Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilot

Armando Fox, Ian Goldberg, Steven D. Gribble, David C. Lee, Anthony Polito, and Eric A. Brewer
445 Soda Hall
University of California at Berkeley
Berkeley, CA, USA 94720-1776
1-510-643-9475, 1-510-642-5775 (fax)
{fox,iang,gribble,lcddave,aspolito,brewer}@cs.berkeley.edu

After an inauspicious debut, communication-enabled personal digital assistants (or PDA's) and handheld PC's are being "rediscovered" as mobile information access terminals. In response, developers have attempted to bring complex applications such as Web browsers to such devices. However, the limited resources available on thin client platforms make them unsuitable for hosting such applications. In this paper, we advocate moving application complexity from thin clients to an *adaptive middleware proxy* (AMWP), an infrastructural application server platform designed to support large populations and diverse applications. We describe one such application, Top Gun Wingman, a graphical, split Web browser for the Palm Pilot PDA that is currently in use by more than 11,000 users around the world. Our discussion focuses on the design philosophy, implementation, performance, and lessons learned from our experience with the Wingman client and the middleware proxy that supports it.

Keywords

middleware, adaptive, proxy, Palm Pilot, topgun wingman

1 MOTIVATION

We would like to connect "thin clients" to the Internet, especially convergent mobile devices such as *smart phones*, while allowing them to leverage at least some of the installed infrastructure of Web content and services. However, the limited capabilities of these devices—smaller displays, more limited user interfaces, lower bandwidth, and limited processing power and memory—complicate direct porting of desktop applications such as Web browsers. In the past (Fox, Gribble, Chawathe & Brewer 1996, Fox, Gribble, Brewer & Amir 1996) we have argued for the use of *transformational proxies*, which perform on-the-fly, datatype-specific lossy compression, as a mechanism for application-level content adaptation that circumvents the limitations of thin clients. We have generalized this approach into a full set of building blocks and programming interfaces for constructing proxy-based applications that adapt to network and client variability. In this paper we describe our *adaptive middleware proxy* (AMWP) approach in the context of Top Gun Wingman, the world's first graphical Web browser for the 3Com PalmPilot.

Existing commercial products such as Microsoft Pocket Internet Explorer for Windows CE and HandWeb for the 3Com PalmPilot attempt to directly port the largest reasonable subset of features from desktop browsers onto thin clients. In contrast, Wingman is not a port, but a true *split application* in which a substantial amount of the application complexity has been pushed to a back end proxy server. The proxy server implements a simple building-block programming model called TACC (Fox 1997), which supports application modules that perform Transformation, Aggregation, Caching, and Customization of Internet content.

We discuss the implementation in detail in Section 2.

1.1 Claims and Contributions

Because we exploit infrastructure computing, Wingman's performance and feature set compare very favorably not only with its direct competitors, but with many desktop browsers as well. In particular, we make the following claims for the AMWP-based implementation of the Wingman browser, which we support with specific examples and later generalize to other thin-client Internet services:

1. The AMWP approach results in **better performance** than a reduced port or least-common-denominator, standards-compliant implementation that runs entirely on the client. We provide evidence for this in Section 2.6.
2. AMWP enables **new features and behaviors** for the client, many of which would be awkward or impossible without proxy support. By hosting the complex parts of the application on the TACC proxy, complex applications can be robustly supported on very simple clients. For example, the AMWP approach has been used to bring the MBONE mediaboard application to the Palm Pilot (Chawathe, Fink, McCanne & Brewer 1998).

3. AMWP **transparently leverages more of the existing infrastructure** of content and services, since on-the-fly transparent translation and compression are effective techniques for delivering acceptable performance between existing servers and thin or legacy clients. Such thin clients might not otherwise be capable of handling content and services designed for modern, “common case” clients.
4. AMWP-based services are **cost effective to operate and expand** because of the simplicity of the client software, the generality of our building-block approach for the middleware proxy, and our proven commodity-PC cluster server implementation (Fox, Gribble, Chawathe, Brewer & Gauthier 1997). Cost efficiency makes widespread deployment of AMWP-based applications feasible even for very large (hundreds of thousands of users) communities.

1.2 Map of Paper

In Section 2 we motivate the three-tier client-proxy-server model as a platform for deploying middleware-based services for mobile and thin clients. We describe our general approach, give quantitative experience from work leading up to Wingman, and introduce the TACC programming model that supports Wingman and other applications. We also give details of the implementation and performance of both the Wingman client and its cluster-based TACC server back-end (discussed previously in (Fox, Gribble, Chawathe, Brewer & Gauthier 1997)), and describe how the AMWP approach enables specific features that make Wingman competitive with both desktop browsers and other thin-client browsers. In Section 3, we attempt to draw some lessons based on our experience in both implementing Wingman (and AMWP applications in general) and serving a worldwide community of over 11,000 users. We discuss related and future work in Section 4.

2 ARCHITECTURE AND IMPLEMENTATION

We argue for a *proxy-based approach* for middleware applications, in which proxy agents placed between clients and servers perform computation- and storage-intensive tasks, such as datatype-specific lossy compression, on behalf of clients. Properly applied, this approach reduces the bandwidth demands on the infrastructure through lossy compression (Fox & Brewer 1996, Fox, Gribble, Chawathe, Brewer & Gauthier 1997), and allows legacy and other nonstandard (including thin) clients to interoperate with existing servers.

