Jonathan Isaac Helfman

Candidate

Department of Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication on microfilm:

Approved by the DISSERTATION Committee:

, Chairperson

Accepted:

Dean, Graduate School

Date

Image Representations for Access and Similarity-Based Organization of Web Information

By

Jonathan Isaac Helfman

A.B., EE and Visual and Environmental Studies, Harvard College, 1981
M.S., Computer Science, Columbia University, 1991

Doctor of Philosophy
Computer Science

July 1999

Dedication

For Beth, Rhianna, Elijah, Sheldon, Muriel, and Ilisha.

For the Mother and the Mystery, Earth and Void.

Acknowledgments

Mandala could not have been completed without the continued support of many friends, colleagues, and family members. In particular, Beth Craig has been an invaluable positive force in my life, before, during and after this project. She has provided emotional strength and daily care of myself, our children, and our family. Her skills as a proofreader and copy-editor have improved this manuscript as have many of our discussions about the ideas underlying this project.

Julia Hirschberg and Ron Brachman at AT&T Labs-Research provided me with the opportunity to work on this project and have continued to support my work despite the complications caused by expanses of time and space.

Jim Hollan, at the UCSD Cognitive Science Department, has generously shared his excitement about information visualization, human-computer interaction, and dynamic user interfaces. Jim's continual support and confidence have given this project a fertile environment in which to grow.

David Ackley, at the UNM Computer Science Department, provided periodic sanity checks on the engineering, philosophical, and documentary aspects of this project.

Everett Rogers, at the UNM Communications and Journalism Department, and Barak Pearlmutter, at the UNM Computer Science Department, provided useful

feedback on an earlier version of this document.

Lance Williams, at the UNM Computer Science Department, shared several enlightening discussions about image scaling. Ron Hightower, Tom Kirk, and Chris Faehl helped use early versions of Mandala'a proxy server and image server. Hugh Bivens shared his knowledge of Windows APIs.

The UNM Computer Science System Support Group maintained the machines and networks, which were provided as part of NSF award CDA-9503064, and which formed the virtual environment for developing Mandala. Additional support was provided as part of Darpa grant N66001-94-C-6039: "Beyond Imitation: A Strategy for Building a New Generation of HCI Design Environments." The Administrative staff at the UNM Computer Science Department, particularly Sandy Blanton, Joann Buehler, Liz Lopez-Gutierrez, and Al Gunn, provided administrative support in New Mexico. Marina Storey, Mary Johnson, and Mary Baldwin at AT&T Labs-Research provided administrative support in New Jersey.

Image Representations for Access and Similarity-Based Organization of Web Information

By

Jonathan Isaac Helfman

Doctor of Philosophy
Computer Science

July 1999

Image Representations for Access and Similarity-Based Organization of Web Information

by

Jonathan Isaac Helfman

A.B., EE and Visual and Environmental Studies, Harvard College, 1981
M.S., Computer Science, Columbia University, 1991
Ph.D., Computer Science, University of New Mexico, 1999

Abstract

Although digital and visual technologies are beginning to converge, most digital systems continue to use text to access and organize information. Textual organizations, such as indexes and conceptual hierarchies, are effective for well-structured information, but seem less effective for vast and poorly-structured sources like the World Wide Web, where information exists in a wide range of formats, styles, and combinations. Images provide a complementary way to represent information that may be better suited for helping people use large poorly-structured information sources. Images are perceived quickly, even when reduced in size. Animation, scaling, and layout techniques can therefore help people access masses of visual information with minimal cognitive overhead. Because images play a special role in the function of human memory, hypermedia systems that use images as links to information may be less susceptible to the classic hypertext problems of spatial disorientation and cognitive

overhead.

Mandala is a platform for studying the utility of images for accessing and organizing web information. Mandala lets people view hundreds of thumbnails of web page images. Selecting any thumbnail signals a web browser to display the associated page. Other systems that use images to represent information focus on server-side applications, because images can take a long time to retrieve over the web. Mandala focuses on client-side applications and minimizes retrieval delays by obtaining images from a proxy server cache. Given a collage of images, people have an intrinsic ability to identify groups of related images quickly. Mandala is designed to make it easy for people to share their visually-identified groups with the system by dragging and dropping thumbnails between windows. Mandala also groups different types of images automatically, computes layouts for groups, and stores groups as imagemaps. With these capabilities, Mandala is, at once, a repository for visual site indexes, a visual history facility, a visual bookmark facility, and a cache visualization application. In addition to helping people access and organize information, Mandala's visualizations can increase cache hit-rates and provide shared access to relevant resources, while revealing the dynamic access patterns of a community.

Contents

List of Figures

List of Tables

Part I

Re-Examining Image Representations

Chapter 1

Introduction

1.1 Why Study Image Representations?

One of the primary ways that people experience the world is through visual perception. "We live in a world of things seen, a world that is visual, and we expend much of our physical and emotional energy on the act of seeing. Like fish, we 'swim' in a sea of images, and these images help shape our perceptions of the world and of ourselves" [7, p. 1]. As we see, we understand the world and ourselves visually. Visual technologies amplify the power of visual perception by showing people images of distant galaxies as well as atomic structures. Visual technologies also inform and entertain by displaying multiple images a second to create an illusion of motion.

Visual technologies are becoming digital. Since the recent "revolution" in desktop publishing, most typography and graphic design is prepared digitally. As high-resolution digital cameras become affordable, they become more convenient than film and video cameras. Telecommunications and cable television companies are merging to provide broad-band digital networks to peoples' homes. As advances in hardware and networking continue, visual technologies will continue to become digital, driving down costs associated with image storage and transmission, and providing people with a wealth of digital visual information.

Although visual and digital technologies are beginning to converge, digital information systems still use text to represent, model, organize, and access information. Text dominates our experience of interacting with digital information for historical and economic reasons. Computers were originally developed to manipulate alpha-numeric symbols. Text enjoys an efficient digital representation that can be stored in a minimum of memory and copied across a network at great speed. Digital images, however, require much more memory and bandwidth for storage and transmission. When networks are slow and digital storage is expensive, the utility of image representations is questionable. As digital storage prices decrease and network speeds and bandwidth increase, the utility of image representations should be re-examined.

Today, typical personal computers sell with more than 8 Gigabytes of disk storage, 64 Megabytes of memory, and internet connections to the World Wide Web (a.k.a. the web). The web is a global network that supports many different types of digital media, such as linked documents, linked images, streaming audio and video, and streaming software applications. Web resources are accessed and organized with textual representations.

A web resource is accessed by its URL (uniform resource locator), a string of alpha-numeric characters [9]. When a person types a URL into a web browser, the browser retrieves and displays the associated resource. From the person's perspective, the browser converts the textual representation (i.e. the URL) into its associated information.

Groups of URLs are organized in ways that make sense for textual representations: lists, menus, taxonomies, and indexes. Most web browsers allow people to maintain personal URL collections as history lists and bookmarks. Public link taxonomies, such as Yahoo!, organize URLs into textually labeled topics, categories, or channels. Public search engines build textual indexes from samples of web page text and match textual queries to textually similar web pages.

In some cases, textual organizations seem adequate for large amounts of infor-

mation. Most libraries have adopted databases that mimic the document classification schema of their card-catalog predecessors. The success of these systems make textual indexes seem like an obvious, natural, and inevitable method for organizing information. Libraries rely, however, on a global organization scheme – a single, unifying database.

Textual strategies for organization seem less effective on the web, where there is no single unifying taxonomy or index. Textual queries result in countless unrelated matches, which must be examined sequentially. Link taxonomies must be maintained manually and always seem out-of-date. Many textual labels in our bookmark lists do not contain enough information to remind us why we made the bookmark. Many textual labels on hyper-links do not contain enough information to help us decide if the link is worth following.

Images provide a complementary way to represent information that may be better suited to browsing very large, poorly-structured information sources like the web. In many cases, images from web pages provide good representations for the content of the page. Web pages about people, particularly public figures, almost always contain a photograph of the person's face. Web pages about places almost always contain a photograph of a visible landmark that clearly identifies the place. Web pages about objects, particularly objects for sale, almost always contain photographs of the objects.

Images have become a common strategy for site differentiation, in part, perhaps, because web browsers supported GIF and JPEG image formats before audio formats or even text style-sheets. Images from the same web site often have a shared style that is indicative of the site's agenda. For example, many old-looking black-and-white photographs may indicate a site with historic content, while many adult color portraits with similar framing and lighting may indicate a site for a corporate board or research lab. Even images used for decoration, navigation, and advertisement often provide additional characterizations of a web site (although these images can often

be recognized and suppressed).

Illustrated web pages are only the latest example of a rich history of using images to illustrate and illuminate manuscripts [86, 97]. Illustrations are often designed to convey the essence of their associated text. Many technical documents contain highly descriptive illustrations, charts, or diagrams. For example, multi-media researchers working on a system for delivering chemistry abstracts have found that the illustrations, mostly plots and diagrams of chemical structures, are so important to their readers that "the interface they would like most would be one in which the articles were represented by images, with the text being hidden under icons" [53].

Images and text are different media. They are perceived in different ways. Unlike a textual description, which requires literacy in a specific language, an image can convey information to a wider audience. Representational images of people, places, and things may be perceived immediately, while text is symbolic and must be perceived and decoded linearly.

A person seeing an image can also perceive structural information about the components of an object. For example, "[we] can draw a picture of an unfamiliar object for a friend, and he will recognize the object if it eventually appears. When we simply tell our friend the name for the object, he cannot usually recognize the object when it appears. A picture of an unfamiliar object can tell our friend how many legs the object has, where its arms and neck are, and so on. Presumably, the picture tells our friend about familiar visual elements in a new arrangement. If so, pictures represent parts of objects, not just the whole object" [49, p. 108].

Minimal visual elements convey large amounts of information to people. Even without texture, shading, and color, "lines can depict any of the visible discontinuities of surface, pigment, illumination, and texture layout. These are the basic features that create the visible environment" [49, p. 132].

Although lines may represent objects, their structure, and their surface discontinuities, images often contain much more information than merely lines. Texture,

shading, and color add multiple levels of information. Images may also carry symbolic content. The letters of an alphabet are, of course, abstract images functioning as symbols. Less abstract images also function symbolically, such as government, corporate, athletic, and religious seals and logos. People have a rich history of imbuing representational imagery with symbolic content, which is reflected in art [35, pp. 40-43] and the analysis of dreams [48, p. 55]. Through line, texture, shading, color, and symbolism, detailed images convey a richness and depth of peripheral and subliminal information that can be appreciated immediately, although the details may take longer to interpret.

Image representations scale better in space than textual representations. Textual representations encode most of their information in the high frequencies – edges and corners of the letter forms. Shrinking text by even one quarter of its original size usually makes it illegible on a computer screen. In contrast, many images encode information in the low frequencies – large areas of similar color. Small versions of images often retain enough low frequency information to allow people to recognize the visual content of the original image. Not only can people perceive small images better than small lines of text, but people can perceive more images than lines of text in the same amount of time. Multiple images can be perceived almost simultaneously, while multiple lines of text must usually be read in succession. In this sense, image representations scale better in both space and time than textual representations. A system that uses large displays of hundreds of selectable images to represent information may therefore help people access more information faster.

There is also another way that image representations scale better in time than textual representations. Psychologists have shown that human visual memory far surpasses human textual memory, allowing people to distinguish between familiar and unfamiliar images easily [49, p. 63][94]. Images also improve human memory for associated textual information [74]. Mnemonics, the practice of improving human memory, has an ancient history of using images [106]. Because people can easily

distinguish between familiar and unfamiliar images, a system that represents information with images may help people identify new information (when seeing an unfamiliar image) and find previously accessed information (by remembering and locating a familiar image). Because images help people remember associated textual information, they should serve better than textual labels as cues to help people remember why they made a bookmark and what information was found.

In short, the role of image representations in digital information systems should be re-examined for a number of reasons. People are already comfortable with using visual technology to inform and entertain. As visual and digital technologies converge, economic barriers to using image representations dissolve. On the web, images from a page provide good representations for the content of the page. Images may be perceived immediately, without requiring literacy in a specific language, while text is symbolic and must be perceived and decoded linearly. Images convey a richness and depth of information while revealing additional information if they are studied longer. Images scale better in space and time than textual representations, allowing more images to be seen, appreciated, and remembered in a single instant. A system that uses images to represent information should therefore allow more people to be more productive by accessing and organizing larger amounts of information.

1.2 Why Study Similarity-Based Grouping?

When people have a lot of information to organize, they tend to group similar things together [62]. Grouping by similarity is a natural strategy for organizing physical objects and abstract concepts, as well as digital information. Grouping manages complexity by hiding differences between group members while emphasizing differences between groups.

Most automatic methods for similarity-based grouping of documents (i.e. content-based clustering) use textual similarity metrics [88, Chap. 8]. Automatic methods for

similarity-based grouping of images have also been developed. While these methods have shown some success for particular types of images [36], they usually identify such low-level graphical features (e.g., color and texture distributions) that they rarely seem to group images with similar content [31, 93]. In contrast to automated image clustering algorithms, a person can group images with similar content quickly and easily.

Effective grouping improves access. Information systems that support grouping often allow an entire group to be accessed at once. For example, atomic operations (such as copying, printing, or deleting) can be applied to each file in a directory or to each message in a mailbox. Some information systems display summaries of group members in the form of an interactive overview, such as a bookmark menu, the results of a search, or a table of selectable images. Interactive overviews help people see patterns in large amounts of information without sacrificing access to individual items. Individual items contributing to any pattern can be selected to access their associated information (and perhaps determine the cause of the pattern).

Using images to represent information changes the way information is organized. Gestalt psychologists have observed that people group related visual stimuli with very little conscious effort or awareness [52, Chapter 5, p. 80][100]. Cognitive psychologists have verified these observations with performance data [47, 82] and cognitive neurobiologists have included perceptual grouping in their theories of vision[64, p. 91]. Grouping enables related visual stimuli to be distinguished from their context, allowing people to differentiate objects from backgrounds and, therefore, begin to make sense of their visual environment. Interactive overviews of image representations take advantage of peoples' hardware for pattern recognition, allowing them to identify groups of similar images much faster than they could identify groups of similar textual representations.

Groups of image representations may also diminish several classic problems with hypertext systems (see Section 3.10). Spatial disorientation, which is caused by

unfamiliarity with possibly complex hypertext structures, may be less of a problem when navigating through groups of similar images, which flatten hypertext structure. Because images provide better memory cues than textual labels, remembering which links have been visited, and which were interesting enough to return to, may be less of a problem when using image representations. Saving session summaries and bookmarks as image representations (see Section 3.9) may make people's histories easier to use and maintain.

The focus of this dissertation is on the creation of a system architecture for letting people use images to represent web pages. Although the architecture is designed to facilitate experiments and user studies that could test the utility of image representations (see Section 9.3), these experiments have yet to be performed.

1.3 Why "Mandala"?

A Mandala is a sacred geometrical structure, used for magical, medical, astrological, and religious purposes in South Asia, particularly by Tibetan Buddhists and other followers of the Hindu tantric tradition. Although the literal meaning of the word in Sanskrit is "circle," Mandalas usually have a central axis, which is the focus of concentric circles and quartered squares. Mandalas often include small pictures, which are integrated with the geometry. For Carl Jung and his contemporaries, the Mandala was a symbol of the wholeness of the self. For Zen Buddhists, the circle is a symbol of enlightenment. The system described in this dissertation is named Mandala because it uses geometry to organize information visually in an attempt to augment human memory and intellect.

1.4 Why Mandala?

Mandala was developed for the following reasons (also referred to later as Mandala's "purposes"):

1. To provide an environment in which people can use image representations to access and organize web resources.
2. To explore visual organization strategies for image representations.
3. To provide additional support for image representation applications, such as real-time cache visualizations, visual web site indexes, and visual bookmark facilities.
4. To provide additional support for evaluating image representations and their organizational strategies.

While most operating systems support storage and manipulation of text, there is minimal standard support for caching, compressing, decompressing, shrinking, or combining images. Basic support is also lacking for building web-based information systems that both extract information from real web pages and interact with unmodified web browsers and servers.

Mandala is an extensible framework of reusable components, which provides basic support for studying image representations on the web. One of Mandala's components is an image server, Imago, which is used to generate thumbnails and imagemaps of GIF and JPEG images (see Chapter 6). Image shrinking is supported with a fast and novel hybrid scaling algorithm. Imagemaps can be created using a variety of layout algorithms.

Another Mandala component is a proxy server, Mirage, which provides support for image caching and has been extended to provide support for describing its cache, parsing web pages, and monitoring unmodified browsers and servers (see Chapter 5). A proxy server is a program that sits between web browsers and servers. Web browsers are easily configured to make requests through a proxy server, rather than

requesting resources directly from web servers. There are several advantages to using a proxy server (e.g., improved security, content filtering), particularly one with a large cache (e.g., increased bandwidth and access, decreased latency and connection charges). Since they receive both client requests and server responses, proxy servers are uniquely positioned to monitor web traffic. Monitoring web traffic with a proxy server is transparent to users and requires no modifications to their web browser code (see Section 5.2).

Mandala's graphical user-interface (GUI) is designed to support the exploration of visual organization strategies for image representations. Mandala's GUI is supported by a Mandala client as shown in Figure 1.1 (see Chapter 8). Mandala clients allow people to visualize groups of web pages by displaying images from many pages simultaneously. The displays function as visual interactive indexes. Each small image is an interactive representation that can be selected to access its associated page in a regular web browser. Groups of representations may be specified in many ways, such as URLs in a bookmark file, the history of a browsing session, the results of a query, etc.

Mandala provides additional support for applications of image representations. For example, since the proxy server can describe its cache and its caching activity, Mandala functions as a real-time cache visualization application (see Section 3.4). The proxy server's monitoring capabilities also allow Mandala to build a visual history of a web browsing session (see Section 3.5) or to display representations of pages that are reachable from the most recently requested page (see Section 8.10). By grouping image representations from the same web site and storing them as imagemaps, Mandala functions as a repository for visual interactive site indexes (see Section 3.6). As an additional example, Mandala functions as a visual bookmarks application by organizing and presenting images from the pages listed in a person's Netscape bookmarks file (see Section 3.7).

Figure 1.1: Mandala client with three views. Each view displays a group of representations. The rightmost view has been toggled to display only textual representations.

Finally, Mandala functions as a testbed for future research in web-based visual information systems (see Chapter 9). Mandala provides facilities to support future experiments for evaluating image representations and their organizational strategies. For example, the views in the graphical interface can be toggled to display representations using images or text, which could facilitate comparisons between visual and textual representations (see Figure 1.1). Mandala components log interactions, allowing detailed analysis of user behavior. In addition, the proxy server provides several opportunities for study. For example, the cache can be loaded with particular sets of pages before being disconnected from the web, restricting web access and providing a controlled environment for experiments. The cache size can also be varied, allowing experiments that compare representation type as a function of available information.

This chapter has described how the experience of interacting with digital information is dominated by textual representations and their organizations. Images have been introduced as a complementary way to represent information that may be better suited to browsing very large, poorly-structured information sources like

the web. People have highly advanced abilities for identifying and grouping relevant visual information, which suggests that the design of visual information systems can benefit from user-centered techniques for grouping images. Mandala has been created to provide an environment for exploring the utility of image representations, their organization strategies, and their various applications.

The remainder of this document describes other web access and organization technologies, highlights unique aspects of image representations, and describes how to use Mandala and how it works. Typical approaches for access and organization on the web are described in Chapter 2 and compared to image representations in Chapter 3. Chapter 3 also provides several examples of using Mandala for frequent tasks. General issues concerning Mandala's architecture are described in Chapter 4, followed by specific descriptions of each of Mandala's components: the proxy server (Chapter 5), the image server (Chapter 6), the Mandala server (Chapter 7), and the Mandala clients (Chapter 8). Future work is described in Chapter 9 and conclusions are discussed in Chapter 10. Appendix A describes related systems implemented by the author and their similarities to Mandala.

Chapter 2

Access and Organization on the Web

The web is a collection of technologies for communicating and accessing multi-media information on a world-wide network of computers. In principle, the web's many technologies could provide multiple modes of interaction and information access. In practice, web technologies have not grown much past the relational access method of the selectable hyper-link. Web clients and servers communicate with a protocol called HTTP, the Hypertext Transfer Protocol [33]. Web documents are written in HTML, the Hypertext Markup Language [8, 83].

In this chapter, the earliest visions of hypertext systems are re-examined to emphasize their author's hopes for augmenting human memory and intellect. Hypertext access strategies and the problems experienced while using hypertext systems are also reviewed. In addition, many web technologies are described and evaluated in terms of how well they address hypertext problems and allow people to access and organize relevant information. Although it is useful here to distinguish various technologies, as the web continues to grow and change, boundaries between technologies continue to blur.

2.1 Hypertext Visions

Hypertext was originally conceived of as a technology for augmenting human intellect and cognition. Consider the following seminal titles:

- "As We May Think," 1945, Bush [17].
- "A Conceptual Framework for the Augmentation of Man's Intellect," 1963, Engelbart [30].
- "Literary Machines: The Report on, and of, Project Xanadu, Concerning Word Processings, Electronic Publishing, Hypertext, Thinkertoys, Tomorrow's Intellectual Revolution, and Certain other Topics Including Knowledge, Education and Freedom," 1981, Nelson [70].

By emphasizing thought, intellect, and knowledge, the titles reveal the hope that hypertext could improve cognition by augmenting human abilities to access and organize information.

Bush was motivated by the associational structure of human thought and its potential advantages over current methods of data indexing:

> Our ineptitude in getting at the record is largely caused by the artificiality of systems of indexing. When data of any sort are placed in storage, they are filed alphabetically or numerically, and information is found (when it is) by tracing it down from subclass to subclass. It can be in only one place, unless duplicates are used; one has to have rules as to which path will locate it, and the rules are cumbersome. Having found one item, moreover, one has to emerge from the system and re-enter on a new path.
>
> The human mind does not work that way. It operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain. It has other characteristics, of course; trails that are not frequently followed are prone to fade, items are not fully permanent, memory is transitory. Yet the speed of action, the intricacy of trails, the detail of mental pictures, is awe-inspiring beyond all else in nature.
>
> ...Selection by association, rather than by indexing, may yet be mechanized. One cannot hope thus to equal the speed and flexibility with which the mind follows an associative trail, but it should be possible to beat the mind decisively in regard to the permanence and clarity of the items resurrected from storage. [17, p. 121]

Bush used the associational structure of human thought as a model for his Memex machine. He was hopeful that associational organizations would circumvent the problems of traditional textual organizations. This theme is echoed in Nelson's "A New Home for the Mind?":

> An alternative to this care and feeding of ever more complex systems based on simplistic frameworks is to seek a framework that holds and deals with ideas and their relationships in their natural form and structure, in their full and exact intricacy. To face squarely and early the natural implications of a process brings simplicity in the long run. [71]

Both Bush and Nelson believed that the relational structure of hypertext could model complex ideas better than traditional methods of indexing. The web embodies a colossal challenge to their vision. Most of the web's relational structure is created by authors and publishers who add hyper-links to their web pages, but they cannot possibly link to all other related pages. The widespread use of search engines (see Section 2.8) suggests that readers need to augment the relational structure of authors and publishers. Readers need tools to let them access and organize information in ways that are relevant to their goals and needs.

Another challenge to the power of the relational structure of hypertext is evidenced by the preponderance of broken hyper-links – links that do not point to resources because the resources have been moved or their servers have become inaccessible. Broken links are an indication of the transitory nature of associational structures that Bush had hoped automation would eliminate.

Bush described the process of reading, collecting, and organizing information by adding new links. He called sets of links *trails*, and he proposed that trails could be shared with other people. Bush recognized hypertext as a tool not only for accessing information but also for organizing information with the intent to share it. Bush's Memex was a single-user machine, however, and he did not propose a plan for allowing multiple Memex systems to communicate and share associations. By contrast, Nelson's Xanadu project is a global organizational structure, like the web,

which allows people to reference other people's information. Unlike the web, however, Xanadu is based on a scheme for collection of royalties in exchange for accessed information.

Engelbart's "A Conceptual Framework for the Augmentation of Man's Intellect" envisions further expanding a person's abilities to organize information:

> ...the symbols with which the human represents the concepts he is manipulating can be arranged before his eyes, moved, stored, recalled, operated upon according to extremely complex rules - all in a very rapid response to a minimum amount of information supplied by the human, by means of special cooperative technological devices. In the limit of what we might now imagine, this could be a computer, with which individuals could communicate rapidly and easily, coupled to a three-dimensional color display with which extremely sophisticated images could be constructed.... [30]

Engelbart envisioned a system for augmenting human intellect in which ideas could be represented and manipulated visually. The focus of the present dissertation on image representations follows directly in the spirit of Engelbart's vision.

2.2 Hypertext Access

The relational structure of hypertext encourages information access to take the form of *browsing* although *analytical search* strategies are also possible. Browsing is generally defined as seeking information in an informal and opportunistic manner. Browsing "depends heavily on the information environment" and "coordinates human physical, emotive, and cognitive resources in the same way that humans monitor the physical world" [63, p. 100]. Browsing is especially effective when the information environment is well organized, the information task is poorly defined or interdisciplinary, or when the goal is to gather a range of information about a topic [63, p. 100].

Foss identified at least three advantages of browsing over sequential reading: 1) to provide an opportunity for the reader to learn how data is organized; 2) to learn about concepts that are related to their main topic of study; and, therefore, 3) provide the potential to broaden or redefine a reader's original goals [37].

At least three general browsing strategies have been identified: 1) random browsing, 2) browsing topics of general interest, and 3) directed search [25].

Analytical search strategies are different from browsing strategies [63, p. 73]. Analytical strategies depend on planning and iterative query reformulation, while browsing strategies are opportunistic and depend on a person's ability to recognize relevant information. For example, if a person has to find a picture of a particular chemical compound in a textbook, they could use directed search, by flipping through the book to try to find the figure, or they could use a more analytical search strategy, by consulting the book's index.

Most information-seeking tasks require a hybrid strategy that combines aspects of analytical and browsing strategies. Evidence suggests that users often combine strategies and change their strategies as they encounter new information. Catledge and Pitkow's study of web usage found the most common browsing strategy to be local backtracking – continually returning to a page with many links, such as browsing through the results of a search engine query [21].

2.3 Hypertext Problems

At least two major types of problems with reading hypertext have been identified [24]. They are described as follows:

1. Spatial disorientation
   (a) Where am I?
   (b) Where should I go next?
2. Cognitive overhead [37]
   (a) Trail-blazing: Does the link label provide enough of a clue to decide if it is worth following?
   (b) Lack of closure: Which pages have I already visited? Are there any unvisited relevant pages nearby?

