
Fifth International World Wide Web Conference
May 6-10, 1996, Paris, France

TeleWeb: Loosely Connected Access to the World Wide Web

Bill N. Schilit, Fred Douglis, David M. Kristol,
Paul Krzyzanowski, James Sienicki, John A. Trotter

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Abstract: The development of the World Wide Web (WWW) has made people reliant on continuous, high-speed, low-cost networks in order to get work done. Ideally, one should be able to browse the Web anytime, anywhere, whether connected to such a network or not. This paper describes the design of TeleWeb, a system we are building to support this goal. We believe that fine-grained cost control is crucial and have developed a "reactive architecture" that supports user-specified adaptation under various operating conditions. There are four key features to TeleWeb's architecture: costs are made visible to the user through annotated HTML; budget monitoring warns the user when operations exceed pre-specified limits; actions may be postponed and later triggered when conditions are met; and finally, user customization and system configuration values may automatically adapt according to the changing conditions of use. These mechanisms work together to provide an asynchronous, email-style of browsing in which users can work disconnected from a cache of documents, or trade off communication cost against information needs.

Key words and phrases: Mobile Computing, World Wide Web (WWW), Ubiquitous Information Access, Ubiquitous Computing.

1. Introduction

The World Wide Web (WWW) has surged to prominence since the introduction of NCSA's Mosaic browser in mid-1993. Users now regularly rely on widely distributed resources and freely add hypertext links into their own documents and hot-lists. One result is that people have become reliant on continuous, high-speed, low-cost networks in order to access information that they need in their work. However, ideally one should be able to

browse the Web anytime, anywhere, whether or not connected to such a network. In addition, when mobile users access the Web over multiple networks with different data rates and fee structures, they should be able to painlessly make time-money tradeoffs given their immediate information needs.

That is the premise for the TeleWeb system, whose design we present in this paper. Our work addresses a number of issues that are discussed in greater detail in later sections, including:

- What would the "look and feel" of such a system be?
- How would the system adapt to the characteristics of different networks?
- How would the system operate when not connected to a network?
- What would the system architecture be?

To illustrate how a laptop computer enhanced with TeleWeb might operate, consider the following scenario:

> It's late Thursday afternoon when Jack Webb gets email from his boss. His last-minute assignment is to fly to Denver to brief the Mayor's office on law enforcement policy. In order to prepare a presentation while en route, Jack looks through his browser's hot-list and finds several multimedia documents from his company's intranet that he would like to download. Jack's system shows him their estimated cost to transfer, based on configuration settings that apply whenever he is connected to his office LAN. Taking these costs into account, Jack rank-orders the resources, and his system starts to load his top choices. Unfortunately, there is not enough time to complete the download before he has to leave for the airport.
>
> In the airport lounge Jack turns on his wireless CDPD modem and starts browsing the Web, organizing the material for his presentation. Customers of this wireless service are charged per-byte, so as long as Jack mainly references cached resources the session will not be too expensive. He glances at the list of pending downloads initiated in his office and notes (with relief) that none are active. This is due to the current high cost of communication. Jack also notes that his current use of the system over CDPD has caused a slight change in the behavior and appearance of his browser. The  icons that Jack now sees in front of some hypertext anchors mark those links that require data transmission to follow because their resources are missing from his laptop's cache. Also his "personalized budget monitor" interrupts every now and then to notify that he is about to traverse an unusually expensive link.
>
> On the airplane Jack opens his laptop and continues to work. He follows a link in his Web browser to a graphic that he would like for the presentation. Since the system is disconnected, it tells Jack that the item is unavailable, but offers him the opportunity to connect via the in-flight telephone to load the graphic, or to postpone the request until later. Jack decides to postpone the request and adds it to the queue of pending downloads set to activate when a lower cost network connection is present.
>
> After he reaches the hotel, Jack connects his laptop's modem to the room telephone and his system automatically retrieves the graphic that he requested while airborne. He incorporates the image into the presentation while his other pending downloads complete in the background. With his presentation finished Jack gets a good night's sleep, knowing he is prepared for tomorrow's meeting.

The scenario above highlights some of the issues the TeleWeb project addresses, and some of the features our design offers. TeleWeb's goal is to allow a user to access the Web anytime and anywhere. In particular, our design copes with browsing the Web from a machine that is disconnected from any network and that may be

connected intermittently to a variety of heterogeneous networks. It employs a user's specifications in conjunction with knowledge about its environment to cope with expensive or nonexistent network connections. It keeps users informed of changes in network status and the underlying costs so they can adapt accordingly.

We saw how Jack adjusted his behavior depending on access costs. In general, a user may want to know which resources are available locally (on the laptop), which can be obtained cheaply, and which are expensive to obtain. The TeleWeb architecture presents cost information to users as they are browsing. For example, by annotating Web documents with small icons near each hypertext link or by redirecting link requests to an "over budget" warning page. In addition, we expect that users will want to dictate how conspicuous or subdued these notices are to appear under different conditions of use. Such fine-grained control is available to the user through setting variables and conditions.

The scenario presents a system that reacts to network-level changes. The cost information that Jack sees when his laptop is connected to his office LAN differs from when it is connected via a modem. TeleWeb permits cost indicators and other variables to adapt automatically according to low-level system changes. Furthermore, we saw TeleWeb react to restored connectivity by loading resources that Jack had left pending. This shows how user-specified conditions may be satisfied by system changes and cause pending actions to be triggered.

