Web Browsing in a Wireless Environment: Disconnected and Asynchronous Operation in ARTour Web Express

Henry Chang, Carl Tait, Norman Cohen,
Moshe Shapiro, Steve Mastrianni
IBM T. J. Watson Research Center
Route 134
Yorktown Heights, NY 10598
{hychang, tait, ncohen, mshapiro, stevemas}@watson.ibm.com

Rick Floyd, Barron Housel, David Lindquist
IBM Corporation
E55A/B502
P.O. Box 12195
Raleigh, NC 27709
{raf, housel, lindqui}@raleigh.ibm.com

Abstract

In a previous paper [*elided*], we described ARTour Web Express, a software system that makes it possible to run World Wide Web applications over wide-area wireless networks. Our earlier paper discussed how our system significantly reduces user cost and response time during online browsing over wireless communications links. However, even with these savings, users may experience slow performance. This is a result of the inherent delay of wireless communication coupled with congestion in the Internet and Web servers, which cannot be masked from users under the synchronous request/response model of browsing. Furthermore, disconnection -- both voluntary and involuntary -- is common in the mobile environment, and the standard browsing model provides no support for disconnected operation. This paper describes how ARTour Web Express has been enhanced to support both disconnected and asynchronous operation.

1 Introduction

The underlying mechanisms and protocols of Web browsing were developed with a traditional network model in mind. It was tacitly assumed that the computers involved were connected via high-bandwidth, inexpensive, reliable links. Mobile computing turns these assumptions on their heads, and requires a fundamental rethinking of both the implementation and semantics of Web browsing.

In stark contrast to a wired LAN or WAN environment, mobile links are typically low-bandwidth, costly, and unreliable. Some mobile connections are less burdensome than others -- for example, a simple dialup modem is both faster and cheaper than packet radio -- but all are dramatically slower than their LAN counterparts. They are also less reliable: dropped connections are not uncommon due to signal degradation, blockage, and other problems. Applications such as browsers that were targeted for a LAN environment often perform very poorly in a network-constrained setting.

Furthermore, the mobile environment raises the issue of *disconnected operation*. Standard Web browsing -- as well as many existing networked applications -- assume that disconnection is a comparatively rare error case. Operations summarily fail when the client is disconnected from the server.

Weak connectivity and the possibility of disconnection lead to yet a third aspect of the mobility problem: the dynamic nature of a user's connectivity. At different times, a single user may be strongly connected (LAN), weakly connected (cellular or other mobile link) or disconnected. Seamless adaptation to changing link characteristics is a major challenge.

The ARTour Web Express system described in [*elided*] focused on various techniques for minimizing data volume and reducing latency of the HTTP protocol when communicating over low-bandwidth, high-latency communication channels such as those used in wide-area wireless networks [4, 5, 6, 7]. We argued that the browser model is becoming the user interface of choice due to its rapid deployment and acceptance and the broad connectivity afforded by the Internet. The objective of ARTour Web Express is to facilitate the use of Web technology to run many typical commercial applications over wireless networks to increase the productivity of the mobile worker. The predictability of this type of usage makes it possible to employ optimizations that make wireless Web access practical from both a usability and cost perspective. The successful deployment of Web Express extends Web technology to a new usage domain.

In spite of our previous improvements, several factors contribute to poor usability and reduced user productivity when using browsers in a resource-constrained or unreliable communication environment typified by wireless communication. First, the browser protocol is synchronous, which means that users must wait until a request completes before another request can be made. When the delay is long due to slow wireless transmission, congested Internet or intranet traffic, or overburdened Web servers, users become frustrated and unproductive.

Second, the natural burstiness of the synchronous request/response scheme becomes a significant problem over a slow link. Over a wired LAN, server response time is usually the primary concern, but in a wireless

environment, bandwidth and latency are typically the dominating factors. (Latency on a packet radio network can be on the order of several *seconds*.) It is therefore important to provide an *asynchronous* browsing model in which some of this burstiness can be smoothed out over time. Requests can be processed in the background while the user continues the traditional request/wait/read model in the foreground.

Third, the usual synchronous request/response model does not work at all in the face of voluntary or involuntary disconnection. If a request cannot be satisfied immediately, an error code is returned and the user must explicitly retry the request at a later time.

1.1 New Functions

The functions described in this paper address the problems discussed above.

Asynchronous Request/Response Mode: This mode permits a user to continue making requests even though previous requests have not completed. Requests are recorded internally for background processing. When requests complete, the results are saved and status is updated asynchronously. The user is (optionally) notified when requests complete and may, at any time, switch to the status page to review the status of one or more requests. The status entry for each request conveys the state of the request (not started, in process, or complete) and contains a link to the response page if the request has completed.