Furthermore, by performing client adaptation at a shared infrastructural proxy, we avoid inserting adaptation machinery at each origin server. Application partitioning arguments have long been used to keep clients simple (as in (Watson 1994*a*)); we simply split the application between client and proxy, rather than between client and server. From the client’s perspective, the proxy is simply a server that gets the data from someplace else.

2.1 TACC: A Programming Model for AMWP Applications

We have evolved a programming model for proxy-based applications called TACC (Fox 1997, Fox, Gribble, Chawathe, Brewer & Gauthier 1997): *transformation* (distillation (Fox, Gribble, Brewer & Amir 1996), filtering, format conversion, etc.), *aggregation* (collecting and collating data from various sources, either offline or on-the-fly), *caching* (both original and transformed content), and *customization* (persistent store of user profiles, allowing the service to tailor its output to each user's needs or device characteristics). Each *TACC worker* specializes in a particular task, for example, scaling/dithering of images in a particular format, conversion between specific data formats, extracting "landmark" information from specific Web pages, etc. Complete applications are built by *composing* workers, by *chaining* them (in the Unix pipeline sense), or allowing one worker to *call* another as a subroutine or (parallel, asynchronous) coroutine. The *dispatch rules* that determine which workers are invoked to satisfy a particular user request can either be hardcoded for each application, or controlled dynamically by the workers themselves. A runtime platform called a *TACC server* is expected to provide mechanisms for hosting workers, instantiating dispatch rules, providing high availability, and recovering from errors such as dispatch loops, infinite mutual recursion, or worker instability; we describe our prototype TACC server in the next section.

Workers serially process tasks from a task queue; each task corresponds to some component of satisfying a user request, e.g. scaling and transforming an inline image on a Web page. Each worker is expected to be *atomic and restartable*, in order that the TACC server (a prototype implementation of which we describe below) be able to exploit a repertoire of availability and scaling mechanisms. There are several ways in which a worker can achieve this goal, including being completely stateless (as in Wingman), performing only idempotent operations, or sharing reconstructable group state via mechanisms such as Scalable Reliable Multicast (SRM) (Floyd, Jacobson, Liu & McCanne 1995).

Customization is central to TACC: the TACC server is expected to manage persistent *user profiles* (lists of key/value pairs) that allow workers to provide user-customized service. The TACC server identifies a user via incoming IP address, cookies, or some other client-specific identifier, and automatically routes the appropriate profile information to the worker(s) that will be used to satisfy the request. API's are provided for workers to modify or access profile data directly, and to generate HTML forms that allow users to edit their profiles without requiring direct worker involvement.

TACC provides a very general programming model that subsumes transformation proxies (Fox, Gribble, Chawathe & Brewer 1997), proxy filters (Zenel 1996), customized information aggregators, and search engines. The TACC programming environment, which currently consists of Unix-hosted interface files and build/test harnesses, provides the inter-worker and intra-worker API's, composition rule API's, and glue that allows workers to be authored in a variety of languages (currently C/C++, Perl 5, Java, and Tcl).

2.2 Scalable Cluster-Based TACC Server

A *TACC server* is a platform that instantiates TACC workers and the dispatch rules for routing network data traffic to and from them, and provides support for the inter-worker calling and chaining API's. Roughly, a TACC server is to TACC workers as the Unix shell and runtime are to Unix programs. Our decision to use a cluster of commodity PC's as the basis for our TACC server implementation, as well as the design goals for the cluster runtime software, were motivated by several straightforward observations:

- The inherent hardware redundancy of clusters can be harnessed to provide high system availability.
- For "embarrassingly parallel" workloads with favorable computation to communication ratios, fast and inexpensive system-area networks such as switched 100 Mb/s Ethernet or Myrinet (Myricom 1995) allow a cluster to be treated as a single large computing resource.
- Using commodity PC's as the unit of scaling allows the service to ride the leading edge of the cost/performance curve as it is incrementally expanded.

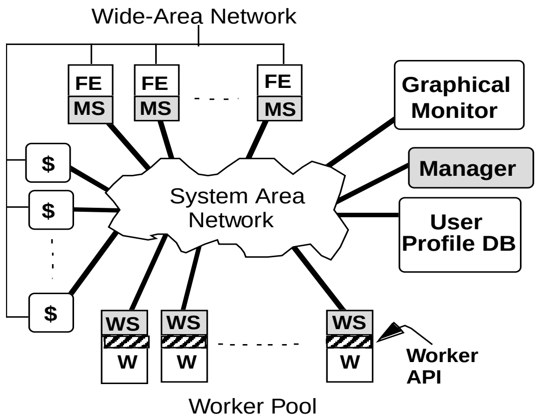


Figure 1 Components of a cluster-based TACC server include front ends (FE), TACC workers (W) and caches ($), a user profile database, a graphical monitor/administration interface, and a fault-tolerant load manager whose functionality logically extends into the manager stubs (MS) and worker stubs (WS).

We describe our system architecture at a high level; a detailed description can be found in (Fox, Gribble, Chawathe, Brewer & Gauthier 1997). The software-component block diagram of our scalable TACC server is shown in Figure 1. *Front* *ends* receive client requests from the outside world and "shepherd" them through the system, by fetching Internet content via the *caches*, matching the request with the appropriate user profile from the *customization database*, and queueing the request for service by one or more *TACC workers* that will process the data according to the user preferences. Front ends maximize system throughput by maintaining state for many simultaneous outstanding requests, and can be replicated for both scalability and availability. The *Load Balancing/Fault Tolerance manager* is responsible for internal reactive load balancing (including exploiting an *overflow pool* of general-use machines), autostarting and reaping TACC workers, and detecting and recovering from various system failures, including crashed components and network partitions, via multicast heartbeat and process-peer fault tolerance. The entire system is administered through a *Graphical Monitor*, which supports asynchronous error notification via email or pager, temporary disabling of system components for hot upgrades, and visualization of the system's behavior using Tcl/Tk (Ousterhout 1994).

