References

1. Roth, Steven F., Chuah, Mei C., Kerpedjiev, Stephan, Kolojejchick, Jake, and Lucas, Peter. Towards an Information Visualization Workspace: Combining Multiple Means of Expression. *Human-Computer Interaction*, Volume 12 (1997).
1. Card, Stuart K., Robertson, George G., and York, William. The Webbook and the Web Forager: An Information Workspace for the World Wide Web. *Proceedings of CHI '96* (New York NY, 1996), ACM Press, 111-117.
2. Mukherjea, S., Hirata, K., and Hara, Y. Vizualizing the Results of Multimedia Web Search Engines. *Proceedings of Information Vizualization 95*, (Los Alamitos CA,1996), IEEE Computer Soc. Press, 64-65
3. Rohrer, Randall M. and Swing, Edward. Web-based Information Vizualization. *IEEE Computer Graphics and Applications* (July/August 1997), 52-59.
4. Ballay, Joseph M. Designing Workscape: An Interdisciplinary Experience. *Proceedings of CHI '94* (New York NY, 1994), ACM Press, 10-15.
5. Dertouzos, Michael. *What Will Be*. Harper Collins, New York, NY, 1997. Brown, Marc H. and Schillner, Robert A.
6. Smith, R.B. Experiences with the Alternate Reality Kit, An Example of the Tension Between Literalism and Magic. *Proceedings ACM CHI + GI 87 Conf. on Human Factors in Computing Systems and Graphics Interface*, (New York, 1987), ACM Press, 61-67.

7. Roth, Steven F., Kolojejchick, Jake, and Lucas, Peter. Information Appliances and Tools in Visage. *IEEE Computer Graphics and Applications* (July/August 1997), 32-41.
8. Tessler, L. The Smalltalk Environment. *Byte*, (Vol. 6., No. 8, August 1981), 90-147.
9. Berners-Lee, T., Cailliau, R., Groff, J. F., and Pollermann, B. World-Wide-Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 1992.
10. Multipurpose Internet Mail Extensions. ftp://ftp.isi.edu/in-notes/rfc1522.txt, ftp://ftp.isi.edu/in-notes/rfc1523.txt
11. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T. Hypertext Transfer Protocol – HTTP/1.1. January, 1997
12. The DeckScape Web Browser. *Proceedings of CHI '96* (New York NY, 1996), Video.
13. Bederson, B., Hollan, J., Stewart, J., Vick, D., Ring, L., Grose, E., Forsythe, C. A Zooming Web Browser. *Human Factors in Web Development*, Eds. Ratner, Grose, and Forsythe, (Lawrence Erlbaum Assoc., 1998), 255-266.
14. Pirolli, P., Pitkow, J., et al. Silk from a sow's ear: Extracting usable structures from the Web. *Conference on Human Factors in Computing Systems, CHI '96*, (Vancouver, Canada)