Disconnected Operation: Users can operate in either synchronous or asynchronous mode. In either case, when

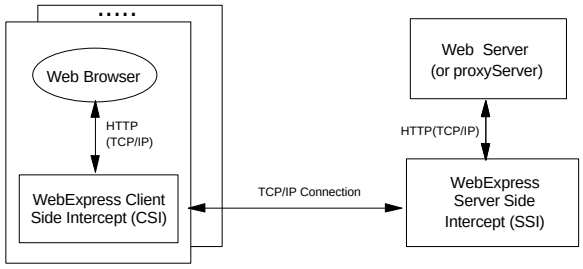


Figure 1: Proxy Model

the loss of a connection is detected, or if communication is not possible (e.g., out of signal range), ARTour Web Express is capable of queuing and holding requests for later processing. When communication is re-established, queued requests are automatically processed in the background, and subsequent responses are handled as described previously for asynchronous requests. This capability enables a user to continue to be productive *offline*: in many cases, all the pages needed for a transaction are stored in the local Web Express cache and no communication is required.

For example, consider an insurance adjuster who must visit a number of clients each day. The standard claim forms can be cached in the adjuster's mobile computer. While discussing a case with a client, the adjuster can bring up the correct form, fill it out, and submit it, all without being connected. (ARTour Web Express even allows queued forms to be re-edited in case of entry error.) When connectivity is re-established (either via wireline connection back at the office or when the wireless signal is strong enough), all the requests are serviced. If the user is using asynchronous request/response mode when connectivity is suddenly lost, there is no need to re-enter requests in process. All requests are remembered on the status page and the requests can be restarted when connectivity is regained.

Implementation of the above functions presents several challenges. We want to run with any browser and all browser (i.e., HTTP) protocols are synchronous. How do we accomplish the asynchronous behavior and conform to the synchronous requirements of the browser? Once an asynchronous request is made, how do we correlate the subsequent response with the original request? How are the user interfaces designed? Should editing or deletion of submitted requests be permitted if the request has not been started? These and other issues are addressed in detail later in the paper.

In the next section, we summarize the architecture of the WebExpress system upon which the asynchronous request/response mode and disconnected operations functions are built. We assume that the reader has general familiarity with HTML and the HTTP protocol (refer to [1, 2, 3] for more details).

1.2 ARTour Web Express Proxy Model

An important objective of ARTour Web Express is to be able to run with *any Web browser* (e.g., Netscape or Internet Explorer) and *any Web server* without requiring any changes to either. To accomplish this we use a proxy technique that enables ARTour Web Express to intercept and control communications over the wireless link for the purposes of reducing traffic volume and optimizing the communications protocol to reduce latency. The components that implement this intercept technique are shown in Figure 1. We insert two proxies into the data path between the Web client and the Web server: the *Client Side Interceptor (CSI)* that runs in the client's mobile device and the *Server Side Interceptor (SSI)* that runs within the wireline network. The CSI intercepts HTTP requests and, together with the SSI, performs optimizations to reduce data transmission over the wireless link.

From the viewpoint of the browser, the CSI appears as a local Web proxy that is co-resident with the Web browser. The CSI communicates with the Web browser over a local TCP connection (using the TCP/IP "loopback" feature) via the HTTP protocol. Therefore, no external communication occurs over the TCP/IP connection between the browser and the CSI. No changes to the browser are required other than specifying the (local) IP address of the CSI as the browser's proxy address. The actual proxy (or *socket server)* address is specified as part of the SSI configuration. The CSI communicates with an SSI process over a TCP connection using a terse, private protocol. The SSI reconstitutes the HTML data stream and forwards it to the designated Web proxy server. Likewise, for responses returned by Web servers (or proxies), the CSI reconstitutes an HTML data stream received from the SSI and sends it to the Web browser over the local TCP connection as though it came directly from the Web server.

The proxy model implemented in ARTour Web Express is transparent to both Web browsers and Web (proxy) servers and, can therefore be employed with any Web browser. Here is a brief summary of the optimizations for online wireless browsing:

- **Caching:** Both the SSI and CSI cache graphic and HTML objects. If the URL specifies an object in the CSI's cache, it is returned immediately as the browser response. The caching functions guarantee cache integrity within a client-specified time interval. The SSI cache is populated by responses from the requested Web servers. If a requested URL received from a CSI is cached in the SSI, it is returned as the response to the request.

- **Differencing:** CGI requests result in responses that normally vary for multiple requests to the same URL (e.g., a stock quote server). The concept of differencing is to cache a common *base* object on both the CSI and SSI. When a response is received, the SSI computes the difference between the base object and the response and then sends the difference to the CSI. The CSI then merges the difference with its base form to create the browser response. This same technique is used to determine the difference between HTML documents.

- **Protocol reduction:** Each CSI connects to its SSI with a single TCP/IP connection. All requests are routed over this connection to avoid the costly connection establishment overhead. Requests and responses are multiplexed over the connection. (Note that this is more powerful than the persistent *per-server* connection of HTTP 1.1.)