(c) Embedded digression: Have I neglected to return from a digression? Have I neglected to pursue a digression? Now that I am here, what was I going to do?

(d) Session summarization: Can I summarize what I have learned in this session?

Spatial disorientation is caused by unfamiliarity with complex hypertext structures. Most strategies for helping people stay oriented utilize either careful design or graphical displays of hypertext structure (see Section 2.9). These strategies reinforce a navigational metaphor in which the reader moves through structured information. When information is structured in a way that makes sense to the reader, spatial disorientation is diminished.

The problem of where to go next is compounded by the issues of cognitive overhead, the general level of complexity associated with multiple choices. The choice can be daunting whether a person is deciding which hyper-link to select on a particular web page, or using one of the web technologies discussed later in this chapter, which offer numerous hyper-links to other possible destinations.

Trail-blazing, the inability to determine if a link is worth following, is a representation problem. Web page authors, taxonomy maintainers, and search engine designers must each devise adequate representations for links to other web pages. Lack of closure is the inability to determine which pages have been visited or if any nearby, unvisited pages are relevant. Embedded digression is the inability to manage multiple, nested digressions. These problems are minimally addressed by most web browsers, which color the representation of visited links to distinguish them from unvisited links. Session summarization is the inability to save the state of a browsing session. Session summarization is rarely addressed by web technologies, which seem to assume that people will be able to recover the state of a browsing session from strategically saved bookmarks or successful search engine queries.

To a large extent, the classic hypertext problems are consequences of the assumption that brief textual labels on hyper-links can represent web pages effectively.

Mandala avoids many aspects of these problems by using images to represent web pages, because images can contain more information and serve as better memory cues than brief textual labels (see Section 3.10).

2.4 Web Pages: Part Information, Part Index

Web pages containing hyper-links can be thought of as embodying relational structure; they are part information and part index to related information.

The relational structure of the web makes it possible (in theory) for pages to be linked to all other related pages in their "full and exact intricacy" [71]. In practice, however, such pages are highly impractical and unlikely for a number of reasons:

- They would require a very large number of links.
- They would be very difficult to maintain and almost always out of date.
- They would be incomplete: their organization could never anticipate all possible types of relationships for all types of users.

Unfortunately, web pages are not likely to live up to their potential of being linked to all other related web pages. Nevertheless, most of our web technologies for accessing and organizing web pages take the form of new pages that act as partial indexes to related information.

A problem with hyper-links is the volatile nature of the web, which makes it extremely difficult to keep links up-to-date. It is not uncommon for web pages to reference broken links (i.e., links that point to inaccessible or non-existent pages). New pages are constantly added, old pages are constantly changed or removed. In addition, links often seem broken because they link to web servers that have either crashed or are so overloaded that they do not respond to requests.

Some programs have been developed to automatically check for broken links. For example, some search engines have an option to check the web pages that match a query and only report matches that are accessible and actually contain the desired

content (i.e., that have not changed too much) [91]. Unfortunately, requesting these options invariably causes search engines to take a longer time to run. Systems also exist to report any broken links in a person's HTML files [29].

Hyper-links give web pages associational structure. Each web page is part of a larger network of related pages. Web "pages," however, are artifacts of a textual metaphor (as are bookmarks, which are described in Section 2.6). The web browser user interface, the primary technology for viewing web pages, is modeled after a book.

2.5 Web Browsers: The Web One Page at a Time

Most web browsers force people to access the web one page at a time. Access at this limited level of granularity makes it difficult to assimilate information. Sometimes only a fragment of a web page is relevant. Other times relevant information spreads across several different pages at several different web sites. Nevertheless, our technologies force us to access the web one page at a time even though the granularity of web pages may be too large in some cases (when only a fragment of the page is relevant) or too small in others (when relevant information is spread across several pages).

As people find information on the web that is relevant to their particular tasks, they make new associations between pages and partial pages. These associations might not be captured by pre-existing hyper-links. Yet these associations are precisely the structures that are most relevant to people. If it were possible, it would be useful to have web browsers that let people capture relevant structures as they use the web.

Web browsers try to help people capture relevant structures by providing history lists and bookmark facilities. A history list is a list of each URL a person visits. Most web browsers maintain a history list and instantiate it as a menu of web page titles. People can select a title from the menu to access the associated page. Bookmarks are like history lists in that they are a list of URLs that a person has visited,

which is instantiated as a menu of web page titles. (Bookmarks are described further in Section 2.6.) While browsers maintain history lists automatically, people add their own bookmarks to relevant pages. Bookmarks are also stored on disk, so they last from session to session, while history lists last for only one session.

History lists have several problems. One problem is that many web page titles do not adequately represent the content of the page. Another problem is that web browser's often truncate history lists unexpectedly (e.g., after a person visits more than some maximum number of pages). Perhaps a more significant problem, however, is that a history list might not capture structures that are relevant to people's goals and needs. People might want to edit their histories and save them. They might not want web browsers to edit their histories in arbitrary ways.

In fact, people might want a much more general and flexible way to record meaningful structures, such as annotation facilities. Bush considered note-taking one of the fundamental advantages of his Memex system [17, p. 121]. Although support of note-taking is implicit in some of the early, personal hypertext systems, such as NoteCards [40], this capability seems largely forgotten in web-based technologies. Annotation is inadvertently supported by web browsers that provide WYSIWYG (What You See Is What You Get) capabilities for editing HTML [3]. WYSIWYG capabilities allow HTML objects to be copied from one web page and pasted into another. Since hyper-links that are copied still link to their associated resource, WYSIWYG HTML editors can be used to take notes and summarize browsing sessions as interactive "scrap-books."

While WYSIWYG capabilities allow people to save portions of web pages, only the prototype web browsers, DeckScape [14] and Web Forager [19], allow people to group web pages. The idea of allowing users to create sets of pages also existed in earlier hypertext systems, such as the NoteCards *FileBox* [40] and the Hypercard *card stack* [72]. Browsers that allow users to create their own sets of pages are a significant improvement over those that just support history lists and bookmarks.

Grouping allows people to expand the level of granularity from the individual page, allowing them to structure relevant information in meaningful ways. Multiple groups of pages are presented in a single window where pages can be classified. Groups can be scanned, edited, and ordered using direct manipulation techniques. Web Forager can be used with another system that has been designed to provide computational support for organizing groups of pages into categories [79]. Groups of pages can be constructed by using a combination of hypertext connectivity, usage patterns, and inter-document similarity metrics.

Mandala lets people capture relevant structures as they browse the web, but it breaks away from the textual page metaphor entirely. Mandala uses images from visited pages to represent the pages, displaying a visual record of people's history as they browse the web (see Section 3.5). Mandala groups images in several ways, and people can also create their own groups by dragging and dropping images between windows (see Section 3.8).

2.6 Bookmarks: The Web as a Book

Most web browsers let people save URLs of pages that they might need to access again later. These client-based URL repositories are termed *Bookmarks* in Netscape Navigator, although each browser seems to require a different name (e.g., *HotLists*, *Quicklists*, *Favorites*). Bookmarked pages are typically accessed through menus that are maintained by the web browser. The Netscape browser allows people to organize bookmarks into personal concept hierarchies, which are then instantiated as a hierarchical menu. People can group URLs together in folders, which can be named and positioned within a hierarchy and accessed together in the same sub-menu. A personal concept hierarchy is stored as a single web page using nested HTML definition lists. When the page is loaded in a browser, the title of each bookmarked page appears as a hyper-link to the bookmarked page. Bookmark editing facilities

are, therefore, very primitive HTML editors. They provide a simple mechanism for creating personal interactive indexes.

Bookmark facilities give people control of their own URL collections. However, the support provided for maintaining bookmarks is limited. There is little support to help assimilate new material or to help organize and maintain the collections when they grow large. Perhaps it is for this reason that Catledge and Pitkow found that bookmark facilities are rarely used [21].

Some work has been done to improve bookmark facilities. Maarek and Ben Shaul have developed a tool that combines manual and automatic organization facilities for bookmarks [58]. It can be used to automatically classify a new bookmark or re-classify a sub-tree of existing bookmarks. Wittenburg et.al. created a system that merges the personal bookmarks of multiple users [103]. Although individual privacy may suffer, this approach allows people to share the benefits of any work done maintaining individual bookmark collections.

Because individuals are responsible for maintaining them, bookmarks are particularly susceptible to the broken link problem (see Section 2.4). Netscape's newer browsers provide support for identifying bookmarked items that have changed and flagging them with little icons in the bookmarks menu. Other, more sophisticated facilities also exist for detecting changes in web pages [29].

Mandala augments traditional bookmark facilities by converting a Netscape bookmarks file into a series of imagemaps containing thumbnails of images from the bookmarked pages (see Section 3.7).

2.7 Taxonomies: The Web as a Conceptual Hierarchy

Taxonomies, such as Yahoo! and the World Wide Yellow Pages, are pages of links that are organized into conceptual hierarchies. Taxonomies are organized manually by a

staff of people who categorize web pages by deciding where they should be positioned in a concept hierarchy. Categorizing pages is not easy. The quantity of new web pages continues to increase. Some new pages may require augmenting or rearranging the existing taxonomy. Also, it is difficult to categorize pages where people will find them, because a person's mental conceptual hierarchy might not match the link taxonomy. The two concept hierarchies are quite likely to differ structurally, and even more likely to use different category labels [38]. To address this problem, multiple links to the same page (i.e., cross-references) may be positioned throughout the taxonomy, but they may need to be minimized for maintenance purposes.

Taxonomies are often a fine place to begin browsing and are generally useful for accessing popular and older sites, but they rarely have links to sites that are new, unusual, or obscure. Even if they did, the links may be hard to find in cases where the user's mental conceptual hierarchy does not match the conceptual hierarchy of the taxonomy. The more a person uses a particular taxonomy, the more familiar he becomes with its structure, and the more likely he is to find what he is looking for (or at least find where in the taxonomy it would be). However, this effect is somewhat negated by the existence of many differently-structured taxonomies and the tendency for taxonomies to be restructured as they grow.

Some systems exist to help people navigate through taxonomies graphically [4]. If it were possible, it would be more useful to have public taxonomies that were customized to individual's tasks and personal concept hierarchies. Bookmarks can be used as customized taxonomies, but they require users to do all the organization and maintenance. Some taxonomies are searchable and, therefore, customizable within the scope of the existing concept hierarchy.

Mandala's facilities for grouping, classifying, and storing groups of image representations also support personal index creation, while offloading some of the organization and maintenance tasks (see Sections 3.8 and 3.9).

2.8 Search Engines: The Web as a Textual Similarity Structure

While taxonomies are generic structures that are general enough for many different users, search engines are personalized. Because search engines automatically generate new pages of links in response to user queries, they have the potential to provide more accurate access to relevant information than taxonomies. In practice, however, it is difficult for people to formulate queries that retrieve the desired results. While perhaps the most popular web access method, search engines can be quite frustrating to use.

People do not have a good model for how web search engines work. Many aspects of web search engines make it very difficult for people to build a model that could explain how they work. For example, the same query will yield different results on different search engines, because they each index a different portion of the web. The same query may also generate different results at different times, because the web changes and indexes are updated constantly.

It is difficult to try the same query on multiple search engines, because syntactic rules for formulating queries vary across search engines. Some meta-search engines address this problem by querying multiple search engines and eliminating duplicates before presenting the user with the coalesced results [89, 91, 99]. Even so, there are often so many pages matching a query that it is difficult to identify the relevant links in all the noise (i.e., recall is high, but precision is low). This is in part because most web documents are short and most web queries are only one or two words [78]. Also, many likely key-words have multiple meanings, which makes them less effective in queries. In addition, people rarely agree on brief textual names for naming concepts or labeling categories [38].

It is also never clear how much of the web is indexed and how much is not. Many web pages have content that may not be accurately represented by any asso-

ciated text (e.g., images, audio, or video). Many pages will not be indexed at the author's request or because they are generated automatically.

One search engine technology that helps people narrow their search to the information they need is AltaVista's "query refinement" facility (also called LiveTopics, la vache, and cow9) [2]. Query refinement uses text classification techniques to group search results into categories. Each category is labeled with its relative size as well as the key words and phrases that distinguish the documents in that category. For example, an initial query, "image classification," retrieved about 875 documents. Accessing the refine interface for this query indicates that 74% of the results have to do with "sensing." Other categories are also indicated along with their relative sizes (e.g., "unsupervised" 49%, "neural" 47%, and "vegetation" 42%). Query refinement lets people narrow their search to relevant categories. For example, the initial query, "image classification," was made significantly more effective by excluding the terms "sensing" and "vegetation." By grouping search results, query refinement increases the granularity of access. Instead of being forced to inspect each result sequentially, people can manipulate groups of results, focusing on a relevant category or eliminating irrelevant categories.

Although Mandala is better suited for browsing than searching, it also groups representations of web pages. In addition, Mandala allows people to edit, save, and share groups.

2.9 Web Maps: The Web as a Hypertext Structure

Graphical displays of hypertext structure are a common strategy for minimizing spatial disorientation. Hypertext "maps" [40] or "overview diagrams" [72] were popular graphical orientation techniques in early hypertext systems. These maps typically take the form of graph diagrams in which nodes correspond to hypertext pages and

edges correspond to links between pages. Most attempts to add maps to hypertext browsers use one view for the map and a separate view for the current textual page [40]. Other systems include two different types of maps: a global view of the entire web structure and a local view of the current page and its immediate context [72, 105]. A variety of maps have also been implemented on the web [28, 44, 67].

One problem with maps is that they do not scale up – as the number of connected nodes increases, maps become an unintelligible tangle of edges [15]. An overview of the entire web would be too complex to alleviate disorientation. For this reason, web maps may be a natural fit for multi-scale environments [44].

Another problem is that while maps display hypertext structure, they hardly ever display semantic information about the contents of a web page or how the contents of one page relate to any other [37]. This problem has been addressed by visualization research that attempts to combine structural abstractions (filtering, grouping, and hierarchization techniques) with user-selectable binding of visual attributes [67].

Perhaps a more fundamental problem with maps is that their utility hinges on the implicit assumption that the hypertext structure of the web is meaningful and relevant to people attempting to access information. Although this may be true within some well-organized web sites, web structure is so chaotic and dynamic that it seems less relevant to people than the structures they generate as they locate and organize relevant information.

In general, the pre-existing hypertext structure of the web may not be as relevant to readers as the mental structures they create while reading. The results of at least two behavioral studies of how people navigate in information systems suggest that users' own internal models of how information is organized may be more important for successful searching than any graphical user-interface cues designed to aid in navigation [18, 51].

2.10 Push: The Web as a Targeted Advertisement

Some web technologies seem to be modeled after traditional broadcast media. Streaming audio seems to be modeled after radio. Shockwave, Flash, and Mbone seem to be modeled after television. Digital media also provide the opportunity for personalization in the form of targeted broadcasting. Push technology (i.e., point-casting, web-casting, channel-casting, or narrow-casting) is a general term for a collection of web technologies that deliver customized information automatically.

One advantage of push technologies is that critical information can be delivered immediately. For example, software upgrades can be performed as soon as a new software version is released, or a new version of a document can be delivered as soon as it is changed. Another advantage is that most push technologies provide filters for letting people indicate their degree of interest in various types of available information.

One disadvantage of push technologies is that they can consume large amounts of network bandwidth. Push seems most appropriate for corporate intranets, where bandwidth considerations can be more easily controlled. Miramba's push technology focuses on updating software over a network automatically [65]. BackWeb's product focuses on delivering business-critical information to a corporate sales force.

Another example of push technology, PointCast, is a news-feed that takes its input from multiple sources such as CNN, CNNfn, Reuters, PR Newswire, Business-Wire, Sportsticker, and Accuweather [81]. After Dark Online has a similar product in the form of a screen saver. These systems allow people to indicate preferences, which are used to create filters for identifying articles that are likely to be relevant. Headlines of these articles are presented in an interactive display along with advertisements. Selection of any headline causes the associated news story to be displayed.

PointCast is similar to Mandala in a few ways:

- new information is filtered;
- the presentation of information is animated; and

- the representations in the presentation (e.g., news headlines) are selectable to access related and more detailed information (e.g., news stories).

However, PointCast also differs from Mandala in several significant ways:

- PointCast information is organized into channels;
- PointCast filters eliminate information;
- PointCast filters do not help people group information;
- PointCast stores and owns a user's filters; and
- PointCast incorporates and pays for itself through advertisements.

PointCast filters weed-out irrelevant information; they do not help people organize information into groups. PointCast information is already organized into groups or channels, such as "sports," "weather," and "foreign." Pre-structured information can be useful, but it also shares the problems found in public taxonomies: it may be difficult to find relevant information that is new, unusual, obscure, or that does not fit into one of the pre-defined channels.

Proponents of push technology argue that targeted broadcasting will improve information access through personalization, despite the potential annoyance of targeted advertising. The inclusion of advertisements into the PointCast model is an indication that the information being pushed is not under the direct control of the user. Perhaps even more significant is the relationship between the advertisements and the storage of user customizations or filter specifications. In the PointCast model, users send their filter specifications to the PointCast Network. The alleged advantage is that this will help minimize network traffic and allow advertisements to be targeted.

Mandala takes the opposite approach: people should be able to maintain their own filter specifications, make their own decisions about the relevance of information, and have support for grouping the information they collect to better understand and assimilate it.

2.11 Agents: The Web as a Personal Assistant

Although the term "agent" has crept into common usage, few researchers in the field seem willing to provide the term with a definition. When definitions can be found, they seem general enough to encompass most software systems. For example: "Autonomous agents are computational systems that inhabit some complex, dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks that they are designed for" [60].

Perhaps a better way to understand what an agent is supposed to be is through the metaphor of a *personal assistant* that collaborates with the user [59]. Such assistants are smarter and friendlier than typical programs. They are able to learn from a user's behavior, anticipate a user's needs, and return useful results without having to be specifically programmed to do so.

The term "web agent" is typically used to refer to a mobile agent, a process that moves freely through a network by running on different host machines. The major problem for mobile agent architectures is the creation of a secure framework that will allow agents to move to different processors without endangering the processors or the agent. Secure frameworks have been proposed [20], and examples have been implemented [101]. The Java programming language, which is supported by most web browsers, provides security by eliminating pointers, peforming byte-code verification, and forcing untrusted code to run in a protected environment or "sandbox" [34, p. 7]. Although Java protects processors from agents, it does not protect agents, which may need to include sensitive information, such as a person's medical history, or shopping preferences.

The term "web agent" also refers to client-side tools for helping people access web information. For example, Letizia is an "autonomous interface agent" that monitors a user's web browsing activity, pre-fetches pages that are reachable from the browser's current page, and, when asked, is able to suggest pages that are likely to be interesting to the user [55].

The term agent is also used for several programs with considerably less intelligence. For example, *offline agents*, such as FreeLoader, WebWhacker, and WebCompass, fetch groups of connected web pages and store them on the user's local disk where they can be browsed offline. Offline agents typically edit the pages they copy so that links reference other local copies wherever possible. By supporting access to groups of connected pages, offline agents expand the granularity of web access. WebCompass also acts as a personal database that indexes and clusters pages using textual similarity metrics [99]. Personal indexes of relevant information may be a good idea, but it seems unwise to maintain a database of web sites on a person's local disk when a proxy server can perform this function more efficiently.

Mandala's design allows local copies to be temporarily cached by a proxy server (see Chapter 5). Using a proxy server also promotes sharing of cached resources. Once a resource is removed from the cache, only meta-information about it is stored, such as its URL, associated web page, or thumbnail. Like Letizia, Mandala also monitors a user's web-browsing activity and pre-fetches pages, but Mandala lets users select relevant images instead of trying to guess which pages are likely to be interesting (see Section 8.10).

2.12 VRML: The Web as a Virtual World

VRML is a language for defining three-dimensional virtual worlds that can be accessed over the web. VRML worlds may be populated by animating three-dimensional objects. The objects may also be interactive. Selecting a VRML object may *teleport* the viewer to a different location or an entirely different world.

VRML systems seem to have focused on navigation and simulation, instead of providing advanced facilities for organizing information. As Mallen points out, VR seems more focused on the photorealistic simulation of reality through computer graphics than with the simulation of complexity (e.g., statistical data visualization)

[61]. This conclusion is supported by recent work on interactive storytelling with VR [76].

Another problem with many 3-D visualizations is that there is no way to see the entire world at once. VRML, and most other 3-D technologies, transform 3-D models onto a 2-D picture plane by using a perspective projection with a single fixed viewpoint. Objects obscure each other, and it is difficult to see more than a fixed angle of view. These problems are addressed by modifying object parameters (e.g., changing transparency dynamically) and by widening virtual viewing angles (e.g., QuickTime VR's panoramic movies).

There are no technical reasons why a future Mandala client could not be implemented in VRML. VRML supports texture-mapping of images onto objects. Texture-mapping could be used to position images on walls within a VRML world. VRML also supports *billboards*, which are two-dimensional objects that reorient themselves to be perpendicular to the user's angle of view automatically. Texture mapping and billboards offer intriguing possibilities for integrating visual representations within 3-D environments. The historical success of the mnemonic technique invented by Simonides suggests that people may benefit from storing information in a 3-D virtual world composed of spaces filled with image representations (see Section 3.13).

2.13 Visual Information Systems: The Web as an Image Database

Visual information systems allow people to access large databases of images. Most visual information systems index images and support *visual queries* (i.e., the system will return images that are similar to a given input image or sketch) [54]. Images are indexed in two ways: 1) textually, by identifying (or generating) associated key words for each image, and 2) by extracting features from the image data using various computer vision techniques.

For example, Interpix Software has a product called Image Surfer, which is used by Lycos, Yahoo!, and Infoseek. Image Surfer categorizes images from web pages and lets people browse through categories of images and their associated web sites. Image Surfer also lets people perform key-word searches over the images. Because it supports browsing by category, Image Surfer similar to a taxonomy, but with selectable images instead of textually-labeled hyper-links. As with a taxonomy, new images must be categorized or associated with searchable key words, and it may be difficult to locate information that falls outside the categorization scheme.

WebSEEk is an effort to catalog web images and video based on features and textures that are recognized automatically [93]. WebSEEk has so far cataloged over 650,000 images and videos from many sites on the web. Given a relevant image, images with similar features can be accessed quickly. For example, using an image of a pink flamingo as a query retrieved images of the milky way, a solar eclipse, and polychlorinated biphenyls. Here image similarity is based on similar color and texture distributions, not necessarily on what people would consider similar content.

Image Surfer and WebSEEk seem typical of previous approaches to organizing image databases. Images are either clustered based on automatically-recognizable features such as color and texture distributions or categorized by manually adding textual labels. The former approach seems too low-level to be useful, while the latter approach provides a more accurate assessment of image content at the expense of manual categorization.

Mandala is less concerned with supporting visual queries through feature extraction and indexing than previous visual information systems. Mandala's architecture does not preclude a component for image analysis and feature extraction, but web images are surrounded by a rich context of meta-information, which seems to provide an ample feature set for indexing [87].

2.14 Summary

The original visions of hypertext were of a technology for augmenting human memory and cognition. Relational structure was seen as a powerful organizing principal, as were facilities for letting people organize and share relevant information. In particular, organizing and manipulating information visually was seen as a critical feature of the ultimate user interface for augmenting human intellect.

In practive, hypertext's relational structure facilitates browsing, but leads to problems of spatial disorientation and cognitive overhead. Most web browsers try to help by supplying history lists and bookmarks. Often, however, the information relevant to a person's needs does not coincide with web page boundaries, while most web browsers force people to access the web one page at a time.

Bookmarks, taxonomies, and search engines are technologies to help people access information on the web by creating pages of links to related information. Bookmarks are personal and require individuals to do all the construction and maintenance. Taxonomies are general and may not contain new and unusual links or correspond to every individual's personal mental taxonomy. Search engines generate personal indexes, but it is often difficult to formulate queries that give the expected results. Web maps may improve navigation, but they make the implicit assumption that existing hypertext structure is meaningful and relevant. Push allows for customization of pre-structured information at the cost of enduring targeted advertisements. Agents that are personal information gatherers represent powerful and useful technologies for collecting and organizing textual information, but standards and platforms are still being developed to allow agents to be autonomous as well as secure. VRML could someday replace the textual representations of web pages with 3-D, graphical, virtual worlds, but it remains unclear how VRML will help people organize and assimilate information.

Visual information systems represent a departure from the traditional assumption that digital information be represented textually. Most visual information

systems, however, have adopted textual organization strategies, such as manually-assigned key-words or vectors of automatically-extracted features. In the next chapter, several alternate visual information systems are presented, which use images to represent information. One of these systems is Mandala – the focus of the present dissertation.

Chapter 3

Image Representations: The Web as Interactive Cinema

Most of the web technologies described in the previous chapter use textual representations to model, organize, and access information. This Chapter describes several systems that use images to represent web pages, including Montage, the first version of Mandala. In most of these systems, images are displayed in rapid succession, much like a fast slide show or movie. The main difference is that each digital image is interactive – it can be selected to access associated information. The Mandala platform, the subject of the present dissertation, is also introduced. Mandala allows people to interact with hundreds of animating interactive images while they access the web with their normal web browser. Mandala's capabilities are described by providing several examples of how Mandala is used. Several consequences of using image representations are also described. Image representations diminish several classic hypertext problems. Image representations may also violate authors' intentions and seem to violate authors' rights. Finally, evidence of imagery's effect on human memory is presented, which indicates that people may be more effective at accessing and organizing information when it is associated with images.

Figure 3.1: Montage with Netscape. The sphere image has just been selected in Montage, signaling Netscape to display the associated web page.

3.1 Using Images to Represent Web Pages

Montage is an X Windows application that uses images from a proxy server cache for visualizing web pages (proxy servers are described in in Chapter 5) [42]. Associations of images to web pages are made by parsing the proxy server's log files – images are assumed to be associated with the closest preceding web page from the same web site. Montage displays multiple images at the same time on 8-bit color displays by quickly remapping their colors to use a common colormap. Colors are remapped with an error diffusion algorithm and a very fast closest-color algorithm. Images are presented in quick succession and at random positions. Selection of any image signals a Netscape web browser to display the associated web page (see Figure 3.1).

New images overlap older ones so visibility indicates average recency. The entire collection of images functions as a visual interactive overview for the content of the web pages in the cache. Caches of web pages and images are meant to minimize

Figure 3.2: Visual similarity structures in Montage. The cars form a group of similar images. The small cars form a sub-group.

access delays, but Montage reveals the cache as a community-wide resource. Like a newspaper in an unfamiliar city, the content of the cache illustrates how a community uses information and which information they find important.

