Dynamic Documents: Mobile Wireless Access to the WWW

M. Frans Kaashoek, Tom Pinckney, and Joshua A. Tauber
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA

{kaashoek, pinckney, josh}@lcs.mit.edu

Abstract

We propose dynamic documents *as an approach to extending and customizing the WWW/Mosaic for mobile computing platforms. Dynamic documents are programs executed on a mobile platform to generate a document; they are implemented as Tcl scripts. We have modified the NCSA Mosaic web client to run the dynamic documents it retrieves through a modified Tcl interpreter. The interpreter is designed to execute only commands that do not violate safety.*

To hide the latencies of slow links we have modified the Mosaic client to perform caching and prefetching. The policies for caching and prefetching can be under control of dynamic documents, allowing the strategies to be document-specific. Using dynamic documents, we have built an adaptive email browser that employs application-specific caching and prefetching strategies. Both the browser and the displayed email messages are dynamically customized to the mobile computing environment in which they run.

1 Introduction

Information retrieval is likely to be one of the driving applications of mobile computing. New, small, mobile devices that integrate the functionality of phones, faxes, pagers, and other communication tools are becoming available at low prices. Popular information retrieval tools need to be supported on this new technology to allow users to have information at their fingertips everywhere. Recently, the World Wide Web (WWW) and NCSA Mosaic have become extremely popular information retrieval tools [7, 4]. Since, Mosaic has been designed for browsers running on workstations connected by high-speed networks, its fixed policies and fixed interfaces prevent it from easily incorporating new applications or running on other computing platforms. In particular, wireless mobile computers are quickly overwhelmed by the data included in many multi-media WWW documents; for example, they often do not provide enough network bandwidth or display area.

Mobile computer environments differ from the workstation milieu for which Mosaic was designed in several key aspects. The physical resources of a mobile machine are usually more restricted than those of a workstation but may vary with location and available equipment. The combination of a slow processor, small memory, and a small disk is a particularly poor environment for the multi-media display common with web clients. Mobile computers often use wireless network links [14]. In general, we assume mobile clients will be connected to information rich networks. Only intermittently will mobile computers be wired or completely disconnected. Web traffic tends to consist of large bursts of activity during retrieval of multi-media documents alternating with quiescent periods while users examine the display. Wireless communication often cannot deliver the peak performance users expect in an interactive multi-media application.

A number of groups are addressing the problem of adapting the WWW/Mosaic by extending the HyperText Transfer Protocol (HTTP) (e.g. Common Gateway Interface), the HyperText Mark-up Language (HTML), or by introducing new protocols (e.g. HTTP+) [2, 3, 1, 20]. Unfortunately, these approaches are rigid and constrained because protocols function as standards for information exchange and, therefore, cannot be easily modified. We address this inflexibility by introducing *dynamic documents*. Dynamic documents are programs executed by Mosaic clients in order to generate the actual information displayed to the user.

Execution of a dynamic document may include any actions related to the document. Other documents may be fetched, new information may be generated, or information local to the client may be accessed. Safety is guaranteed by using an interpreter that executes only safe commands. We have implemented dynamic documents by storing documents as Tcl scripts and adding a new MIME content type [19, 8, 18]. We modified the NCSA Mosaic client to

To appear in *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

run dynamic documents it retrieves through a Tcl interpreter. The resulting document is then displayed.

In addition, we have altered the NCSA Mosaic client to hide the latency of wireless networks by caching and prefetching. The caching and prefetching can be made more "intelligent" by using dynamic documents to control the policies. Dynamic documents can even be used to "hoard" information to take advantage of brief periods of wired connection or to hide brief periods of disconnection. It is still unclear whether the network bandwidth consumed by document prefetching could be better used in other ways to provide interactive performance to the user.

In the next section we describe how dynamic documents can be used to address adapting Mosaic to mobile environments, and how we have implemented dynamic documents. We then describe the mobile Mosaic client that implements dynamic documents, caching, and prefetching, and how these features are used in an adaptive email browser. An important note: although the Mosaic client is designed for mobile computers, it is currently running on a stationary computer; we are in the process of porting it to a mobile wireless environment. Finally we describe related work and conclude.

2 Dynamic Documents

In order to address the variable resource requirements of mobile computers accessing the WWW, we propose *dynamic documents*. Dynamic documents are programs executed by WWW clients in order to generate the actual information displayed to the user. Execution of a dynamic document may cause the client to perform any number of actions in order to generate a final presentation to the user; other documents may be fetched, new HTML may be generated, information local to the client may accessed, or an entire X interface may be generated.