- **Header reduction:** The HTTP protocol is stateless, requiring that each request contain the browser's capabilities (called *access lists* in HTTP). For a given browser, this information is the same for all requests. When the CSI establishes a connection with its SSI, it sends its capabilities only on the first request. This information is maintained by the SSI for the duration of the connection. The SSI includes the capabilities as part of the HTTP request that it forwards to the target server (in the wireline network).

2 Request Queuing

2.1 User Model

Unlike our earlier version of ARTour Web Express, which operated transparently to improve wireless performance, disconnected and asynchronous operations require new user interfaces. If a user's request can be satisfied from the cache, we do so immediately and the standard browser interface remains unchanged. On a cache miss, however, we extend the browser's semantics with our own mechanisms and interfaces.

Since the browser is an immutable piece of code, our solution is to return a stand-in page whenever we are in asynchronous or disconnected mode and cannot satisfy a user's request from the cache. This page contains an explanation of what has happened -- "Your request has been queued for later processing" -- and, if requested, displays the current status of all pending requests. Note that as far as the browser is concerned, this stand-in page *is* the response to the request. In other words, the browser retains its simple request/response mentality, while we handle the details under the covers. Also note that we use the same mechanism for both disconnected *and* asynchronous requests: an informational page is returned to the browser as the response to the request.

There is one additional feature primarily useful in asynchronous browsing: an option to return to the current page rather than being presented with an intermediate acknowledgment. This is implemented by returning code 204 to the browser. From the user's point of view, a link is clicked but the browser remains on the same page; the request has been quietly queued in the background. Since this option can lead to considerable confusion for the novice user, it must be explicitly configured.

2.2 Processing Queued Requests

ARTour Web Express maintains an internal queue of requests that have been deferred for later processing. Requests end up on this queue for one of two reasons:

1. A connection was not available, and so the request could not be processed at the time it was received (disconnected operation); or
2. The requested information was not in the cache, and the user has configured ARTour Web Express to never block on requests to the network.

Regardless of the source of the request, the processing is identical once the request is queued. The remainder of this section describes how these queued requests are handled.

Logically, the deferred processing support consists of the queue of requests, the processing engine, and the cached results (see Figure 2).

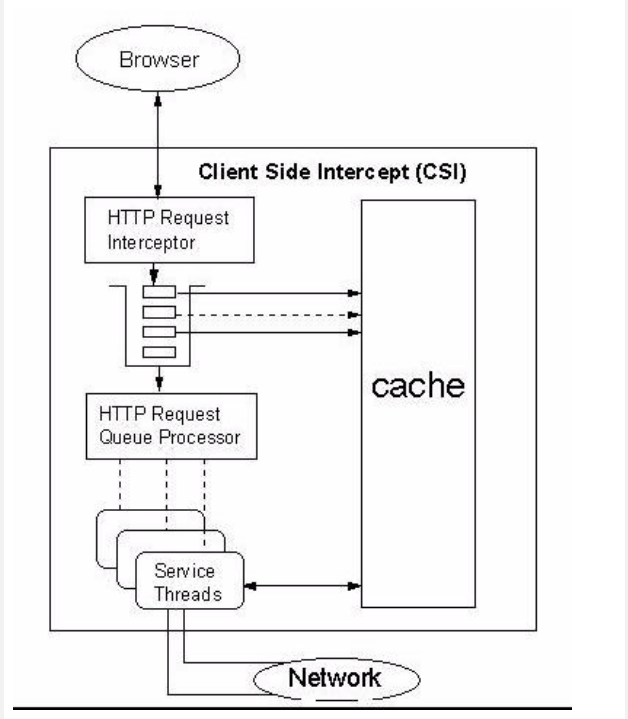


Figure 2: Background Processing Architecture

The ARTour Web Express request queue is a list of requests that have been received from the browser, along with status and control information. Each request element includes all of the information received from the browser (the HTTP headers and any body). This allows the request to be replayed to the network at a later time. Each request element also holds state associated with the request. This includes a summary of the progress that has been made in processing the request, and a list of remaining work to be done.

It is also possible to associate special processing instructions with a request. For example, in a wireless environment, it is usually too expensive to download graphics embedded in a page. If graphics have been suppressed for this request, this processing control information will be included in the request element. The request queue persists across client sessions.

Internally, a request consists of a set of *attributes*. Each attribute is a name-value pair. When the request is created, it is given attributes that contain the browser request. As processing proceeds, attributes are added describing the progress, any status information returned from the browser, and so on. Some attributes, such as the progress indication, are *per request*. Attributes may also be *per queue*, and in this case control processing for all elements added to the queue. Per-queue attributes are used by ARTour Web Express to support multiple queues that handle requests in different ways, based on their source. An example of this is the cache refresh capability (described in a later section), which has different requirements for graphics retrieval and cache coherency.