Montage continues to monitor the cache, displaying new images as they are cached. Streams of related images identify active hypertext trails (e.g., the cars in Figure 3.2). These patterns provide interactive cues for people with shared interests. When multiple streams of related images appear, Montage illustrates the surfing behavior of a community.

The "Web Page Caricatures" of Wynblatt and Benson also use images from web pages to represent the pages [104]. By over-emphasizing key elements of a web page, Wynblatt and Benson convert web pages into visual representations, which are "easily scanned" and allow people to "more quickly select the relevant documents of interest" [104].

Two other systems also use images from web pages to represent the pages. The PolyNav system, created by Wittenburg et. al., presents web images in quick succession and at random positions to aid in hypertext navigation [102]. The hypothesis

Figure 3.3: Postcard illustrating stardust.jpl.nasa.gov.

motivating PolyNav is that the rapid presentation of images can help people preview previously-structured information. Improved previewing capabilities will help people choose where to go next, and thereby, improve navigation. A VRML PolyNav client plays GIF animations, MIDI and WAV audio files, as well as AVI, MPEG, and QuickTime movies.

The CollageMachine, created by Kerne, presents people with dynamic displays of images and text sampled from web pages [50]. Motivated by postmodern art methods of indeterminacy and found objects, CollageMachine provides an "evolving art work," the goal of which is to "create a user experience of the WWW which is both entertaining and useful" [50]. CollageMachine chooses objects to display based on a combination of user input (selections of previous objects) and random noise functions. While Web Page Caricatures and PolyNav are intended to help people access previously-structured information, CollageMachine uses random juxtaposition to help people define new information structures in a manner that is both visual and serendipitous.

The authors of all three systems note that retrieving images over the web is slow. This problem causes Wynblatt and Benson to claim that their system is

"too slow for interactive use" [104]. Similarly, Kerne suggests that a collage of web images is "an ideal way to use computing resources after initiating a search, and while pausing to perform some other task or get a cup of coffee" [50]. Performance issues also cause Wittenburg et. al. to adopt "preprocessing" strategies and to suggest server-side applications for PolyNav: "Our work is directed towards providers of Web information who are able to select, structure, and cache images in advance in order to support a previewing function over indexed material" [102].

Montage is able to avoid the delays associated with retrieving web images by obtaining images from a proxy server cache. In most corporate network configurations, the files in a proxy server cache are accessible over a local network. Because there are minimal delays when obtaining images from a proxy server cache, there is little reason to limit Montage to server-side applications. The focus of Montage has always been to function as a client-side tool for a community of end-users, readers of web pages, who can use it to access vast amounts of information quickly and with a minimum of cognitive overhead. Like CollageMachine, Montage displays images outside of their intended context and uses random juxtaposition to encourage people to identify new information structures visually. For example, the cars in Figure 3.2 represent a hypertext trail of an anonymous colleague using the cache, which is visible as a cluster of similar images.

Unlike the other systems that use images to represent information, Montage organizes related image representations automatically. Thumbnails of images from the same web site can be annotated by users, positioned using a variation of a 2-D bin-packing algorithm [23], and automatically installed as an imagemap on a web server. These annotated imagemaps are thought of as *Postcards*, illustrating the web site (see Figure 3.3). Postcards function as a visual site index – they give a visual overview of a web site while preserving interactivity. Postcards can be shared by sending a colleague the Postcard's new URL. Postcards are useful for illustrating web site reviews and ratings.

Figure 3.4: Mandala client with six views. Each view displays the members of a different group (clockwise from lower-right): the cache group, a user's group, a visual site index for pds.jpl.nasa.gov, the imagemap group, a visual site index for bluenote.com, and a session history group.

3.2 Mandala

Mandala is a complete rewrite and generalization of Montage (Mandala's implementation is described in Part II). Mandala uses a special proxy server, which has extensions for HTML parsing, allowing Mandala to determine associations more accurately than Montage. Mandala's proxy server also has extensions for HTTP monitoring, allowing Mandala to work with any web browsers and servers. Unlike Montage, Mandala uses a separate image server with a fast hybrid scaling algorithm for thumbnail generation and several layout algorithms for imagemap generation. Mandala also generalizes Montage's image display capabilities. While Montage has two types of views (the cache view and the web site or Postcard view), Mandala has multiple views that each display the members of a group. Mandala builds and maintains more types of groups

than Montage. Mandala not only maintains a cache group and groups for each web site, but it also maintains groups for individual browsing sessions and groups for each of a user's bookmark categories.

Figure 3.4 is a screen-shot of a Mandala client with six views. Each view displays the members of a different Mandala group. The view in the lower-right, for example, displays thumbnails of each image in the proxy server cache. The view in the lower-center displays the thumbnails of a user's group. The views in the lower-left and upper-center are groups of images from the same web site. These views function as visual site indexes. The view in the upper left displays each Mandala imagemap. Because each imagemap corresponds to another Mandala group, viewing the imagemap group is a fast way to see all the other groups. The view in the upper-right displays thumbnails of images that have been cached as a result of the user's current browsing session.

The following sections describe Mandala's basic capabilities by providing several examples of how Mandala is used.

3.3 Accessing Associated Information

The Mandala server builds and maintains associations of images to web page URLs. New associations are derived from cached and bookmarked pages. Groups of associations are maintained as imagemaps.

Whenever someone double-clicks on an image in a Mandala client view, Mandala identifies the web page URL associated with the selected image and signals a web browser to display the corresponding web page (see Section 8.2 for implementation details of Mandala's access mechanism). For example, Figure 3.5 shows the result of double-clicking on the thumbnail of the Albanian tank cannon in the top-center of the upper-left view – the default web browser appears, displaying the web page with the full-sized image of the cannon.

Figure 3.5: Double-click on an image to access the associated information. In this case, double-clicking on the thumbnail of Saturn in the Mandala client view (on the left) signals the Netscape browser to display the associated web page (on the right).

The ability to access the information associated with any image, in any Mandala view, is what makes Mandala's images function as interactive representations of associated information. This ability adds value to Mandala applications because each image in a cache visualization, visual bookmark, and visual session history can be used as a starting point for accessing the web.

3.4 Cache Visualization

One of the groups that Mandala builds and maintains automatically is the "cache" group. The cache group contains a representation for each image and page in the proxy server cache. Figure 3.6 shows a Mandala client view displaying the cache group. As with a Montage cache view, the images are displayed with random horizontal and vertical positions. Like Montage, Mandala continues to monitor the cache, displaying thumbnails of new images as they are cached.

3.5 Browsing Session Visualization

A Mandala "session" group contains representations of web resources that have been retrieved for web clients running on a particular machine. Session groups are named after the client's machine. For example, in Figure 3.7, a session view is shown on the left, for the group "tattoo.cs.unm.edu" (Netscape's web browser is shown on the right). As a user surfs the web with the browser, the proxy server informs the Mandala server of any newly-cached resources. The Mandala server builds new associations, updates the session group, and informs the Mandala clients, which update any views displaying the session group.

Figure 3.6: Mandala client cache view. Image position is random, but images are displayed as they are cached so visibility indicates average recency.

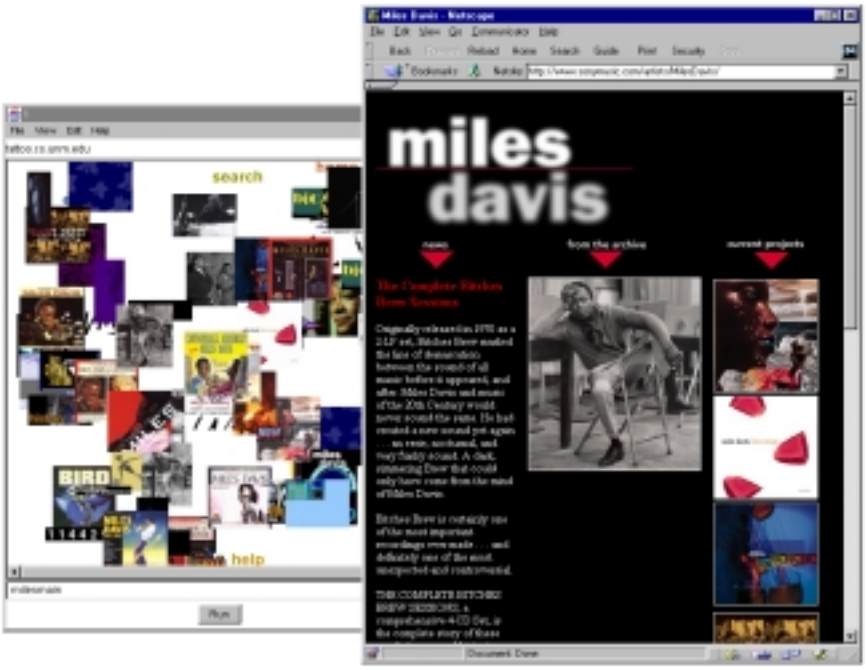


Figure 3.7: Mandala client session view maintains a visual history while you surf. As pages are visited in the web browser (on the right), thumbnails of the page's images appear in the Mandala client's session view (on the left).

Figure 3.8: Visual site index for `pds.jpl.nasa.gov`. A spiral layout algorithm positions large, colorful, square images near the center to make it easy to see most of the information in a single glance.

Figure 3.9: Visual site index for `www.carta.org`. Another example of the spiral layout algorithm.

Figure 3.10: Visual bookmark for `jon/People`. Multiple invocations of a spiral layout algorithm cluster images from the same web site.

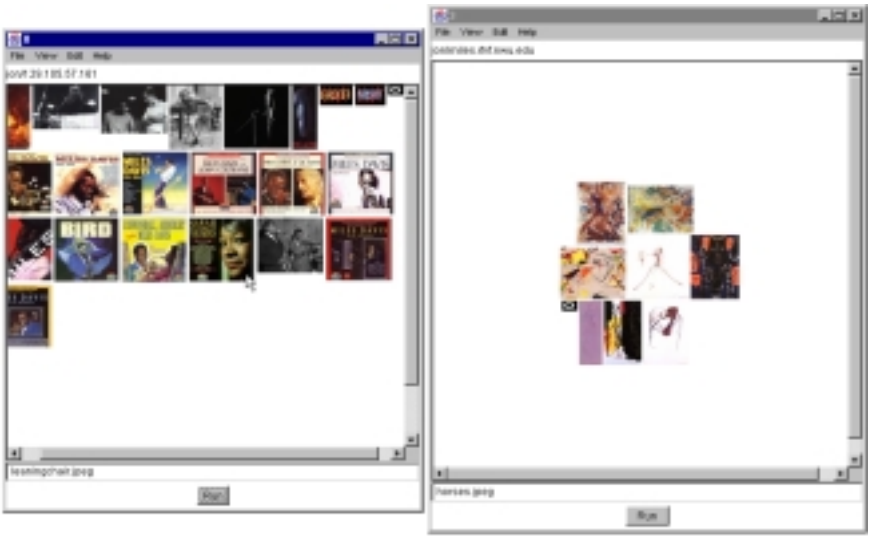
3.6 Web Site Visualization

Mandala groups representations from the same web site automatically. Representations from newly-cached pages or previously-stored imagemaps with the same host part of their image URLs are collected in a group (which is named after the host part of their image URL) and saved together as an imagemap. The imagemap is a visual, interactive index for the web site. Visual site indexes archive image representations – they continue to grow as new parts of the web are explored. Figure 3.8 shows a visual site index for pds.jpl.nasa.gov. In this case, the image layout is computed using a spiral algorithm, which positions large, colorful, square images near the center (see Section 6.2). Another example of a visual site index is shown in Figure 3.9. In general, visual site indexes flatten a site's hypertext structure, allowing immediate access to pages that may be many links from the site's main page.

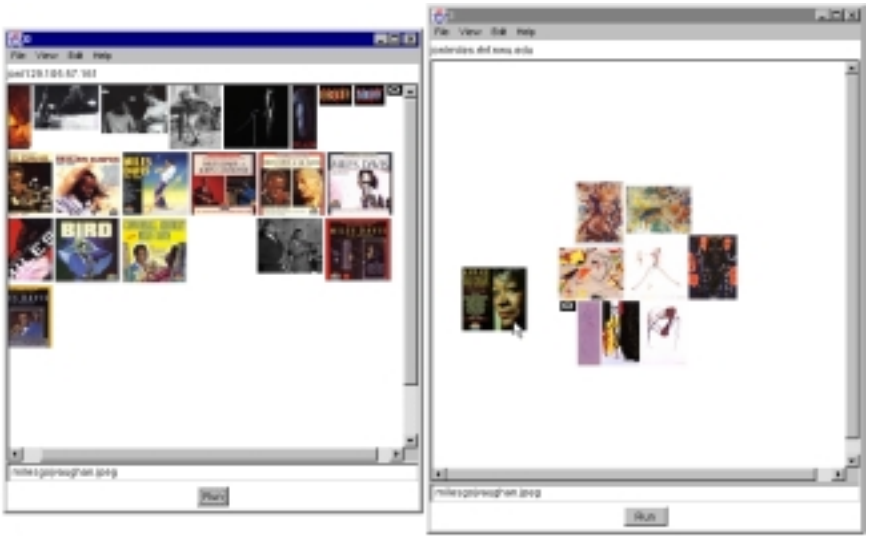
Figure 3.11: Visual bookmark for `jon/News`. Another example of the double-spiral layout algorithm clustering images from the same web site.

3.7 Bookmark Visualization

Netscape's browsers allow people to store their bookmarks in different categories and access them through hierarchical menus where each category is a separate sub-menu (sse Section 2.6). Bookmarks and categories are stored on disk in an HTML file. Mandala reads a user's bookmarks file and builds a new group for each category. For each bookmarked page in a category, if the page can be accessed, Mandala adds thumbnails of images from the page to the category's group. For example, Figure 3.10 shows a Mandala group corresponding to the category "People," which was created from the bookmarks file of user "jon." In this example the thumbnails are positioned using multiple invocations of a spiral layout algorithm, which clusters images from the same web site and centers the largest clusters within a larger spiral (see Section 6.2).



a) The thumbnail image of Sarah Vaughn is selected and dragged to the right.



b) The thumbnail is dropped in a different view causing Mandala to remove the associated representation from the group shown on the left and add it to the group on the right.

Figure 3.12: Edit groups by dragging and dropping images between views.

3.8 Editing and Defining Groups

People edit groups by dragging and dropping images between views. For example, in Figure 3.12 upper left, the thumbnail image of Sarah Vaughn is selected and dragged to the right. The result is shown in the bottom of Figure 3.12 where the Sarah Vaughn representation has been removed from the group on the left and added to the group on the right. Holding down the control key while dragging an image representation into a new view causes a copy of the representation to be added to the new view's group (see Section 8.8).

Because it is possible to see many images at once, it is also possible to recognize patterns of similarity that segment a collection of images into groups. For example, images of cars form a group in Figure 3.2. Images of people form a group in Figure 3.6. It is also possible to see structural relationships between and among groups of similar images. For example, in Figure 3.2, small cars form a sub-group of the images of cars. In Figure 3.6, images of the president form a sub-group of the images of people. It would be difficult to see any structural relationships if the web pages represented by their images in Figures 3.2 and 3.6 were instead represented textually.

The same drag-and-drop facilities used to edit are used to define new groups, allowing people to share their visually-perceived groups with the system easily. For example, if a person determined that the images of cars in Figure 3.2 formed a relevant group, they could share the group with the system by opening a new, empty view and dragging the car images into it.

3.9 Saving a Group as an Imagemap

People save groups as imagemaps by selecting the "save" option from the "view" menu (see Section 8.6). For example, in Figure 3.13, a group displayed in a Mandala client view has been saved as an imagemap, which is shown in the Netscape browser on the right. Imagemaps provide Mandala with a persistent representation for groups

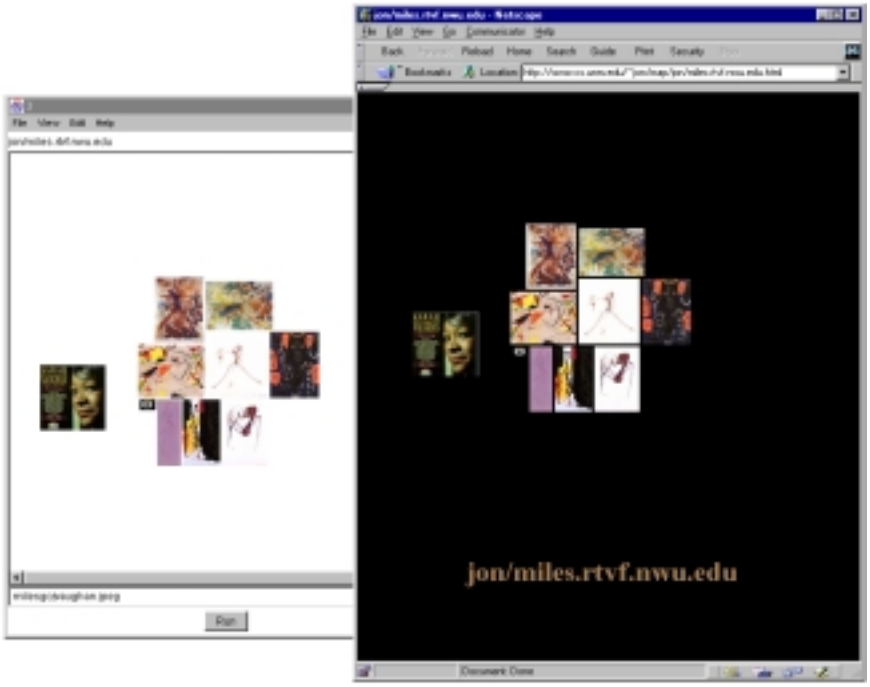


Figure 3.13: Save a group as an imagemap.

of selectable images that can be used in any web browser. Since an imagemap acts as multiple selectable images that can each link to a different URL, imagemaps make it easy for people to maintain their groups in a form that preserves interactivity.

3.10 Diminishing Classic Hypertext Problems

Image representations diminish several of the classic problems identified with accessing hypertext (see Section 2.3). Image representations change the way the web is seen and understood, which changes the nature of reading hypertext. Because image representations de-emphasize hypertext structure, people navigate through clusters of similar images instead of structures of hypertext links. Images may provide better navigational cues than textually-labeled links, because images have been shown to improve human memory for associated information [74]. Instead of asking "where

am I?" and "where should I go next?" users ask "where were those pictures of ___?" Locating a familiar group of pictures is facilitated by animating displays of multiple small images (see Section 8.5). Layouts that preserve or emphasize grouping can also help people locate images (see Sections 6.2 and 8.4).

Trail-blazing is a representation problem that is hardly ever addressed on the web. It is entirely up to web page authors to devise adequate representations for links to other web pages. The trail-blazing problem may be diminished when using image representations, because an image has the potential to provide a better indication if a link is worth following than a brief textual label. It is also possible to pre-fetch images to facilitate trail-blazing by illustrating hyper-links automatically (see Section 8.10).

The lack-of-closure problem and the embedded digression problem are significantly changed in systems that use image representations. Because people have a remarkable memory for images, they can distinguish quickly between familiar and unfamiliar images [94]. A system that uses image representations may therefore help people identify new information, because new information will be associated with unfamiliar images. It may also help people find previously accessed information, because people could remember and locate familiar images. Instead of wondering about digressions or visited pages, users utilize their visual ability to associate images with information. They see a familiar image and have a very good chance of recalling the associated information. They see an unfamiliar image and wonder "what else is associated with this strange image?"

The hypertext problem of session summarization, the inability to save the state of a browsing session, is alleviated by storing the image representations associated with a browsing session as an imagemap, which provides a visual session summary that preserves the interactive nature of the browsing experience.

3.11 Violating the Author's Intentions

Removing images from their pages and letting people identify their own structures may violate the intentions of web page authors and designers. This violation is intentional. The similarity structures that can be visually perceived in a collage of images will no doubt be different from the structures revealed by other web technologies (e.g., the conceptual hierarchies of taxonomies, the similarity structures of search engines, the hypertext structures depicted in web maps). Visually-perceived structures will vary from viewer to viewer, while hypertext structures and textual similarity structures will not. One person looking at a collage may see pictures of buildings, but another person who knows about buildings may see one group of post-modern buildings and another group of Greco-Roman buildings. Visually-perceived structures depend on the viewer's past experience and would, therefore, seem to be more relevant to the viewer than structures defined by taxonomy architects, textual clustering algorithms, or web page authors.

3.12 Violating the Author's Rights

In some situations a publically viewable cache may seem like an invasion of privacy. A publically viewable cache eliminates private sources of information, which may be sources of power. There may also be a concern that cache visualizations might be used to monitor the surfing behavior of unconsenting employees. The only basis for this concern with Mandala is that Mandala's proxy server logs transactions. There is no specific information in the log files about individual users. There is, however, a field in each log entry that identifies the host computer that initiated each request. Although Mirage ignores the host information, in some cases it may not be difficult to convert the host name to a user name. The very existence of these log files may, in some organizations, be considered a violation of privacy.

One consequence of Mandala's architecture is that it can avoid relying on proxy

server logs to associate pages with their images (as did Montage, the first version of Mandala, see Section 3.1). Mandala uses the proxy server's HTML parser to identify the image references on web pages (see Section 5.3).

3.13 Images and Memory

There is a rich history of using visual representations to augment human memory. Imagery has been used within memory enhancing strategies dating back to at least 556-468 B.C. when a particular mnemonic process was invented by Simonides of Ceos (according to Cicero, Quintillian, Pliny, and others). Carefully documented by Yates, who terms it "the Art of Memory," this mnemonic process used mental images of things in places to help orators remember long speeches [106]. Images were chosen to evoke memories of things. The places had an associated natural linear order (i.e., a path connecting each place) that stored the linear order of the memories. Enormous amounts of information could be memorized and recalled in linear order by mentally traveling along the path, allowing each place to evoke the chosen images, which in turn evoked the next set of desired memories. Some examples of remarkable memories that were probably enhanced by this technique (or its variations) include the elder Seneca, a teacher of rhetoric, who could repeat two thousand names in the correct order. Augustine tells of his friend Simplicius, who could recite Virgil backwards. Cicero writes of Charmadas at Athens and Metrodorus of Scepsis whose memory skills were "almost divine " [106]. Metrodorus, Yates relates, used a variation of the process that was based on the signs of the Zodiac, an appropriate framework for organizing information because of its natural order and associations with mythic images.

There appears to be strong scientific evidence to support the use of imagery to enhance memory. Several studies in the field of psychology indicate that people have an astonishingly high capacity for remembering images. For example, when 2560 photographs were shown to 21 students for 10 seconds each, performance ex-

ceeded 90% when, after 3 days the students were tested by showing them pairs of photographs, one of which they had seen before, and asked to identify which image they had seen (presentation time could be reduced to 1 second without seriously affecting performance) [94].

These results were repeated on a smaller scale with children (as reported by Kennedy):

> ...Hoffman (1971) presented groups of children as young as three and five years old with 100 pictures and later tested them for recognition. All the pictures were in color, and none had a single dominant object. Each picture had as much diversity of line and color as possible. No inherently attention-grabbing devices like letters or numbers were displayed, nor were people shown in the pictures. Having seen 100 pictures, the children were asked to select from 20 pairs of pictures any member of the pair that had been shown previously. Even the three-year-olds selected 75 percent correctly, and the five-year-olds scored 82 percent. [49, p. 63].

Additional studies indicate that images improve recall of word pairs, especially images that illustrate the words interacting. For example, if subjects are trying to remember the word pair "hammer" and "pencil," seeing (or even imagining) a picture of a hammer hitting a pencil makes it easier to remember the word "hammer" when presented with the word "pencil" [11, 74, 85].

The focus of much research in the psychology of imagery is to determine if there are structural or functional differences in the brain for verbal and visual processes. Memory tests are used to help determine if verbal and visual representations are stored or accessed differently. Although a precise characterization of the mechanisms underlying memory is not the focus of the present dissertation, the psychology of imagery provides scientific evidence that visual representations are fundamental to human memory. Visual representations, therefore, are worth exploring for use within systems that extend human memories by helping people access and organize information.

3.14 Summary

This chapter has described several systems that use images to represent web information. One of these systems, Mandala, is the focus of the present dissertation. Most of these systems allow the web to be accessed as a stream of images, much like a digital video sequence or movie. Selecting any image, however, signals a web browser to display the associated page.

The authors of some of these systems argue that image representations are best suited to server-side applications such as previewing previously-structured information because of the delays associated with retrieving images over the web.

The Montage system avoids retrieval delays by obtaining images from a proxy server cache. Montage is a client-side tool to help people identify patterns of relevant information on the web with minimal cognitive overhead.

Mandala, which is an improved version of Montage, also uses a proxy server and is also able to avoid retrieval delays for cache visualizations. Mandala has additional capabilities for organizing groups of images and their associated web pages. Mandala groups images from categories of bookmarked pages, from the same web site, and from the same browsing session. Mandala uses spiral layout algorithms to visually cluster related images. In some cases retrieval delays are unavoidable, such as when accessing images referenced by pages in a user's bookmarks file. In most other cases, however, a fast image scaling algorithm and a dynamic view updating policy allow Mandala to function as a client-side tool with minimal delays.

Several consequences of using image representations have also been described. Image representations diminish classic hypertext problems. A small image can store more information than a small string of text and therefore has the opportunity to provide a more evocative representation for associated information. People also group similar images visually. Visually-defined groups of image representations will vary with each person's particular experience and current task. Visually-defined groups may therefore be more relevant for people than pre-defined hypertext structure. Inter-

acting with groups of images shifts the activity of information access from the logical untangling of hypertext structure to a more direct mode of visual sense-making that makes better use of human visual memory and pattern-recognition skills.

Image representations also violate authors' intentions (and may seem to violate author's rights) by providing a powerful technology for readers to re-interpret the web according to their own needs. Readers need such tools because although the web is dynamic and chaotic, most web technologies are designed to help web page authors and web site publishers. When it comes to making sense out of the web, readers are own their own.

Finally, the profound effect of imagery on human memory has been described. People cannot only remember a lot of images, they can also use images to help them remember associated information. Because images are such effective cues for memory, people may be more effective at accessing and organizing information when it is represented by images.