Our working prototype of the cluster runtime provides a set of mechanisms (described in (Fox, Gribble, Chawathe, Brewer & Gauthier 1997)) to support incremental scaling, internal load balancing, and high availability for TACC applications. Hiding this machinery behind the TACC API's keeps TACC applications easy to write even though they will be deployed to large user communities who will expect continuous availability (so-called "24 $\times$ 7 operation"). Our cluster has been hosting Wingman on the Berkeley NOW (Network of Workstations) since October 1997, using four dedicated SPARCstation-10's and an overflow pool of up to 100 SPARC Ultra-1 Enterprise servers, about five of which are in use at any given time.

The availability mechanisms in the cluster runtime have kept the system running virtually continuously* despite a variety of observed failures (worker crashes, temporary interconnect partitions, and unexpected node reboots, among others), in addition to the aggressive fault-injection experiments described in (Fox, Gribble, Chawathe, Brewer & Gauthier 1997). Although (as described in that paper) the cluster's availability mechanisms cannot completely mask front-end failures, Wingman masks them using a client-side front-end failover mechanism, similar to Netscape proxy auto-configuration (Netscape Communications Corporation 1998).

2.3 Wingman as a TACC Application

The 3Com PalmPilot's austere graphics library supports a single native bitmap format, and text objects are not automatically word-wrapped and must consist entirely of text in a single visual style (bold, etc.). The principle of client competence (that the client should only perform tasks for which it is competent) therefore suggests that HTML parsing and tag-to-font mapping, page layout with client-specific font metrics, and image scaling with conversion to native bitmap format all be left to the TACC proxy. The proxy side of the Wingman browser uses four TACC workers: the

* During February 1998, we experienced frequent sporadic downtime, which was ultimately traced to a configuration change on the cluster that caused remote execution to sometimes break.

image processor, the HTML processor, the zip processor, and the *aggregator request service.* The TACC model and API's allow parallelism to be exploited in various ways; for example, all inline images are prefetched in parallel with HTML parsing, and whenever possible multiple images are converted in parallel by calling multiple workers.

2.3.1 Image and HTML Processors

The **image processor** reads GIF and JPEG images (the two most common Web formats (Gribble & Brewer 1997)), converts them to an intermediate bitmap form, optionally scales, color-quantizes and dithers, and finally outputs the result in either the PalmPilot's native image format (Tbmp) or our enhanced 2-bit-per-pixel format.

The **HTML processor** parses HTML markup, maps HTML tags to supported font attributes, and generates an intermediate-form page layout. Because the HTML processor knows the client's font metrics and display properties, it can wrap text, flow text around inline images, etc. (A client identification token in the initial client-to-proxy handshake is used to select the correct client profile. The layout code and intermediate layout form are client-independent.)

When inline image tags are encountered, the HTML processor fetches the image, determines how much it should be scaled down using a simple set of heuristics we developed, * and dispatches the original image to an image processor to do the actual scaling and format conversion. Image processing is done asynchronously and in parallel for all images; the HTML parser continues laying out the page in the meantime, and arranges to rendezvous with the image data before returning to the client a simple display list of images and text, using a tokenized markup we have developed that allows the complete page (including all inline images) to be sent as a single object.

2.3.2 Aggregators

An *aggregator* queries one or more Web sites for specific content and collates and formats the results for presentation to the client. (The Metacrawler (go2net, Inc. 1997) service on the Web is a recent example of a Web-based aggregator.) Although Wingman's aggregator mechanism was originally designed as a substitute for HTML forms support, aggregators have become our general mechanism for retrieving information from multiple content sources and filtering that information to best suit the needs of the PalmPilot user. The **aggregator request service** allows access to proxy-based content-aggregation applications.

In Wingman, an aggregator consists of a very simple PalmPilot-native UI that allows the user to write some text describing what is being requested (e.g. the query terms for a Web search, the ticker symbol for a stock quote, etc.). When the "OK" button is tapped, the contents of the form are transmitted to the Wingman TACC proxy's aggregation worker. The worker uses the content of the first box to determine which service the user is requesting (search engine, stock quote, etc.) and the

---

* Roughly speaking, larger images are shrunk more aggressively, imagemaps are shrunk less aggressively to preserve embedded text, and images are pinned to the screen size of the PalmPilot.

content of the second box as parameters to the aggregator. A site-specific aggregator handler then converts this data to an HTML form submission on the proxy side, and the results from the origin server are formatted by the HTML processor. We have deployed aggregators for Yahoo, HotBot, AltaVista, DejaNews, Yahoo's stock quotes service, and TripQuest.

2.3.3 Zip, PalmOS, and Doc Support

If the client fetches a Zip file (a popular format for PalmPilot software archives), the **zip processor** formats a listing of the archive contents in HTML, such that following the link for a particular archive member will cause the zip processor to return the selected member. The HTML created by the zip processor is then parsed by the HTML processor for conversion to our binary markup, just as for normal Web pages. This mechanism exploits TACC's ability to easily *compose* different workers to quickly create new client abilities.