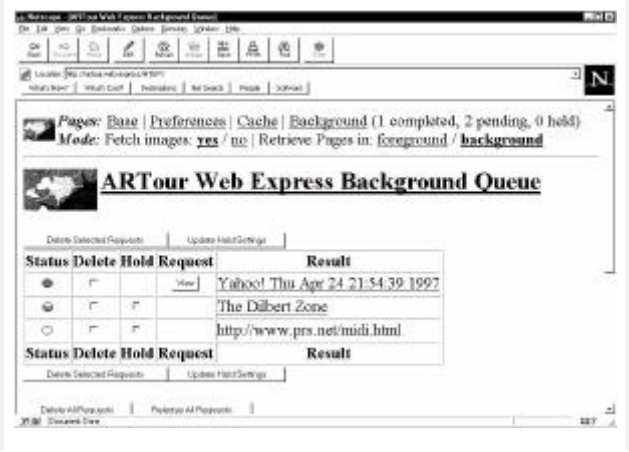
Queue attributes allow us to easily tune request processing based on the user's preferences, the source of the request, and the configuration of ARTour Web Express. They have been a key to providing us with the flexibility to effectively handle requests in a resource-constrained environment.

Processing of queued requests is handled by threads that run independently of any other activity in ARTour Web Express. Requests on the queue are usually handled on a FIFO basis. However, it is possible to mark selected requests as *held*, or to indicate that all new requests should be held. This allows the user to indicate that processing on these requests should be deferred until a later time, even if a connection is available.

For example, some requests may be too expensive to handle on a wireless link, and so the user would prefer to defer processing them until a LAN or dialup connection is available. The hold attribute is also applied to any outstanding queued requests remaining when the client is first started. This avoids nasty surprises when switching from an inexpensive link to a potentially much more expensive wireless link. We would prefer to make the hold/release decision automatically, based on the connection characteristics. However, this information is not readily available on commercial platforms, so we give users the capability of controlling this behavior themselves.

The first step in handling a queued request is acquiring a connection to the ARTour Web Express server. An attempt is made to get a connection when a request is first received. This will continue until a connection is made, either by the queue processor or due to other ARTour Web Express activity. An exponential backoff algorithm is used to control the delay between connection attempts. This allows us to be responsive in the case where communication failures are transient without burdening the network when the failures are long term.

Once we have a connection, we use the information we originally saved from the browser to reconstitute the



request and send it off. At this point the request looks to servers just like it would if it came directly from the browser. If the request fails due to failures in the wireless link, we try again at a later time. Retrying these requests allows us to transparently mask transient communications failures. Other failures are recorded for later return to the user. If the request succeeds, the returned page is stored in the ARTour Web Express cache, and any status information returned by the server is saved with the request.

We still may not have enough information to display the page for the user, however. The use of graphics, applets, and other embeds is now nearly universal on Web pages. If the user has asked to see this information (rare in a wireless environment, but typical if a wired connection is being used to prewarm the cache), the returned page is parsed for embeds. Each embed is then retrieved and added to the cache. At this point the request is complete and the user can be notified that it is available.

2.3 Reporting Request Status

ARTour Web Express was designed as a transparent HTTP proxy that will work with any browser. Queuing requests for background processing doesn't match the model that browsers expect, but we address this by returning a *stand-in* response, via HTML or an HTTP code, so that the browser and the user can continue. Similarly, we use HTML pages and embedded HTML information to report status in most cases so that we can maintain this browser independence.

Status of an outstanding request is reported in one of three ways: an optional completion popup, an optional status bar embedded in the top of returned Web pages, or on an HTML page that summarizes the state of the queue.

The popup let users know that there is a newly completed page to view on the background page, and lists the URL of the page. One of these popups is generated the first time new information becomes available, and then further messages are suppressed until the user has visited the page and viewed the information. This allows the user to get an asynchronous completion notification without being overwhelmed on faster links.

The user may also chose to embed a status bar describing the state of ARTour Web Express in each returned page. This status bar includes information on the number of requests that have been completed, the number outstanding, and the number held. It also includes a textual version of the popup stating that new results are available, and links to various generated status pages.

One of these links takes the user to a page summarizing the state of the request queue (see Figure 3).

Figure 3: Background Queue Page

The page displays the queue, one line per request. Each request includes a graphical representation of the progress that has been made processing the request, using the model of a traffic light:

- Red: Request has not been sent
- Yellow: Base page received; embed(s) are pending
- Green: Request complete

For both black-and-white displays and color-blind users, each graphic has a distinctive look apart from its color. Red is an open circle; yellow is a half-moon; green is a solid ball. In case of error (e.g., the request was sent but the base page could not be retrieved), an X is placed across the traffic light symbol to indicate failure.

Along with the status graphics are options to delete or hold a request. In the case of forms, the user may also view or re-edit the request. If the request has completed, a link to the cached result is included on this page.