The scenario hints at the outline of TeleWeb's architecture. For Jack to be able to work independent of a network, his laptop clearly must have a cache of information that can be pre-loaded. Besides a cache, TeleWeb must also have support for system- and user-defined variables and expressions, and a way to evaluate and act on them in response to changing conditions. For reasons described below, we want people to be able to use their choice of Web browser, so some part of TeleWeb must intercept requests coming from the browser (to check for costs) and content going to the browser (to annotate links with cost information).

The remaining sections of this paper elaborate on these topics. Section 2 explains why it is challenging to support mobile users. Section 3 surveys the work of other researchers in the area. Section 4 presents the TeleWeb system architecture. Section 5 reports on the current implementation and how we expect the system to evolve, and Section 6 concludes the paper.

2. The Challenges of Mobile Web Browsing

In our current wired environment we expect to be connected to the network all the time through a predictable connection that costs a fixed amount of money. We expect resources to be available, and if we retrieved a document a week ago we should still be able to retrieve it today. Unfortunately high-bandwidth, low-cost communication is not ubiquitous. A mobile user may therefore deal with multiple networks of different technologies exhibiting widely differing bandwidth, latency, availability and fee structures, or they may expect to be disconnected for long periods.

Consequently, there is no truly *transparent* way to make a Web browser running on a mobile host operate the same as a Web browser running on a desktop system. The challenge in designing such a system is finding new ways in which we might access the Web that provide graceful degradation of service across increasingly difficult conditions. Such a system should be unobtrusive, predictable and always meet the expectations of users.

This section presents challenges divided into three areas: managing caches, controlling costs, and surviving change. In exploring this problem space we suggest some attributes that a successful system might have, and some techniques for attaining those qualities. The following section presents the TeleWeb architecture and describes in more detail how our system addresses a subset of these problems.

Managing Caches

Current browsers employ caches to avoid having to reload resources from a slower network if the user chooses to revisit them. These caches are optimized for a connected environment where it is reasonable to expect that a document ejected from the cache can be easily retrieved. However, for a mobile system that might be connected over an expensive or slow link, or be completely disconnected, such strategies are less suitable. We present a number of specific issues below.

Trading off cost & consistency. In a high-speed, connected environment it is reasonable to validate every resource to ensure that the latest version will be presented to the user. If the cost of validation is high however, the user may be content with a recent, but not necessarily up-to-date version. In other words, users may prefer to trade-off consistency for cost. For example, it may be appropriate to check documents on every reference when connected over an ethernet but only once per session when using a high cost cellular modem. Similarly, an expiration algorithm that discards resources based on their age might consider aging differently when communication costs are high.

Aggressively using storage. Since existing browsers are able to load documents as necessary from the network, they do not have to be too concerned about purging documents from their caches. A mobile-host cache may be less inclined to remove a document and may choose to keep it and consume more disk space as a consequence. Moreover, it may be reasonable for a mobile-host cache to aggressively use disk space and, instead of using a fixed allocation, use a variable sized amount of storage. The problem, of course, is that other applications the user runs may fail when they find there is no disk space available. The cache must therefore be able to monitor resource usage and relinquish disk space (or memory) if other applications need it.

Interacting with users. When a host is connected to a high-speed network, purging an "important" document is not a problem because it can always be retrieved. However, if the host becomes disconnected and the document is still missing the user might be inconvenienced. A mobile cache should therefore allow the user to advise how replacement of specific resources should operate. If the user believes a resource is important then it should be marked "keep." Others could be marked "keep as long as possible," and still others could be explicitly purged. Also, as noted by Goldberg and Tso [Goldberg93], it might be desirable to simply ask the user which resources to remove when it is necessary to make room in the cache for a new entry.

Controlling Costs

Existing browsers are not expected to control costs because they are usually connected over a high-speed, low-cost connection. However, for a mobile system cost control is essential so a number of techniques are discussed below.

Exposing Costs. Exposing costs to the user is an important function of a mobile-host based system and offers three benefits: users can decide if retrieving a particular piece of information is worthwhile; users can live within a budget; and the effect of user interactions will be made less surprising when time and expense are exposed. Potentially useful information includes cumulative costs, time and costs remaining for the current operation, and

time and costs for potential operations. An important specific instance of exposing costs is tagging the hypertext links which are not in the cache and which must be fetched from the network. By marking these "expensive" links, users can more easily control costs and also adjust their expectations of the system performance. Exposing caches in this way was first proposed by Goldberg and Tso [Goldberg93].

Monitoring costs. Although it is useful for users to see costs at all times, it can also be distracting. Quietly monitoring costs until they exceed some threshold or budget and then informing the user is one technique to address this problem. For example, accessing a link that will fetch an unusually large audio file might invoke a dialog asking the user for confirmation.

Limiting results. Costs might also be controlled by toggling parameters such as "image downloading" to reduce the amount of information returned. However, while graphics in some documents are superfluous, they are absolutely necessary in others. These problems argue for enhancements to the HTML language to identify images that *must* be downloaded to make the page understandable. A different approach is to use a wired-side image filter that can convert large, high-resolution images into black-and-white thumbnail-sized images able to be downloaded in a fraction of the time [Fox95].