The proxy passes PalmOS *databases* (persistent object stores on the PalmPilot) unmodified, which allows users to use Wingman to install new data and applications onto their PalmPilots; for example, a proxy-side AportisDoc (Aportis Inc. 1998) converter module allows users to immediately save the text of a Web page in this popular e-book format.

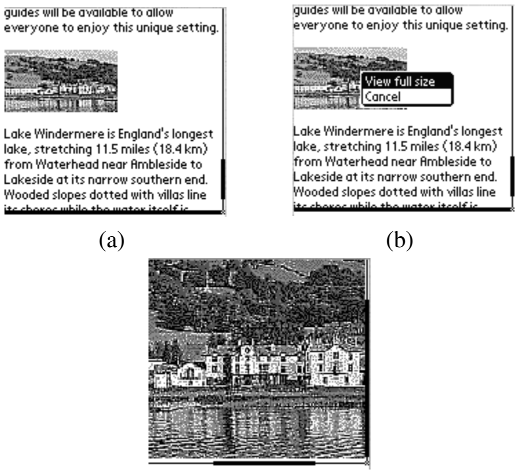
2.4 Client Implementation and User Experience

The "look and feel" of the Wingman browser intentionally resembles desktop browsers, but with specific additions and modifications motivated by the nature of the PalmPilot platform and small device UI's in general. Like desktop browsers, Wingman has a hotlist (bookmark) feature, a local user-controllable cache, and an automatic history cache. Aggregators (described above) are handled by a special aggregator request window containing a popup menu of all available aggregators and a text area for entering aggregation specific data, such as a search string.

The *refinement interface* allows users to zoom in on scaled-down images (Figure 2). Clicking on an image brings up a popup menu that allows the user to view the original image or follow the link (if applicable). Even for images that have no link to follow, we chose to have the popup menu appear anyway (with the "follow link" item disabled) to avoid disorienting the user with different click behaviors for different image types. We originally proposed a mechanism of this kind for regular Web surfing in (Fox & Brewer 1996), but Wingman completely integrates the refinement interface into the user experience.

2.5 Noteworthy Implementation Details

Following the long-standing recommendations of (Clark & Tennenhouse 1990), complete, ready-to-render pages are delivered as application data units (ADU's). This simplifies the client logic considerably: when the network layer delivers a network event, it means that an entire, self-describing ADU is waiting in a buffer ready to



(c)

Figure 2 Top Gun Wingman screenshots: (a) the Middleware '98 conference Web page, (b) the refinement popup menu, and (c) the refined image.

be rendered. Contrast this with traditional HTTP: in that protocol, a network event merely indicates that some number of bytes has arrived, but framing is left entirely to the application. Furthermore, in HTTP, inline images are generally fetched using independent transactions (perhaps in parallel), requiring a nontrivial amount of connection management overhead on the client. Application-level framing eliminates these concerns, makes the client simpler, and enables a "push-friendly" model of asynchrony that even allows out-of-order ADU delivery. Although Wingman was not explicitly designed to support disconnected operation, the ADU-based network layer has no notion of underlying "connection state", so it is possible for a page to be split into ADU's whose arrival is separated by complete (but temporary) network disconnection. This behavior makes it possible to retarget a Wingman-like application for connectionless datagram networks, such as the paging network. We return to these ideas in Section 3.

2.6 Performance

We measured the relative performance of Top Gun Wingman and its competitors over the same bandwidth connection: Netscape Navigator for Windows 95, and two PalmPilot-specific Web browsers, HandWeb and Palmscape. Note that of the three PalmPilot browsers tested, only Wingman supports display of images. As we show below, even with this significant additional functionality, Wingman's performance is still often superior because of its split design.

We measured the average end-to-end latency (elapsed time between clicking on a link and completion of loading the page) for each of a collection of 15 different Web pages chosen to represent the various types of pages on the Web today; we selected pages with varying but representative page sizes, quantity of inline images, content, and location. The pages were initially fetched into the proxy's local cache, to eliminate the highly-variable Internet latencies from the measurements.

Our Internet gateway is a Pentium 133/64MB running Red Hat Linux 5.0 and connected to the Internet via a 128Kb/s ISDN connection to UC Berkeley. To simulate a variety of connection speeds, the PalmPilot browsers were connected to the gateway machine via PPP over an RS-232 serial link. By varying the serial port speed between 57600 and 19200, we simulated low speed connections with slightly lower latency than a typical modem (120ms). 19200 bps is representative of current wide-area and metropolitan-area wireless networks, including Metricom Ricochet (Metricom Corp. 1998), CDPD, and PCS. We believe that such technologies are of particular interest to the mobile professionals who currently comprise a large market segment for devices such as the PalmPilot. 57600 was chosen to represent fast wireline (56K) modems as well as a variety of emerging metropolitan-area picocell wireless data services.

2.6.1 Comparison to Other PalmPilot Browsers

Despite the fact that Wingman is a graphical browser while Palmscape is a text only browser, Wingman was still faster than Palmscape in 93% of the sites tested, not including those sites that caused Palmscape to crash (see below). At 19200 bps, Wingman was at least twice as fast as Palmscape 43% of the time. At 57600 bps, as the time to display a page becomes more CPU bound, this number rises to 54%. The PalmPilot's slow 16 MHz Motorola DragonBall 68328 CPU and limited memory make HTML parsing and layout painfully slow; Wingman relies on the cluster-based proxy for HTML parsing and layout.

