VisageWeb:An Information-Centric Web Browser 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 user interface challenges. We conclude with a presentation of the VisageWeb prototype itself.

Keywords
World Wide Web, User Interface, Information Visualization

INTRODUCTION
Visage is an information-centric environment for exploring and visualizing data [5]. 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.

Why develop yet another environment for using the Web? Visage is an "information-centric" platform. Its principled means of representing information allows unprecedented flexibility in the display and manipulation of that information. By modeling all information as directly manipulable UI objects, Visage provides the user an unprecedented sense of direct-access – of "getting your hands on" the data. We are re-inventing interaction with the Web within this architecture, hoping to both enrich users' experiences and glean deeper insights into the utility and expressiveness of our system.

In this paper 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 interacting with 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*. Information-centricity is the next logical step in environment evolution from application-centric environments to document-centric environments.

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 [2] and Workscape [1]) allow visual and spatial operations to be performed on their documents. OLE and OpenDoc are commercial attempts to provide this sort of environment.

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* [3]. 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.

Scaling, which means that every visual element in Visage can be scaled, or magnified, 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." 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."

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.

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.

The Web, however, has its own information architecture. Since it is a highly distributed entity, 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:

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. 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.

HTML made a good stab at separating content and presentation, as does our u-form-frame model. We can certainly support "make Web pages look like Netscape does" as a particular visualization.

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. 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 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). 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

The Web is a big and complicated place. Building a browser that follows our information-centric architecture, supplies a direct-manipulation interface, and deals respectably with a reasonable subset of the Web is a major challenge. We found that it exercised our capabilities as data architects, interface designers, engineers, and visual designers. It has profoundly influenced our most recent series of interface revisions within Visage and has led to major system improvements.

Project Requirements

- 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.
- 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.
- Users should be able to directly manipulate collections of Web pages, visualizing these collections in other frames.
- 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.
- 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.
- 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.

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.

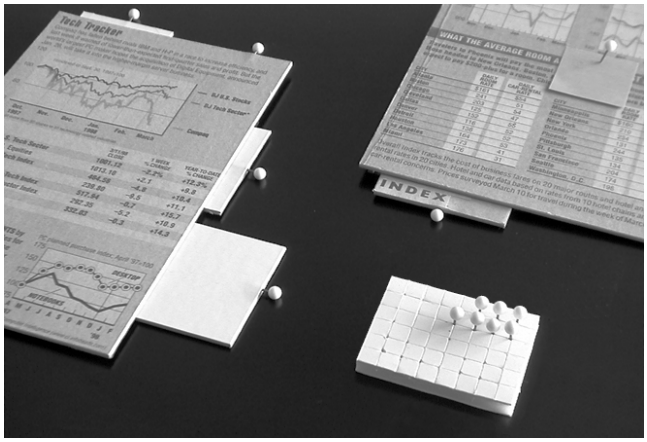

Figure 1: Photograph of early physical mockups of attachments.

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.

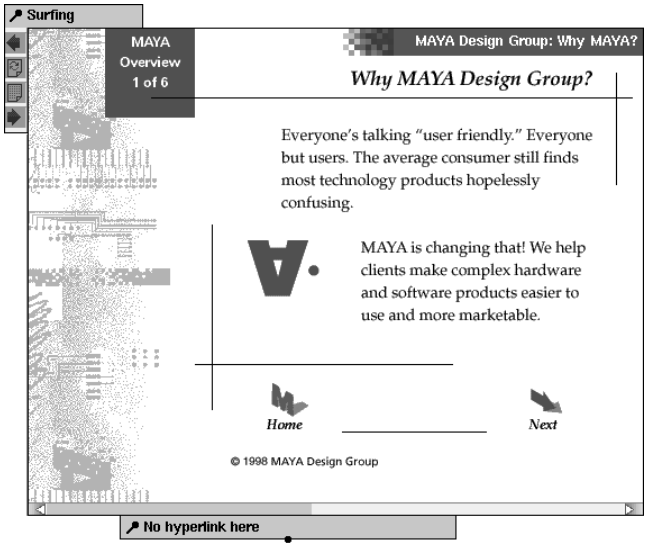

Figure 2: A Web page with an attachment that handles Web navigation and one that displays URLs when the mouse rolls over a hyperlink.

THE CURRENT USER INTERFACE

In the long run it may be that the most important thing to come out of our experiments with the Web are the new user-interface design and information architectural insights we've gained. While it's early to fully evaluate those, in the meantime we have built a fairly remarkable Web browsing environment. Let's walk through a typical user experience.

The first thing many users do when browsing the Web is to initiate a search using a Web 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 3: 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. 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.

