The Importance of Translucence in Mobile Computing Systems

MARIA R. EBLING, BONNIE E. JOHN and M. SATYANARAYANAN
Carnegie Mellon University

Mobile computing has been an active area of research for the past decade, but its importance will increase substantially in the decade to come. One problem faced by designers of mobile systems is that of maintaining the illusion of connectivity even when network performance is poor or non-existent. The Coda file system uses its cache to maintain this illusion. Extensive experience with the system suggests that, although users find the functionality provided by the system extremely valuable, new users face an arduous learning curve and even experienced users are sometimes confused by the system's behavior. The fundamental problem is that the lack of a strong network connection causes the system to violate a key property of caching: transparency. To overcome this problem, we have built an interface, called the CodaConsole, that makes caching translucent to users through controlled exposure of cache management internals. The interface exposes critical aspects of caching to support the mobile user while hiding noncritical details to preserve usability. This article presents the design, implementation, and usability evaluation of this interface. The CodaConsole successfully makes caching translucent in the presence of disconnected or weakly connected operation. The most surprising result was that novice Coda users performed almost as well as experienced Coda users.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User interfaces—*evaluation/methodology*; D.4.3 [**Operating Systems**]: File Systems Management—*distributed file systems*

General Terms: Design, Experimentation, Human Factors

Additional Key Words and Phrases: Coda, disconnected operation, mobile computing, translucent cache management, weakly connected operation

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA), Air Force Materiel Command, USAF under agreement numbers F19628-96-C-0061 and F19628-93-C-0193, the National Science Foundation Award #IRI-9457628, the Xerox Corporation, the Intel Corporation, and the IBM Corporation. Some of the equipment used in the process of conducting this research was acquired through the National Science Foundation Equipment Grant #9022511. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Intel, Xerox, DARPA, or the U.S. Government.
Authors' addresses: M. R. Ebling, IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY, 10598; email: ebling@us.ibm.com; B. E. John, Human Computer Interaction Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213; email: bej@cs.cmu.edu; M. Satyanarayanan, Department of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213; email: satya@cs.cmu.edu.

1. INTRODUCTION

Mobile users need access to information stored in a variety of places. They might need to look up information from a corporate database, distributed file system, or intranet. They might need to access data from the Internet or the Web. Mobile users might also need to access information from a variety of computing devices. These users may be using laptop computers or handheld devices. No matter what type of device the user happens to have and regardless of how much disk space or memory it has, people can always find a way to fill it to capacity, making mobile information access vital to these users.

Mobile information access poses some unique problems, one of which is that mobile clients experience a wide range of network connectivity [Satyanarayanan 1996], from fast, reliable, and cheap networks (*strong connectivity*) at one extreme to slow, intermittent, or expensive ones (*weak connectivity*) at the other. Mobile clients may also experience periods of disconnection. This problem is intrinsic to mobile computing and will not simply disappear over time. Consequently, systems must cope gracefully with changing network conditions. The Coda File System supports mobile access to a distributed file system and provides mechanisms that allow the file system to adjust transparently to changing network conditions, thus insulating users from the vagaries of network connectivity [Satyanarayanan et al. 1990]. Although the mechanisms Coda provides are effective, they introduce a new problem: because important information is masked, users are sometimes confused by behaviors not seen in other systems. Although Coda explores these topics in the realm of a file system, the lessons learned in Coda can be applied more widely, including to the Web and to handheld devices.

Dix advocates an alternative approach that makes users aware of the state of connectivity [Dix 1995; Dix and Beale 1996]. He argues that such systems can afford to use more complex caching algorithms and that caching decisions should be a cooperative activity between the system and the user. Furthermore, he argues that transparency is precisely the wrong approach in these systems because it hides the information that users need to cooperate effectively. He proposes the need for systems that make users aware of appropriate information in a low-effort, even subconscious, way that does not interfere with the user's primary tasks. The approach that he believes will lead to the best interfaces is that of adding information to points of shared focus, such as showing a temporarily disconnected user's cursor grayed out.

In this article, we describe how the usability of systems such as Coda can be improved through *translucent caching*. This technique exposes critical details of caching to support availability while continuing to hide noncritical details to preserve usability. Our approach effectively introduces a shared focus, allowing the file system to communicate directly with its users. This article begins with a discussion of the concerns that a translucent system must consider to avoid exposing too many details. We then present an overview of a graphical interface, called the CodaConsole, that makes caching translucent to Coda users. We continue with a presentation of the results of a usability study that validates our approach. Our study confirms that translucent caching is a promising approach to improving the usability of highly available systems. In fact, in our study, new Coda users performed nearly as well as experienced users.

2. BACKGROUND

The research on Coda has been extensively reported in the systems community, but this work extends its applicability to the HCI community as well. Because Coda's functionality is widely known [Kistler and Satyanarayanan 1992; Mummert et al. 1995], we have chosen not to include a detailed description of Coda here. Instead, we refer readers looking for additional background to ACM's digital library where a brief overview of Coda is available [ACMDL 2002]. In addition, because readers may not be fluent in Coda terminology, we include footnotes that define Coda terms as they are introduced. In this section we present anecdotal experiences of Coda users. We then discuss related work, from both the file systems and the computer-supported cooperative work communities.

2.1 Anecdotal Experiences with Coda

The original version of Coda offered users detailed control over the contents of the cache.[1] Unfortunately, the feedback provided by the system about the cache was minimal. Experience with the deployed system revealed that, although experienced users found the functionality extremely useful, they continued to be confused by the system's behavior even after months of regular use.