In 10% of the test cases, Palmscape crashed while loading the page. We suspect the crashes occur because of the intricate HTML constructs often found on commercial sites; stably capturing all HTML corner cases is difficult, and unlike our TACC server, the PalmPilot client cannot gracefully shield the user from such failures. In this sense, our proxy makes Wingman a more stable and reliable application than its peers.

2.6.2 Comparison to Desktop Browsers

Compared to Netscape running on a Pentium II 300/128MB, Wingman was still faster at downloading and displaying pages in 77% of the test cases. In this situation the latency savings come from the image processor reducing images' resolution and color depth of image before transmission over a slow link (we demonstrated this savings originally in (Fox & Brewer 1996)). Although Netscape receives and displays more data, that excess data is useless to a device with limited display resolution and depth, such as the PalmPilot. At 57600 bps Wingman is at least two times faster than Netscape 33% of the time, and at 19200 bps this number increases to 73%. This re-

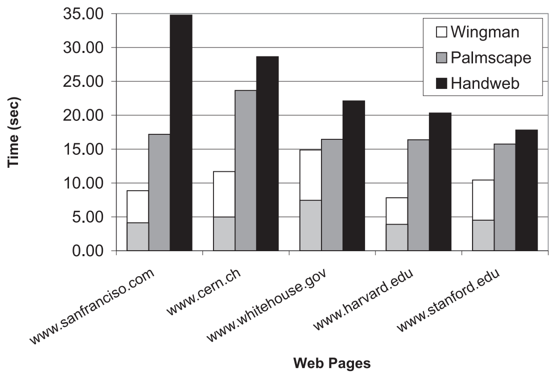


Figure 3 Comparison of end-to-end latency among Wingman, Palmscape, and HandWeb at 57600 bps.

sult reinforces our original hypothesis that a proxy system for thin clients is useful even when the client CPU's are fast enough to display and parse complex pages.

Figure 4 compares Wingman and Netscape Navigator at 19200 and 57600 bps for a number of different Web pages. The top number above each pair of bars is the total number of bytes transferred from the TACC proxy to the Wingman client, i.e. the size of the Wingman ADU corresponding to the delivered page with all inline images. The bottom number is the number of bytes transferred from the origin server to the Netscape client, i.e. the total size of the HTML, all inline images, and HTTP header overhead. The relatively small differences in Wingman's performance at 19200 and 57600 bps suggest that back-end processing dominates latency, whereas with Netscape the network bandwidth dominates latency.

Wingman's HTML processor was written in Perl for fast prototyping and to exploit Perl's superior regular-expression and string manipulation facilities. We conservatively estimate that a factor of five to ten in performance would result from rewriting the HTML processor in a faster language (C/C++); this estimate is compatible with past work comparing the performance of interpreted, compiled, and byte-compiled languages (Romer, Lee, Voelker, Wolman, Wong, Baer, Bershad & Levy 1996). This would make Wingman the fastest browser in virtually all the test cases, even considering image processing latency, leaving only the link bandwidth as the limiting factor in performance.

3 LESSONS AND EXPERIENCE

Top Gun Wingman has been deployed and in use by the general Internet population since October of 1997. Since then, we have processed tens of millions of requests

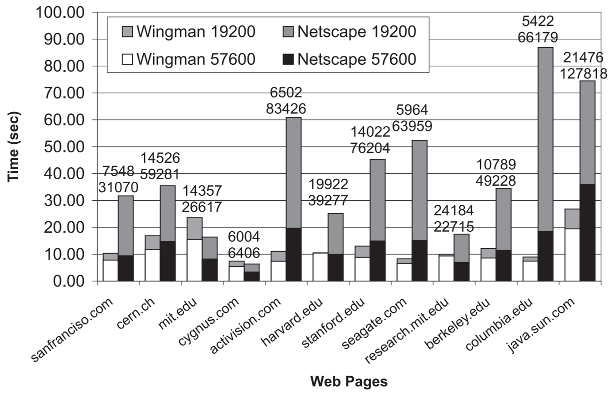


Figure 4 End-to-end latency: Wingman vs. Netscape Navigator at 19200 and 57600 bps for a selection Web pages.

from over 11,000 users.* In this section we share the insights into design, implementation, and operational issues that we have gained from running such a well-used middleware service.

3.1 Moving Complexity to the Proxy

A graphical Web browser is a complex application. The Web consists of many intricate and verbose data formats, hastily designed protocols, and pervasive malformed HTML and image formats that the browser must robustly handle. Even page layout is difficult, given the complex elements such as nested tables, frames, and forms that exist in modern HTML. Implementing a browser is daunting even on a desktop platform with strong operating system support and abundant computation and storage; porting one to a device such as the PalmPilot is difficult to imagine. With only 1-2MB of SRAM, such a device can't even hold all the code. The small heap size (32KB), limited code segment size (32KB relative jump maximum), and lack of native OS support for shared libraries further complicate development.

However, all that the PalmPilot really needs to do is render primitive data types such as images, formatted text, and links. By offloading all of the complex Web browsing functionality (such as HTML parsing, GIF decompression, JPEG rendering, HTTP processing, etc.) to our middleware proxy, we have successfully avoided dealing with all of the PDA resource limitations. Our resulting implementation ef-

*At its peak, the service was processing several requests per second, and was seeing requests from more than a thousand unique IP addresses per day. We estimate that there have been more than 11,000 downloads of the Wingman client software to date.

forts were concentrated on the Unix TACC server implementation, which was rapid and robust because of the rich and mature Unix development environment.