Dynamic documents are flexible enough to address many mobile computing resource constraints. Documents can be customized at the client depending on available resources. For example, if the screen size is small, the script can select some core subset of information to display to the user. If the bandwidth is limited, the fetching of large multi-media objects can be delayed and placed in the background while the user views more accessible information. Any location information available to the client can also be used to customize the document.

To a certain extent, the largest difficulty with dynamic documents is that they may be too powerful. The variety and complexity of possible dynamic documents raise three issues: safety, portability, and authoring. We are able to ensure the safety of downloaded documents be using an interpreter that limits the operations a script may perform. Similarly, we ensure portability by selecting a well known standard scripting language: Tcl. Since dynamic documents are so general, we are investigating the possibility of assisting authors by automatically generating certain types of scripts. For example, it is possible to process standard HTML documents into scripts that restrict the amount of data initially transmitted to the client while fetching the rest in the background.

Dynamic documents do not entirely eliminate the need to extend protocols in Mosaic. In order to add functionality to the client we need to define the primitives available to be called by a dynamic document. For example, we specify the manner in which dynamic documents access local information. For our prototype, we have specified a simple protocol that gives dynamic documents access to local architecture information and the ability to display and fetch documents.

We decided to place a great deal of functionality at the mobile computer. The client must perform all document processing in addition to providing the Mosaic interface. Other configurations are possible. With proper protocol specifications dynamic documents could execute at an intermediate relay server or at the document server itself (similar to the NCSA Common Gateway Interface). However, for the foreseeable future, wireless bandwidth will be a more scarce resource than mobile computing processing cycles. In addition, placing processing on the client side is inherently more scalable. Processing at the client does not impede server performance. Indeed, since mobile clients can elect to retrieve only small documents, server load may be further reduced.

2.1 Implementation of Dynamic Documents

Dynamic documents are implemented as Tcl scripts that are executed upon reception from a WWW server. We have modified the NCSA Mosaic client to provide hooks to the Tcl interpreter in order to fetch or display HTML. In addition, a script can register callbacks so that it can receive notification of events from or provide needed information to the web client. Additionally with tk scripts, dynamic documents can provide the full functionality of X programs.

Our web client handles scripts as just another MIME type. Every document retrieved via HTTP has an associated MIME type that gives the format of the data contained in the document. For each format, Mosaic has an interpreter, either internal or external, that is capable of presenting a view of the data. The "viewer" for scripts is simply the Tcl interpreter.

The script and WWW client interact in several ways. First, we altered Mosaic to provide access to a number of variables that describe the hardware platform on which the client is running. Second, several new Tcl commands have been added to support the fetching of HTTP objects and the display of HTML from within scripts. Third, scripts can register procedures in the script as callbacks. The caching and prefetching sections below discuss several possible uses of the callback mechanism.

One use of callbacks and dynamic documents is to perform interface processing locally on the client. For example, a document may register callbacks to receive certain mouse-click events, the contents of fill-out forms, or keystrokes. In this way, a document may present an interface to the user without necessarily requiring any server interaction, thus decreasing the client's need for bandwidth to the server and reducing the server's load.

If even more interface flexibility is needed, tk scripts may be used. Tk is a set of extensions of Tcl that allows X applications to be quickly created. Thus, by using a tk script, a document author may present an arbitrary X-based interface, albeit in a separate window.

3 A Mobile Mosaic Client

We have designed and implemented a mobile WWW client based on NCSA Mosaic version 2.4 that supports dynamic documents, caching, and prefetching. We initially implemented the client on stationary SPARCstation 10 running Solaris 1.0. We have ported the client to IBM ThinkPad 350Cs running Linux-1.1.49 connected by a 1 M-bit/s WaveLAN. By running it on a stationary computer we were easily able to attract a user community to test out the client and to collect traces on the usage of dynamic documents, caching, and prefetching. We are using these traces to compare policies.

In this section we will discuss the implementation of caching and prefetching, as well as an example application—an adaptive e-mail browser—that employs dynamic documents for application-controlled caching and prefetching and for dynamic-extension of the Web client interface.

3.1 Caching

Caching is an effective technique to improve interactive performance. In a wired environment, the document cache is probably best stored at a proxy server so that it can be shared by an entire group of users. This architecture benefits from the locality due to the common interests and communications among users sharing the proxy server. In contrast, wireless connections often have low variable bandwidths that are less reliable. It is unclear where to place a proxy server for a mobile user who may roam among several wireless connections. Thus, the cache in a wireless environment can be better placed at the mobile computer itself. We have chosen to make the cache persistent across sessions in order to benefit from locality between Mosaic sessions. Additionally, a persistent cache may be pre-loaded for disconnected operation, as in the Coda file system [15].