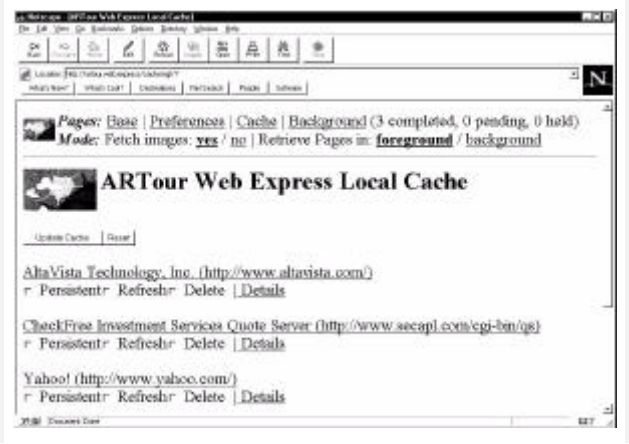
In order to provide URL-based access to the background queue and other internally-generated pages, we use our own domain name -- `artour.web.express` -- coupled with other options as appropriate. For example, the background queue is accessed via the URL `http://artour.web.express/HTEP/`. Rather than using HTTP and a reserved domain name, it was tempting to define our own protocol name for such requests. Netscape, however, rejects protocols it is unfamiliar with, so we could not use that approach.

There is also an ARTour Web Express architecture for reporting the progress of request handling. Requests move through a series of states as they are processed, starting with submitted, to processing begun, to initial page

retrieved, and finally to completed. As a request moves into a state, an event describing the transition is sent to an internal event manager. Other components may register to receive events, filtered by event state and other criteria. Processing events can be used to generate dynamic interfaces to ARTour Web Express. The popup notification is one example of the use of these events.

3 Cache Management in a Weakly-Connected Environment

The ARTour Web Express cache provides, as described earlier, the basic support we need to effectively cache and manage Web pages to support browsing in a wireless environment. We have extended the cache to provide



additional support for handling queued requests, storing the associated results, and to allow the user to browse and manage the cache to enhance disconnected use.

When a browser makes a request and receives a response, it handles it in one of two ways. If the response is expected to be relatively static, it is cached by the browser so that future requests to the page can be handled quickly. However, if the page is a response to a forms request, or is otherwise generated (so-called cgi-bin requests), the browser only displays the response, and doesn't cache it, since the response is typically different from one cgi-bin request to the next.

ARTour Web Express normally uses the same strategy with cgi-bin requests, only saving enough information to allow the optimization algorithms to work. However, when we are processing queued requests, this is no longer adequate. We don't know when the user will view the results of these requests, and so we need to save them indefinitely. Another case where this problem arises is with objects that the source server marks "no-cache." Browsers and proxies, such as ARTour Web Express, that observe this directive do not save these items. However, when one is retrieved as part of processing a queued request, it must be saved for later viewing.

We have addressed the need to save these normally transient objects by defining a new category of cached information: *user data*. These are data that have been retrieved by ARTour Web Express in response to a user request, and only have meaning in the context of that request. Unlike normal Web data, user data objects are only accessible as responses on the queue status page. They are not used to handle other requests, since they are transient time- or request-sensitive responses. User data are not subject to the normal coherency and aging algorithms used to manage the cache. They persist and are valid until deleted by the user.

Note that the concept of user data is a significant extension to the normal Web model. Cgi-bin results are no longer fleeting entities. Users can make these requests, and then at any later time go back and view (or review) the results for any of the requests that were handled by the queue. We have extended the Web data model with a component that captures the users temporal and query history. This is a surprisingly powerful and useful concept that was not at all obvious to us when we first started working on support for request queuing.

Effective use of ARTour Web Express in a disconnected mode requires that the cache contain the items needed by the user, and that these items be as up to date as possible. ARTour Web Express provides a generated HTML page that gives users access to cache management functions that help enable effective disconnected operation (see Figure 4).

Figure 4: Cache Management Page

The cache management page contains a list of all HTML entries in the cache. Each entry is a hyperlink to the corresponding cache entry, and so may be used for browsing local pages when disconnected. Using this page, users can indicate that an entry should be *persistent*. Persistent entries, along with their embeds, will never be deleted from the cache as part of the normal LRU management of the cache. This ensures that they will always be available for disconnected use.

Users may also use the cache page to *refresh* entries. Marking an entry for refresh instructs ARTour Web Express to contact the source server and verify that the cached copy is current. Any or all of the entries in the cache (except for the user data described earlier in this section) may be marked for refresh. Refresh requests are actually queued on an internal *refresh queue* for later processing. The refresh queue uses the queue support described earlier, but with queue attributes tailored to refreshing entries. The combination of refresh capability and persistence can be used to insure that the information

present in the cache when disconnected is both correct and current.

4 Form Requests

Web page designers often employ HTML forms for data entry that requires only simple interaction. For example, a Web search may start with a text input field. User logon in a restricted Web area may be accomplished through text input fields requiring a user's name and password. Registration, purchasing, and survey forms may use a combination of lists, check boxes, and radio buttons. Moreover, dynamic HTML forms can be used for electronic Internet commerce or intranet applications, where different users receive customized forms that are dynamically formulated by the Web application.