The partitioning of complexity away from the client has had a number of powerful benefits:

- **Client independent back end:** The implementation of the Wingman back end is structured as a pipeline of transformations, only the last of which makes any assumptions about client-specific data formats. This means that porting Wingman to a new PDA platform requires two development efforts: writing the rendering shell on the PDA itself, and creating a new final back end pipeline stage corresponding to that shell. We have made the job of application porting simpler through a well designed application partitioning.
- **Transparent functionality upgrades:** Because most of the complex components of the Wingman implementation reside in the proxy back end rather than the client, it became possible for us to extend the browser's functionality without changing any of the code running at the client. The first example of this was when we improved the graphics transformer (Haeberli 1997) that converts GIF and JPEG images into PalmPilot images (and which also performs lossy compression through resolution and color depth reduction). The improved converter performs edge enhancement, giving users the impression of greater contrast and better image clarity. By simply dropping the new transformer into our proxy, every Wingman user instantly began seeing improved quality images. The zip archive processor and AportisDoc processor are further examples of this mechanism: we have been able to extend the browser to understand new data types without modifying the client implementation. We can extend this strategy in the future to support new technologies such as XML, again without browser changes.
- **Backward compatibility:** During our initial deployment, we rapidly released new versions of the client-side implementation containing bug fixes, UI enhancements, and new features such as a client-side cache and imagemap support. The proxy uses the client version number (embedded in the initial handshake from the client) to determine what objects the client can handle. For example, this mechanism ensures that image maps are only sent to clients whose version number indicates they are capable of handling this feature. The mechanism is similar in spirit to HTTP feature negotiation, but the version-to-feature binding is done late (at the proxy) and can be easily changed. By putting this capability in our middleware, we shielded end servers from having to deal with this additional client heterogeneity.

3.2 Middleware Availablility

As any designer of highly available services will attest, the devil is often in the implementation details. In our AMWP architecture, it is the responsibility of the TACC server to provide availability and robustness mechanisms that are independent of any worker; the mechanisms employed in our cluster-based TACC server give rise to the requirement that each TACC worker be restartable and atomic (alluded to in section 2.1). We have found that despite this programming restriction, the orthogonal separation of availability/robustness from worker code has made the overall system simpler to engineer and more robust, and allows new workers to be prototyped more quickly since they inherit these behaviors for free.

The 11,000 real users stressing our proxy implementation revealed a number of interesting corner cases in the TACC server design and Wingman implementation that caused service delays and even interruptions. For example, our initial design of the Wingman transformation pipeline included a "dispatch" worker that would route data to subordinates based on the type of the data it receives (HTML, Zip files, etc.), and then block until the subordinate(s) finished. The long blocking latency made it appear to the TACC server that the dispatcher worker was overloaded, causing more of them to be spawned until there was almost one dispatcher per outstanding outstanding user request. This resulted in hundreds of processes being spawned across our cluster, filling the cluster nodes' process tables and causing the service to become unavailable.

Except for failures due to such rare but interesting cases, all Wingman outages have been due to factors beyond our control, such as disrupted service from our campus ISP and power failures that weren't caught by the departmental UPS.

4 RELATED AND FUTURE WORK

Middleware services such as filtering and on-the-fly compression have become particularly popular for HTTP (Internet Engineering Task Force 1997), whose proxy mechanism was originally intended for users behind security firewalls. The mechanism has been used to shield clients from the effects of poor (especially wireless) networks (Fox & Brewer 1996, Liljeberg, M., et al. 1996), perform filtering (Zenel 1996) and anonymization, and perform value-added transformations on content, including Kanji transcoding (Sato 1994), Kanji-to-GIF conversion (Yee 1995), application-level stream transducers (Brooks, Mazer, Meeks & Miller 1995), and personalized agent services for Web browsing (Barrett, Maglio & Kellem 1995). Infrastructure proxies have also been used as a mechanism for application partitioning, as in the Wit project (Watson 1994*a*, Watson 1994*b*) which focuses on partitioning between the mobile and fixed infrastructure. The Video Gateway (Amir, McCanne & Zhang 1995) is an example of an infrastructure proxy that performs application-level adaptation and serves as a multicast to unicast protocol gateway.

Application design and implementation on PDA or smartphone class devices has recently generated significant research effort. In (Fox, Gribble, Chawathe, Polito, Ling, Huang & Brewer 1997, Fox & Brewer 1996, Shimada, Iwami & Tomokane 1997), the user interface issues associated with introducing proxy controls as an orthogonal extension to a Web browser interface are explored. (Schilit & Bickmore 1997) describes an intelligent proxy that attempts to classify HTML pages into a fixed number of bins, and performs semantic compression and layout modification based on the chosen bin to repurpose the page for a PDA.

Our TACC middleware has actually been used to implement a variety of distinct applications besides Wingman. TranSend (Fox, Gribble, Chawathe & Brewer 1997) is a Web acceleration proxy that performs on-the-fly, datatype-specific compression, in particular on GIF and JPEG images; it has been available to the public since April 1997 and some of its main ideas have been instantiated commercially as Intel QuickWeb.* Top Gun Mediaboard is an electronic shared whiteboard (a derivation of the desktop *mediaboard* (Chawathe et al. 1998) application) for the PalmPilot, in which TACC workers perform protocol adaptation as well as data transformation; it is in prealpha use at UC Berkeley, and performs satisfactorily even over slow links such as the Metricom Ricochet wireless packet radio modem (Metricom Corp. 1998). Top Gun Mediaboard extends Wingman's markup and protocol to accommodate the draw operations and user interactions required by a shared whiteboard.