Postponing operations. Costs might also be controlled by allowing operations to be invoked at a later time when inexpensive communication is possible. When a user comes across a link of interest that is not in the cache, it should be possible to queue the link for later traversal. Queued links might be retrieved during idle transmission periods or delayed until the user is connected to a low-cost, high-bandwidth network such as Ethernet. To support delayed interactions a manager program might show pending and recently executed operations, pointing out resources that have been added to the cache as a result.

Interrupting transfers. Costs can also be controlled by allowing downloads to be interrupted midstream and resumed later or by simply postponing the response part for later. For example, you might contact a multimedia news server to initiate the compilation of a custom video, but you don't want to wait for the result or transfer it immediately. Instead, you would prefer to delay the transmission until you are connected over a fast link. Possible solutions to this problem include servers that permit a part of a document to be downloaded at a time (*e.g.*, by specifying a byte range as part of the URL).

Maximizing channel utilization. Another area for cost cutting is a communication channel's long spans of inactivity that may occur between transmissions. In some cases these can make pay-per-minute channels a poor value. We can reduce idle spans by issuing many user requests together in one batch, or since the time is being charged whether used or not, by performing cache validation or cache pre-fetching on links that are "likely" to be taken. The actual link-following strategy will depend greatly on the parameters of the link itself, especially its cost model.

Surviving Change

Changing Interfaces. Another set of challenges is presented by the way a mobile user connects to the network. Ideally the user can plug into any network that is available (and for which they are authorized) and the system will be able to switch to the new network seamlessly and without the user's intervention. For instance a mobile user may be connected to an Ethernet in the office, then unplug the Ethernet and rely on a wireless LAN for a meeting, then go home and use a telephone line. All of these networks have different interfaces and the

system should be able to switch among them without a software restart. In addition the possibility that the "best" link may change over time should also be taken into account. If a telephone was in use and an Ethernet is then plugged in as well, the system should use the ethernet LAN to establish a faster and cheaper connection.

Changing networks. Another impediment to seamless operation is the use of firewalls (and other security measures based on point-of-access) which prevent a browser from connecting to resources in the same way on each access. When navigating through a firewall a host may have to authenticate itself or give a different address for URLs. In some cases resources that are behind firewalls may simply not be available. These issues might be addressed with a system that is constantly aware of the different connections available, including their security characteristics, and is able to choose the best one. Such a system should know about different login procedures for different networks from different locations, and must be able to authenticate itself to gateway machines.

Changing settings. In general, it is desirable that configuration parameters adapt themselves based on the current network. For instance, the choice of a proxy-caching server may change depending on whether one is inside or outside a firewall, or simply because different servers are best for particular networks.

Changing hosts. People tend to browse the Web from several different hosts. They might switch between one or more workstations in the office, use a laptop during meetings or while on the road, and at home they might access the Web using yet another host. Unfortunately, when the user has been browsing on one host and then wants to use another there is no way to "move" the session. A system supporting mobile users should endevour to make transitions to new execution platforms appear seamless. One way to do this is for hosts to checkpoint and transfer all useful browser state, including hot-lists, history, and cached pages whenever the user moves.

Changing capabilities. People may also browse the Web over many different *kinds* of machines such as workstations, desktop systems, laptops, and personal digital assistants (PDAs). The computation, communication, and graphics capabilities of these platforms vary widely. A workstation may have high resolution color while a laptop may have a grey-scale display. A more extreme example would be a PDA with no graphics capability or a telephone with only voice output. Clearly it will be difficult to present the same content across all platforms. One solution is to expect servers to provide different sets of documents for differents platforms, say color as opposed to black-and-white systems. Another approach is to translate sources into documents that meet the specific capabilities of the systems on which they will be displayed, for example, for a telephone the page might be converted into audio to be read to the user.

3. Related Work

The prospect of browsing the Web on mobile hosts has appealed to a number of researchers. Bartlett's Wireless World Wide Web (W4) [Bartlett95] and Gessler's PDA WWW [Gessler94] use Apple Newton PDAs as mobile WWW browsers. In these systems the PDA performs as a specialized graphical terminal communicating with a dedicated stationary host. This design is similar to that employed by the general purpose PARCTAB PDA [Want95]. These PDA projects demonstrate that deciding on the correct cut for the client-server function (and the resulting protocols) is critical since it defines both bandwidth and local processing requirements. The W4 system uses a low-level cut based on "screen" descriptions and then augments the client with a simple cache and prefetching of the next sequential screen in the document to provide reasonable performance.

A number of groups have been exploring browsing the Web on more powerful notebook class computers [Baquero95, Liljeberg95]. For the most part this work focuses on the question of which part of the system should run on the mobile host and which part on stationary servers. Both MobiScape [Baquero95] and Mowgli [Liljeberg95] use a proxy running on the mobile host in conjunction with a dedicated process running on the stationary server. Caching at the stationary support station reduces wait periods caused by fetching remote documents, and a mobile host cache supports disconnected operation. The Mowgli system uses HTTP header compression for improved bandwidth and long lived connections to avoid the TCP slow-start problem. MobiScape uses system profilers on both the mobile and support station that read user supplied lists of URLs and fetch those documents based on URL specific "recycling periods." Another project has looked at how to define the split dynamically through programmable documents [Kaashoek94].