We have implemented caching at the HTTP level for ease of implementation. When an HTTP request is made for a document that is not currently cached, the document is requested from the server, and an exact copy of the server's response is recorded. Later, when requests for this Uniform Resource Locator (URL) are made, the cached data is simply replayed as if it had come from the network layer [5]. When the document cache is full, a randomly picked entry is discarded.

Since WWW servers do not provide callbacks, stale cache data cannot be detected. To attempt to maintain the correctness of cached data, entries are cached for only a document-specific period of time. The period of time used is an estimate of the time until the next modification. The estimate is made by assuming that the time between when a document is fetched from a server and when it was last modified on that server is proportional to the time until the next modification will be made. We provide dynamic documents with the additional ability to explicitly flush items from the cache. This is the same basic method used by Glassman for the caching relay server [10].

This global caching policy is adequate. Several users have used our modified Mosaic on a SPARCstation 10 during the three-week period of our trace and have experienced a cache hit rate of about 40%. The document cache is secondary to the standard Mosaic image cache and the Mosaic history cache. Thus our caching statistics only reflect the load presented to our document cache. All other references for images are first handled by the image cache. Thus, the three caches may combine to display a hit ratio higher than that measured at the level of the document cache.

However, the global caching policy is not always correct. At times stale data is used while at other times data is needlessly fetched. We solve this problem by allowing the cache to be controlled by dynamic documents. A dynamic document can register a callback with the cache subsystem. This callback will be used by Mosaic to determine if the cached entry is fresh or stale. If the callback returns that the entry is stale, Mosaic will fetch the document again from the server. Otherwise, the cached copy will be used. Dynamic documents can also invalidate particular entries, forcing the documents to be fetched from the server the next time they are referenced. This arrangement allows dynamic documents to implement their own cache coherency protocols. The adaptive e-mail browser described in Section 3.3 below uses this mechanism to implement a caching policy that only fetches data when it has changed on the server.

3.2 Prefetching

To hide the latencies of slow wireless links we have implemented document prefetching. Documents are prefetched by forking off a copy of Mosaic and then performing a document fetch in the forked copy, which loads the document into the cache. This implementation is not an efficient way of prefetching, but because the current implementations of Mosaic are single-threaded, we are forced to use this approach. Future versions of Mosaic promise to lift this restriction and thus allow much more efficient prefetching. If a user requests a document while prefetches are in progress, the prefetches are suspended so that the full network bandwidth may be given to the user's request. After the user's request completes, any prefetches are allowed to continue.

Currently the default prefetching strategy is to prefetch documents based on the user's Mosaic history list. Unfortunately, this strategy generates relatively few future cache hits while consuming a considerable amount of network bandwidth. Our traces show that on average 0.5 objects are prefetched for each object explictly fetched by the user. Of these prefetched objects only 2% are actually used in the future. We are investigating better strategies. One alternative solution is to have a local proxy server monitor access patterns and then periodically augment documents with information that specifies what documents a client should prefetch, given that that client has requested a particular document.

It is also possible for scripts to implement their own prefetching policy. Prefetching is only effective when future access patterns can be determined. Additionally prefetching uses network bandwidth - a scare resource in the mobile environment. Since scripts have detailed document specific knowledge they are sometimes best equipped to make prefetching decisions. We have implemented and begun testing prefetching policies. However it is still unclear how prefetching in Mosaic will affect other networked applications - or even other mobile computers - in the wireless environment.

3.3 An Adaptive Email Browser

To investigate the potential of dynamic documents, we have built an adaptive email browser in which email is accessed through a Mosaic client. Dynamic documents in this application are used to dynamically generate the browser and views of email messages. In addition, dynamic documents are used to implement application specific caching and prefetching strategies. This application did not require any changes to the mobile Mosaic client itself.

The adaptive email browser is organized as follows. The browser is a set of dynamic documents that is retrieved from an HHTP server when the user opens the URL for the browser. The dynamic documents then run on the mobile computer in order to generate the browser. The browser's interface runs entirely on the client so button clicks and typing do not necessarily result in messages to the server. Envi-

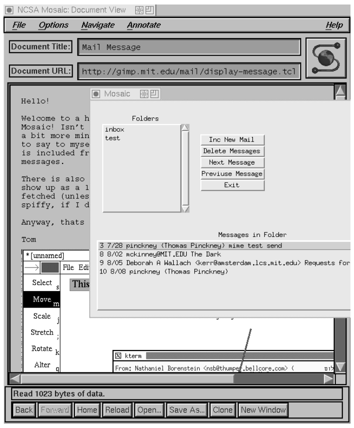

Figure 1. With a large screen and fast network, the browser displays the messages with text and graphics, and has an elaborate user interface in a separate window.