The commercial ProxiWeb client (ProxiNet, Inc. 1998) adds proxy-based implementations of features missing from Wingman, including secure connections, cookies, and HTML forms. Future research includes merging Top Gun Wingman and Top Gun Mediaboard, resulting in a "generic thin client" application that acts as the drawing, rendering, and interaction shell for proxied applications such as MBONE access, Web access, or an environment-aware interaction device (Hodes, Katz, Servan-Schreiber & Rowe 1997). We will also further explore the security and privacy implications of middleware in general (in comparison to end-to-end security) and Wingman in particular, attempting to apply some ideas from our earlier work on security protocol adaptation for thin clients (Fox & Gribble 1996).

5 CONCLUSIONS

This paper described the proxy-based architecture and implementation of Top Gun Wingman, the first graphical Web browser for the PalmPilot PDA. We described our middleware philosophy, which we refer to as the Adaptive Middleware Proxy (AMWP) approach: let the client do only what it does well (client competence), and push all other functions to a transparent infrastructure proxy. The proxy side of Wingman was described in terms of our TACC (transformation, aggregation, customization, caching) programming model for interactive Internet services, and discussed our experience with a publicly-available cluster-based TACC server that has been operational since late 1997 with virtually no unscheduled downtime.

Our experience with Wingman, as well as with the other TACC applications we briefly described, supports our claims for the Adaptive Middleware Proxy (AMWP) approach. Wingman has better performance than a reduced client-only port, has new features and behaviors (such as browsing software archives and saving e-books, in the case of Wingman), and has the ability to almost completely leverage the entire

* There is no formal connection between QuickWeb and TranSend.

existing Web-based content and services infrastructure. Our proxy implementation has low administration cost (due to our commodity-PC cluster design and graphical administration interface) that makes it realistic to operate services on a 24×7 basis for large user communities, and the ability to add features at the proxy largely eliminates the client software distribution problem and preserves backward compatibility when client upgrades do occur.

In addition, our experience with a user community of over 11,000 suggests that users will have no problem accepting middleware as a pervasive element of Internet services. For these reasons, and because Wingman demonstrates key technologies needed to rapidly deploy specialized information-delivery services for thin clients, we believe that the principles successfully demonstrated by Top Gun Wingman will play a prominent role in the coming wave of applications for convergent thin client devices.

5.1 Acknowledgments

We extend our appreciation to the many students, staff, and faculty at UC Berkeley who have helped us support our proxy cluster, debug our applications, suggest feature enhancements, and act as reviewers for this paper. Special thanks go to Tim Kimball for all of his support efforts and Paul Haeberli for allowing us to use his high quality graphics transformation code. Finally, we would like to thank the PalmPilot user community for their tireless support and encouragement.

REFERENCES

Amir, E., McCanne, S. & Zhang, H. (1995), An Application Level Video Gateway, *in* 'Proceedings ACM Multimedia 1995'. Available at `http://http.cs.berkeley.edu/~elan/articles/pub/vgw.ps`.

Aportis Inc. (1998), 'AportisDoc Overview'. `http://www.aportis.com/products/AportisDoc/benefits.html`.

Barrett, R., Maglio, P. P. & Kellem, D. C. (1995), How To Personalize the Web, *in* 'Conference on Human Factors in Computing Systems (CHI 95)', Denver, CO. WBI, developed at IBM Almaden; see `http://www.raleigh.ibm.com/wbi/wbisoft.htm`.

Brooks, C., Mazer, M. S., Meeks, S. & Miller, J. (1995), Application-Specific Proxy Servers as HTTP Stream Transducers, *in* 'Proceedings of the Fourth International World Wide Web Conference'.

Chawathe, Y., Fink, S., McCanne, S. & Brewer, E. A. (1998), A Proxy Architecture for Reliable Multicast in Heterogeneous Environments. Currently under review.

Clark, D. & Tennenhouse, D. (1990), 'Architectural Considerations for a New Generation of Protocols', *Computer Communication Review* **20**(4), 200–208.

Floyd, S., Jacobson, V., Liu, C. & McCanne, S. (1995), A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing, *in* 'ACM SIGCOMM '95', Boston, MA, pp. 342–356.

Fox, A. (1997), The Case for TACC: Scalable Servers for Transformation, Aggregation, Caching and Customization. Qualifying Exam Proposal, UC Berkeley Computer Science Division. http://www.cs.berkeley.edu/~fox/papers/quals.ps.

Fox, A. & Brewer, E. A. (1996), Reducing WWW Latency and Bandwidth Requirements via Real-Time Distillation, *in* 'Proceedings of the Fifth International World Wide Web Conference', World Wide Web Consortium, Paris, France.

Fox, A. & Gribble, S. D. (1996), Security On the Move: Indirect Authentication Using Kerberos, *in* 'Proc. Second International Conference on Wireless Networking and Mobile Computing (MobiCom '96)', Rye, NY.

Fox, A., Gribble, S. D., Brewer, E. A. & Amir, E. (1996), Adapting to Network and Client Variability via On-Demand Dynamic Distillation, *in* 'Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)', Cambridge MA.

Fox, A., Gribble, S. D., Chawathe, Y. & Brewer, E. (1997), TranSend Web Accelerator Proxy. Free service deployed by UC Berkeley. See http://transend.cs.berkeley.edu.