Part II

A Platform for Using Image Representations

Chapter 4

Introduction: System Architecture

4.1 Technical Challenges

Creating a system that uses images to represent web information requires overcoming several technical challenges. Additional challenges must be overcome in attempt to support Mandala's particular purposes (see Section 1.3). Mandala's technical challenges are as follows:

1. How to best support digital image compression, decompression, scaling, grouping, and layout?
2. How to allow people to interact with the web in their normal fashion, with their familiar web browsers, in real time or with minimal noticeable delay?
3. How to build associations between images and the web pages that they represent?
4. How to group associations in meaningful ways?
5. How to display multiple, selectable images so people can identify groups of similar images quickly and easily?
6. How to let people share their visually-identified groups with the system?
7. How to let people edit, save, and revisit groups of images?
8. How to visualize a cache, web site, and bookmarks file?

An additional technical challenge was to write as much of Mandala in Java as possible, as an experiment to see if Java was up to the task.

The technical challenges listed above have been addressed by Mandala's modular design, which provides a flexible and general platform for future visual information research on the web.

4.2 Modular Design

Given a wide range of technical challenges, it is not obvious how to begin designing a system. Mandala's design began in a bottom-up manner by exploring different possibilities for overcoming one of the most significant challenges, number 2, allowing people to interact with the web in their normal fashion.

One possibility would be to adapt the source code for the most popular web browser, but it is rarely clear which web browser is most popular. Modifying source code also has the drawback that modifications must be made to each new release of the browser's code.

A more general possibility would be to use a modified proxy server. A proxy server operates at the level of the HTTP protocol. Situated between web servers and web browsers, proxy servers forward client requests on to servers and forward server responses back to clients. A proxy server could be modified to send summaries of HTTP traffic to separate applications (e.g., Mandala). Using a proxy server meets challenge 2 by allowing people to interact with the web in their normal fashion, because people are free to use their favorite web browser or switch browsers at any time. A proxy server could also be modified to describe its cache, making it possible to visualize the cache and thereby helping to meet challenge 8. Finally, a proxy server could also be modified to parse HTML. For example, by identifying each image reference on a web page, a proxy server could help meet challenge 3.

Building a visual information system around a proxy server has many archi-

tectural advantages. Using a proxy server simplifies the system because no other system component interacts with the web. Isolating web interaction to one component has security advantages when the proxy server is run as part of a firewall [56, p. 5]. The proxy server also caches images and therefore provides essential performance improvements for any system that needs to access web images repeatedly.

Mandala uses a special proxy server, Mirage, which was written entirely in Java. In retrospect, it may have been wiser to modify an existing proxy server, such as Squid [77], which is designed for very fast response time. Implementing a new proxy server, however, provided an opportunity to learn as much as possible about HTTP and proxy servers. Mirage is described further in Chapter 5.

Mirage is useful by itself. It has been used as a caching HTTP proxy server for the University of New Mexico's Computer Science community since September 1998. Mirage is also useful in conjunction with other systems. It can be used as a component of any application that requires HTML parsing or HTTP monitoring capabilities. For example, Mirage is used as a component of PadPrints, a research project at the University of California, San Diego. Mirage monitors a user's requests and informs PadPrints when and how to update its hypertext map of the user's browsing history [44].

Once it was decided to use a proxy server to help meet technical challenges 2, 3, and 8, the rest of Mandala's design began to fall into place.

The second major design decision was to support the image handling functions of challenge 1 (compression, decompression, scaling, etc.) with a second server component written in C. Reasons for writing the image server in C include the absence of GIF and JPEG image compression facilities in Java and the poor reliability of image decompression facilities in Java. By contrast, C code for GIF and JPEG compression and decompression is publically available and reliable. The speed of compiled C is also an advantage for any process that needs to manipulate many large images. Because Imago runs as a self-contained server, its implementation language has no

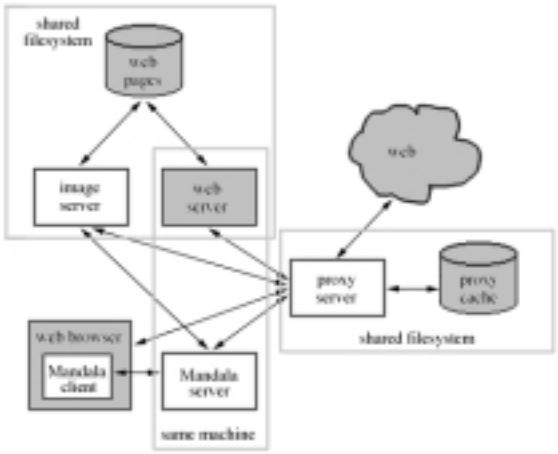


Figure 4.1: Mandala's component structure.

effect on any other part of the system.

The third major design decision was to meet challenges 5-8 by client processes that support various graphical interfaces and afford various interaction modalities. As separate processes, Mandala clients can exist in different environments enabling different capabilities. A Mandala client in the form of a Java applet might be appropriate for very simple applications, such as monitoring a cache, while a Pad++ Mandala client would be more appropriate to explore visual, multi-scale organization strategies (Pad++ is a sophisticated programming environment, which supports image scaling) [6].

The fourth major design decision was to meet the remaining challenge, 3, with a final separate process: a Mandala server. The Mandala server provides a single point of contact for each of the clients. Having a single Mandala server also allows people to share their associations, groups, and imagemaps.

Mandala's component structure is shown if Figure 4.1. Splitting Mandala into components has many other obvious advantages (e.g., each component can run on a separate processor, components can be developed and debugged independently, components can be re-used in other systems, components can be substituted with

others that provide the same programming interface).

The Mandala components are subject to various run-time constraints (indicated by light grey rectangular boundaries in Figure 4.1), which were also a factor in Mandala's modular design. When the Mandala client is implemented as a Java applet, the Mandala server must run on the same machine as the web server due to security restrictions.[1] The image server must be able to write images in the file system served by the web server. This constraint implies that the image server run on a machine that mounts the web server's file system with "write" permission. The proxy server must be able to read and write its cache. If Mandala were written as a monolithic application, it would be subject to each of the run-time constraints. Because Mandala is implemented as separate components, each constraint can be satisfied by a single component, freeing most of the components from most of the constraints.

For consistency, Mandala components communicate by reading and writing HTTP-like messages – messages with ASCII headers and either ASCII or binary data. Web images, for example, consist of an ASCII header followed by binary image data. Reading such a message consists of reading lines of ASCII characters, until a blank line is reached, which signals the end of the header. The binary image data can then be read in large blocks. Interestingly, reading a binary message with an ASCII header amounts to mixing buffered I/O and unbuffered I/O. Buffered I/O is typically used to read lines of ASCII, while unbuffered I/O is typically used to read blocks of binary data.

The DataInputStream class in Java 1.0 read HTTP-like messages quite easily by calling its `readLine()` method to read lines of ASCII and then calling its `read()` method to read blocks of data. Unfortunately, Java 1.1 deprecated the DataInputStream's `readLine()` method and replaced it with a `readLine()` method in the BufferedReader class, which converts 8-bit bytes into 16-bit characters in the name of internationalization. Doubling the size of each byte makes sense when the bytes are

[1] One way around this constraint is to run a local "stub server" on the same machine as the web server and have the stub server forward requests to the Mandala server.

to be interpreted as text (all Java text strings use 16-bit characters), but not when the bytes are to be interpreted as binary data. A new `ByteInputStream` class had to be written to read HTTP messages. It reads from an unbuffered `FilterInputStream` and uses an internal buffer to support reading lines of ASCII via a `readLine()` method. When its `read()` method is called, if there is any valid data in the internal buffer, it is returned. Otherwise, the `FilterInputStream.read()` method is called to read and return bytes without buffering.

Mandala has been fully implemented. Mandala's architecture allows it to function in multiple ways (e.g., as a GUI for organizing and maintaining information represented by images, a visual bookmark facility, an imagemap editor, a cache visualization tool, a visual look-ahead cache).

Mandala's architecture also supports collaboration. Multiple Mandala users can take advantage of each other's browsing activity by viewing each other's thumbnails from the proxy server cache. They can also take advantage of each other's group definitions by viewing each other's imagemaps. Imagemaps are useful for saving and sharing interactive visual summaries of any set of URLs. This design is efficient for users, who can easily share the product of each other's work. The design is also efficient for Mandala and the web, because it promotes accessing locally-cached resources, which can be provided quickly, without increasing web traffic.

This chapter has been an overview of the technical challenges posed by Mandala and how they were met by a modular design. The following chapters describe the functions of each Mandala component in greater detail.

Chapter 5

Mirage: Mandala's Proxy Server

This chapter describes the reasons for using and extending a proxy server. Mandala's proxy server, Mirage, is described, as are several issues related to the implementation of Mirage's cache and Mirage's extensions for monitoring HTTP and parsing HTML. Details related to using Mirage are also described.

A proxy server is a program that sits between web servers and web browsers or other web clients. Requests are copied from the clients to the servers. Responses (or errors) are copied back to the clients. There are several compelling reasons for using a proxy server.

Proxy servers were originally developed to run on firewall machines so people on a LAN or intranet could surf the web while maintaining a measure of security[57, Sec. 1.1]. Proxy servers are also useful for monitoring HTTP requests and responses. For example, proxy servers can be used to prevent children from accessing unauthorized sites or to filter out advertisements from retrieved pages.

Not long after they were deployed on the web, proxy servers were extended to *cache* web resources (mostly HTML pages and images) by copying them to a local disk. Caching increases a proxy server's value to its community of users: if someone requests a resource that happens to be in the cache (a cache *hit*), it can be copied over the local network without having to connect to the remote server and copy the

resource over the web. The percentage of requests satisfied out of the cache is called the cache *hit-rate*. Cache hit-rates have been measured between 30-50% [1, 39]. These measurements indicate that resources are requested repeatedly and caching is worth the trouble. The time spent waiting for a response to a request (i.e., the *latency*), is reduced when requests are satisfied from the cache. Similarly, the overall bandwidth of the web is increased, because fewer resources are copied over the web. Access to web resources is increased, because if the remote server is inaccessible, requests can still be satisfied out of the cache. Proxy caching can save more disk space than client caching, because only one copy of popular resources is saved. A caching proxy server also saves money – the entire country of New Zealand surfs the web through proxy servers primarily to save money on long-distance connection charges [69].

Several researchers have argued that extended proxy servers are valuable tools for supporting a wide range of web utilities and applications. In 1995 extended proxy servers were considered to be *HTTP Stream Transducers*, because they modify information within the network, adding value by changing the information to a more suitable form (e.g., filtering out advertisements, filtering out images for bandwidth-limited clients)[13]. More recently, proxy servers have been considered to be *Intermediaries* for producing and manipulating web content in a variety of applications (e.g., personal history managers, cookie managers)[5].

A number of proxy server extensions have been implemented. Some extensions store and organize the history of a series of browsing sessions to enrich the browsing experience, such as annotation servers, full text indexers, and subject-matter indexers[5, 13, 27]. Other proxy server extensions curtail the browsing experience. Commercial proxy servers, used by internet service providers, have been extended to filter objectionable web sites as a service for parents[46]. These various extensions indicate the flexibility and power of a system architecture for web research that uses a proxy server.

Mandala's Proxy Server is called *Mirage*, as a reminder that the cache is just

request:	argument:	description:
APPLICATION	-	register as an application connection
ANCHOR_FILTER	web page URL	return link references
FRAME_FILTER	web page URL	return frame references
IMAGE_FILTER	web page URL	return image references
TITLE_FILTER	web page URL	return page title
DUMP_CACHE	-	return ASCII cache listing
HTML_CACHE_SUMMARY	-	return HTML list of freshest resources

Table 5.1: Mirage's Request Interface

a representation of the web. Many researchers believe that the continued growth of the web will not be possible without a mesh of cooperating proxy servers to minimize latency and maximize bandwidth. When most pages are served from proxies, it may become increasingly difficult to know if the pages are up-to-date or even if they still exist on the origin server. The problem of how to keep cached files up-to-date is discussed further in Section 5.1.1.

Mirage supports the standard HTTP requests: `GET`, `POST`, `HEAD`, `PUT`, `LINK`, `UNLINK`, `DELETE`, `OPTIONS`, and `TRACE`. The most common of these is the `GET` request for retrieving web resources.

Mirage's extensions are accessed through the same TCP socket interface as the standard HTTP requests. They use the same HTTP-like messages composed of ASCII headers with optional binary or ASCII data. Mirage's extended requests are listed in Table 5.1. The `APPLICATION` request allows separate applications to monitor HTTP traffic (see Section 5.2). The `ANCHOR_FILTER`, `FRAME_FILTER`, `IMAGE_FILTER`, and `TITLE_FILTER` requests are for parsing and filtering web resources (see Section 5.3). The `DUMP_CACHE` and `HTML_CACHE_SUMMARY` requests allow Mirage to describe the contents of its cache.

5.1 Cache

Once a user's web browser is configured to use a proxy server, web resources will get cached automatically. The two main problems for a caching proxy server are 1) how to determine if a cached file is fresh? and 2) which files to remove when the cache gets full? In addition, caching significantly complicates the most common HTTP request: `GET`. Caching also requires storing meta-data about cached files, which complicates the design of any platform-independent cache implementation. Mirage's solutions to the problems of determining freshness, removing cached files, implementing `GET`, and implementing a platform-independent cache, are addressed in the following subsections.

5.1.1 How Long to Cache?

Freshness of cached resources is difficult to determine, because few servers use the `Expires` HTTP header, and there is no way for a web server to inform a proxy server when a resource has been updated [66]. When the `Expires` header is not used, Mirage uses a typical hueristic, which estimates freshness time as a factor of the last-modified time [56, p. 163]. If the server does not transmit a last-modified time, Mirage assigns the resource a maximum age (the maximum age is currently set to 3 days).

The pseudo-code in Figure 5.1 sketches the implementation of a subroutine that determines if a cached file is stale. In this example, `last_checked_time` is the time the file was last checked on the original server. The `last_checked_time` starts out being the time the file was first cached, but it is updated for each conditional `GET` (see Section 5.1.3, below).

The `last_modified_time` is the time the file was last changed on the server. This value is usually specified by the web server in the file's HTTP header as the value of the `Last Modified` field. If the file arrives without a `Last Modified` header field, Mirage sets `last_modified_time` to the value of the current time. The value of

```
file is stale{
  if(the file has an Expires value){
    if(expires_time > time_of_request)
      return false
    else
      return true
  }
  else if(the file has an Last-modified value)
    freshness_time = last_modified_factor * // default 0.1
                  (last_checked_time - last_modified_time)
  else
    freshness_time = maximum_age  // default: 3 days
  if(last_checked_time + freshness_time >= time_of_request)
    return false
  else
    return true
}
```

Figure 5.1: Pseudo-code to determine whether a file is stale.

`last_modified_time` is updated each time the file is copied from the server (e.g., if a conditional `GET` returns the entire file).

5.1.2 How to Uncache?

When the cache gets full, the proxy server must decide which resources to discard. A common policy is to discard resources based on their *age* or recency of use. This policy is based on the assumption that if a resource has not been requested in a long time, it is unlikely to be requested soon. This policy is also called LRU, because the **l**east-**r**ecently-**u**sed resource is discarded. Some researchers have found LRU to result in more cache hits than other approaches [80]. Another policy, called *size*, is to discard the largest resource, because it will create the most free space in the cache. An improvement over *size* is to discard the resource with the highest value of *size*age*, where *size* is the file size in bytes and *age* is as above. A slightly more complicated policy takes into account the fact that, on the web, some servers may respond very

slowly, even if they are serving relatively small files. This policy discards the resource with the highest value of *size*age/cost*, where *cost* is the time needed to fetch the document from its original web server [75].

Much work has been done evaluating the performance of proxy server caching policies, but little of this work evaluates, or even describes, the implementation of these policies as algorithms (e.g., [75, 80]). A natural way to implement an uncaching algorithm is to keep an ordered list of resources, sorted by a value corresponding to the policy's particular metric or score (e.g., *size*, *age*, *size*age/cost*). Once the list is sorted, objects with the highest scores can be quickly identified by consecutive entries at one end of the list. The list takes space (list entries might be the characters of the object's URL and its numeric score), and it also takes time to maintain as the cache changes. When new objects are cached, their score must be calculated and their entry must be inserted at the right position in the list. When cached objects are accessed (e.g., a cache hit), their score must be recalculated and the file's entry must be removed and re-inserted.

A cache algorithm that must maintain a list may be a problem, because when the proxy server is busy, and the cache is full, it should be able to uncache immediately, without having to take time to calculate a score and maintain a list.[1] Fortunately, LRU has an implementation that is more efficient in both space and time when compared to other algorithms.

In an LRU implementation, the list entries may be sorted by increasing access time (i.e., the time the object was last accessed). Entries at the beginning of the list (those with the smallest access time) are the oldest and may be removed when the cache gets full. The time required to maintain the LRU list is minimal. When a new object is cached, its score does not need to be calculated, its position in the list does not need to be determined, and it never needs to be inserted in the middle of a list. Because it must be newer than the other cached files, its entry can be appended to

[1] An alternative design would use a separate process or thread for uncaching, but coordinating multiple processes could add unneeded complexity.

the list. For a cache hit, the object's entry must still be removed from the list, but its score does not need to be recalculated, its new position in the list does not need to be determined, and it never needs to be inserted in the middle of a list. Because a cache hit changes a resource's access time to the current time, the resource must be newer than the other cached files, and its entry can be appended to the list. Because there is no need to recalculate scores, there is no need to store them in the list. The position of an item in the list is sufficient to store its relative age.

5.1.3 How to `GET` when Caching?

If a proxy server did not have to cache, retrieving resources would always consist of requesting them from the associated web server (or another proxy). Caching significantly complicates resource retrieval. For example, the decision to cache a file usually requires examining the characters of the requested URL and the response code, as well as the values of several response fields (e.g., `Expires`, `Pragma`, `Cache-control`).

The decision to use a cached file usually requires the values of the `Pragma` and `Cache-control` request fields and the values of the `Last-Modified` field of the cached file (as well as, possibly, the time the cached file was last accessed and the time its freshness was last checked on the server).

Figure 5.2 provides a sketch of how Mirage implements `GET`, the most common HTTP request. If the cached file can be used, it must first be determined whether or not it is stale (see Figure 5.1). If the cached file is stale, its last-modified time can be checked on its associated web server by performing a "conditional `GET`" (see Figure 5.3).

A conditional `GET` request is a `GET` request that uses an `If-Modified-Since` field with a value taken from the `Last-Modified` field of the cached file. If the server sends back a `200 Ok` response, the body of the message contains the updated page and should be copied to the client, but if the server sends back a `304 Not Modified` response, the cached file is still fresh.

```
if(cached file exists && request permits)
 if(get file from the cache)
   if(cached file is stale)
     conditional get
get file from server
```

Figure 5.2: Pseudo-code to determine how to `GET` a web resource.

```
conditional get
  update header
    set If-Modified-Since value to
      Last-Modified value or current time // use GMT
  get file from server
  if(response_code == 304){ // Not-Modified
    get file from cache
    update cached file's Last-Checked value
  }
  else if(response code == 200) // Msg body is new object
    read file from stream
```

Figure 5.3: Pseudo-code to conditionally `GET` a web resource.

One trick to getting web servers to respond accurately to conditional `GET` requests is to convert the date to a fixed-width interpretation of the RFC 1123 and RFC 822 date specifications [12, Sec. 5.2.14, pp. 55-56] [26, Sec. 5, p. 26]. The date must be converted to Greenwich Mean Time, being sure to use two digits for the month and day of the month (e.g., "02" for the second month, or day of the month, instead of "2").

5.1.4 Cache Implementation

A major challenge for a proxy server cache design is handling meta-data for cached resources. In particular, the file's access time is required to implement the uncaching algorithm described in Section 5.1.2, and the file's last-modified and last-checked times are required to determine if the file is stale as described in Section 5.1.1. These

times must be updated appropriately. For example, the file's access time must be updated each time it is copied out of the cache.

Storing the cached file meta-data in a big list in memory would take up space and require a strategy for saving it on disk. Ideally, no such list should be necessary. In the case of access time, for example, it may be possible to use the file's last-accessed time as stored by the operating system. An early version of Mirage determined access time with a C native method that called the UNIX™ `stat` command to obtain the file's UNIX access time (there is no way to obtain a file's access time with Java!). Unfortunately, native methods must be compiled and configured as shared libraries on each platform they are to run on. This usually requires figuring out how shared libraries work on each platform. In addition, while native methods could also be used to obtain a UNIX file's last-modified time, the last-checked time must also be stored to compute accurate freshness times.

To store times consistently and avoid the platform dependencies of a native method, each cached file in the current version of Mirage stores its last-accessed, last-checked, and last-modified times in its own header fields. Times are converted to Greenwich Mean Time and stored using the same fixed-width interpretation of the RFC 1123 and RFC 822 date specifications as used in conditional `GET` requests (see Section 5.1.3). The times are stored in fixed-width fields, so they can be updated without ever having to rewrite the entire file.

One disadvantage of this approach is that when the proxy server starts, each cached file must be read to determine its access time. For very large caches, the startup time can be several minutes. To minimize the agony associated with such a long startup time, the access list is initialized in a separate Java thread. As soon as Mirage starts, it begins honoring requests without using its cache, while the thread initializes the access list in the background. When the thread is finished determining the access time of each cached file, it sets a flag, which indicates that the cache is ready for use.

The many advantages of this approach outweigh the disadvantage. The advantages are that 1) no meta-data needs to be stored in a special file, 2) all three times are stored in a consistent manner, 3) the last-checked time is stored so Mirage can determine freshness times more-accurately, and 4) Mirage can run on both Windows and UNIX machines.

5.2 Monitoring Requests

The proxy server allows separate processes to monitor the access patterns of regular web browsers. Once a process makes a socket connection to the proxy server and identifies itself as an *application*, it can read messages on the socket to determine when other processes obtain web resources from the proxy server.

Applications connect on the same port as browsers, but send the ASCII string `"APPLICATION\n\n"` before listening. While Mirage normally closes socket connections after sending a response, an application connection is only closed when the application disconnects.

There can be multiple applications, but Mirage currently keeps track of the name of the requesting machine and makes the assumption that only one web browser will be running on the requesting machine (it should probably use a real client identification scheme, such as client-side cookies, to avoid having to make this assumption).[2]

Mirage writes strings on the application channel after successfully reading the response data. For example, here is a typical line of output announcing the successful retrieval and caching of a GIF image:

```
thread: Thread-1572 client: surf.cs.unm.edu url: \
http://www.metacreations.com/home_ui/mid_left_p55_on.gif type: \
image/gif rc: 200 ref: http://www.metacreations.com/ cache_status: WRITTEN
```

[2]Client-side cookies are identification strings that are sent from web servers in HTTP headers, stored in a file on a client's disk, and sent back to web servers in the HTTP headers of each client request.

error code:	error type:	description:
502	Bad Gateway	can not write request
500	Internal Server Error	can not read response

Table 5.2: Mirage's Error Interface

option:	argument:	type:	default:
-p	port	int	8888
-l	log directory	string	-
-d	cache directory	string	-
-s	cache size	int	8000000

Table 5.3: Mirage's Command-Line Options

To ensure that the proxy server receives each of the web browser's requests, the browser must be configured to use no other caches (e.g., by setting the size of the browser's memory and disk caches to zero). In this way an application can monitor a regular web browser to determine its currently viewable web page. Mandala uses Mirage's cache monitoring extensions to build groups of associations from the same browsing session (see Section 7.11).

5.3 Parsing HTML

The design of the HTML parser was more challenging than expected. The parser must be able to return the text within any unpaired tag (e.g., `img`), as well as the text within and between any paired tags (e.g., `title`). The parser must also be able to resolve relative URLs correctly, which includes identifying and honoring any `<base>` tags (which change the root of all subsequent relative URLs). In addition, the parser must be able to identify multiple nested tags in the same pass. For example, to find all image references, not only must the `<img>` tags be identified, but it is also useful to determine if the `<img>` tag occurs within a pair of `<a>` tags, indicating that the image

is a link to another resource. Furthermore, images can also be referenced within the `<body>`, `<table>`, `<tr>`, and `<td>` tags. Finally, the parser must be able to handle a wide variety of anomalies permitted by the HTML specification, including 1) optional quotation marks around URLs, 2) optional newline, linefeed, and carriage return characters within tags and between tag pairs, and 3) a poorly-specified comment syntax.

Mandala's HTML parser borrows a design concept from Steve Uhler's HTML displayer, which is written in Tcl and renders HTML/2.0 documents into a Tk text widget [96]. Each time the parser is called, it returns a "chunk" consisting of a tag, a hash table of any attribute/value pairs associated with the tag, and any text up to the start of the next tag. To parse an entire page, the parser is called repeatedly. This design keeps the low-level inner loop simple and general. Any support for paired or unpaired tags must be implemented by the code that calls the parser.

Any URLs returned by the parser are first fully-qualified. For example, relative URLs with occurrences of "." and ".." are resolved; URL fragments (i.e., URL suffixes beginning with "#") are removed; and URLs beginning with a "/" are appended to the protocol and hostname. The code for fully-qualifying URLs is complex. Network Working Group RFC 1808 "Relative Uniform Resource Locators" section 4: "Resolving Relative URLs" provides a seven-step recipe for the subtle code needed to get the right answer [32]. Implementing this algorithm in Java is straightforward, but clumsy. The exercise demonstrates that Java could benefit from the sort of regular-expression support found in other programming languages (e.g., Tcl/Tk, Awk, Perl, Unix Shell).

The parser must also address the problem of parsing HTML comments. This is complicated by the fact that although HTML comments were meant to take the form of SGML comments, web browsers have to ignore SGML comments and devise their own, ad hoc, HTML comment interpretation. The HTML 2.0 specification states "A comment declaration consists of '`<!`' followed by zero or more comments followed

by '>'. Each comment starts with '--' and includes text up to and including the next occurrence of '--'. In a comment declaration, white space is allowed after each comment, but not before the first comment" [8, Sec. 3.2.5]. One of the main uses of comments in HTML, however, is to hide Javascript code, which can contain any number of ">" and "--" substrings.

Web browsers have adopted their own strategies for parsing HTML comments. For example, when Microsoft's HTML parser reads the string "<!--", it interprets everything up to the next "-->" or "->" as a comment. When Netscape's parser reads the string "<!--", it interprets everything up to the next "-->" as a comment. If there is no next "-->" in the file, everything up to the next ">" is interpreted as a comment. This behavior may be useful if an author forgets to terminate a comment, but it is an example of the sort of problem encountered when trying to parse a dynamically-evolving language with a lenient specification.

Mirage parses HTML comments by mimicking the behavior of Netscape's parser. When Mirage's parser reads the string "<!--", everything up to the next "-->" is copied into a buffer. If the end of the file is reached before reading "-->", the parser copies the buffer, converts it to a new input stream, and reads from the stream, interpreting the first ">" as the comment terminator. Copying and converting the buffer to a new input stream allows the parser to handle multiple, poorly-terminated comments.