HTML form tags provide a convenient way for building electronic forms for Web interaction. HTML form tags allow a Web user to make selections from a list, to check on/off boxes, to select from radio buttons, to enter text into a text field or a large multi-line text area, and to push action buttons. When the user presses an action button, the entered data is sent to a Web server designated by the action with name/value pairs, where each name represents an input field and each value represents the user's input in the field. In addition, there may be hidden fields, which carry preset values that a Web server sent along with the form. These hidden values will be sent back to the server together with the values in visible fields.

Like any HTML page, an HTML form can be cached for future use. A cached HTML form may be edited for submission later or resubmission again with different user input. For example, a Yahoo search input form could be edited again and again to send out different search requests. Similarly, an intranet data entry form such as a patient admission form could be edited to correct data entry errors, or resubmitted with new data for a different patient.

Most forms can be meaningfully cached for independent future submission because they either have a simple one-form interaction model or contain self-sufficient hidden fields so that the Web application can accept a submission in isolation. However, it is worth noting that some form requests may not be useful when taken out of a sequence of Web interactions. Certain Web applications disallow caching of HTML forms because they are dynamically tailored for the user at that specific time. ARTour Web Express focuses on the more usual case of forms that are independent requests, and can be cached without problems. This provides a rich base for typical form-based access to enterprise data.

4.1 Disconnected Form Operations

In a mobile environment, ARTour Web Express supports disconnected form submission to extend the productivity of users even when the Web is not reachable. This allows multiple data entry Web pages to be filled without connecting to any network. Also, with the re-editing function supported by ARTour Web Express, a user can draft a few forms and have a chance to review, approve, or edit them before they finally go out to the Web.

Like regular HTML pages, the user obtains a cached HTML form using a URL. The user fills in the form and submits it using one of the action buttons. When a form is submitted, the name/value pairs together with the originating form URL are stored and queued. If the user requests the same URL form again and submits a new result, the new submission is kept separate from the previous submission. Every submission is counted as a separate entry in the queue for automatic submission when a connection becomes available. These queued submissions are displayed on the Background Queue page (see Figure 3).

Form re-editing is a surprisingly tricky business. In order to reconstruct the form as the user submitted it, ARTour Web Express must compose a new HTML page using both the original blank form from the cache and the submitted name/value pairs. When the blank form is first fetched from the Web, the CSI inserts hidden values into the HTML before returning the page to the browser. The hidden information includes both the originating URL and the form number within the page. To re-edit the form, the blank form is retrieved using the hidden value containing the form's URL. Then this original form is scanned to match name/value pairs with named fields in the original form. If any match is found, the default selection or input value is then changed to reflect the user's submission.

Special care must be taken with Web pages that consist of multiple HTML forms, since they may use the same field names in different subforms. The parsing and transforming operation is complicated by the fact that industry-standard browsers are very forgiving of HTML errors. There is a large amount of malformed HTML available on the Web, and we attempt to handle the bulk of it as gracefully as possible.

After an asynchronous or disconnected form request has completed, the user may wish to review the original request ("I forgot what this search was for"). The form is reconstructed using the same procedure as for re-editing before submission, but to avoid user confusion, the submission buttons are removed from the form. They are replaced by a link to the result of the request.

5 Related Work

Offline Web-browsing is a particularly hot area [24]. Most commercial products support both the Microsoft Internet Explorer and Netscape Navigator. Some, such as Lotus's Weblicator [10] and Traveling Software's WebEx [27] (once known as Milktruck), communicate with the user through the browser window like ARTour Web Express. Most, however, provide their own GUI controls (and we are considering this change for ARTour Web Express as well).

The Mowgli project at the University of Helsinki [17] was an early example of the use of client and server proxies communicating through a private protocol to optimize the retrieval of Web pages. In contrast to the dual-proxy architecture of ARTour Web Express, most commercial Web-browsing products reside entirely at the client. An exception is the WebMirror Communications Server [20], which has both a client component and a server component. These components communicate through a compressed and encrypted data stream, supporting mobile use.

The Rover project at MIT [8] is a toolkit for building mobile applications and for writing proxies to enable legacy applications for mobility. Part of this project includes a mobile-enabled Web browser, which supports a form of asynchronous browsing. When a page is not immediately available, the Rover-based proxy returns code 204 so that the browser will remain on the same page. (This is a configurable feature under ARTour Web Express.)

A feature common to many offline browsers, but not found in the first release of ARTour Web Express, is a sophisticated prewarming mechanism. Such a mechanism can download a prespecified set of pages in the background, or overnight. Some products -- for example NetAttaché Pro [28] -- offer a mechanism for supplying form information, such as passwords and search-engine query strings, to be submitted automatically during this process. Other products, such as InContext FlashSite [16], import the browser's bookmarks file, or provide bookmarking mechanisms of their own, to gather lists of URLs to be loaded automatically.