The TeleWeb system differs from previous mobile WWW browsing projects in its goals and architecture. TeleWeb advances the issue of monetary cost control to a prime concern. It is our belief that a truly useful mobile Web browser must give the user fine control over managing costs. This has led to a "reactive architecture" that provides user controlled adaption under various operating conditions. For example, TeleWeb supports not only disconnected browsing of cached documents, but also a queue of user requests that will trigger when certain conditions are met, such as a high speed network connection. This introduces an asynchronous email style to mobile browsing that is not present in other designs.

In addition, whereas other systems concentrate on improving latency by communicating with stationary servers running special protocols, we believe that requiring dedicated servers reduces availability. Indeed, we see wireless modem compression technology, widely available Web caches, HTTP-NG's virtual sessions, and other efforts in the wireless and networking arena addressing many of the same general network level problems that make stationary proxies desirable. These are general problems that warrant, and are receiving, general treatment, making more specific solutions devised for wireless proxies less durable. Therefore, our research agenda is to concentrate on higher level "assistants" that through short term interaction provide added value for mobile users.

The TeleWeb system draws influence from work in a number of areas. This includes adaptive mobile systems work done at Columbia [Schilit91], Xerox PARC [Schilit93], and more recently CMU [Noble95]. Our initial emphasis is on system level adaptation but we believe the architecture can also support the related notion of location and context adaption [Schilit94, Schilit95, Voelker94]. Goldberg and Tso's proposal for *intelligently autonomous* user interfaces [Goldberg93] motivates our belief that operational costs must, at all times, be visible and comprehensible to users. The benefit of email-style interaction has been shown in a number of systems including Magic Cap [Boone95]. The Rover project is exploring similar issues using a tool-kit approach [Joseph95]. Mary Baker's MosquitoNet project [Baker96] is exploring network level issues for seemingly continuous network connectivity. Supporting our notion of casual assistants is Douglis and Ball's AIDE system for notification of changing URLs [Douglis96], and InfoPad's stationary HTTP proxies that distill images and postscript from HTML for limited bandwidth mobile hosts [Fox95].

4. TeleWeb Architecture

The TeleWeb system addresses several of the challenges that were described in Section 2. In particular it provides:

- **HTML annotation** that changes the appearance of documents so users can see which links are in the TeleWeb cache before they are traversed.
- **Budget monitoring** that warns users when operations will exceed pre-specified limits.
- **Conditional actions** that allow the user to postpone communication (and other actions) by queueing monitored requests that trigger actions when the associated conditions are met.
- **Dynamic customization and configuration settings** that can change values automatically as an effect of the changing conditions of use.

The principles driving TeleWeb's architecture are simplicity, predictability, and flexibility. We want to provide an environment that has predictable performance while allowing the user the flexibility to fully customize its operations. For instance, the user can set arbitrary operations to be triggered by changing connectivity or other variables changes. We also feel it is important to support commercial Web browsers instead of requiring a special TeleWeb browser. In our system all interaction, including TeleWeb's user interface, is through the user's browser. This means that the TeleWeb system does not have to keep pace with additions and changes to HTML, thereby easing portability and allowing the user to interact with a familiar environment. Finally, to ensure that TeleWeb can be run from any platform and connect to any network, the design does not require special network protocols or a dedicated wired-side proxy.

System Overview

TeleWeb is implemented as a daemon process run on the portable host that is interposed between the browser and the network. Because we assume that the daemon runs on the same machine as the browser, bandwidth and connectivity are not an issue. The TeleWeb daemon acts like a caching proxy [Loutonen94] in that it receives requests from the browser and either passes them onto the network or, if the page is in the TeleWeb cache, services the request itself. The TeleWeb daemon differs from standard caching proxies in that it will annotate the HTML sent back to browsers, intercept distinguished URLs for the purpose of user interface, and also, when certain conditions are met, redirect HTTP requests to built-in pages.

To employ TeleWeb, a user must configure their commercial browser by setting the HTTP proxy to be the local host's TeleWeb daemon and disabling their browser's disk and memory caches. This forces the browser to contact the TeleWeb daemon for each resource, keeping the HTML annotation accurate and only introducing a small delay since both processes run on the same host. It should be noted that the TeleWeb daemon takes control over a number of browser related preference variables, permitting automatic and dynamic control over their settings. Two examples are the proxy and cache consistency level. The TeleWeb daemon's proxy setting will normally be the original value from the user's browser but may be set to automatically change according to the location of use. The cache consistency level, either "once per session," "everytime," or "never" determines how accurate the user wants TeleWeb's cache to perform and may also vary depending on the cost of communication.

A block diagram of the TeleWeb system is shown in Figure 4.1. The interaction of the different TeleWeb components is described below.