The HTML parser is written as a set of reusable Java classes. For example, the visual bookmarks process uses the parser classes to parse a user's bookmarks file (see Section 6.8). Mirage uses the parser classes to implement four new parsing services: `ANCHOR_FILTER`, `FRAME_FILTER`, `IMAGE_FILTER`, and `TITLE_FILTER`.

The `IMAGE_FILTER` command takes a URL as an argument, retrieves the associated resource, verifies that it is an HTML web page (i.e., its mime-type is "text/html"), and returns URLs of each referenced image, including images used as anchors. The `ANCHOR_FILTER`, `FRAME_FILTER`, and `TITLE_FILTER` commands work the same way,

returning each link, each frame reference, and the web page title, respectively.

The code for parsing HTML is complex, so keeping it in one place is very helpful for systems and applications that need to extract information from web pages. For example, putting parsing capabilities in the proxy server simplifies the Mandala code. Mandala never needs to retrieve web pages. The Mandala server builds associations between web pages and images by issuing `IMAGE_FILTER` requests to the proxy server. It builds associations between web pages by issuing `ANCHOR_FILTER` requests. In most cases these commands are likely to be fast, because the requested web pages will already be in the cache.

5.4 Usage Details

This section describes miscellaneous detail associated with using Mirage, such as its log file format, error format, and command-line options.

Although the proxy server does not log the names of users requesting web pages, it does log a variety of information related to each request using a common log format [98].

Normally, Mirage forwards a request from a client to a web server, and then forwards the response back to to the client. If the web server encounters an error while trying to service the request, Mirage forwards the error back to the client. If Mirage has trouble communicating with a web server, however, it will generate its own error. Mirage errors are listed in Table 5.2. Mirage generates `502` errors if it cannot connect or write to the web server. Mirage generates `500` errors if it cannot read the web server's response.

Mirage honors the command-line arguments listed in Table 5.3. These can be used to modify various default configuration parameters. The `-p` option specifies the port number that Mirage should use to listen for requests. The `-l` option specifies the directory that Mirage should use to store its log file. The `-d` option specifies the

directory Mirage should use to store its cache. The `-s` option specifies the size of the cache in bytes.

5.5 Summary

This chapter has described proxy servers as software components that were originally designed to offer a measure of security and, with the addition of large caches, have since become useful for improving web access. In general, proxy servers increase network bandwidth and access speeds while reducing latency and access charges. Because proxy servers intercept HTTP traffic between web browsers and servers, a number of researchers recommend extending proxy servers to further improve the web access experience.

The implementation of Mandala's proxy server, Mirage, has been described. Proxy servers need to determine how long to cache resources, and which resources to uncache when the cache gets full. Mirage uses standard freshness heuristics (Section 5.1.1) and an optimized LRU uncaching algorithm (Section 5.1.2). Caching significantly complicates a proxy server's most common request – `GET`. The implementation of Mirage's `GET` request has been described, including details of implementing conditional-`GET` (Section 5.1.3). Mirage stores various cache-related times in the header fields of cached files, allowing Mirage to run on UNIX as well as Windows machines (Section 5.1.4). Mirage's application interface allows separate applications to monitor HTTP traffic (Section 5.2). Mirage also parses HTML, handling a wide range of variable syntax due, in part, to HTML's lenient specification (Section 5.3). Details were also described related to using Mirage, such as Mirage's log format, command-line options, and errors.

Chapter 6

Imago: Mandala's Image Server

This chapter describes Mandala's image server, *Imago*, which provides services to clients that manipulate web images. Sections are included for each of Imago's requests. Imago's `GET_THUMB` request generates thumbnails, small versions of any web images, with a fast and accurate scaling algorithm. Imago's `MAKE_MAP` request generates imagemaps, single images with multiple selectable areas. Imago's `GET_INFO` request returns meta-information about images, such as an image's width, height, color space, and number of colors. The `GET_MAP_INFO` request returns meta-information about the imagemap's image as well as the coordinates and associated URL for each of the imagemap's areas. The `GET_RATED_MAPS` requests returns names of highly-rated maps, where an imagemap's rating is the average of it's thumbnails' ratings, which are, in turn, a function of the thumbnail's area and number of colors. In addition, Imago acts as a limited HTTP proxy; the `GET` command retrieves any web resource, a capability that may simplify the design of some clients. After describing each of Imago's requests, additional sections describe Imago's errors and command-line options. Applications that use Imago are also described, as are issues related to Imago's implementation language.

Clients communicate with Imago through an HTTP-like protocol (requests contain ASCII headers with optional ASCII or binary data). Like Mirage, Imago's

request:	argument:	return value:
GET_THUMB	image URL	thumbnail URL or data
MAKE_MAP	map name	imagemap URL or data
GET_INFO	image URL	list of attribute/value pairs
GET_MAP_INFO	stub name	list of attribute/value pairs
GET_RATED_MAPS	integer n	first n map names sorted by rate
GET	URL	associated resource

Table 6.1: Imago's Request Interface

protocol extends HTTP by supporting additional requests (see Table 6.1). Imago's requests take a single argument and, in most cases, consist of a two-line header (the second line is blank and signals the end of the request; the version is optional and typically omitted). Imago's responses consist of a two-line header (the second line is blank) and a message body, which may be either binary image data, an ASCII list of attribute/value pairs, or an HTTP error. Imago's requests are described in the following sections.

6.1 GET_THUMB: Thumbnail Generation

Imago creates thumbnails according to a client specification, which takes the form of a GET_THUMB request. The simplest form of GET_THUMB request is a URL for an input image. For example:

```
GET_THUMB http://www.cs.unm.edu/~jon/test.jpg
```

Note that the request is two lines long. The last line is blank. This request yields the following output (also two lines long):

```
GET_THUMB 200 http://www.cs.unm.edu/~jon/thumb/www.cs.unm.edu/g.100._7ejon_2ftest.jpg
```

Imago caches thumbnails by first building a new name. The new name is used to determine if the requested thumbnail already exists. The name is built from the

attribute:	value:	description:
return	bits	return the thumbnail's image data
return	url	return the thumbnail's URL
replace	create	ensure thumbnail exists
replace	unique	store with a unique name
replace	overwrite	ensure updated thumbnail exists
filter	none	uniform point sampling with no filtering
filter	gaussian	fast HDC halving, with minimal Lanczos
filter	lanczos	sharp, slower, for arbitrary scaling
thumb_max_size	integer	integer dimension of thumbnail's largest side
thumb_min_width	integer	lower bound on thumbnail width
thumb_min_height	integer	lower bound on thumbnail height
thumb_src_min_width	integer	lower bound on input width
thumb_src_min_height	integer	lower bound on input height
thumb_src_min_colors	integer	lower bound on input total-colors

Table 6.2: Imago's Thumbnail Specification Interface

URL of the input image, the first character of the filter type ('g' in this case for the gaussian filter option, which is described below), and the digits of the maximum dimension (in this case the default maximum dimension of 100 is used). If the thumbnail does not yet exist, Imago obtains the input image, scales it to the specified maximum dimension (while maintaining its aspect ratio), and installs the result on a local web server. Imago then returns the URL of the thumbnail to the client and closes the client connection. The next time the thumbnail is requested, it will be available from the local web server and will not need to be recomputed.

Clients can override the default scaling behavior by specifying the maximum thumbnail dimension, filter algorithm, return style (whether to return the URL or the data), and replacement style (whether to overwrite an exiting thumbnail of the same name or generate a new name). Additional options ignore input images that do not meet minimal requirements for size or number of colors. Thumbnail specifications are formed from combinations of the attributes and values listed in Table 6.2. For example, the following thumbnail specification specifies several non-default options:

```
GET_THUMB http://www.christusrex.org/www1/sistine/0-A.jpg
filter: lanczos
thumb_max_size: 75
replace: overwrite
return: bits
```

The above specification will cause Imago to shrink the input using the `lanczos` filter (described below) so that its largest dimension is 75 pixels. The `overwrite` option instructs Imago to generate the thumbnail even if it already exists, while the `bits` option instructs Imago to return the compressed thumbnail image data, instead of merely the thumbnail's URL.

Imago's `none` filter option scales images using the fastest and most common algorithm: uniform point sampling. For example, if the input is four times the size of the desired output, then the uniform sampling algorithm will select every fourth pixel from every fourth scan line of the input.

Uniform sampling may be adequate for blurry images, but it violates sampling theory for inputs with high frequencies. In the previous example, where the input is four times the size of the output, patterns of light and dark in the input that alternate at rates higher than every eighth pixel will not be sampled at a high enough frequency and are likely to result in jagged artifacts as the high input frequencies *alias* to lower frequencies in the output (see Figure 6.1.a).

Aliasing artifacts associated with uniform sampling can be minimized by expanding each sampling point to a weighted sum of pixel values in the neighborhood around each point. Sampling theory indicates that neighboring pixel values should be weighted according to an infinite sync function, centered at the sample point. In practice, sync functions are approximated with finite functions. Many different finite functions have been proposed for approximating sync functions [95]. They are typically instantiated as a real-valued matrix, which is thought of as a weight matrix or a filter kernel. One typical optimization is to use filter kernels that are separable, so the two dimensions of the input image can be scaled in separate passes.

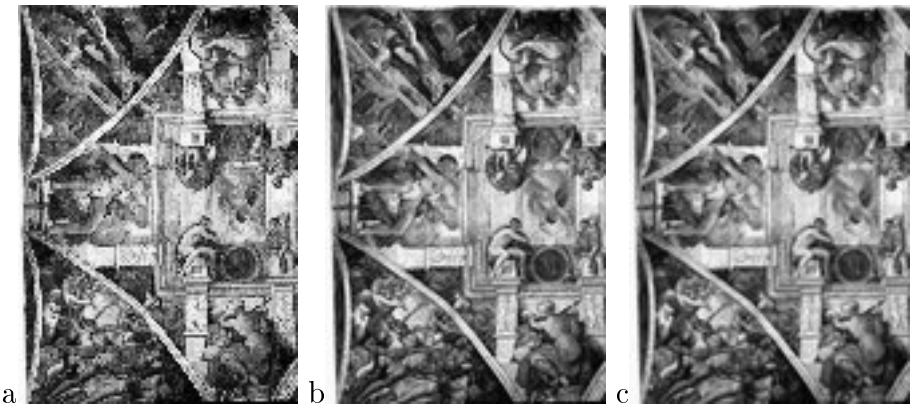


Figure 6.1: Output of scaling by a factor of 5.8 with different filters: a) `none`, b) `lanczos`, c) `gaussian`.

Sampling theory dictates using kernels that are large enough to sample at twice the rate of the highest possible input frequency. Large kernels can be accurate, but they can also make the process of shrinking an image very slow.

For example, Imago's `lanczos` filter option uses a general image rescaling algorithm that creates and scales filter kernels based on the scale factor [90].[1] This algorithm uses separable kernels and, as an additional optimization, precomputes the kernel contributions for each row and column in advance. Despite these optimizations, however, scaling images with the `lanczos` option typically takes five to ten times longer than uniform point sampling (the `none` option).

Imago's `gaussian` filter option invokes a novel hybrid scaling algorithm that is both fast and accurate. Imago uses a form of hierarchical discrete correlation (HDC) that scales an input image by one half [16]. The HDC kernel approximates a Gaussian function when it is applied multiple times. To scale images by arbitrary amounts, Imago uses successive invocations of HDC and a single invocation of the general image rescaling algorithm with the Lanczos kernel. The possible slowness of the final pass is minimized, because scale factors are guaranteed to be less than one half. Scaling

[1] Although the underlying algorithm is general enough to use a wide range of kernels, it is currently set to only use a Lanczos kernel.

images with the `gaussian` option typically takes only a few seconds longer than uniform point sampling, but it produces results that are difficult to distinguish from the slower `lanczos` option.

Imago's filter options are visually compared in Figure 6.1, which shows the output images for an input measuring 616x877 pixels, using a scale factor of approximately 5.8. Aliasing artifacts are noticeable in the result of uniform point sampling (Figure 6.1a). While it is difficult to distinguish differences in the other results, in general the `lanczos` filter makes images look very sharp because it tends to "ring" (it passes resonant energy in its stop band), while the `gaussian` filter makes images look a little less sharp because it tends to "blur" (it does not pass enough energy in its pass band).

6.2 `MAKE_MAP`: Imagemap Generation

Like thumbnails, Imago creates imagemaps according to a client specification (imagemaps are single images that act like multiple, selectable images by associating image areas with URLs). A minimal imagemap specification is a stub name for the output and a list of URL pairs for web images and associated resources. For each URL pair, Imago attempts to add a thumbnail of the web image to the imagemap and store the URL of the associated resource in the imagemap's HTML file. For example, the following is a `MAKE_MAP` input specification:

```
MAKE_MAP imagemap_example
area: image=http://www.cs.unm.edu/~jon/k11thumb.gif href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/tophathumb2.gif href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/montage.thumb.jpg href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/montage/montage.gif href=http://www.cs.unm.edu/~jon/montage/
area: image=http://www.cs.unm.edu/~jon/mandala/image/bhutmara.100.jpg href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/rhianna/glasses.2.jpg href=http://www.cs.unm.edu/~jon/rhianna/rhianna2.html
area: image=http://www.cs.unm.edu/~jon/dotplot/ama.color.gif href=http://www.cs.unm.edu/~jon/dotplot/ama.html
area: image=http://www.cs.unm.edu/~jon/hansard.thumb.gif href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/dotplot/shake.color.gif href=http://www.cs.unm.edu/~jon/dotplot/shake.html
```

When the above request is submitted, Imago returns the following output:

```
MAKE_MAP 200 http://www.cs.unm.edu/~jon/map/imagemap_example.html
```

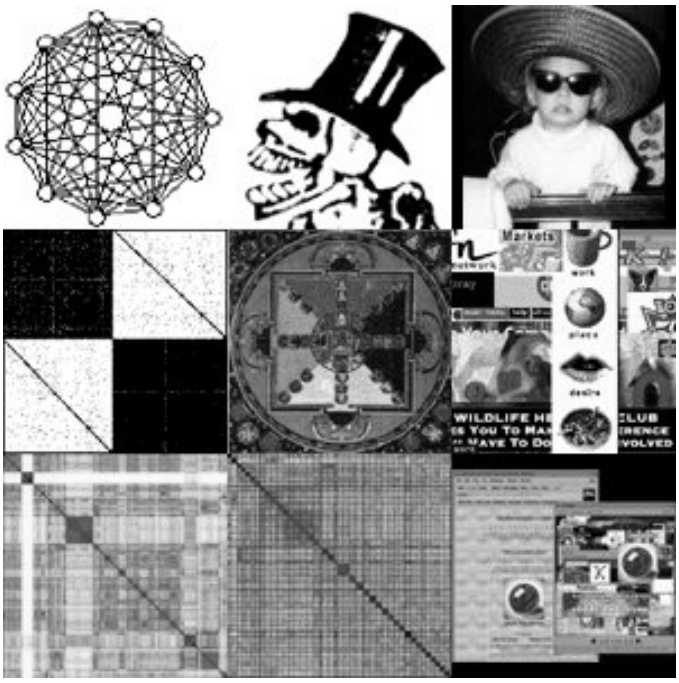


Figure 6.2: Imagemap computed with default parameters.

By default, the imagemap command returns the URL of a new web page that defines the imagemap. In this case, the HTML code for the new web page looks like this:

```
<html>
<head>
<title>imagemap_example</title>
</head>
<body bgcolor="#000000" link="#1f9d00" text="#a67e53" vlink="#4f9177">
<center>
<img src="http://www.cs.unm.edu/~jon/map/imagemap_example.jpg" border=0 usemap="#map1">
<h1>imagemap_example</h1>
</center>
<map name="map1">
<area href="http://www.cs.unm.edu/~jon/"
 image="http://www.cs.unm.edu/~jon/k11thumb.gif" coords="0,0,100,100">
<area href="http://www.cs.unm.edu/~jon/"
 image="http://www.cs.unm.edu/~jon/tophathumb2.gif" coords="100,0,200,100">
<area href="http://www.cs.unm.edu/~jon/"
 image="http://www.cs.unm.edu/~jon/montage.thumb.jpg" coords="200,100,300,200">
<area href="http://www.cs.unm.edu/~jon/montage/"
 image="http://www.cs.unm.edu/~jon/montage/montage.gif" coords="200,207,300,293">
<area href="http://www.cs.unm.edu/~jon/"
 image="http://www.cs.unm.edu/~jon/mandala/image/bhutmara.100.jpg" coords="100,100,200,200">
<area href="http://www.cs.unm.edu/~jon/rhianna/rhianna2.html"
 image="http://www.cs.unm.edu/~jon/rhianna/glasses.2.jpg" coords="205,0,295,100">
<area href="http://www.cs.unm.edu/~jon/dotplot/ama.html"
 image="http://www.cs.unm.edu/~jon/dotplot/ama.color.gif" coords="0,200,100,300">
```

attribute:	type:	description:
background_color	hex Color	color for background
background_thumb_url	image URL	image for background
default_url	URL	for selections off of defined areas
dimensions	width, height	integer dimensions of imagemap
area	(see below)	attribute/value pairs

Table 6.3: Imago's Imagemap Specification Interface (attributes with typed values)

attribute:	value:	description:
replace	augment	add areas to existing imagemap
dimension_type	fixed	respect dimensions values
dimension_type	min-square	use smallest possible bounding box
layout	random	position images randomly
layout	grid	center images in grid
layout	spiral	wrap rated images around center
layout	spiral2	wrap rated images around center
layout	spiral3	recursive spiral2
layout	preset	respect thumb coordinates
rate	area-colors	rating scheme for imagemap
background	solid	respect background_color value
background	tile	tile with background_thumb_url
background	center	center background_thumb_url

Table 6.4: Imago's Imagemap Specification Interface (attributes with literal values)

```
<area href="http://www.cs.unm.edu/~jon/"
 image="http://www.cs.unm.edu/~jon/hansard.thumb.gif" coords="0,100,100,200">
<area href="http://www.cs.unm.edu/~jon/dotplot/shake.html"
 image="http://www.cs.unm.edu/~jon/dotplot/shake.color.gif" coords="100,200,200,300">
</map>
</body>
</html>
```

The web page references the imagemap's image, `imagemap_example.jpg`, which is shown in Figure 6.2. The image is composed of thumbnails of each input image, scaled to a default size (100 pixels), and organized according to the default layout algorithm (`spiral2`).

The `MAKE_MAP` request supports each of the thumbnail attributes listed in Table

attribute:	type:	description:
image	text	original image URL
thumb	text	small image URL
other	text	associated page URL
href	text	same as other
alt	text	comment
coords	text	"x,y,w,h"
max_size	integer	maximum dimension
rate	double	system-computed score

Table 6.5: Imago's Area Specification Interface

6.2, although they change their meaning to apply to the map as a whole. For example, the `return` attribute applies to the imagemap's image data or URL. The `replace` attribute applies to the imagemap's HTML file and its image file. The values of the other attributes are applied to each thumbnail in the map.

The `MAKE_MAP` request also supports the attributes listed in Tables 6.3 and 6.4. Each area within the imagemap must be specified with a separate `area` specification, which is formed from the attributes and values shown in Table 6.5. For example, the following imagemap specification uses the same input images as before, but specifies several non-default parameters. The resulting imagemap is shown in Figure 6.3. It was created by positioning thumbnails randomly over a white background. The thumbnails were computed using the `lanczos` filter and scaled so their maximum dimension is 75 pixels.

```
MAKE_MAP imagemap_example2
layout: random
filter: lanczos
thumb_max_size: 75
background_color: 0xffffff
replace: overwrite
area: image=http://www.cs.unm.edu/~jon/k11thumb.gif href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/tophathumb2.gif href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/montage.thumb.jpg href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/montage/montage.gif href=http://www.cs.unm.edu/~jon/montage/
area: image=http://www.cs.unm.edu/~jon/mandala/image/bhutmara.100.jpg href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/rhianna/glasses.2.jpg href=http://www.cs.unm.edu/~jon/rhianna/rhianna2.html
area: image=http://www.cs.unm.edu/~jon/dotplot/ama.color.gif href=http://www.cs.unm.edu/~jon/dotplot/ama.html
area: image=http://www.cs.unm.edu/~jon/hansard.thumb.gif href=http://www.cs.unm.edu/~jon/
area: image=http://www.cs.unm.edu/~jon/dotplot/shake.color.gif href=http://www.cs.unm.edu/~jon/dotplot/shake.html
```

Figure 6.3: Imagemap computed with non-default parameters.

The `MAKE_MAP` request supports two different values for the `dimension_type` attribute. The `fixed` value indicates that the integer values of the `dimensions` attribute should be respected. The `min-square` value, which is the default, indicates that the dimensions of the imagemap should be the smallest possible bounding box of the thumbnail positions within the imagemap.

The `MAKE_MAP` request supports six different values for the `layout` attribute, corresponding to five different layout algorithms for positioning images – the `preset` value instructs Imago not to perform a layout, but to instead honor the coordinates specified in the `coords` attribute of each `area` specification.

The `random` value causes images to be positioned randomly. The `grid` value causes Imago to position images within a square grid in which each grid box is the size of the largest image. The `spiral` value causes images to be sorted (according to the current rating scheme) and positioned in a spiral so that the highest-rated image is in the center and the lowest-rated images are along the edges. Both the grid and spiral algorithms can waste space when there are many small images, because

images are centered within grid boxes. The `spiral2` algorithm is just like `spiral`, but images are positioned as close together as possible.

The `spiral3` algorithm partitions the area specifications into groups, runs the `spiral2` layout on each group, and then runs the `spiral2` layout again on all the groups keeping the thumbnails of each group together as if they were a single image. The `spiral3` algorithm is useful for displaying visual bookmarks, because it groups thumbnails from the same web site (see Figure 3.10). Thumbnails are first clustered according to the host part of their `image` URL. This process separates images on a page that load from a different host, which are often advertisements. Clustering can also be determined by the file part of each image's URL or the host part of each area's `href` URL. These clustering alternatives may be more appropriate for other applications.

The `MAKE_MAP` request supports three different values for the `background` attribute. The `solid` value indicates that the background should be filled with the solid color specified by the hex value of the `background_color` attribute. The `tile` value indicates that the image specified by the `background_thumb_url` attribute should be used to tile the background. The `center` value indicates that the background should first be filled with the solid color specified by the hex value of the `background_color` attribute (if any) and then the image specified by the `background_thumb_url` attribute should be centered in the background.

The `MAKE_MAP` request also supports `augment` as an additional value for the `replace` attribute. When `augment` is specified, Imago attempts to merge new area specifications with existing ones stored in an imagemap with the same stub-name. This capability allows imagemaps to grow. It is used by Mandala to keep groups of image representations up-to-date. For example, when the Mandala server builds groups of associations (between cached images and web pages) from a particular web site, it updates the imagemap associated with the web site (i.e., the visual site index) by using "`replace: augment`" in a `MAKE_MAP` request. Mandala does not need to

determine if the associations already appear in the site index, because Imago deletes duplicate associations when it merges the area specifications.

6.3 GET_INFO: Image Meta-Data

Imago's GET_INFO request takes an image URL as an argument and returns a list of image meta-data in the form of attribute/value pairs. For example, the following is a valid GET_INFO request:

```
GET_INFO http://www.cs.unm.edu/~jon/mandala/image/bhutmara.75.jpg
```

The above request, when submitted to Imago, yields the following output:

```
INFO http://www.cs.unm.edu/~jon/mandala/image/bhutmara.75.jpg

Content type: image/jpeg
Content length: 12015
width: 70
height: 70
total components: 3
input color space: Y/Cb/Cr (yuv)
output color space: rgb
total colors: 16777216
transparent color: -1
interlace: false
```

Image meta-data is useful for clients that are not equipped to retrieve images and read their headers.

6.4 GET_MAP_INFO: Imagemap Meta-Data

Imago's GET_MAP_INFO request takes a map stub-name as an argument. It returns a list of attribute/value pairs just like the GET_INFO request, but with additional area specifications. The area specifications are formed from the attributes and values shown in Table 6.5, just like the ones in the MAKE_MAP request.

6.5 GET_RATED_MAPS: Image and Imagemap Rating

Imago also rates imagemaps according to a heuristic that attempts to identify images that are most likely to represent information and least likely to be used for decoration, navigation, or advertisement. The rating scheme is based on image meta-information, as opposed to image content analysis. It ranks large square images with many colors higher than small narrow images with few colors. The rate of an imagemap is the average rate of the imagemap's thumbnails. Ratings allow clients to identify images and imagemaps that are most likely to contain useful representations.

One possible value for such a rating scheme might be image area. Image area expresses size, but not aspect ratio. While aspect ratio is usually considered to be $\frac{width}{height}$, the rating scheme does not need to distinguish between the direction of the maximum dimension, so aspect ratio can be defined as:

$$aspect\_ratio = \frac{max(width, height)}{min(width, height)}$$

Instead of using image area, Imago's rating scheme uses thumbnail area. Thumbnails are created by shrinking each image so its maximum dimension is a common size, while preserving its aspect ratio. Thumbnail area can be defined in terms of aspect ratio:

$$thumbnail\_area = \frac{max\_dimension^2}{aspect\_ratio}$$

In general, for large images, one edge of their thumbnail will equal the maximum dimension, while the other edge will be smaller. Thumbnails of large square images (i.e., that have an aspect ratio of one) will have the maximum thumbnail area, which is equal to the maximum dimension squared. Images with aspect ratios that approach one will have thumbnail areas that approach the maximum, while images with aspect ratios less than one will have thumbnail areas that are less than the maximum. In other words, thumbnail area expresses aspect ratio.

In some cases, for images that are too small to require a thumbnail, both dimensions may be less than the maximum. The "thumbnail area" of these small images is taken to be their image area, which is always less than that of any actual thumbnail. To some extent, then, thumbnail area also expresses size.

Imago's rating scheme multiplies the thumbnail area times the percentage of colors used in the image, out of the maximum possible number of colors:

$$rate = thumb\_area * \frac{total\_colors}{MAX\_COLORS} * 100$$

The maximum possible number of colors is taken to be 16,777,216, the total number of different colors that may be stored by the JPEG image file format. Color JPEG images are assumed to have 16,777,216 colors. Greyscale JPEG images are assumed to have 256 colors. The total colors used for GIF images is obtained by reading the image's header.