Some products preload pages linked to from other preloaded pages, following links to a specified maximum depth. Some products provide the capability to download all the pages at a given Web site. FlashSite and DocuMagix Internet HotSuite [9] provide visual displays of the pages at a web site and the links among them. NetAttaché Pro uses its multiple TCP/IP connections to request pages from different Web sites, to avoid overwhelming one server with multiple simultaneous requests. Surfbot [25] (acquired by Oracle in late March 1997) allows Web-crawling algorithms to be specified by programmed agents.

Peak Net.Jet [22] is not an offline browser, but it uses predictive caching to speed up a connected browsing session. It uses nongraphic links from the current page and a history of the user's past browsing patterns to preload additional pages likely to be requested in the near future. This preloading occurs during lulls in communication. (This approach is, of course, poorly suited for a mobile environment with high per-packet communications charges.) Enabling the loading of graphics significantly decreases the effectiveness of Net.Jet.

Closely related to automatic caching is the monitoring of interesting pages to determine when they have changed. In connected mode, ARTour Web Express uses coherency intervals and checksums to determine whether cached copies of pages should still be considered fresh. Products oriented towards machines that are *usually* connected can be more aggressive, automatically downloading the most recent versions of designated pages. Tierra Highlights2 [26] and FirstFloor Smart Bookmarks [11], for example, provide automatic notification when a monitored page has changed. Smart Bookmarks even checks whether the result of a form submission has changed. NetAttaché Pro and Highlights2 compute the differences between old and new versions of a changed page, not to optimize communication like ARTour Web Express, but to highlight the differences to the reader and make it easy for the reader to track how a page has been modified.

Some products, notably WebWhacker [13], support the hoarding of files on one machine, presumably connected to a LAN, and the transfer of the resulting cache by diskette to a second, mobile machine. In contrast, ARTour Web Express presumes a mobile machine that is capable of connecting to the Web, albeit possibly through a slow or expensive communications link. We focus on mitigating the negative effects of this link. We also provide the user with feedback on the progress of a download, something that products oriented toward unattended cache loading generally do not do (FlashSite being a notable exception).

IBM's Web Browser Intelligence (WBI) software [15] supports an architecture for composing programmed agents to enhance web browsing. This architecture can be used to support offline browsing along with the ancillary activities performed by many offline browsing products. Some agents monitor issued HTTP requests or responses to requests, possibly recording information such as a history of sites visited or the contents of retrieved pages. Other agents intercept a request and pass on a modified request (for example to correct a URL for a page that is known to have moved) or intercept a response and pass on a modified response (for example to add controls like the ARTour WebExpress navigation bar to a page or to use PICS labels to filter the pages that arrive at a browser).

Still other agents act like Web servers, accepting requests and issuing responses (for example to generate pages like the ARTour WebExpress background queue status page, or to satisfy a request from a local cache rather than passing it on to the Internet). Finally, there are agents that are triggered by events such as time-based alarms (for example to periodically scan a set of Web pages for updates).

There are many products that optimize wireless communication, but are not specifically concerned with Web pages. Examples are Ardis Mobile Office [18], Nettech InstantRF [21], Paragon Précis Link [19], Racotek Keyware [23], and XcelleNet Remoteware [29]. Unlike these general-purpose mobile middleware products, ARTour Web Express is specifically concerned with the retrieval of web pages, so it can use HTTP-specific techniques to optimize communication.

6 Future Work

Request queuing is useful in more strongly connected environments as well. In these environments, slow performance is usually due to server processing delays and congestion, not connection speed. However, the request queue can grow quickly in a higher-speed environment. We would like to develop mechanisms that allow users to easily navigate among large numbers of completed requests, and to more easily discover when "interesting" requests have finished.

Some of the request queuing support, such as hold options and startup behavior, were inserted to insure that queued requests would not overwhelm slower wireless networks. Managing these options requires user intervention. If ARTour Web Express were aware of the connection characteristics and of *changes* in connections, it would be possible to automate much of this management. We are investigating both the acquisition and utilization of connection information in ARTour Web Express. This information could be used to drive not only queue management, but also control the use of optimization techniques, image retrieval, lookahead, and other potentially connection-sensitive behavior.

Finally, we are looking at ways in which we can help users prewarm their caches with the Web data they need for their applications. One option we are investigating is the use of scripting technology to track user requests and construct *packages* of requests for playback and refresh at a later time.

7 Conclusion

The initial implementation of ARTour Web Express [*elided*] showed that it was feasible, from a cost standpoint, to run commercial Web applications over wide-area wireless networks. However, transmission delays, congestion, and intermittent or lost connections can make it difficult to make use of Web technology in this environment.

In this paper, we have described a set of techniques that allows users to work productively even in the face of delays and connection uncertainty. We have implemented disconnected and asynchronous browsing using a form of request queuing. This allows the user to continue with productive work rather than waiting for each request to complete in a synchronous fashion. It also allows us to effectively mask intermittent connections. Disconnected operation allows the user to effectively utilize cached information to browse, submit, and re-edit requests in the absence of connections. To support these operations, we have extended caching of Web data to include information that has temporal and request-specific validity.