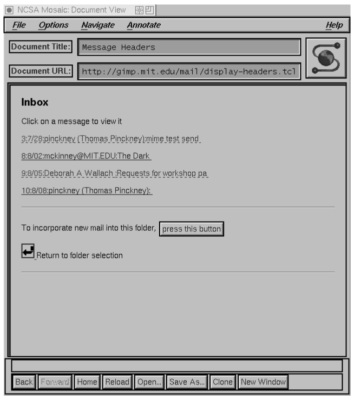

Figure 2. The user interface is adapted to a mobile environment; only text and links are displayed.

Figure 3. The message is adapted to a mobile environment; only text and links are displayed.

ronment variables (e.g., screen size and type of network) determine which parts of email messages the browser retrieves. At the client, the browser then generates an HTML page for the message to display. This setup dynamically moves the application interface from the server to the client. As a result, we reduce the load on the server, we reduce the network traffic between the client and server, and we improve interactive performance.

The browser adapts to the screen size and to the network speed of the platform on which it runs. Depending on the screen size, it either displays a menu of URLs that point to individual mail folders and messages, or it generates a more elaborate interface in a second window that is much more user friendly. In addition, messages adapt to screen size and network capabilities. For example, if the screen space is limited or the network speed slow, the browser generates links to pieces of the message rather than actually fetching and displaying the data completely.

Figures 1, 2, and 3 illustrate how the browser adapts to different environments. Figure 1 shows the browser for a platform with a big screen and a fast network. In this configuration, the browser displays multiple windows and displays the message with graphics and text. Figure 2 shows how the browser adapts to a mobile environment. It has only one window with a more primitive interface. Figure 3 shows the display of a mail message in mobile configuration of the browser; only the text and links are displayed. The important point about these figures is that the browser and mail message adapt by using dynamic documents and environment parameters. No changes are needed to the mobile Mosaic client.

The browser employs caching and prefetching strategies based on the objects that are included in email messages. The browser understands the MIME standard for bundling and transporting sequences of multi-media objects in a normal RFC822 compliant mail message [9]. When a message is turned into a collection of links, the browser will prefetch the data in the background while displaying the messages. In this way the fetching of potentially large objects (e.g., GIF or PostScript images) can be overlapped with the user reading text. The browser can do a better job at prefetching than an application-independent scheme, as the dynamic documents can take advantage of knowledge about the object type (e.g., text, GIF, PostScript).

Just as prefetching is under document control, so is cache invalidation. The browser controls which of its pieces are invalidated since only it has complete knowledge about when documents have changed. For example, the contents of different mail folders are cached and only invalidated when the user deletes mail from or incorporates new mail into the folder. This is vastly better than relying on a general caching and invalidation policy built into Mosaic.

4 Related Work

Using scripts for extending and customizing applications is not a new idea. Many examples of this approach can be found. A relatively successful example is PostScript [13]. Other examples include the Andrew Mail system, in which messages can contain fragments of LISP code that are executed at the receiver. This approach has been adopted for some more recent mailers that defined a safe subset of Tcl to do flexible mail processing. Recent work uses scripting languages to implement intelligent agents [6].

A number of projects have been extending the applicability of the WWW and its clients. In particular, Houh et al. implemented capabilities for shipping code from the client to the server to implement, for example, queries [12]. Mallery discusses an implementation of an HTTP server in Common Lisp and argues for dynamically moving functionality from servers to client using fragments of Lisp code [17]. Our work differs from this work in that we propose dynamic documents as a general approach for customizing and extending the WWW clients, and apply them to a mobile computing environment.

In the mobile arena, a number of groups have been working on WWW clients (or other user interfaces) for mobile computers. Most of this work focuses on the question what part of the client to run on the mobile computer and what part to run on a stationary computer [11, 22]. Landay and Kaufmann have used Scheme to ease the implementation of a split user interface [16]. Our work currently assumes that clients have enough processing power to run the client completely on the mobile computer. In the future we are planning to investigate how to implement dynamic documents on smaller mobile computers, and use Tcl scripts to split the interface, which should work particularly well for WWW clients that are being written using Tcl. Schilit et al. propose dynamic environment variables for customizing mobile applications while they are moving around; this idea could easily be incorporated into dynamic documents [22].