Imago's rating scheme is a very crude estimate of the amount of possible information in the image. It has a number of problems. To begin with, an image's number of possible different colors cannot be larger than its area – only a very large JPEG image can actually have 16,777,216 different colors. Another problem is the abrupt discontinuity between images that are just small enough not to require a thumbnail and all larger images. Images that are just small enough not to require a thumbnail may be over-rated. Imago is not limited to the current rating scheme, however. As with layout algorithms, additional rating schemes can be implemented and included within Imago as time permits.

6.6 `GET`

Imago's `GET` request are simply forwarded to the web server specified in the request's URL. Results (or errors) are forwarded back to clients. The `GET` request is offered as a convenience for Imago clients. It may simplify the design of clients that require

option:	argument:	type:	default:
-i	port	int	6666
-P	proxy host	string	-
-p	proxy port	int	8888
-d	server disk dir	string	-
-u	server URL dir	string	-

Table 6.6: Imago's Command-Line Options and Arguments

limited web access, because they need only connect to Imago instead of multiple, remote servers.

6.7 Usage Details

This section describes Imago's command-line options and errors. Command-line options are used to initialize Imago. Under certain conditions, Imago generates errors. The form of of the errors, and the conditions that cause them are also described.

Imago honors the command-line options listed in Table 6.6. Options are listed together with the expected type of their arguments, and their default values, if any. The values of the arguments determine Imago's behavior. The `-i` option specifies the port number that Imago should use to listen for requests. The `-P` option specifies the host name of a machine running a proxy server. The `-p` option specifies the port number that Imago should use to connect to the proxy server. The `-d` option specifies the full path name of a directory where Imago should store its thumbnails and imagemaps. If this directory is exported by a web server, the `-u` option specifies the directory's fully-qualified URL path.

Imago generates errors as HTTP messages with HTML data. For example, the query "`GET_INFO http://www.cs.unm.edu/foo.jpg\n\n`" yields the following error:

```
HTTP/1.1 404 Not Found
Date: Mon, 15 Mar 1999 21:47:09 GMT
Server: Apache/1.3.0 (Unix)
```

```
Connection: close
Content-Type: text/html
Last-modified: Mon, 15 Mar 1999 21:38:20 GMT
Last-checked: Mon, 15 Mar 1999 21:38:20 GMT
Last-accessed: Mon, 15 Mar 1999 21:38:20 GMT

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /foo.jpg was not found on this server.<P>
</BODY></HTML>
```

Imago generates additional errors as shown in Table 6.7. Errors with a 400 code are generated if Imago is issued an unrecognized request or if a request had trouble obtaining memory or opening files. The `MAKE_MAP` request generates an error with a 400 code if no areas are specified or if each area specification fails.

The other requests generate errors for problems related to single images. Errors with a 406 code are generated for images that do not meet the `thumb_src_min_width`, `thumb_src_min_height`, or `thumb_src_min_colors` requirements. Errors with a 406 code are also generated for images with extreme aspect ratios that, once scaled, would not meet the `thumb_min_width` and `thumb_min_height` requirements. Errors with a 414 code are generated when requests include URLs that are longer than 1024 characters. Errors with a 500 code are generated when Imago cannot connect to the proxy server or cannot decompress an image. Errors with a 501 code are generated for requests that specify images stored in unsupported formats (only GIF and JPG formats are supported).

6.8 Applications

Imago can be used as a component of any system that requires image thumbnails, imagemaps, or image meta-data. Imago is used as a component of Cospace, a research

error code:	error type:	description:
400	Bad Request	bad request, can not read files
406	Not Acceptable	image too small
414	Request-URI Too Long	requested URL too long
500	Internal Server Error	can not get or decompress image
501	Not Implemented	unsupported file type requested

Table 6.7: Imago's Error Interface

project at AT&T Labs that uses imagemaps as texture-maps to cover the walls of automatically-generated VRML worlds [92].

Imago is also used by Mandala's Bookmarks process to build imagemaps of thumbnails from each bookmarked page. Special diagnostic thumbnails are used for pages that are inaccessible or that have no images (clusters of diagnostic images are visible in the lower left corners of Figures 3.10 and 3.11). Since the diagnostic thumbnails are linked to the bookmarked page in the imagemap, they serve as useful indicators of broken links.

Imago is also used at the University of California, San Diego, to improve the quality of thumbnail images used in PadPrints [44] and Pad++ [6].

In addition, Imago has been used through a `telnet` interface, by administrators of the University of New Mexico Computer Science Department's web site, to make thumbnails of portraits for the faculty and graduate student web pages. The `telnet` program provides a simple socket interface, which connects to a server, copies any typed text to the server, and prints server responses on the screen.

6.9 Implementation

While the other Mandala components are written in Java, Imago is written in C, for reasons of speed and reliability. Imago uses publicly available C code: `libjpeg`, JPEG software from the Independent JPEG Group [45], and `gd`, GIF software from

boutell.com [10]. These libraries support both reading and writing GIF and JPG images, while the Java classes support only reading GIF and JPG images. The C code is also relatively robust compared to the Java decompression classes, which frequently throw undocumented exceptions when trying to read GIF or JPG variations that they do not fully support. The C code is not perfect (several problems have been fixed in `gd`), but at least problems can be fixed, because the code is available in source-code form, unlike the Java classes, which are only available in byte-compiled form. The C code also makes it possible to read header information from images; image header information is hidden by the Java image classes.

Imago has been tested on SGI systems running IRIX 6.5 and Sun systems running SunOS 4.1.4 and Solaris 5.5. Imago has also been extensively tested with Purify, a run-time error and memory leak detection tool from Rational Software [84].

6.10 Summary

This chapter has described Mandala's image server, *Imago*. Imago's client interface has been described as have each of Imago's requests. Imago uses a fast and accurate scaling algorithm, which combines repeated fast halving with a single invocation of a slower, more general, scaling algorithm (Section 6.1). Imago uses multiple layout algorithms to position thumbnails in imagemaps (Section 6.2). Spiral layout algorithms position the highest rated images near the center (spiral and spiral2). Another algorithm (spiral3) divides the thumbnails into clusters, uses a spiral layout algorithm to position the thumbnails within each cluster, and then uses the spiral layout algorithm again to position the clusters within a larger spiral. Imago also returns meta-information about images (Section 6.3) and imagemaps (Section 6.4), and rates images as a function of their thumbnail area and number of colors (Section 6.5). Applications that use Imago have been described (Section 6.8), as have issues related to Imago's implementation language (Section 6.9).

Chapter 7

Mandala Server

The Mandala server manages three main types of abstract objects:

Group: A set of associations.

Association: Part representation and part list of URLs.

Representation: A text string or URL.

The Mandala server builds and maintains *groups* of *associations*. An association consists of a *representation* and one or more URLs – links to associated information. A representation may be a text string (e.g., a web page title) or the URL of a web-accessible image. The Mandala server gets associations from any previously-stored imagemaps and builds new associations from the proxy server's cached pages. The Mandala server groups associations in several ways and builds imagemaps of the groups.

The Mandala server also supports multiple Mandala clients, manages their requests, and stores their associations and group definitions as imagemaps on a web server. The Mandala server's client interface is shown in Table 7.1. The requests are described in the immediately following sections. The remainder of this chapter describes how the Mandala server builds associations, groups, and imagemaps. The

request:	argument:	return value:
GET_GROUPS	user name	group specifications
GET_DATA	association id	image data
GET_INFO	association id	attribute/value pairs
MAKE_MAP	group name	status
GROUP	(see below)	-
SELECTION	association id	-
DISCONNECT	-	-

Table 7.1: The Mandala Server's Client Interface

method used to monitor the proxy server is described, as is the server's procedure for managing multiple clients. Miscellaneous usage details are also described.

7.1 GET_GROUPS: Group Data

When clients start, they issue a GET_GROUPS request to the Mandala server, which returns a summary of its group database. The following is an example of a result from a GET_GROUPS request:

```
GROUP 200 new groups follow

add 9 io.womanonfire.com site 110 111
add 10 www.sonymusic.com site 113 114 115 116 117 118 119 120 121 122
add 11 www.inkblotmagazine.com site 124 125 126 127 128
add 0 text_only system 1612 1683 1796
add 100 jon/new_groupX user
```

The general form of each line of the message body is:

add group_id group_name group_type member_id_1 ... member_id_n

7.2 GET_DATA: Image Data

Before Mandala clients can display images, they need to obtain the image data from the Mandala server by issuing a GET_DATA request. The response header for a typical

GET_DATA request is shown below (the binary image data, which follows the header, is omitted). On the first line of the response, the second token is the representation id; the third token is the representation type (either map or image).

```
DATA 260 map
Date: Thu, 25 Feb 1999 22:37:26 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Thu, 25 Feb 1999 22:37:09 GMT
ETag: "cfc9a-2db73-36d5d095"
Accept-Ranges: bytes
Content-Length: 187251
Connection: close
Content-Type: image/jpeg
Last-checked: Thu, 25 Feb 1999 22:28:39 GMT
Last-accessed: Thu, 25 Feb 1999 22:28:39 GMT
Content-Location: http://www.cs.unm.edu/~jon/map/www.entropy8.com.jpg
Content-Length: 187251
Referer: http://www.cs.unm.edu/~jon/map/www.entropy8.com.html
```

7.3 GET_INFO: Image and Imagemap Meta-Data

Mandala clients use the GET_INFO request to obtain meta-information about images and imagemaps. Meta-information about imagemaps is used to split an imagemap into its individual thumbnails (this trick assumes the imagemap has been stored using a non-overlapping layout).

The GET_INFO request is handled by first using the argument to determine the representation and its type. If information is requested for an image, Mandala issues a GET_INFO request to Imago. If information is requested for an imagemap, Mandala issues a GET_MAP_INFO request to Imago. Mandala takes the attribute/value pairs returned from Imago, adds the representation's id, image URL, and associated URLs, and then passes the result back to the client.

7.4 MAKE_MAP: Imagemap Generation

Mandala clients use the MAKE_MAP request to save groups of image representations as imagemaps. The MAKE_MAP request is a minimal imagemap specification, in which one line indicates the desired dimensions and the other lines indicate the association id and position of each thumbnail. The Mandala server converts the client's minimal imagemap specification into one that can be sent to Imago by specifying the layout type as preset, the thumb_max_size as 75 pixels, the filter as gaussian, and the dimension_type as fixed. Lines that start with "position:" are converted to area specifications by replacing the word "position:" with "area:" and converting the association id to values for the image and other attributes.

Imago attempts to build the imagemap, store it in a web server's export directory, and return the new URL to the Mandala server (see Section 6.2). If imago was successful, the Mandala server generates the HTML client-side imagemap and stores it in the web server's export directory. The Mandala server returns Imago's status to the client in the form shown below. Errors take the same form, but substitute 204 for 200.

```
MAP 200 group_name
```

7.5 GROUP: Group Editing

The GROUP request is used by clients to add, copy, and edit groups of associations. Each line in a GROUP request is a separate operation of the form shown in Table 7.2. For example if a person using a Mandala client wishes to copy group 10 and its five members (associations 13, 4, 1234, 89, and 12) to a new group named "jon/Paco", the client sends the following GROUP request:

```
GROUP 200 edit

copy 10 jon/Paco
```

operation:	id:	arg1	arg2	additional args
add	group id	group name	group type	members
add_member	group id	group name	member 1	members
remove_member	group id	group name	member 1	members
copy	group id	new group name	-	-

Table 7.2: The Mandala Server's GROUP Request Interface

The Mandala server's response is shown below. Note that the response is itself another GROUP request (as is the response to the GET_GROUPS request in Section 7.1), because the Mandala server and clients share the same parser for GROUP requests.

```
GROUP 200 new

add 9178 jon/Paco user 13 4 1234 89 12
```

Mandala clients send GROUP requests that use the add_member and remove_member operations when people edit groups by dragging and dropping image representations between client windows.

7.6 SELECTION

The SELECTION request signals the Mandala server that a person has double-clicked on a representation in a client. No response is generated. The Mandala server logs each selection.

7.7 DISCONNECT

The DISCONNECT request is used when a client exits, freeing resources on the Mandala server. It is especially useful when debugging for terminating a telnet session.

7.8 Building Associations

When the Mandala server starts, it issues a `GET_RATED_MAPS` request to Imago to determine the highest-rated imagemaps. Each imagemap is a source of previously-stored associations.

To find the associations, the Mandala server issues a `GET_MAP_INFO` request for each imagemap. Each area specification in the response (i.e., each line that starts with "`area:`") that has values for the `href` and `image` attributes contains an association.

After reading any existing imagemaps, Mandala also builds associations from cached resources. Mandala issues a `DUMP_CACHE` request to the proxy server, Mirage. For each cached page, Mandala issues an `IMAGE_FILTER` request to Mirage. Mirage returns a list of URLs of images that would appear on the page in a web browser and an indication if any of the images are links to additional URLs. Mandala preserves image links by building an association of the image to the other URL. To ensure that each cached page is represented by at least one of its images, Mandala always associates the first image on the page with that page. Any remaining images on the page are associated with the page unless the image is a link to another URL, in which case the image is associated with the URL.

Mandala also builds associations from newly-cached resources. When a web browser loads a new page, it makes additional requests for any inline images. When requesting inline images, most web browsers identify the referring web page in the value of a `Referrer` header in the HTTP request.

As the proxy server services requests, it announces its activity to application processes (see Section 5.2). Each line of output describes a single request for a URL and the proxy server's response. Mirage indicates the host name of the requesting client, the requested URL, its mime type, the response code, the URL of a referrer, and the cache activity. For each cached image, if a referrer URL is indicated, Mandala builds a new association between the cached image and its referrer.

7.9 Building Groups

The Mandala server builds groups from imagemaps. When the server first starts and reads imagemaps to find stored associations, it builds a special group called "maps." An association is created for each imagemap and added to the group "maps." Mandala groups have a type. The "maps" group is of type "system," because it is maintained by the Mandala server.

A new system group is also created for each imagemap. Imagemap groups are named after the imagemap's stub name. As associations are built from the imagemap's area specifications, they are added to the imagemap's system group.

The Mandala server also builds groups from cached resources. Cached pages without inline images are added to a system group called "text_only." For each cached page with at least one inline image, a new "site" group is named using the host part of the page's URL, and associations are made (between the images and the page) and added to the site group.

Associations for newly-cached images are also added to the corresponding site group. In addition, the Mandala server builds "session" groups for newly-cached resources. A session group is named after the host of the proxy server client that caused the resource to be cached.

Site groups and session groups are actually system groups, however there are so many system groups that it is convenient to split them up into sub-groups.

Because the Mandala server maintains system groups, people cannot edit them directly. Instead, system groups must be copied. The copies are of type "user" and can be edited by the user who created them.

7.10 Building Imagemaps

After the Mandala server builds groups of associations, it may have to rebuild the imagemaps of certain site groups. Imagemaps that should be rebuilt are identified

option:	type:	default:
-proxy_host	string	-
-proxy_port	int	8888
-server_disk_dir	string	-
-image_server_host	string	-
-image_server_port	int	6666

Table 7.3: The Mandala Server's Command-Line Options

while building associations from cached resources. Mandala builds associations from imagemaps before cached resources. Because Mandala does not build duplicate associations, if a new association is built for a cached resource, it must be a new association that is not in any imagemap. The cached resource comes from a particular web site. If Imago has an imagemap that corresponds to the web site, the imagemap is marked to be rebuilt. After the associations and groups are built, any marked imagemaps are also rebuilt. Imagemaps are rebuilt by converting the new associations to area specifications and issuing a `MAKE_MAP` request to Imago. The "`replace: augment`" header is specified, allowing the visual site indexes to grow as new parts of the site are explored by any member of the community.

7.11 Monitoring the Proxy Server

The Mandala server connects to Mirage as an application to be informed of newly-cached resources (see Section 5.2). The Mandala server builds new associations for each newly-cached image that has a `Referrer` field (see Section 7.8). A new site group is created for the association (if necessary), and the association is added to the site group (see Section 7.9). In addition, a new session group is created for the association (again, if necessary) and the association is added to the session group. Finally, clients are sent `GROUP` messages, so that they can update their groups and associated views.

7.12 Managing Multiple Clients

The Mandala server "listens" for client connections on port 7777 and keeps a list of each connected client process. The Mandala server sends a `GROUP` message to each client whenever a new group is added (e.g., because a new session started, resources were cached from a new site, or a client copied a group) or an existing group changes (e.g., because an association was added or deleted), .

7.13 Usage Details

The Mandala server honors the command-line arguments listed in Table 7.3. These arguments can be used to modify various default configuration parameters.

When the Mandala server encounters an error while trying to service a client's request, it replies with a `204 No Content` error. For example, if a client issued a `GET_DATA` request and passed an invalid representation identifier, the server would respond:

```
DATA 204 No Content
```

Similar `204 No Content` errors are generated for bad `GET_INFO` requests or if, for some reason, a `GET_GROUPS` request could not be serviced.

7.14 Summary

This Chapter has described the Mandala server. The Mandala server's client requests have each been described (Sections 7.1-7.7). In addition, the server's initialization process has been described. The Mandala server starts by building associations from any stored imagemaps and any images in the proxy server's cache (Section 7.8). The Mandala server builds several groups of image representations (Section 7.9) and stores them as imagemaps (Section 7.10). The Mandala server continues to monitor the

proxy server's HTTP traffic to create groups of images cached by the same browsing session (Section 7.11). The Mandala server also manages multiple clients, servicing their requests, and keeping them informed as groups change (Section 7.12).

Chapter 8

Mandala Clients

The Mandala clients handle layout, animation, and user interactions. User interactions include direct manipulation of images, saving imagemaps, and selection of images to access associated information.

Mandala's client/server architecture can support many different types of clients. One advantage of this architecture is that clients can run on different types of hardware (e.g., television, touch-screen, palm-top) each with different interaction capabilities. One type of client has been implemented, which runs as a Java application. A future client will run as a Java applet within Java-enabled web browsers (as shown schematically in Figure 4.1).

Java applets, however, operate under severe security constraints. In particular, a Java applet cannot read or write local files, and it can only open socket connections to the web server that served the applet's code. Because an applet cannot read or write files, user-defined objects must be stored on a server machine. Although these security constraints apply to Java applets only, the Mandala server assumes that all clients may be similarly challenged. Mandala clients need only one connection to the Mandala server. Also, Mandala clients do not need to write any files, because the Mandala server handles saving associations, groups, and imagemaps.

This chapter begins by explaining how any Mandala client communicates with

a Mandala server and signals a web browser to display web pages. Next, several issues related to the current Mandala client's graphical interface are introduced. In particular, reasons are provided for why each client view works the same way. View options for layout and animation of image representations are described as is the view menu interface. The implementation of image repositioning and drag-and-drop capabilities are outlined. In addition, the procedure for updating views dynamically is described as is the function of a view as a visual look-ahead cache.

8.1 Communicating with a Mandala Server

Current Mandala clients communicate over a single connection to the Mandala server. The connection is established when the client starts. It should stay open as long as the client is running. Because the connection stays open, a sender must indicate where in the data stream a message ends by specifying the message's length in its header.

Mandala clients read messages that take the form of the requests listed in Table 8.1. Note, however, that these messages are not requests for information from the client. Consequently, no response is given or expected. Rather, these messages are responses from the Mandala server. They contain information that was previously requested by the client.

An earlier version of the Mandala client attempted to read messages in a low-priority thread, but it was still easy for a server to send so much data to the client that the client refused to reply to normal input events. Thread priority seems to be honored differently on different implementations of Java.

In an attempt to prevent the Mandala server from overloading its clients, the following transmission policies are enforced: 1) a client must specifically ask for individual resources, and 2) unsolicited messages sent from the server to the client must be brief. For example, when a Mandala client starts, it issues a `GET_GROUPS` request

request:	argument1:	argument2:
DATA	representation id	-
INFO	return code	representation id
GROUP	return code	-
MAP	return code	-

Table 8.1: The Mandala Client's Message Interface

to the Mandala server. The server responds with a GROUP message containing the server's group database (see Section 7.1). To display a new image representation, a client would issue a GET_DATA request to the server, which would later respond with a DATA message containing the compressed image data (see Section 7.2). A client that needed additional information about a representation would issue a GET_INFO request to the server. The server would later respond with an INFO message containing image or imagemap meta-data in a list of attribute/value pairs (see Section 7.3). MAP messages are also sent by the server to indicate the status of a previous MAKE_MAP request (see Section 7.4). The only unsolicited messages sent by the server are GROUP messages, which indicate that the corresponding view (if any) should be updated (see Section 8.9).

8.2 Accessing Associated Information

Each selectable object displayed in a Mandala client has a unique identifier. When the object is selected, the Mandala client sends the identifier to the Mandala server as an argument to a SELECTION request so the server can log the selection.

Applet clients can drive the web browser to a new page with the showDocument method of the AppletContext interface. Application clients may require platform-dependent code to signal web browsers to load a new page. Most web browsers allow separate application programs to control some of their behavior through an API. The Mosaic web browser, for example, can be controlled through a TCP/IP socket

interface [68].

Unfortunately, Netscape's web browsers cannot be controlled through a socket interface. Netscape's browsers run on three different operating systems and have four different APIs. The UNIX API requires application programs to send it X Events – an inter-process communication scheme that requires using the X Window System. The Macintosh API requires application programs to send it Apple Events. The Windows API requires application programs to send it DDE (Dynamic Data Exchange) messages or OLE2 (object linking and embedding) object methods.

Netscape's multiple APIs are a problem for application clients. Although Java is designed to be platform-independent, Java applications that need to control Netscape must use the API that corresponds to their current environment. This means that the Java applications need to include platform-dependent code for each supported platform. They need to determine their current environment at runtime to execute the appropriate platform-dependent code.

For example, on a UNIX platform, the Java code invokes the runtime's `exec()` method, which invokes the `openURL()` method of the Netscape browser's `-remote` option, passing the address of the associated URL as an argument. The `openURL()` method identifies an instance of the browser running on the X server and signals it to display the resource with an X Event.

On a Windows platform, the Java code invokes the runtime's `exec()` method, which invokes a small C program that calls the Microsoft Foundation Class `ShellExecute()` method, specifying the "open" option and passing the address of the associated URL as an argument. The `ShellExecute()` method is a handy way to avoid DDE or OLE2. Once it identifies the argument as a URL from its ".html" extension, `ShellExecute()` signals the Windows default web browser (presumably with a DDE or OLE2 message) to retrieve and display the associated web resource.

8.3 View Uniformity in the Graphical Interface

Mandala clients interact with people through a graphical interface. The major challenges of the graphical interface are 1) to let people display hundreds of images in ways that let them identify groups of similar images easily and 2) to let people share their visually-identified groups with the system easily.

One reason ease-of-use is difficult to achieve is that Java and most other graphical interface toolkits make it easy to build interfaces in which the initial view behaves differently than subsequent views. In most graphical interface toolkits, a reference to an existing window is a required argument for many common operations. In Java, for example, before a process can decompress an image, it needs to establish a window on the screen. Because many applications have initialization constraints of their own, it is usually simplest for designers to make the initial view be the application's "main" view. Unfortunately for users, this means there are at least two different types of user interface objects to learn how to use, because the main view behaves differently than the other views. In many applications, for example, when the main view is closed, the application exits, while the other views can be opened and closed without side-effects.

By carefully examining initialization constraints, current Mandala clients are designed so that each view, even the initial view, runs the same code. Each view has an initial splash screen. While the client displays the initial splash view, it obtains group definitions from the server. Each view has the same configuration of menu options (see Section 8.6), group label, display area, current association label, and pause/run toggle (see Figure 1.1 and most of the figures in Chapter 3). Each view displays the members of a group. Different types of groups may be used for different purposes, but these differences are not reflected in the user interface. Keeping the user interface consistent makes the system easy for people to learn to use, and easy to remember how to use, because each view works exactly the same way.

8.4 Layout

Layout is the process of determining a position within the display window for each selectable object. Although animation can also be used to display selectable objects (see the next section), layout is useful for archiving and static displays, which cannot overlap without sacrificing visibility. For the purpose of layout, each selectable object in Mandala is modeled as a rectangle with the dimensions of the associated image or text. Mandala clients support three layout algorithms: random, tiled, and centered. Layout algorithms are specified in the "view options" dialog, which is accessed by selecting `options` on the View menu (see Section 8.6).

The random algorithm positions images randomly, but subject to the constraint that images be completely visible within the display window (see Figures 3.6, 3.2, and 3.7). The random algorithm does not prevent images from overlapping, so it is most useful when combined with cycling animation (as described in the next section).

The tile layout algorithm is a variant of offline 2-D bin-packing algorithms, which minimize total area without overlapping [23]. The tile algorithm sorts rectangles by height. Then it positions them from left to right across the top of the view. If there are too many rectangles to fit across the view, it positions them in successively lower horizontal strips. If there are too many rectangles to fit in the entire view, the tile algorithm starts positioning them across the top again, overlapping the tallest images. The tile algorithm is most useful for applications that wish to minimize overlapping and maximize visibility.

Layout algorithms are either offline (requiring the complete set of objects in advance) or online (able to position objects incrementally). The random algorithm is an example of an online layout algorithm. It is useful when the complete set of objects can not be known in advance (e.g., for monitoring a cache or browsing session). The tile layout algorithm is an example of an offline layout algorithm, because it needs to start by sorting the rectangles by height. It is useful for positioning the members of small user-defined groups before storing them as imagemaps.

The centered option centers each representation in the middle of the view. The centered option, obviously a simple layout algorithm, is useful for groups of large images when combined with cycling animation. The centered option is the default layout for a group of imagemaps.

Although Mandala clients support only three layout algorithms, they honor the area coordinates of imagemaps, allowing them to take advantage of Imago's layout algorithms (e.g., gird, spiral, spiral2, etc.) for archived groups of image representations (see Section 6.2).

8.5 Animation

Mandala clients use a very primitive sort of animation to allow people to see more images than will fit on the screen at once. Images are displayed by a separate thread, the Displayer, which repeatedly displays images from a list. If the image has no coordinates, the Displayer computes them based on the view's current layout algorithm. When the Displayer reaches the end of its list, it continues displaying from the beginning again. This sort of animation has little effect when all group members can fit in a view without overlapping. However, for large groups that cannot fit in a view without overlapping, the Displayer allows each image to be seen repeatedly.

Mandala clients provide a single toggle button for pausing the animation and restarting it.