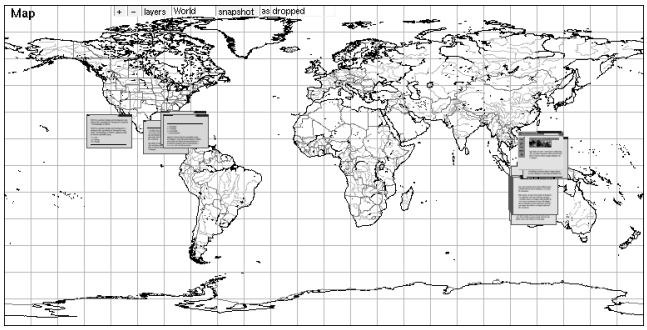

Figure 4: Web pages scattered on a map of the world.

Once the search is narrowed down to a page that the user actually wants to read, he or she can present it in the browser format now familiar to everyone. Links can be traversed with a click, just like in any other browser. Suppose, however, that the user is not interested in all the content on the page. Flashing banner ads can be particularly annoying. In VisageWeb, the user can simply grab the offending content and throw it away.

Content can be taken from the page and analyzed in other frames. While the data may need to be normalized for consumption by other frames, there is no question of data format conversion – every Web page element is a u-form and every Visage frame is designed to interpret u-forms. A script can easily be written in Visage to perform the normalization, making it available for use in other frames. Many general frames, such as the outliner, require no such normalization.

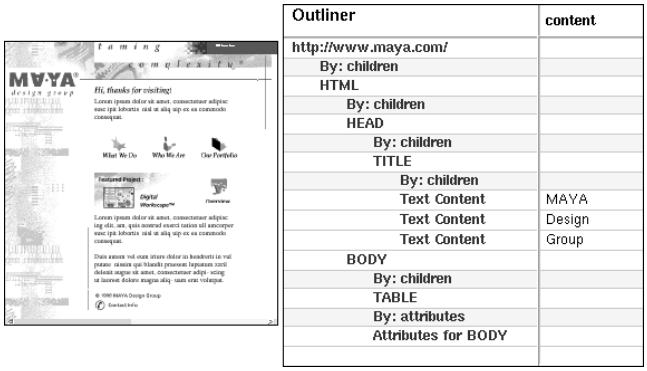

Figure 5: A Web page, left, with an alternate visualization of it in the outliner frame.

New Web pages can be composed of pieces of old Web pages. Each piece "remembers" where it came from, since the u-form contains identity and location information, as described in the information architecture section above. This gives the user a comfortable drag-and-drop interface for building customized Web pages. When it comes time to update the content on the custom page, the heuristic identity analysis described above can re-fetch each piece of content independently and put them together again. Alternatively, the user can copy the Web page in order to update the copy while keeping the original.

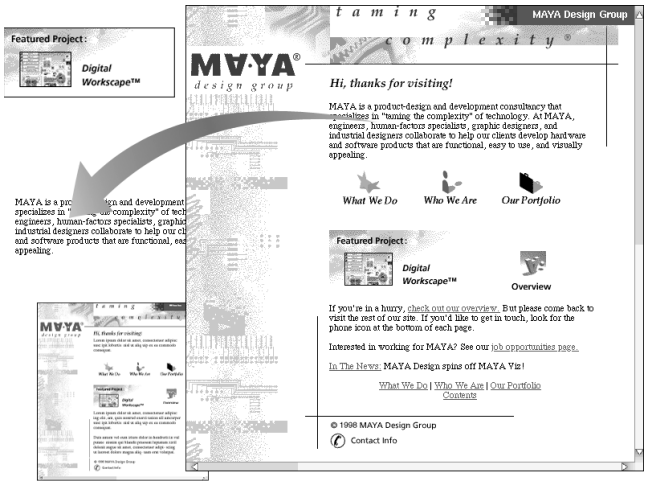

Figure 6: A Web page as retrieved (left), and a larger copy that is being edited using direct manipulation techniques.

Our ongoing research is exploring effective ways to design these heuristics and ways to publish such ad hoc documents back to the Web. Since Visage makes no distinction among frames, a user can perform data analyses on data from any source and create a Web page out of the resulting Visage frames. We now have the technology to publish these pages containing Visage frames to the Web. When normal browsers encounter such documents, they render static visualizations. But when VisageWeb encounters such a document, it resuscitates the frame embedded in the HTML, creating a "live" visualization again. This works in much the same way that older browsers can ignore

embedded Javascript, while newer ones take advantage of it.

CONCLUSIONS

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

Creating this browser 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.

ACKNOWLEDGEMENTS

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, and Susan Salis for editorial assistance. We would also like to thank Jeremiah Blatz and Noah Guyot for help in formatting our screenshots and photographs.

REFERENCES

1. Ballay, Joseph M. Designing Workscape: An Interdisciplinary Experience. *Proceedings of CHI '94* (New York NY, 1994), ACM Press, 10-15.
2. 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.
3. Dertouzos, Michael. *What Will Be*. Harper Collins, New York, NY, 1997.
4. Roth, Steven F., Kolojejchick, Jake, and Lucas, Peter. Information Appliances and Tools in Visage. *IEEE Computer Graphics and Applications* (July/August 1997), 32-41.