Caching and prefetching have been used for mobile computers. In particular, the Coda project has worked on making a file system that runs well on mobile computers [15, 21]. One of the potential advantages that dynamic documents offer is that they allow document-specific caching and prefetching. Such domain knowledge might reduce bandwidth consumption and might increase the number of cache hits. Glassman reports on a relay server that caches documents and gives a detailed analysis of its performance [10].

5 Conclusions

We have described the design and implementation of a mobile web client in both stationary and mobile computing environments. It supports dynamic documents, caching, and prefetching. Dynamic document are an approach to allow WWW clients to adapt easily to different computing environments; they contain programs, which are run at the client and generate the document to be displayed. We have used dynamic documents to implement a mobile mail browser that gracefully adapts to mobile computing environments. In addition, the browser uses application-controlled caching and prefetching strategies.

Our initial results based on the usage of our WWW client on stationary computers by a group of ten persons for three weeks indicate that dynamic documents are a powerful approach to customizability and extensibility. We have used application specific information with dynamic documents to customize information display and interface processing in a scalable manner. We have shown that a persistent cache works well. With dynamic documents we can guarantee the correctness of a cache while increasing its hit rate. At the same time, application-specific information is extremely useful to formulating a prefetching strategy that works well.

Acknowledgments

We thank David Gifford and James O'Toole for suggestions and ideas in the initial phase of this project. We also thank Deborah A. Wallach, Wilson Hsieh, and Sanjay Ghemawat for their comments on versions of this paper.

References

[1] Common gateway interface. *http://hoohoo.ncsa.uiuc.edu/cgi/intro.html*.

[2] Http: a protocol for networked information. *http://info.cern.ch/hypertext/WWW/Protocols/HTTP/-HTTP2.html*.

[3] Hypertext markup language (html). *http://-info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html*.

[4] Mosaic. *http://www.ncsa.uiuc.edu/SDG/Software/-Mosaic/NCSAMosaicHome.html*.

[5] Uniform resource locators. *http://info.cern.ch/-hypertext/WWW/Addressing/URL/Overview.html*.

[6] Intelligent agents issues. *CACM*, 37(7), July 1994.

[7] T. Berners-Lee, R. Caililiau, A. Luotonen, H.F. Nielsen, and A. Secret. The world-wide web. *Comm. ACM*, 37(8):76–82, Aug. 1994.

[8] N. Borenstein and N. Freed. Mime (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies. *Internet RFC 1521*, Sep 1993. Obsoletes RFC1341, Updated by RFC1590.

[9] David H. Crocker. Standard for the format of arpa internet text messages. *RFC 822*, Aug. 1982.

[10] S. Glassman. A caching relay for the world wibe web. In *Proc. First International World-Wide Web Conference*, pages 60–76, Geneva, May 1994.

[11] D. Goldberg and M. Tso. How to program networked portable computers. In *Proc. Fourth Workshop on Workstation Operating Systems*, pages 30–33, Napa, California, Oct. 1993.

[12] H. Houh, C. Lindblad, and D. Wetherall. Active pages. In *Proc. First International World-Wide Web Conference*, pages 265–270, Geneva, May 1994.

[13] Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, Reading, MA, 1985.

[14] Randy H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1):6–17, 1994.

[15] J. J. Kistler and M. Satyanarayananan. Disconnected operation in the coda file system. In *Proc. Thirteenth Symposium on Operating System Principles*, pages 213–225, Pacific Grove, CA, Oct 1991.

[16] J.A. Landay and T.R. Kaufmann. User interface issues in mobile computing. In *Proc. Fourth Workshop on Workstation Operating Systems*, pages 40–48, Napa, California, Oct 1993.

[17] J.C. Mallery. A common lisp hypermedia server. In *Proc. First International World-Wide Web Conference*, pages 239–247, Geneva, May 1994.

[18] K. Moore. Mime: Part two: Message header extensions for non-ascii text. *Internet RFC 1522*, Sep 1993. Obsoletes RFC1342.

[19] J.K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA, 1994.

[20] Dave Raggett. A review of the html+ document format. In *Proc. First International World-Wide Web Conference*, May 1994.

[21] M. Satyanarayanan, J. J. Kistler, L. B. Mummert, M. R. Ebling, P. Kumar, and Q. Lu. Experience with disconnected operation in a mobile environement. In *Proc. Symposium on Mobile and Location-Independent Computing*, pages 11–28, Cambridge, MA, Aug 1993.

[22] B. N. Schilit, M. M. Theimer, and B. B. Welch. Customizing mobile applications. In *Proc. Symposium on Mobile and Location-Independent Computing*, pages 129–138, Cambridge, MA, Aug 1993.