A browser sends a request to the daemon where the *caching proxy* checks whether the request can be satisfied from the cache and if so returns the cached information. Otherwise, the URL is examined to see if it has a distinguished prefix matching `http://localhost/teleweb` and if so the request is handed off to TeleWeb's built-in *user interface (UI) server* for processing. At this point the request can only be satisfied by network

communication so a check is made to see if the network is available and whether fetching the link from the network would meet the user's "budget monitor" criteria. If these conditions are not met then an HTTP temporary redirect (302) is made to the UI server's Conditional Action form. This form serves to notify the user and allows them to continue, cancel, or to create a new conditional action, thereby postponing the operation. In the case when the pre-specified budget level is acceptable, the request passes to a remote server via the *communication manager*. The communications manager is responsible for setting up and tearing down connections (*e.g.*, modem connections) and informing the rest of the system of changes in network cost, bandwidth, and connectivity. In all cases, when the daemon returns HTML documents to the client browser, it first inserts user specified annotation strings around anchor tags letting the user see which links are currently in the cache.

In addition to the Conditional Action form produced on the "over budget" situation described above, the user interface server makes a number of forms available for examining, creating, and modifying TeleWeb variables and conditional actions as well as managing the disk cache. Providing a UI integrated with the daemon in this way means that the user need not leave the domain of the Web browser to configure and interact with TeleWeb.

All components in the TeleWeb daemon are tied together into a reactive architecture by *TeleWeb variables*. These variables are central to our design and are used at both a user and system level. TeleWeb variables provide a way to reflect system state, such as connectivity status and link charactertistics, as well as a way to manage user customization and configuration preferences. Changing a TeleWeb variable may trigger *conditional actions* which are pending actions associated with a condition. The condition part is defined by the user as an expression of TeleWeb variables, operators, and constants. For instance, one can request that certain URLs be downloaded if the bandwidth increases to over 1Mbps.

Finally, the TeleWeb daemon can be loosely coupled with any number of *wired assistants.* These processes execute on the wired side of the network and perform communication intensive high level functions. An assistant, for example, might monitor a set of Web pages for changes and provide a list of those that have been recently modified.

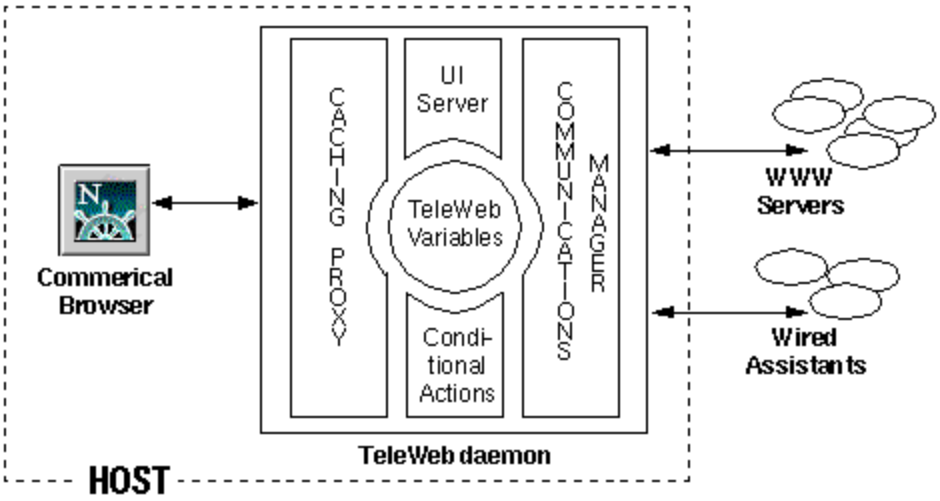

Figure 4.1: TeleWeb System Architecture

Components

The following sections describe the components of the TeleWeb system in more detail.

TeleWeb Variables

TeleWeb variables reflect the internal state of the system as well as the current customization and configuration settings. Each variable has a type (boolean, integer, float, text), a value, and a descriptive text used for documentation purposes. TeleWeb users can create, set, examine, or delete variables using forms produced by the TeleWeb UI server and viewed by the user's Web browser.

A number of TeleWeb variables are defined and maintained by system components. The communication manager maintains variables that reflect the communication bandwidth, communication cost, connectivity, as well as related user preference. If the system connectivity changes then the communication manager sets new values for all appropriate variables.

TeleWeb variables provide the basic mechanism for supporting a reactive architecture. They resemble events in an event-based model but afford persistence which we find useful for reflecting the state of a system. All interested parties are notified whenever a variable is set by the user or by a system component. Because variables are contained in the condition part of a conditional action, setting a variable may trigger a user-level action. Notification of variable changes is implemented efficiently by requiring that procedures declare ahead of time their interest in a particular variable. For example, when the data rate changes the conditional action component will be notified only when a user condition includes that particular variable. This means that TeleWeb variables can be used to reflect even fine-grained system state changes efficiently.

In addition to reflecting the internal state, TeleWeb variables are used to store *all* configuration and customization settings. The text used to annotate anchors absent from the cache is stored as a TeleWeb variable as is the name of a remote proxy. If the user desires, these preference variables can be used in the condition that triggers an action, or an action can be used to automatically set them, or they can simply be examined and changed by the user through the UI.

People will often have different preferences when browsing from home, or the office, or a hotel. We envision TeleWeb users creating one HTML page for each preference group. The page might have the title "Office Settings" and include a form with TeleWeb variables and default values for each situation. Invoking the form can set all variables at once using the UI server's built-in interface for changing TeleWeb variables. Of course, given the appropriate low level TeleWeb variables, we expect the user will want to create conditional actions that automatically invoke their preference forms.