We have found that these techniques, working together, have greatly extended the reach and usability of ARTour Web Express. We are no longer at the mercy of the "world wide wait" or of uncertain network connectivity. We can, instead, work productively while ARTour Web Express masks these problems for us.

References

1. T. Berners-Lee et al., *The World-Wide Web*, CACM 37(8):76-82., August 1994.
2. T. Berners-Lee, D. Connolly, *Hypertext Markup Language Specification - 2.0*, Internet-Draft, Internet Engineering Task Force (IETF), HTML Working Group, June 1995. Available at <http://www.ics.uci.edu/pub/ietf/html/html2spec.ps.gz>.
3. T. Berners-Lee, R. Fielding, H. Frystyk, *Hypertext Transfer Protocol - HTTP/1.0 Specification*, IETF, Internet-Draft, August 13, 1995. Available at <http://www.ics.uci.edu/pub/ietf/http/draft-fielding-http-spec-01.ps.Z>.
4. *An Introduction to Wireless Technology*, IBM International Technical Support Center SG24-4465-01, October 1995.
5. George Calhoun, *Wireless Access and the Local Telephone Network*, Boston: Artech House Inc., 1992.
6. *ARDIS Network Connectivity Guide*, Illinois: ARDIS, March 1992.
7. *RAM Mobile Data System Overview*, RAM Mobile Data Limited Partnership USA RMDUS 031-RMDSO-RM Release 5.2, October 1994.
8. Anthony D. Joseph, et al., *Rover: A Toolkit for Mobile Information Access*, Proc. of the Fifteenth

Symposium on Operating Systems Principles, December 1995.
9. DocuMagix, Inc. “DocuMagix HotPage”, http://www.documagix.com/products/hotsuite.htm, April 8, 1997
10. Edge Research. “Weblicator beta newsroom”, http://www.edge.lotus.com/weblicator/beta_news.html, March 21, 1997
11. FirstFloor, Inc. “FirstFloor Smart Bookmarks 3.0 Data Sheet”, http://www.firstfloor.com/sb30data.html, April 4, 1997
12. Folio Corporation. “Document - Folio Infobase”, http://www.folio.com/cgi-bin/folioisa.dll/retrvewp.nfo?, April 7, 1997
13. ForeFront Group, Inc. “ForeFront’s WebWhacker!”, http://www.ffg.com/whacker/, April 7, 1997
14. FreeLoader, Inc. “FreeLoader -- Product Information”, http://www.freeloader.net/software.htm, March 20, 1997
15. IBM Corporation. “IBM Web Browser Intelligence”, http://www.networking.ibm.com/wbi/wbisoft.htm, March 13, 1997
16. InContext Systems. “FlashSite Index”, http://www.incontext.ca/products/flashsite/, January 31, 1997
17. Liljeberg, Mika, Alanko, Timo, Kojo, Aarkky, Laamanen, Heimo, and Raatikainen, Kimmo. Optimizing world-wide web for weakly connected mobile workstations: an indirect approach. Proceedings of the Second International Workshop on Services in Distributed and Networked Environments (SDNE ‘95), Whistler, Canada, June 5-6, 1995
18. Microsoft Corporation. “Ardis Mobile Office”, http://www.microsoft.com/exchange/exisv/ardispdmobile.htm, January 7, 1997
19. Microsoft Corporation. “Précis Link from Paragon Software”, http://www.microsoft.com/exchange/exisv/paragonpd.htm, January 7, 1997
20. MobileWare Corporation. “WebMirror Communications Server Datasheet”, http://www.mobileware.com/products/wmcs/wmcs_data.htm, February 20, 1997
21. Nettech Systems, Inc. “Nettech Systems Home Page”, http://www.nettechrf.com/, April 1, 1997
22. Peak Technologies, Inc. “Peak Net.Jet”, http://www.peak-media.com/netjet/netjet.html, March 27, 1997
23. Racotek Mobile Data Communications. “Racotek - Building, Enabling, & Supporting Data Mobility”, http://www.racotek.com/index.html, April 7, 1997
24. Stevenson, Ted, and Venditto, Gus. Speed browsing. *Internet World* **8**, No. 4 (April 1997), 72+
25. Surflogic LLC. “Surfbot 3.0: Data Sheet”, http://www.surflogic.com/sb30datasheet.html, March 28, 1997
26. Tierra Communications, Inc. “Tierra Highlights 2”,http://www.tierra.com/products2/highlights2.html, March 26, 1997
27. Traveling Software, Inc. “WebEx - New Version 2”,http://www.travsoft.com/products/webex/2.0/factsheet.htm, April 7, 1997
28. Tympani Development. “Tympani Development - NetAttaché Pro Product Page”, http://www.tympani.com/products/NAPro/NAPro.html, February 17, 1997
29. XcelleNet.“Xcellenet Products”, http://165.113.193.121/xcellenet/products/index.htm, February 17, 1997