Fox, A., Gribble, S. D., Chawathe, Y. & Brewer, E. A. (1996), 'Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives', *IEEE Personal Communications* . Special issue on adapting to network and client variability.

Fox, A., Gribble, S. D., Chawathe, Y., Brewer, E. A. & Gauthier, P. (1997), Cluster-Based Scalable Network Services, *in* 'Proceedings of the 16th ACM Symposium on Operating Systems Principles', St.-Malo, France.

Fox, A., Gribble, S. D., Chawathe, Y., Polito, A., Ling, B., Huang, A. C. & Brewer, E. A. (1997), Orthogonal Extensions to the WWW User Interface Using Client-Side Technologies, *in* 'User Interface Software and Technology (UIST) 97', Banff, Canada.

go2net, Inc. (1997), 'Metacrawler search service'. http://www.metacrawler.com.

Gribble, S. D. & Brewer, E. A. (1997), System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace, *in* 'Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems', Monterey, California, USA.

Haeberli, P. (1997), 'Personal communication'.

Hodes, T. D., Katz, R. H., Servan-Schreiber, E. & Rowe, L. (1997), Composable Ad-hoc Mobile Services for Universal Interaction, *in* 'Proc. Third International Conference on Mobile Computing and Wireless Networking (MobiCom '97)', Budapest, Hungary.

Internet Engineering Task Force (1997), 'Hypertext transfer protocol - http 1.1', RFC 2068.

Liljeberg, M., et al. (1996), Enhanced Services for World Wide Web in Mobile WAN Environment, Technical Report C-1996-28, University of Helsinki CS Department.

Metricom Corp. (1998), 'Ricochet Wireless Modem'. http://www.ricochet.net.

Myricom (1995), Myrinet: A Gigabit Per Second Local Area Network, *in* 'IEEE Micro'.

Netscape Communications Corporation (1998), 'Netscape Proxy Automatic Configuration', http://home.netscape.com/eng/mozilla/2.02/relnotes/unix-2.02.html#Proxi%es.

Ousterhout, J. K. (1994), *Tcl and the Tk Toolkit*, Addison-Wesley.

ProxiNet, Inc. (1998), 'ProxiWeb Thin Client Web Browser', http://www.proxinet.com.

Romer, T. H., Lee, D., Voelker, G., Wolman, A., Wong, W. A., Baer, J.-L., Bershad, B. N. & Levy, H. (1996), The Structure and Performance of Interpreters, *in* 'Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)', Cambridge, MA.

Sato, Y. (1994), 'DeleGate Server'. http://wall.etl.go.jp/delegate/.

Schilit, B. & Bickmore, T. (1997), Digestor: Device-Independent Access to the World Wide Web, *in* 'Proc. Sixth International World Wide Web Conference (WWW-6)', Santa Clara, CA.

Shimada, T., Iwami, N. & Tomokane, T. (1997), Interactive Scaling Control Mechanism for World Wide Web Systems, *in* 'Sixth International World Wide Web Conference (WWW-6)', World Wide Web Consortium, Santa Clara, CA.

Watson, T. (1994*a*), Application Design for Wireless Computing, *in* 'Mobile Computing Systems and Applications Workshop'.

Watson, T. (1994*b*), Wit: An Infrastructure for Wireless Palmtop Computing, Technical Report CSE-94-11-08, University of Washington. `http://snapple.cs.washington.edu:600/papers/wit.ps`.

Yee, K.-P. (1995), 'Shoduoka Mediator Service'. `http://www.shoduoka.com`.

Zenel, B. (1996), A Proxy Based Filtering Mechanism for the Mobile Environment. Thesis Proposal.

6 BIOGRAPHY

Armando Fox received a BSEE from M.I.T. and an MSEE from the University of Illinois, has worked as a CPU architect at Intel Corp., and is currently completing his Ph.D. at the University of California, Berkeley, as a researcher in the Daedalus/BARWAN and InfoPad projects. He will be an assistant professor at Stanford University starting in January 1999. His primary interests are application-level support for adaptive mobile computing, design of reliable distributed systems to support infrastructural applications, and user interface issues related to mobile computing.

Ian Goldberg is a Graduate Student Researcher and founding member of the Internet Security, Applications, Authentication and Cryptography (ISAAC) research group at UC Berkeley. His research areas include cryptography, security, privacy systems, and digital cash. He holds a B.Math. from the University of Waterloo, Canada, and a M.Sc. from UC Berkeley, USA.

Steven Gribble received a combined Computer Science and Physics B.Sc. from the University of British Columbia in 1995, and a Computer Science M.Sc. from the University of California, Berkeley in 1997. He is currently pursuing his Ph.D. at Berkeley, and expects to graduate in May of 2000. His interests include application-level support for adaptive mobile computing, and system and compiler support for scalable, highly-available infrastructure services.

David C. Lee received a Bachelor of Science in Electrical Engineering Computer Science from the University of California, Berkeley, in May 1998. Though currently working in industry, he plans to go to graduate school within a few years.

Anthony Polito received a Bachelor of Arts in Computer Science from the University of California, Berkeley in 1998. He plans to go to graduate school in 1999.

Eric A. Brewer is an Assistant Professor of Computer Science at U.C. Berkeley, and received his Ph.D. in CS from MIT in 1994. Interests include mobile and wireless computing (the InfoPad and Daedalus projects); scalable servers (the NOW, Inktomi, and TranSend projects); and application- and system-level security (the ISAAC project). Previous work includes multiprocessor-network software and topologies, and high-performance multiprocessor simulation.