Conditional Action

The TeleWeb system lets users set triggers to invoke future operations. The condition part is an expression composed of TeleWeb variables and constants. The action part is a qualified HTTP GET operation. For example, the user can postpone fetching a multi-media document until they are connected to an Ethernet. Another use of conditional actions is to automatically change TeleWeb preference variables, such as turning off HTML annotation. This latter change is possible because the TeleWeb UI itself interacts through HTTP operations.

Pending and completed conditional actions can be examined through the UI server. The result from a completed action shows the status plus a hypertext link to the cached resource (if one was returned). Normally, conditional actions are created through a dialog like the one shown in Figure 4.2. This form occurs as a redirection when the requested URL is not in the cache and the host is either disconnected or the operation does not meet the budget monitor constraint. When the user gets this form the URL and other information are already stored in hidden variables, thus simplifying the process of creating the conditional action.

One aspect of conditional actions that we are experimenting with is the notion of disposition that lets the conditional action be continually or periodically active. We think this would be useful, for example, to keep a recent copy of a newspaper cached on your computer whenever you are connected to an Ethernet.

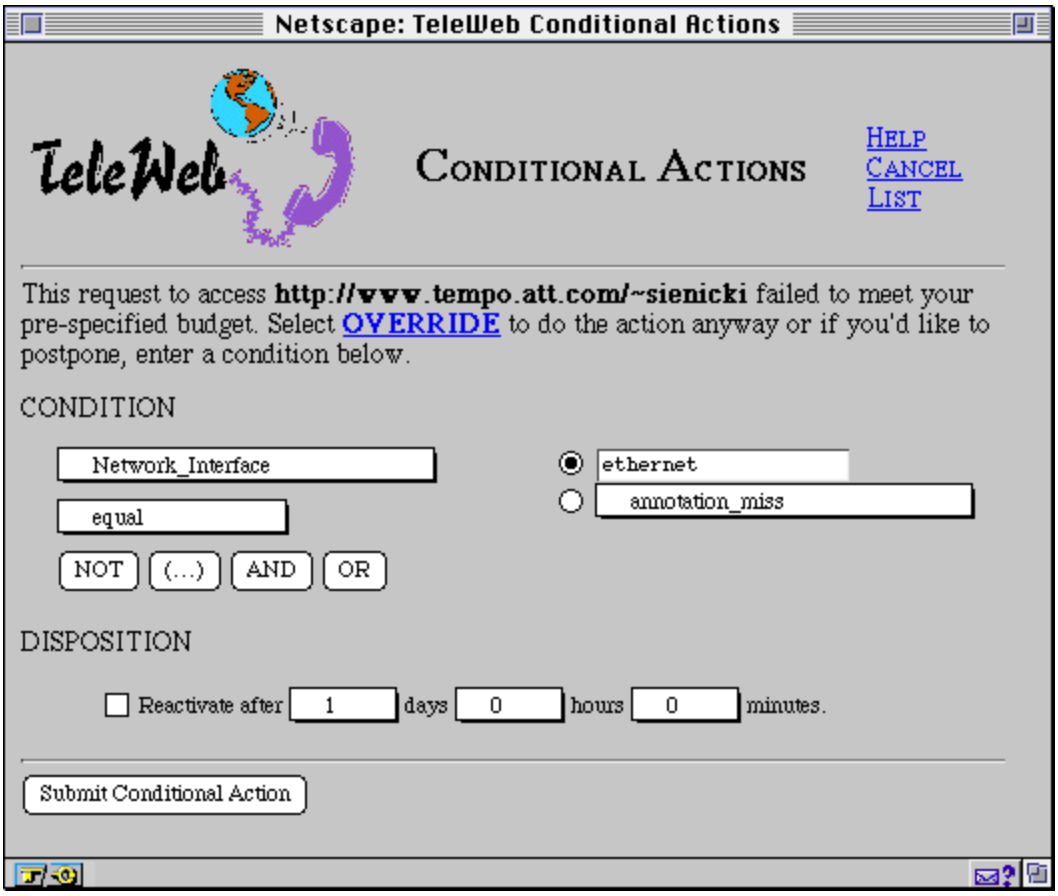

Figure 4.2: Creating a Conditional Action

Caching Proxy

The caching proxy decides how requests are serviced. It accepts URL requests from the browser, and either supplies the result from the cache, fetches it from the network, or uses an HTTP redirect to intercede the Conditional Action form shown in Figure 4.2.

In order to decide whether a request should be passed to the network or redirected, a check is first made to see if the host is connected. If it is, then a budget-monitor expression supplied by the user is evaluated and if the budget is met, the network transfer is initiated, otherwise redirection occurs.

The caching proxy is also responsible for annotating HTML documents as they are returned to the client browser. The reason this operation occurs at the last minute is because both the contents of the cache as well as the annotation texts may change, thus invalidating any stored annotated HTML. Currently the daemon annotates anchor tags using two pairs of TeleWeb variables. A prefix and suffix TeleWeb variable is used for anchor tags whose `HREF` is in the cache and a prefix and suffix for ones not in the cache. The use of the two TeleWeb variables permit users to insert arbitrary HTML before, after, or around anchors. Figure 4.3 shows a document that has been annotated to show the anchors that are absent from the cache.

Through TeleWeb variables, the user can specify the size of the cache as well as various tunable settings for replacement policies. The cache also has a user interface for examining the cache contents, pinning entries, and purging entries. These functions give the user complete control over the contents of the cache.