8.6 Menu Interface

Mandala clients have a menu interface that lets people manipulate views, obtain meta-information about a representation, and terminate the client. Items from the View menu are shown in Table 8.2. Selecting "`open`" lets people open a new view, which displays the members of any group (the group's name is selected from a scrolling

menu item:	result:
open	select from group list
new	new, empty user group
options	edit view options
text	toggle view to text mode
save	save current view as imagemap
clear	clear member coordinates
close	close view

Table 8.2: The Mandala Client's View Menu Interface

list). Selecting "new" creates a new group that appears as an empty view. Selecting "options" lets people switch the layout algorithm for positioning members (see Section 8.4). Selecting "text" toggles the view to stop displaying images, and instead, displays the group as a scrolling list of textual representations (see Figure 1.1). Selecting "save" lets people save a group as an imagemap. To save a group, the Mandala client uses the members of the view's current group to generate a list of association ids and coordinates, which it sends to the server as a MAKE_MAP request (see Section 7.4). Selecting "clear" clears each member's coordinates, which will be recalculated according to the current layout algorithm as the displayer runs (see Section 8.5). Selecting "close" closes the view.

8.7 Using Direct Manipulation to Edit Imagemaps

Current Mandala clients are designed so that people can select and drag images to reposition them within a view. This capability is useful for rearranging portions of an imagemap.

When a person clicks the mouse, the client determines which image was selected by comparing the mouse coordinates with the bounding box of each image. Because image may overlap, they must be checked in the inverse of their display order. If the view is animating, its Displayer is first paused (see Section 8.5). The selected image

is erased from the display by clearing its area to the background color and redrawing the parts of any images that overlap the area. The newly-drawn area, which had been obscured by the image, is saved, and then the image is drawn again on the display. As a person drags the mouse, the client cycles through the following actions: 1) it determines the new position of the dragged image from the mouse coordinates, 2) it erases the image in its old position by drawing the saved background over it, 3) it saves the background area that will be obscured by the new position of the dragged image, 4) it draws the image in its new position, and 5) it updates the display. To avoid flicker, the computer graphics technique of *double-buffering* is used – drawing occurs in an off-screen image, and the display is not updated between erasing the old image and drawing the new one.

8.8 Using Direct Manipulation to Edit Groups

Current Mandala clients are designed so that people can drag-and-drop images between views. This capability is useful for editing groups (see Figure 3.12). In particular, because people can drag-and-drop images into a new, empty group, this capability is useful for sharing visually-identified groups with the system. It should be easy to learn, because dragging-and-dropping is an action supported by most graphical user interfaces. In addition, since dragging-and-dropping is almost the same action as dragging, which is used to reposition images within a view, using drag-and-drop actions to edit groups keeps the user interface consistent.

Although drag-and-drop capabilities are supported in the next version of Java [22, pp. 339-340], in an effort to make Mandala clients immediately useful on a wide range of platforms, drag-and-drop capabilities were implemented in Java 1.1. In order for drag-and-drop to work, the Java runtime environment needs to generate WINDOW_ENTERED events, when the mouse is pressed.[1]

[1]The drag-and-drop capabilities do not work on SGI machines because the IRIX JDK does not generate WINDOW_ENTERED events when a mouse button is pressed.

When a user drags an image over a new view, a WINDOW_ENTERED event is generated, signaling the client to begin dragging the image in the new view. When a user drops an image, a MOUSE_RELEASED event is generated. If the group displayed in the view where the mouse was pressed (the source view) is the same as the group displayed in the view where the mouse is released (the destination view), no group editing occurs.

If the group displayed in the source view is different than the group displayed in the destination view, the following actions occur. If the destination view is animating, it is paused. If the source view is displaying a group of type system, it cannot be edited directly, so the dragged image is returned to its original position in the source view. If the user was not holding down the control key while dragging the mouse, they are prompted with a dialog box, asking if they would like to copy the group displayed in the source view. Holding down the control key allows people to copy image representations from any group. If the destination view is displaying a group of type system, it cannot be edited directly, so the dragged image is returned to its original position in the source view, and the user is prompted with a dialog box, asking if they would like to copy the group displayed in the destination view. Otherwise, the image representation is added to the group displayed in the destination view. If the source view is not displaying a system group, the image representation is removed from the group displayed in the source view. The client then sends the server `GROUP` requests to indicate that the user has edited their groups.

8.9 Updating Views

When a Mandala client starts, it first interprets any command-line options. The command-line options are listed in Table 8.3. These can be used to modify various default configuration parameters. The `group` option is used to specify the initial group that the client should display. By default, Mandala clients display the group `maps`.

option:	type:	default:
-server_host	string	-
-server_port	int	7777
-group	string	maps

Table 8.3: Mandala Client's Command-Line Options

The server_host option is used to specify the host name of the machine running the Mandala server. The server_port option is used to specify the port number on the server_host to use, when connecting to the Mandala server. By default, Mandala clients attempt to connect on port 7777.

After processing any command-line arguments, a Mandala client makes its first view and obtains some utility images, one of which it displays as its splash screen. While the splash screen is being displayed, the client creates a RepManager class, which will keep track of requesting and decompressing representations from the Mandala server. The client then connects to the server and issues a GET_GROUPS request. After getting the groups database, it removes the splash screen and switches the view to display the initial group, which is, by default, the group of imagemaps. The client's main thread then enters a loop (a.k.a. the client's *main loop*), where it repeatedly waits for and services messages from the Mandala server.

Displaying the group of imagemaps is handled the same way as any other group of representations. The initial view is handled in the same way as any other view. In fact, the code for obtaining representations and updating views works the same way for each view and for each type of group.

When a view is first created, it displays its splash screen and calls the RepManager's maybe_get_reps_from_map() method to see if it should obtain any representations from an imagemap. If the view's group name corresponds to a local imagemap and thumbnails have not yet been extracted from the imagemap, then the RepManager submits a GET_INFO request to the server, passing the group's id as an argument.

If no `GET_INFO` request is sent, the view removes the splash screen and displays the group of representations as described below.

On the other hand, if a `GET_INFO` request is sent, control returns to the client's main loop, while the view continues to display the splash screen. The server's response arrives in the form of an `INFO` message, which is handled by the main loop. The main loop processes the message like any other message of type `INFO`. It builds a hashtable of the attribute/value pairs and parses any area specifications. Each `INFO` message includes, in its header, a representation id. If the representation is of type "map" (as it is in this case), then the client checks each of its views to see if any are displaying a splash screen. If the client finds a view displaying a splash screen and the view's group name matches the representation's name, then the client converts each area specification into a new representation by copying the corresponding area from the imagemap into a new thumbnail. Each new representation is registered with the RepManager. The view then removes the splash screen and displays the group of representations.

A view displays a group of representations by first calling the RepManager's `get_existing_reps()` method. If any members of the group have already been decompressed, the RepManager copies their references to the view's display list, and the Displayer displays them immediately.

Any members of the group that have not yet been requested are identified, requested, and displayed by a variation of Java's Observer/Observable protocol. In the normal version of the protocol, the Java Observer interface implements an `update()` method. An Observable object extends the Observable class, which keeps a list of Observers and an internal flag. The Observable's `setChanged()` method marks the internal flag as having changed. The Observable's`notifyObservers()` method calls each Observer's `update()` method if the internal flag has changed.

In the Mandala client, each view implements an Observer interface, and each group extends the Observable class. As the Displayer loops through the representa-

tions, it calls the `notifyObservers()` method of the view's group. If there are more group members to request, the `notifyObservers()` method calls the view's `update()` method, which finds an unrequested representation and requests it from the server. The server's response arrives in the form of a `DATA` message, which is processed by the client's main loop. The main loop processes the message in the same way that it processes any other message of type `DATA`. The client reads and decompresses the data and converts it into a new representation. It then signals the RepManager that the representation arrived. Finally, the client checks each view to see if the representation belongs to its group. If it does, the representation is added to the view's display list, and the Displayer displays it immediately.

The view-updating process may seem complex, but it allows views to be updated as groups change. When the client processes a `GROUP` message that indicates a group should change, it calls the appropriate group method. For example, if a `GROUP` message indicates that a new member should be added to a group, the client calls the group's `addMember()` method. Because the group is an Observable, the `addMember()` method calls `setChanged()`, which marks the Observable's internal flag as having changed and calls the view's `update()` method. The `update()` method obtains the representation and displays it as described above.

The ability to update views dynamically allows Mandala clients to function as dynamic cache visualizations, sessions histories, and look-ahead caches. It also allows people to share their new imagemaps, because each time a person saves a new imagemap, the Mandala server adds it to the maps group and updates the clients as described above.

In addition to updating views dynamically, the client's view-updating process encourages representations to be requested one at a time. This allows requests from multiple client views to be interleaved, so multiple views can be updating at once. In addition, because there is a short delay between client requests, there is also a short delay between server responses. This is important for the client's main loop,

which processes incoming messages, as well as user input. User interactions remain responsive when messages from the server are short and intermittent.

8.10 Session Groups as a Look-Ahead Cache

The Mandala server can tell which web page was most-recently requested in a browsing session by monitoring the proxy server's application channel. The Mandala server can also tell which web pages are one link away from the current page by issuing a `ANCHOR_FILTER` request to the proxy server with the current page as an argument. Additional `IMAGE_FILTER` requests indicate those images on pages that are one link away from the current page. By displaying these images in a client's session view, Mandala session groups can function as a visual look-ahead cache.

A visual look-ahead cache may be particularly useful when used in combination with a search engine. For example, after querying a search engine in a web browser, the search engine returns a web page listing the query results in textual form. Usually the web pages that match the query are displayed with their titles, URLs, and a short summary of text from the page. It is often difficult to determine which of the query results are relevant, because the textual representations must be read iteratively, they may be redundant, and the hyper-links to them may be broken (search engine problems are reviewed in Section 2.8).

By contrast, when selectable images from the pages that match a query are displayed, it may be much easier to determine which of the query results are relevant. Relevant images have a good chance of corresponding to relevant pages. It should also be easier to identify relevant images in a large collage than to identify relevant textual links in a large page of text.

Mandala's look-ahead cache has another advantage. In some sense, a very general way to form groups of indexed information is to use a query language. Query languages identify the subset or group of information that matches the query. A

general way to let people define groups of images, therefore, would be to incorporate a query language into the Mandala server. But when Mandala is used as a look-ahead cache, any web search engine's query language can be used to define groups. Not only is this more general than using a single query language, but it also has the advantage that Mandala needs no explicit code for supporting query languages.

8.11 Summary

This chapter has described how Mandala's client/server architecture supports multiple clients with different capabilities. Each client must adhere to the protocol described in Section 8.1. Clients must connect to a Mandala server and send it requests of the form shown in Table 7.1, to which the server responds with messages of the form shown in Table 8.1. Details have been provided for how to signal web browsers to display web pages, which should also be useful for future Mandala clients (Section 8.2).

The remainder of the chapter described how the current Mandala client works. The current client runs as a Java application on a range of platforms. The client supports multiple views that each work the same way – each view displays the members of a group of image representations (Section 8.3). View uniformity makes the client easy for people to learn to use. The layout and animation settings, which are used by views to display images, have been described (Sections 8.4 and 8.5). Each view has the same menu interface, which is described in Section 8.6. Each view has the same drag-and-drop facilities, which let people edit imagemaps and groups (Sections 8.7 and 8.8). Finally, each view implements the same dynamic updating procedure (Section 8.9). Dynamic view updating allows people to use image representations to monitor caches and histories. The use of dynamic view updating for implementing a visual look-ahead cache has also been described (Section 8.10).

Part III

Future Directions for Image Representations

Chapter 9

Future Work

This dissertation focused on creating a platform for supporting the investigation of image representations and their visual organization. One way to measure the success of a system is by the number of ideas it inspires for future work. Although a few applications of image representations have been explored, there is much more work to be done. This chapter describes future extensions for the Mandala system, several of the many possible directions for future research, and possible experiments to evaluate the utility of image representations for access and organization of web information.

9.1 Future Extensions

There are many possible future extensions to the current Mandala system. It would be particularly interesting to devise more ways to group representations automatically. One possibility would be to extend the proxy server to build full-text indexes of retrieved pages incrementally and group representations based on the similarity of their associated text. Another possibility would be to group images based on a person's interaction history with them. For example, by grouping representations from commonly accessed paths of hyper-links.

While Mandala has so far focused only on two-dimensional groupings of image

representations, images can also be grouped in time. Mandala client views have a "centered" layout option, which centers images in the view while it cycles through them – much like a slide-show. It would be useful to explore further options for displaying groups as animating cycles of centered images. Particular orderings may be more useful than others. Groups could be stored as a single GIF89 animation, although they would lose much of the interactivity afforded by an imagemap representation.

There are several basic improvements planned for the current Mandala client. Although double-buffering is used to prevent flickering, images still appear to "pop on" the screen abruptly. Allowing images to dissolve on to the screen would be much more aesthetically pleasing. Currently, people can save groups as imagemaps, but the imagemap dimensions are taken to be the size of the group's view. It would be better to allow people to sweep a box to specify the imagemap's dimensions. Also, the number and type of layout algorithms on the client are limited. It would be convenient to include a version of the image server's spiral layout algorithm on the client. It would also be convenient to provide a method for allowing people's manually-positioned representations to be aligned to a grid.

There are a large number of digital video files on the web. It would be very interesting to extend a Mandala client to let people access and organize video representations.

9.2 Possible Directions

One possible future direction would be to implement additional Mandala clients. An applet client has long been planned. A single applet client might be a good way to monitor a single session group. Multiple applets per page might even allow people to edit groups without leaving their browser. A pad++ client would be useful for exploring multi-scale aspects of layout and grouping by piling (instead of using a

separate window for each group). A VRML client would be useful for exploring whether or not positioning images in a 3-D environment truly helps people remember more information.

While Mandala has focused thus far only on visualizing web images, one possible future direction would be to attempt to build an audio browser to help people hear web audio files. Imagery and audio have a large number of commonalities. As with images, people learn from audio and are comfortable using audio to interact with technology. Digital audio representations are less efficient than digital textual representations. Audio, particularly music, conveys a richness and depth of peripheral and subliminal information, which like imagery, can be appreciated immediately, although the details may take longer to interpret. With music, multiple tracks can be appreciated at once, although this is less possible with more symbolic audio streams, such as conversations. Also, of course, audio has duration and requires time to be appreciated. For these reasons audio representations may not scale as well in space or time as image representations.

Finally, this project has focused almost exclusively on the drawbacks of textual representations and the benefits of image representations. This focus has allowed progress to be made in the areas of user-interface design and system design for web-based visual information systems. In practice, however, visual and textual representations are complementary. More work needs to be done exploring different combinations of visual and textual representations and identifying applications where the different combinations would be most appropriate.

9.3 Image Representation Evaluation

Mandala provides an excellent environment for studying the utility of image representations. The proxy server cache can be loaded with particular sets of pages before being disconnected from the web, restricting web access and providing a controlled

environment for experiments.

For example, an experimenter could supply different groups of people with different tools and record the time required to perform common tasks. One group might be given only a web browser, while another might be given a web browser with Mandala, a third might also be given a web browser and Mandala, but Mandala would be set to use textual representations only. The tasks might include finding a familiar web page, finding an unfamiliar web page in a familiar web site, finding an unfamiliar web page in an unfamiliar web site, browsing a web site for particular topics, comparing two pages, or comparing two sites.

The cache size can also be varied, allowing experiments that compare representation type as a function of available information. For example, the experiment could be repeated using different sizes of proxy server cache (e.g., 10K, 100K, 1M, 10M, 100M). It would seem from the psychological evidence presented in Section 3.13, that while textual representations would be more effective for some tasks, the overall effectiveness of image representations should increase as the size of the available information increases.

A second type of experiment might involve people's bookmarks. It would be useful to know how well web page titles and URLs help people remember what they have bookmarked. One possible experiment would be to show people randomly-selected web page titles and URLs from their own bookmarks files and ask them if what they are seeing is familiar. Another group of subjects could be shown images from their bookmarked pages to see if the images serve as better cues for remembering what they bookmarked.

A third type of experiment might study how people perceive groups of images. A survey could contain collages and questions. Some questions could be very general (such as "Write down a description of what appears above") to determine if people naturally see groups of similar images. Other questions could be more specific (such as "Write down a description of each group of similar images that appears above")

to determine what sorts of structures people see in groups of images.

9.4 System Evaluation

Mandala's logging facilities can be used to help determine how people use the system. Analysis of the Mandala server and proxy server log files could be used to correlate user activity in Mandala clients with proxy server requests from the user's web browser. Time-stamped log entries could give an indication of the sorts of higher-level tasks being performed by users. For example, a typical usage pattern might consist of users performing searches in a web browser, followed immediately by organizing images into groups and saving groups as imagemaps. Other usage patterns might yield deeper insights into how people use Mandala.

Image representations are an engaging way for children to surf the web. In order to determine if children prefer image representations to textual representations, and to determine if children have additional requirements for information systems, an informal user-study could be performed in which children using Mandala are encouraged to "think aloud" [73].

An online survey could also be used to solicit informal evaluations of Mandala. There could be questions on the survey asking for specific evaluations of various features of the system. There could also be general questions to help determine if people like using the system and if people think it is useful.

Chapter 10

Conclusions

This dissertation begins with the observation that while people are expert at interpreting and remembering images, almost all digital technologies use text to represent information. Using digital images to represent information is inefficient for computers, especially when compared to textual or other symbolic representations. Nonetheless, images are very efficient representations for humans, who can perceive and remember many more images than words. Engelbart recognized the importance of user-interfaces that let people represent and manipulate ideas visually; he considered "extremely sophisticated images" to be a necessary part of the ultimate interface for augmenting human memory and intellect [30] (see also Section 2.1). As digital storage prices decrease and network bandwidth increases, image representations become an affordable alternative to textual representations.

As digital technologies continue to converge with visual technologies, there will be even more visual information available and more need for people to access and organize it effortlessly. Traditional approaches to organizing visual information require the manual addition of codes or key-words, which are indexed with standard (text-based) techniques. Automatic approaches for organizing visual information use vectors of features, such as color or texture distributions, which are extracted from the image data. As with text-based document feature vectors, each vector determines an

angle in a high-dimensional space, and the similarity of two images (or documents) is determined by the closeness of their angles. Vector models support "visual queries," in which images are returned that are similar to an input image or sketch. Although these models do succeed for restricted types of images, the features they detect are generally poor indicators of image content for the wide variety of images on the web (see Section 2.13).

Images from web pages are often good representations for the content of the pages. A few systems use web images to represent web pages (see Section 3.1). Because retrieving images over the web can be slow, most of these systems have not been designed for real-time online use. Rather, the authors of these systems expect that image representations will be most useful for server-side applications. For example, on a web server, hypertext structure and associations of images to web pages may be known in advance. It is easy, in this case, to prepare image representations in advance, and use them to help improve navigation through the hypertext structure.

Montage uses web images to represent web pages, but avoids image retrieval delays, and their associated server-side restrictions, by obtaining images from a proxy server cache. Montage displays hundreds of images in a single view, making the contents of the cache visible, and immediately accessible. Cache visualizations reveal how a community uses the web and which information the community values. Cache visualizations are also a unique way to share information anonymously within a community. Montage also creates Postcards (i.e., visual site indexes) by grouping thumbnails of images from the same web site and saving them as imagemaps. Postcards can be annotated and are useful for sharing comments about web sites and providing visual bookmarks.

Displaying images out of context may violate the intentions of web page authors and publishers (see Section 3.11). Indeed, cache and web site visualizations are powerful client-side tools. Image representations not only help people access more information faster, they may also be able to help people see organizational patterns

in information by taking advantage of the human visual system's intrinsic ability to group related visual stimuli.

A new system has been created, called Mandala, which is a complete rewrite and generalization of Montage (see Section 3.2). Mandala is a platform for exploring the capabilities of using images to access and organize web information. Designed as a powerful architecture of reusable servers, Mandala is general and flexible enough to be used in a variety of different ways for a variety of different applications (e.g., cache visualization, browsing session visualization, web site visualization, etc., see Chapter 4).

Mandala provides basic support for image compression, decompression, scaling, meta-data extraction, meta-data rating, and imagemap creation, in the form of an image server component, called Imago (see Chapter 6). Imago works with any GIF or JPEG images on the web and can be used by any process that needs images converted into thumbnails or organized into imagemaps. Thumbnails are created using a novel scaling algorithm, which combines multiple invocations of a fast halving algorithm with, at most, one invocation of a slower more-general scaling algorithm. The quality of the resulting thumbnails rivals the results of much slower algorithms, while the results are computed almost as quickly as scaling by uniform point sampling.

Mandala also provides basic support for monitoring HTTP traffic, parsing HTML, and caching resources in the form of a proxy server component, called Mirage (see Chapter 5). Mirage responds to requests to describe its cache, which makes it possible to create cache visualization applications.

Mandala's graphical user interface (GUI) is implemented by multiple Mandala clients (see Chapter 8), which communicate with a single Mandala server (see Chapter 7). Mandala's client/server architecture allows different clients to have different capabilities. For example, fully-featured clients have been implemented as Java applications. Mandala's client/server architecture also allows clients to share information about each other's groups and imagemaps. Mandala's GUI provides basic support

for displaying hundreds of images. Each image in the GUI represents an associated web page, which can be accessed in a regular web browser by selecting the image (see Section 8.2).

Visual representations require organizational strategies that differ from familiar textual lists, indexes, and hierarchies. When people can see hundreds of images at once, they can determine visual similarity for themselves. The ability to perceive groups of similar images easily seems to be a fundamental aspect of human vision. Mandala is designed to exploit this human ability to perceive groups and make it easy to share visually-identified groups with the system. Each view in the GUI displays the members of a group of image representations (see Section 8.3). People can edit groups by dragging and dropping images between views (see Section 8.8). People can also save groups as imagemaps. When people have an easy way to indicate, rearrange, and save their visually-identified groups, they can organize information in ways that are meaningful and relevant to their tasks and goals.

The Mandala server creates and maintains certain types of groups, which people can copy and edit (see Section 7.9). Images from the same web site are grouped, as are images that were cached as a result of the same browsing session. Additional groups are created for each category in a user's Netscape bookmarks file, each cached image, and each cached page that has no images. The Mandala server stores groups as imagemaps (see Section 7.10). When the Mandala server starts, if cached images are found that belong in a group with an existing imagemap, the imagemap is updated to include thumbnails of the new images. By continually growing imagemaps, they become a cumulative public repository. For example, the visual site indexes in Figures 3.8 and 3.9 encourage people to explore the sites and will continue to grow as new pages are explored.

As groups of image representations grow, they take on additional structure. Some of these structures are easier for a person to perceive than for a machine to determine automatically. For example, it is easy to distinguish the images of planets

in Figure 3.8. On the other hand, some structures may be easier for a machine to determine automatically, particularly as the number of image representations increases. For example, it is difficult to distinguish the few paintings that were not painted by Picasso in Figure 3.9, although it would be easy for a machine, because they are on a different web page.

When viewing structured groups of image representations, people can benefit from layout algorithms that preserve their structure. While the present dissertation has explored several two-dimensional layout algorithms (e.g., random, grid, tile, spiral, etc.), only one preserves structure by using multiple invocations of a spiral layout algorithm (see Figures 3.10 and 3.11, as well as Section 6.2). By clustering images from the same web site, these layouts preserve information about the image's origin, helping people perceive higher-level patterns related to the web sites. For example, in Figures 3.10 and 3.11, the clusters of images function as multiple mini site indexes within a single imagemap.

In conclusion, the main contributions of this dissertation are as follows:

1. An argument for why image representations may be better suited that textual representations for helping people access and organize relevant information on the web.
2. An analysis of web technologies in terms of how well they help people access and organize relevant information.
3. A platform for studying how images can help people access and organize relevant web information.
4. A caching HTTP proxy server with extensions for describing its cache, parsing HTML, and monitoring HTTP traffic.
5. An image server that creates thumbnails with a unique, fast hybrid scaling algorithm, and that creates imagemaps with spiral layout algorithms.
6. A graphical user interface that 1) allows people to perceive groups of similar images quickly, and 2) allows people to share their visually-perceived groups easily.
7. An argument for view consistency in user interfaces as a way to promote reuse and ease-of-use.

8. A mechanism that allows animating views to update dynamically.
9. Working prototypes of several applications: a repository for visual site indexes, a visual history facility, a visual bookmark facility, and a cache visualization application.
10. An argument for cache visualizations as a way to increase cache hit-rates, increase access to relevant resources, and increase resource sharing, while revealing the dynamic access patterns of a community.

Although many of the techniques described in the present dissertation can be used in server-side applications to help publishers organize information for readers, the focus of this dissertation has been on client-side tools to help people access and organize information for themselves. Because the web is huge, chaotic, disjointed, and fragmented, organization strategies designed for small well-understood information sources seem less effective. Human visual perception, however, is well adapted to extracting complex information from complex images, quickly and accurately. The manner in which people distinguish groups of images is also significant – to a large extent, people see what they want to see, what they have been conditioned to see, and what they are looking for. Images provide a representation for information that is therefore incredibly accurate and efficient for helping people identify and organize relevant information. By harnessing the intrinsic powers of human visual perception, information systems that let people organize visual information for themselves are powerful tools for extending human memory and intellect.

Appendix A

Related Systems Implemented by the Author

Mandala has borrowed ideas from earlier systems implemented by the author. Dotplot is a system for visualizing textual similarity structures. Ishmail is a system for reading and classifying electronic mail. This appendix describes these earlier systems and how they influenced the design of Mandala.

A.1 Dotplot: Visualizing Textual Similarity Structures

Dotplot is a data visualization system that displays string-match patterns in millions of lines of text or code[41]. Input streams are tokenized (e.g., split into short strings by identifying space characters or newline characters) and plotted from left to right and top to bottom with a dot where tokens match. Various weighting and approximation techniques are used to allow plots to be calculated fast enough for use in an interactive browser (see Figure A.1). Dotplot's displays are interactive overviews of enormous amounts of data. The overviews allow people to see patterns that would be difficult, if not impossible, to identify with other techniques.

For example, Figure A.1 shows three views of 3500 lines of C code. The large window in back is an overview of the entire file (dark areas indicate matching lines). A black square near the middle indicates the viewing area of a detailed view (about 330 lines), shown in the window in the upper right. A smaller black square in the detailed view indicates the viewing area of a text view (13 lines), shown in the bottom window. Dotplot's linked views make it easy to determine that the pattern of shrinking diagonals in the detailed view is caused by a pattern of data structure initializations in the code. Because this pattern spans more than 300 lines of code, it would be difficult to appreciate in a text editor. The pattern would also be difficult to identify with related techniques, such as the UNIX *diff* utility or algorithms that compute longest common substrings.

Dotplot's interactive overviews are similar to Mandala's. Both present compressed representations of large quantities of data in a manner that allows people to identify high-level patterns. Both overviews are selectable to allow people to quickly determine more-detailed information and to help interpret the cause of a pattern. Dotplot's overviews, however, display multiple patterns of textual similarity. Multiple patterns of textual similarity form "textual similarity structures," much like those stored in full-text indexes used by IR systems to match textual queries with documents (see Section 2.8).