Figure 4.3: A document annotated to show hypertext anchors absent from the cache

Communication Manager

The communication manager sits between the proxy and the operating system which ultimately interfaces to the actual network connections. The proxy simply sends and receives data to the communication manager which transparently sets up and tears down connections using the most efficient interface available. This level of

indirection allows the communication manager to temporarily disconnect and then reconnect idle communications channels without interrupting the proxy. This is useful for modem or wireless connections where open idle links often cost money.

The communication manager informs the proxy of changing network conditions by setting action variables. The user can similarly modify the behavior of the communication manager (*e.g.*, specify preferred network interfaces and configure their costs) by setting action variables. Since the communication manager supplies information about the connection status, it assumes that if it is invoked by the proxy then it should make every effort to establish a connection to satisfy the request.

Wired Assistants

Part of the design philosophy of the TeleWeb system was to avoid a dedicated wired-side proxy because of decreased availability and increased maintenance required. Although there are many desirable functions a wired side proxy can perform, our view is that it is better to define a mechanism for temporarily connecting to added value services, which we call "wired assistants," rather than require a single dedicated proxy. We expect the user will connect and disconnect from these assistants as necessary according to the functions that they provide and the operating conditions in which users finds themselves. For example, there is no need for a wired-side proxy when the mobile host itself is on the wired side of the network docked at the user's desktop. One desirable aspect of this approach is that it creates a "market" for servers supporting mobile users similar to the market that has developed for Web search and index services.

Although we have yet to fully explore the issues of wired assistants, we are in the process of designing a number of value-added services described below:

- One example of a wired assistant is a server that offloads processing of delayed requests from the TeleWeb system to a backend server. The assistant stores a queue of pending operations that it subsequently retrieves and makes available to the TeleWeb system. The list of requests can be viewed or modified by the user. The user has the ability to prioritize, reorder, and remove pending requests.

- Another wired assistant we are designing reduces the amount of information from returned resources. When the TeleWeb system changes to a slow link, the `Proxy` TeleWeb variable can be set to a "Web Reader's Digestor" proxy. This wired side proxy reduces image depth, rewrites HTML, removes advertisements, and uses other techniques to reduce communication. A similar proxy has been built by [Fox95].

- AIDE is a system for tracking and viewing modifications to content [Douglis95]. Making these changes available to a mobile user is a natural and useful extension. Since the set of new pages is well-defined, it can be prefetched when the user is well-connected. Differences to pages can be precomputed and cached. Differences to pages that have already been cached can be transmitted to the TeleWeb cache, reducing communication time and expense while being transparent to the browser.

Summary

The TeleWeb architecture provides a simple, predicatable, and flexible environment for browsing the Web over a mobile host. The "reactive" design lets users define actions that execute when the conditions of use change. The system is structured around a daemon that runs on the mobile host and acts as an intermediary between a commercial browser and the network. In addition, the daemon may employ a number of loosely coupled wired-side assistants that can change according to the conditions of use.

5. Status and Future Work

We have implemented a basic level of functionality in our system including caching, TeleWeb variables, HTML annotation, and a restricted form of budget monitoring and conditional actions. Currently the communication manager requires the user to specify a change in network interfaces. Our current implementation employs the caching proxy server from the Harvest Project [Bowman94].

Our immediate plans are to implement the remainder of the architecture described in this paper including full budget monitoring, conditional actions, and user cache maintenance. We are also working on a more intelligent communication manager that interacts with the operating system device drivers to sense network changes and automatically change TeleWeb variables. Finally, we're working on allowing HTTP POST functions (in addition to GET) as part of pending actions.

There are a number of areas we would like to pursue in the future. We would like TeleWeb to include intelligent caching and prefetching strategies as well as better developed cost-based cache consistency techniques. We would like to explore multi-platform browsing and the issues of moving state from one host to another. In addition we would like to develop more powerful technique for interacting with wired-assistants which we believe can create a market for value-added services geared towards mobile users.

6. Conclusions

We have described a system architecture for accessing the World Wide Web from loosely connected mobile hosts. Because high-bandwidth, low-cost communication will not be ubiquitous any time soon, if at all, we believe that fine-grained cost control is a crucial aspect of such a system. We propose a "reactive architecture" that supports user-specified adaptation under various operating conditions and provides a platform for exploring many of the problems described in the challenges section. The current TeleWeb architecture incorporates the key features described at the beginning of this paper: costs are made visible to the user through annotated HTML; budget monitoring warns the user when operations exceed pre-specified limits; actions may be postponed and later triggered when conditions are met; and finally, user customization and system configuration values may automatically adapt according to the changing conditions of use. These mechanisms work together to provide an asynchronous, email-style of browsing in which users can work disconnected from a cache of documents, or trade off communication cost against information needs.

References

[Baker96] Mary G. Baker, Xinhua Zhao, Stuart Cheshire and Jonathan Stone. Supporting Mobility in MosquitoNet. In *Proceedings of the 1996 USENIX Technical Conference*, San Diego, California. January 22-26, 1996. USENIX Association.
<URL: http://plastique.stanford.edu/~mgbaker/publications/>

[Baquero95] Carlos Baquero, Vitor Fonte, Rui Oliveira. MobiScape: WWW Browsing under disconnected and semi-connected operation, In *Proceedings Portuguese WWW National Conference*, Braga, July 1995. <URL: http://www.di.uminho.pt/~cbm/ps/MobiScape.ps>

[Bartlett95] Joel F. Bartlett. Experience with a Wireless World Wide Web Client. Technical Note TN-46, March 1995, Digital Western Research Laboratory.
<URL: http://www.research.digital.com/wrl/techreports/abstracts/TN-46.html>

[Boone95] Barry Boone. Magic Cap Programmer's Cookbook. Addison-Wesley, 1995.
<URL: http://www.genmagic.com/MagicCap/cookbook.html>

[Bowman94] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The Harvest Information Discovery and Access System. *Proceedings of the Second International World Wide Web Conference*, Chicago, Illinois. October, 1994.
<URL:ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Conf.ps.Z>

[Douglis96] Fred Douglis and Thomas Ball. Tracking and Viewing Changes on the Web. *Proceedings of the 1996 USENIX Technical Conference*, San Diego, California. January 22-26, 1996. USENIX Association.
<URL:http://www.research.att.com:80/orgs/ssr/people/douglis/papers>

[Fox95] Armando Fox and Eric A. Brewer, GloMop: Global Mobile Computing By Proxy. Department of Computer Science, University of California, Berkeley.
<URL:http://HTTP.CS.Berkeley.EDU/~fox/glomop/>

[Gessler94] Stefan Gessler and Andreas Kotulla. PDAs as mobile WWW browsers. *Computer Networks and ISDN Systems*, Vol. 28, No. 1-2, 1995, pp. 53-59.
<URL: http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/gessler/www_pda.html>

[Goldberg93] David Goldberg and Michael Tso. How to Program Networked Portable Computers, In *Proceedings Fourth Workshop on Workstation Operating Systems* (WWOS-IV), pp. 30-33, October, 1993.

[Joseph95] Anthony D. Joseph Alan F. deLespinasse Joshua A. Tauber David K. Gifford, and M. Frans Kaashoek. Rover: A Toolkit for Mobile Information Access. In *Proceedings of the Fifteenth Symposium on Operating Systems Principles* (SOSP), December 1995.
<URL: http://www.pdos.lcs.mit.edu/rover>

[Kaashoek94] Frans Kaashoek, Tom Pinckney and Joshua Tauber. Dynamic Documents: Mobile Wireless Access to the WWW. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994. IEEE Computer Society.
<URL: http://www.pdos.lcs.mit.edu/papers/www94.html>

[Liljeberg95] M. Liljeberg, T. Alanko, M. Kojo, H. Laamanen, K. Raatikainen. Optimizing World-Wide Web for Weakly Connected Mobile Workstations: An Indirect Approach. In *Proceedings of the 2nd International Workshop on Services in Distributed and Networked Environments* (SDNE'95), June 5-6, 1995, Whistler, Canada.
<URL: http://www.cs.helsinki.fi/research/mowgli/mowgli-papers.html>

[Luotonen94] Ari Luotonen and Kevin Altis. World-Wide Web Proxies. *Computer Networks and ISDN Systems*, Vol. 27, No. 2, 1994, pp. 147-154.
<URL: http://www.w3.org/hypertext/WWW/Proxies/Overview.html>

[Noble95] Brian D. Noble, Morgan Price, and M. Satyanarayanan. A Programming Interface for Application-Aware Adaptation in Mobile Computing. Technical report CMU-CS-95-119, February 1995, School of Computer Science, Carnegie Mellon University.
<URL: http://www.cs.cmu.edu/Web/Reports/1995.html>

[Schilit91] Bill N. Schilit and D. Duchamp. Adaptive Remote Paging for Mobile Computers. Technical report CUCS-004-91, February 1991, Department of Computer Science, Columbia University.
<URL: ftp://ftp.cs.columbia.edu/reports/reports-1991/cucs-004-91.ps.Z>

[Schilit93] Bill N. Schilit, Marvin M. Theimer and Brent B. Welch. Customizing Mobile Applications. In *Proceedings of the USENIX Symposium on Mobile and Location-independent Computing*, pp. 129-138, Cambridge, MA, August 1993. USENIX Association.
<URL: ftp://ftp.parc.xerox.com/pub/schilit/usmlic-93-schilit.ps.Z>

[Schilit94] Bill N. Schilit, Norman I. Adams, and Roy Want. Context-Aware Computing Applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994. IEEE Computer Society.
<URL: ftp://ftp.parc.xerox.com/pub/schilit/wmc-94-schilit.ps>

[Schilit95] Bill N. Schilit. A Context-Aware System Architecture for Mobile Distributed Computing, PhD Thesis, Columbia University, Department of Computer Science, May 1995.

[Voelker94] Geoffrey Voelker and Brian Bershad. Mobisaic: An Information System for a Mobile Wireless Computing Environment, In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994. IEEE Computer Society.
<URL: http://www.cs.washington.edu/homes/voelker/mobisaic/papers.html>

[Want95] Roy Want, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis and Mark Weiser. An Overview of the Parctab Ubiquitous Computing Experiment, *IEEE Personal Communications*, December 1995, Vol. 2, No. 6, pp. 28-43.
<URL: http://www.ubiq.com/parctab>