A.2 Ishmail: Reading and Classifying Electronic Mail

Ishmail is a mail reading system that helps people manage hundreds of messages a day[43]. Unlike typical mail-reading programs, which are designed to make it easy to read messages one at a time in the order they were received, Ishmail lets people define groups of messages (i.e., mailboxes or folders) and classifies incoming messages into groups automatically. Grouping is a flexible strategy for organization that

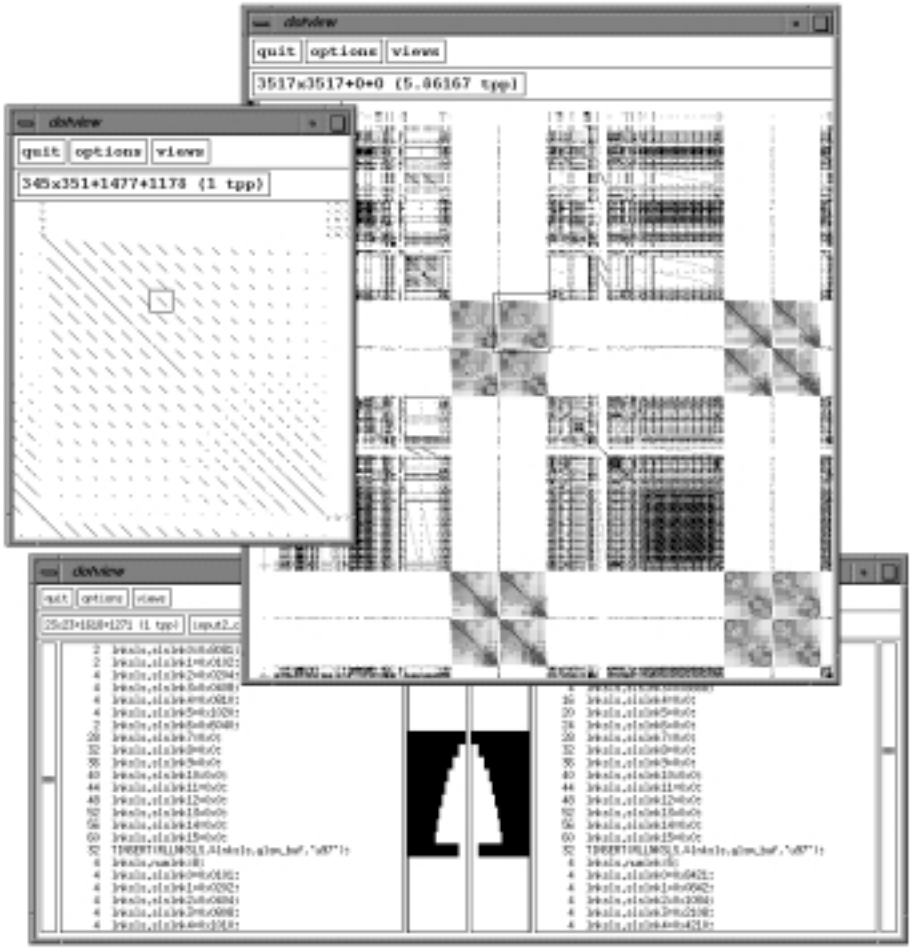


Figure A.1: Dotplot's user interface showing three views of 3500 lines of C code. Dotplot's linked views make it easy to determine that the pattern of shrinking diagonals in the detailed view is caused by a pattern of data structure initializations in the code.

helps people understand and assimilate larger volumes of information by changing the level of granularity from individual units (e.g., messages) to collections of units (e.g., mailboxes or folders). Grouping is also a natural strategy: Ishmail users had been grouping messages manually before they switched to automatic classification. It is appropriate to automate classification of messages into groups, because humans find classification of hundreds of messages a day time-consuming, tedious, and error-prone. Ishmail lets people access groups through textual interactive overviews – collections of textual summaries that can be ordered, filtered, and selected to access the content of the corresponding group member.

Ishmail is built on top of EMACS, a customizable text editor, and RMAIL, the standard mail-reading client for EMACS users. Figure A.2 shows Ishmail's four views: a message view and three interactive overviews. The message view (Figure A.2.a) and the mailbox summary view (Figure A.2.b) comprise the standard RMAIL user interface. The mailbox summary view is an interactive overview of the mailbox contents: selecting any line in the mailbox summary causes the current message window to display the corresponding message. Ishmail adds two additional interactive overviews, the summary of mailboxes view (Figure A.2.c) and the log view (Figure A.2.d). The summary of mailboxes view indicates the status of each mailbox, while the log view indicates when each message arrives and which mailbox it was sorted into. Selecting any line in the log or the summary of mailboxes views, causes the RMAIL windows to update, displaying the associated mailbox and it's current message.

Ishmail has been in continuous use by many researchers for several years. Based on enthusiastic user feedback, Ishmail seems to be very useful for the following reasons:

- grouping messages is a natural strategy for organizing large volumes of mail
- manual grouping of messages is tedious and error-prone
- automatic grouping of messages saves people time, is not too hard to specify, and is accurate enough to be worthwhile
- interactive overviews at multiple levels of granularity are effective mechanisms for organizing and providing access to information

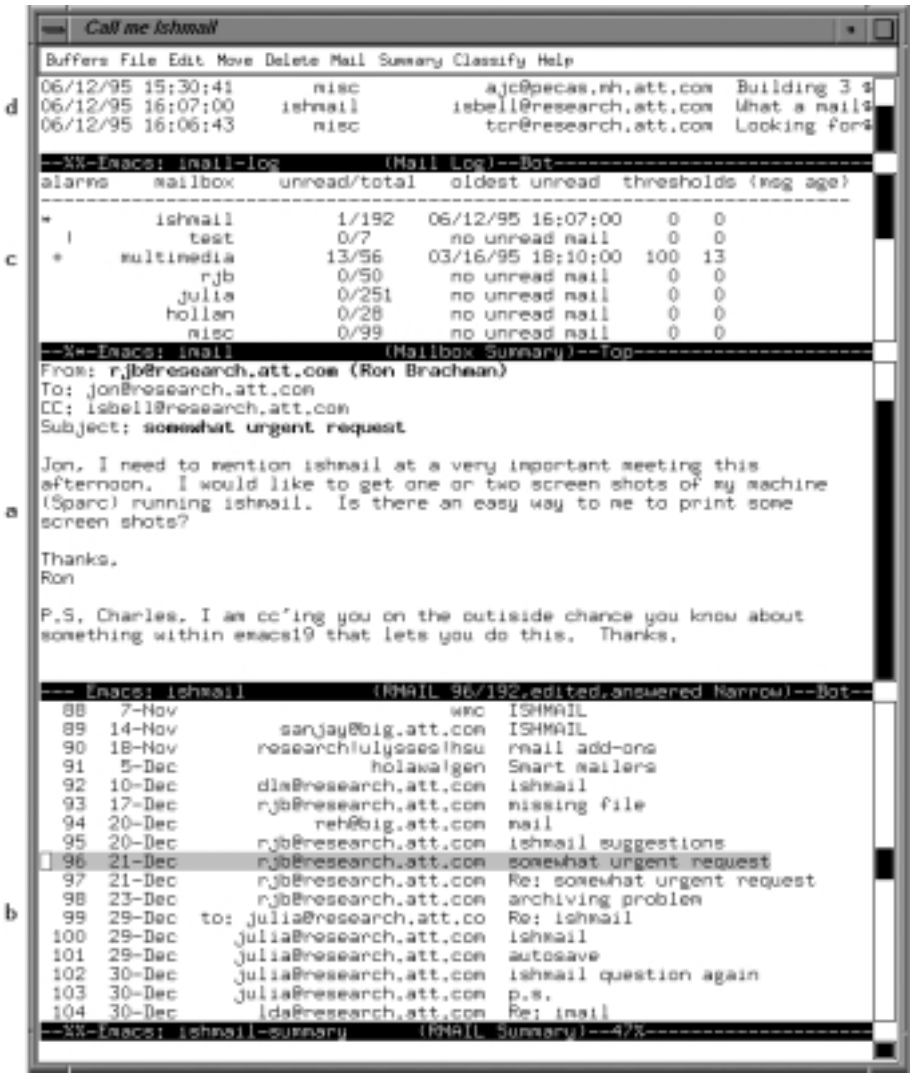


Figure A.2: Ishmail's user interface has a) a message view and three interactive overviews: b) a summary of the messages in a mailbox (bottom window), c) a summary of the mailboxes (second window from top), and d) a log of how each message was sorted (top window).

- it improves upon a system people were already using

Ishmail's success at grouping and classifying messages was one strong motivation for exploring the use of grouping and classifying image representations in Mandala. Ishmail's use of interactive textual overviews was also a motivation for exploring the use of interactive overviews of image representations.

Bibliography

[1] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: Limitations and potentials. In *Proceedings of the Fourth International World Wide Web Conference*, December 1995. `http://www.w3.org/pub/Conferences/WWW4/Papers/155/`.

[2] Our technology: Refine or cow9?, 1998. `http://www.cma.ensmp.fr/cow9/` and `http://altavista.digital.com/av/content/about_our_technology_cow9.htm`.

[3] AOLPress, 1995. `http://www.aolpress.com/press/2.0features.html`.

[4] Apple Research. Hotsauce (previously called Project X), 1997. `http://mcf.research.apple.com/ProjectX/`.

[5] Rob Barrett and Paul Maglio. Intermediaries: New places for producing and manipulating web content. In *Proceedings of the Seventh International World Wide Web Conference*, 1998. `http://wwwcssrv.almaden.ibm.com/wbi/www7/306.html`.

[6] Ben Bederson, Jim Hollan, Ken Perlin, Jon Meyer, David Bacon, and George Furnas. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing*, 7:3–31, 1996.

[7] Arthur Asa Berger. *Seeing Is Believing: An Introduction to Visual Communication.* Mayfield Publishing Company, 1989.

[8] Tim Berners-Lee and D. Connolly. RFC 1866 hypertext markup language – 2.0, November 1995. `ftp://ds.internic.net/rfc/rfc1866.txt`.

[9] Tim Berners-Lee and R. Fielding. RFC 2396 Uniform Resource Indentifiers (URI): Generic syntax, August 1998. `http://info.internet.isi.edu/in-notes/rfc/files/rfc2396.txt`.

[10] T. Boutell. `http://www.boutell.com/gd/`.

[11] G. Bower. Analysis of a mnemonic device. *American Scientist*, 58:496–510, 1970.

[12] RFC 1123 requirements for internet hosts – application and support, October 1989. R. Braden, editor. http://www.kisco.co.kr/~hollobit/RFC/rfc/rfc1123.html.

[13] Charles Brooks, Murray S. Mazer, Scott Meeks, and Jim Miller. Application-specific proxy servers as HTTP stream transducers. In *Proceedings of the Fourth International World Wide Web Conference*, December 1995. http://www.w3.org/pub/Conferences/WWW4/Papers/56/.

[14] Marc Brown and Robert Shillner. Deckscape: An experimental web browser. In *Proceedings of the Third International World Wide Web Conference*, April 1995. http://www.igd.fhg.de/www/www95/proceedings/papers/90/deckscape-final-v1/paper.html.

[15] P. J. Brown. Do we need maps to navigate round hypertext documents? *Electronic Publishing*, 2(2), July 1989.

[16] Peter J. Burt. Fast filter transforms for image processing. *Computer Graphics and Image Processing*, 16(1):20–51, 1981.

[17] Vannevar Bush. As we may think: A top U.S. scientist foresees a possible future world in which man-made machines will start to think. *LIFE*, 19(11):112–124, September 1945. Condensed from the Atlantic Monthly, July 1945.

[18] F. Campagnoni and K. Ehrlich. Information retrieval using a hypertext-based help system. *ACM Transactions on Information Systems*, 7(3):271–291, July 1989.

[19] Stuart Card, George Robertson, and William York. The webbook and the web forager: An information workspace for the world-wide web. In *CHI '96 proceedings*, pages 111–117, 1996.

[20] Luca Cardelli. Abstractions for mobile computation. Technical Report MSR-TR-98-34, Microsoft Research, 1998.

[21] Lara Catledge and James Pitkow. Characterizing browsing strategies in the world-wide web. In *Proceedings of the Third International World-Wide Web Conference*, April 1995. http://www.igd.fhg.de/www/www95/proceedings/papers/80/userpatterns/UserPatterns.Paper4.formatted.html.

[22] Patrick Chan. *The Java Developers Almanac 1998*. Addison-Wesley, 1998.

[23] F. R. K. Chung, M. R. Garey, and D. S. Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, 3(1):66–76, March 1982.

[24] Jeff Conklin. A survey of hypertext. Technical Report STP-356-86, MCC, February 1987.

[25] J. Cove and B. Walsh. Online text retrieval via browsing. *Information Processing and Management*, 24(1):31–37, 1988.

[26] David Crocker. Rfc 822 standard for the format of arpa internet text messages, August 1982. `http://www.kisco.co.kr/~hollobit/RFC/rfc/rfc822.html`.

[27] Chris Dodge, Beate Marx, and Hans Pfeiffenberger. Web cataloguing through cache exploitation and steps toward consistency maintenance. In *Proceedings of the Third International World-Wide Web Conference*, April 1995. `http://www.igd.fhg.de/www/www95/proceedings/papers/50/AWI_Database/AWI_Database.html`.

[28] P. Domel. Webmap – a graphical hypertext navigation tool. In *Proceedings of the 2nd International World-Wide Web Conference*, 1994.

[29] Fred Douglis, Thomas Ball, Yih-Farn Chen, and Eleftherious Koutsofious. Webguide: Querying and navigating changes in web repositories. In *Proceedings of the Fifth International World Wide Web Conference*, May 1996. `http://www5conf.inria.fr/fich_html/papers/P38/Overview.html`.

[30] Doug Engelbart. A conceptual framework for the augmentation of man's intellect. In P. Howerton and D. Weeks, editors, *Vistas in Information Handling*, volume 1. Spartan Books, 1963.

[31] Excalibur Technologies. Excalibur visual retrievalware technical summary, 1999. `http://www.excalib.com/products/vrw/vrwtechsum.html`.

[32] R. Fielding. RFC 1808 relative uniform resource locators. `ftp://ds.internic.net/rfc/rfc1808.txt`.

[33] R. Fielding, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1, HTTP working group, internet engineering task force, 1996. `http://www.w3.org/Protocols/HTTP/1.1/spec.html`.

[34] David Flanagan. *Java in a Nutshell*. O'Reilly, 1997.

[35] David Fontana. *The Secret Language of Symbols*. Chronicle Books, 1994.

[36] David A. Forsyth and Margaret M. Fleck. Identifying nude pictures. In *IEEE Workshop on the Applications of Computer Vision*, pages 103–108, 1996.

[37] Carolyn Foss. Tools for reading and browsing hypertext. *Information Processing and Management*, 25(4):407–418, 1989.

[38] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, November 1987.

[39] Steven Glassman. A caching relay for the World Wide Web. In *First International World-Wide Web Conference*, pages 69–76, May 1994. `http://www.research.digital.com/SRC/personal/Steve_Glassman/CachingTheWeb/paper.html`.

[40] Frank G. Halasz, Thomas P. Moran, and Randall H. Trigg. Notecards in a nutshell. In *ACM CHI+GI '87*, pages 45–52, April 1987.

[41] Jonathan Helfman. Dotplot patterns: A literal look at pattern languages. *Theory and Practice of Object Systems, Special Issue on Patterns*, 2(1), 1996. `http://www.cs.unm.edu/~jon/dotplot/tapos.ps`.

[42] Jonathan Helfman. Passive surfing in the communal cache, February 1996. Human Computer Interaction Consortium Workshop, `http://www.cs.unm.edu/~jon/montage/`.

[43] Jonathan Helfman and Charles Isbell. Ishmail: Immediate identification of important information, October 1995. `http://www.cs.unm.edu/~jon/ishmail/concepts.ps`.

[44] Ron Hightower, Laura Ring, Jonathan Helfman, Ben Bederson, and Jim Hollan. Graphical multiscale web histories: A study of PadPrints. In *Hypertext '98 Proceedings*, pages 58–65, 1998.

[45] Independent JPEG Group. `ftp://ftp.uu.net/graphics/jpeg/`.

[46] Islandnet. Web guardian, 1998. `http://www.islandnet.com/guardian.html`.

[47] Bela Julesz. Figure and ground perception in briefly presented isopodal textures. In Micheal Kubovy and James R. Pomerantz, editors, *Perceptual Organization*, pages 27–54. Lawrence Erlbaum Associates, 1981.

[48] Carl G. Jung. *Man and his Symbols*. Doubleday, 1964.

[49] John M. Kennedy. *A Psychology of Picture Perception: Images and Information*. Jossey-Bass Publishers, 1974.

[50] Andruid Kerne. CollageMachine: Temporality and indeterminacy in media browsing via interface ecology. In *CHI '97: Human Factors in Computing Systems: Late-Breaking/Interactive Posters*, pages 238–239. ACM Press, March 1997.

[51] S. Kerr. Wayfinding in an electronic database: The relative importance of navigational cues vs. mental models. *Information Processing and Management*, 26(4):511–523, 1990.

[52] Wolfgang Köhler. *Gestalt Psychology: An Introduction to New Concepts in Modern Psychology.* Mentor Books, 1947.

[53] Michael Lesk, Dennis Eagan, Dan Ketchum, and Carol Lochbaum. Better things for better chemistry through multi-media, October 1992. `http://www.lesk.com/mlesk/waterloo92/w92.html`.

[54] Clement H. C. Leung and W. W. S. So. Characteristics and architectural components of visual information systems. In Clement Leung, editor, *Visual Information Systems, Lecture Notes in Computer Science 1306.* Springer, 1997.

[55] Henry Lieberman. Autonomous interface agents. In *Proceedings of the ACM Conference on Computers and Human Interface, CHI '97*, March 1997.

[56] Ari Luotonen. *Web Proxy Servers.* Prentice Hall PTR, 1998.

[57] Ari Luotonen and Kevin Altis. World-wide web proxies. In *Proceedings of the First International Conference on the World-Wide Web, WWW '94*, 1994. `http://www1.cern.ch/PapersWWW94/luotonen.ps`.

[58] Yoelle S. Maarek and Isreal Z. Ben Shaul. Automatically organizing bookmarks per contents. In *Proceedings of the Fifth International World Wide Web Conference*, 1996. `http://www5conf.inria.fr/fich_html/papers/P37/Overview.html`.

[59] Patti Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[60] Patti Maes. Artificial life meets entertainment: Interacting with lifelike autonomous agents. *Communications of the ACM, Special Issue on New Horizons of Commercial and Industrial AI*, 38(11):108–114, November 1995.

[61] George Mallen. Back to the cave – cultural perspectives on virtual reality. In M. A. Gigante, H. Jones, and Rae A. Earnshaw, editors, *Virtual Reality Systems*, pages 265–272. Academic Press, June 1993.

[62] Richard Mander, Gitta Salomon, and Yin Yin Wong. A 'pile' metaphor for supporting casual organization of information. In *CHI '92 Conference Proceedings*, pages 627–634. ACM SIGCHI, 1992.

[63] Gary Marchionini. *Information Seeking in Electronic Environments.* Cambridge University Press, 1995. Cambridge Series on Human-Computer Interaction 9.

[64] David Marr. *Vision*. W. H. Freeman and Company, 1982.

[65] Miramba. Castanet, 1997. `http://www.marimba.com/products/castanet-tuner.html`.

[66] J. C. Mogul. Forcing http/1.1 proxies to revalidate responses, May 1997. `http://www.es.net/pub/internet-drafts/draft-mogul-http-revalidate-01.txt`.

[67] Sougata Mukherjea and James Foley. Visualizing the world-wide web with the navigational view builder. In *Proceedings of the Third International World-Wide Web Conference*, April 1995. `http://www.igd.fhg.de/www/www95/proceedings/papers/44/mukh/mukh.html`.

[68] NCSA. Ncsa mosaic common client interface, version 1.1, March 1995. `http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/CCI/cci-spec.html`.

[69] Donald Neal. The harvest object cache in New Zealand. In *Proceedings of the Fifth International World Wide Web Conference*, 1996. `http://www5conf.inria.fr/fich_html/papers/P46/Overview.html`.

[70] Ted Nelson. Literary machines: The report on, and of, project xanadu, concerning word processings, electronic publishing, hypertext, thinkertoys, tomorrow's intellectual revolution, and certain other topics including knowledge, education and freedom, 1981. Available from the author (8480 Fredericksburg #138, San Antonio, TX 78229).

[71] Ted Nelson. A new home for the mind? *DATAMATION*, March 1992. `http://www.datamation.com/PlugIn/issues/bestof/xanadu.html`.

[72] Jakob Nielson. The art of navigating through hypertext. *Communications of the ACM*, 33(3):296–310, March 1990.

[73] Jakob Nielson. Evaluating the thinking aloud technique for use by computer scientists. In Hartson and Hix, editors, *Advances in Human-Computer Interaction*, volume 3, pages 75–88. Ablex, 1992.

[74] Allan Paivio. *Imagery and Verbal Processes*. Holt, Rinehart, & Winston, 1971.

[75] Tomas Partl and Adam Dingle. A comparison of WWW caching algorithm efficiency. `http://webcache.ms.mff.cuni.cz:8080/paper/paper.html`.

[76] Randy Pausch, J. Snoddy, R. Taylor, S Watson, and E. Haseltine. Disney's Aladdin: First steps toward storytelling in virtual reality. *Computer Graphics, SIGGRAPH '96 Proceedings*, 1996.

[77] Oskar Pearson. Squid users guide, September 1997. http://cache.is.co.za/squid/.

[78] Brian Pinkerton. Finding what people want: Experiences with the webcrawler. In *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, 1994. http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/pinkerton/WebCrawler.html.

[79] P. Pirolli, J. Pitkow, and R. Rao. Silk from a sow's ear: Extracting usable structures from the web. In *CHI '96 proceedings*, pages 118–125, 1996.

[80] James E. Pitkow and Margaret M Recker. A simple yet robust caching algorithm based on dynamic access patterns. In *Proceedings of the Second World Wide Web Conference*, 1994. http://www.ncsa.uiuc.edu/SGD/IT94/Proceedings/DDay/pitkow/caching.html.

[81] Pointcast, 1996. http://www.pointcast.com.

[82] James R. Pomerantz. Perceptual organization in information processing. In Micheal Kubovy and James R. Pomerantz, editors, *Perceptual Organization*, pages 141–180. Lawrence Erlbaum Associates, 1981.

[83] D. Raggett. HTML 3.2 reference specification, W3C recommendation, January 1997. http://www.w3.org/TR/REC-html32.html#map.

[84] Rational Software. Purify, 1998. http://www.pureatria.com/products/purify/.

[85] H. Reese. Imagery in children's learning: A symposium. *Psychological Bulletin*, 73:383–421, 1970.

[86] David Metheny Robb. *The Art of the Illuminated Manuscript.* A. S. Barnes, 1973.

[87] Neil C. Rowe and Brian Frew. Finding photograph captions multimodally on the World Wide Web. In *AAAI-97 Spring Symposium Series, Intelligent Integration and Use of Text, Image, Video, and Audio Corpora*, pages 45–51, March 1997.

[88] Gerard Salton. *Dynamic Information and Library Processing.* Prentice-Hall, Inc., 1975.

[89] SavvySearch, 1995. http://guaraldi.cs.colostate.edu:2000/.

[90] Dale Schumacher. General filtered image rescaling. In David Kirk, editor, *Graphics Gems III*, pages 8–16. Academic Press, 1994.

[91] Erik Selberg and Oren Etzioni. Multi-service search and comparison using Metacrawler. In *Proceedings of the Fourth International World Wide Web Conference*, 1995. http://www.cs.washington.edu/research/projects/softbots/papers/metacrawler/www4/html/Overview.html.

[92] Peter Selfridge and Thomas Kirk. Cospace: Combining web browsing and dynamically generated, 3D, multiuser environments. *intelligence: New Visions of AI in Practice*, 10(1):24–32, 1999. see also http://cospace.research.att.com.

[93] J. Smith. *Integrated Spatial and Feature Image Systems: Retrieval, Analysis and Compression.* PhD thesis, Columbia University, 1997. http://disney.ctr.columbia.edu/jrsthesis/ and http://www.ctr.columbia.edu/webseek/.

[94] Lionel Standing, Jerry Conezio, and Ralph Norman Haber. Perception and memory for pictures: Single-trial learning of 2500 visual stimuli. *Psychonomic Science*, 19(10):73–74, 1970.

[95] Ken Turkowski. Filters for common resampling tasks. In Andrew S. Glassner, editor, *Graphics Gems*, pages 147–165. Academic Press, 1990.

[96] Steve Uhler. Web library help page. http://www.sunlabs.com/~suhler/html_library/help.html.

[97] Christopher De Hamel. *A History of Illuminated Manuscripts.* Phaidon, 1986.

[98] Logging control in W3C httpd, 1995. http://www.w3.org/Daemon/User/Config/Logging.html.

[99] WebCompass 1996. http://www.quarterdeck.com/qdeck/products/webcompass/.

[100] Max Wertheimer. Principles of perceptual organization. In David Beardslee and Michael Wertheimer, editors, *Readings in Perception*, pages 115–135. D. Van Nostrand Company, Princeton, 1958.

[101] James E. White. Mobile agents. In Jeffrey M. Bradshaw, editor, *Software Agents.* MIT Press, April 1997.

[102] Kent Wittenburg, Wissam Ali-Ahmad, Daniel LaLiberte, and Tom Lanning. Rapid-fire image previews for information navigation. In *Proceedings of AVI '98, Advanced Visual Interfaces*, May 1998.

[103] Kent Wittenburg, Duco Das, Will Hill, and Larry Stead. Group asynchronous browsing on the Worldwide Web. In *The World Wide Web Journal, Proceeding of the Fourth International World Wide Web Conference*, pages 51–62, 1995. `http://www.w3.org/pub/Conferences/WWW4/Papers/98/`.

[104] Michael Wynblatt and Dan Benson. Web page caricatures: Visual summaries for WWW documents. In *IEEE International Conference on Multimedia Computing and Systems*, 1998.

[105] Nicole Yankelovich, Norman Meyrowitz, and Andries van Dam. Reading and writing the electronic book. *IEEE Computer*, 18(10):15–30, October 1985.

[106] Frances Yates. *The Art of Memory*. Pimlico, 1964.