What events caused confusion among experienced users? One common cause of confusion occurred when the network conditions degraded unbeknownst to the user. Because the client transparently transitioned from strong to weak connectivity, users continued to expect strongly connected behavior when, in fact, the system was only capable of providing weakly connected behavior. In particular, users expected their updates to appear on the servers immediately. When a user noticed that the servers did not have the latest changes (e.g., those made from another machine), the user understandably became very concerned for the safety of those updates. Multiple users observed this particular problem (often multiple times).

Another procedure common among experienced users highlights the difficulty that even they have in preparing their caches for disconnected and weakly connected operation. Once they are done preparing, they have difficulty determining whether their hoarded[2] data are available for use. Because the penalty for discovering missing files during disconnection is high, experienced users frequently test their preparation by disconnecting and attempting to access their work before actually leaving their offices. This strategy allows them to reconnect and correct any problems they encounter, but it imposes a great burden. From this evidence, it is clear that users need better feedback regarding what data are available for use.

[1]The *cache* stores a subset of the files maintained by the Coda servers.

[2]Coda allows users to specify files and directories that the cache manager should maintain in the cache at all times, if possible. These files and directories are called *hoarded* data.

The fundamental problem seems to be that Coda takes an application-transparent approach. However, as much as we might like systems to exhibit the same behaviors when operating over a modem as they do when operating over an Ethernet, they simply cannot do so. A missing file requested during a period of weak connectivity can take a substantial amount of time to fetch. No system today can avoid paying this penalty and no system can maintain the illusion of connectivity in the face of such penalties.

One challenge then is to explain the system's behavior to its users. Coda's application-transparent approach necessarily introduces modal behaviors, which have long been known to cause usability problems [Tesler 1981]. These modal behaviors, which are critical to supporting network-oblivious applications, cannot simply be removed. The key to solving the usability problems such behaviors introduce is exposing them to users in meaningful and apparent ways [Monk 1986]. A second challenge is to provide users with control over how network resources get used. A good example of how users have demanded this control is the ability of Web browsers to disable the automatic download of images. Users of weakly connected file systems need controls akin to those provided by Web browsers.

2.2 Related Work

Although much work has been done on highly available file systems [Alonso et al. 1990; Huston and Honeyman 1993, 1995; Kistler and Satyanarayanan 1992; Kuenning and Popek 1997; Mummert et al. 1995; Tait et al. 1995], few systems address their usability. Certain major pieces of related work are important to mention. The first is a system from Columbia University, which we call transparent analytic spying [Tait et al. 1995]. Tait and his colleagues provided a simple user interface that offered users the ability to aggregate hoarded data and to control which aggregates were hoarded, but this interface offered no feedback to users and no studies evaluating the usability of the interface were ever done. The second is a system from UCLA, called Seer [Kuenning and Popek 1997]. Seer applied multidimensional clustering to hoard files automatically. This system required almost no intervention by the user and the results were quite impressive. Seer's weakness was in its inability to build clusters of files that were meaningful to users. This weakness makes it difficult to expose meaningful availability information to the user.

File systems are not the only systems to face the vagaries of network connectivity and to consider ways in which to expose connectivity to their users. Researchers working in the area of CSCW have also been considering these problems (e.g., Cheverst et al. [1999], Davies et al. [1996], Friday et al. [1999], Pascoe et al. [1998a,b], Rodden et al. [1998], Walker [1998], and Wax [1996]). It is this work that offers the most insight into the problem at hand. One piece of work deserves particular mention. Cheverst and his colleagues exposed the connectivity of field engineers in a collaborative application, called MOST, that provided access to safety-critical information regarding a power plant [Davies et al. 1996]. From their initial evaluation of the interface, they learned that engineers felt that the interface provided insufficient information concerning the constraints imposed by the mobile communications environment. For example, engineers felt the interface did not give them an appreciation of the fact that establishing a connection with a remote user could take a considerable amount of time (over 10 seconds). They also felt that the interface did not give them feedback regarding the state of communication with other group members. Finally, they wanted to have control over the costs associated with their group collaborations. A subsequent interface gave users feedback regarding the state of connectivity and also gave them control over the associated costs by allowing the engineers to specify quality of service requirements on a per-user as well as per-group basis [Cheverst et al. 1999]. As with our work, these researchers also found the need to make the state of connectivity visible to their users and to give them more control over certain aspects of the system.

2.3 Summary

Anecdotal evidence suggested that even experienced users were having difficulty in understanding Coda's behavior as well as in exploiting its features. Our approach to addressing these problems is to make aspects of the system translucent, rather than transparent, to users. This approach is in direct contrast to other file system projects that tried to further hide the details of the system from the users or simply tried to add a graphical representation of the existing controls. It is more similar to the approach taken by the MOST project, which used translucence to set the expectations of field engineers and to give those engineers control over the system. It is also consistent with the position taken by Dix that systems need to give feedback in ways that do not interfere with users' primary tasks. However, Dix assumes that systems interact directly with the user. Because file systems do not typically interact directly with the user, our approach introduces a shared focus, which we call the CodaConsole. The next section presents the design rationale for the CodaConsole.

3. DESIGN RATIONALE

Before adding a shared focus to make a system translucent, one must identify which aspects of the system to expose to users as well as consider the risks involved in exposing those details. In this section, we summarize the rationale for those details the system exposes to the user. We then discuss the risks involved in exposing those details and the steps we took to minimize the impact on the user.

Our goals for translucent caching were many. We wanted to eliminate the confusion users experienced when certain events occurred. We wanted to give users the ability to influence the way in which the system handled their requests, when appropriate. We wanted to allow users to prepare easily for disconnected and weakly connected operation and to be confident in their preparation. Furthermore, we wanted to teach users about Coda and its operation in a reasonable amount of time. In this section, we explore these issues in more detail. In later sections, we operationalize these goals for the purposes of evaluation.

3.1 What Aspects to Expose?

In identifying the details to expose to users, we considered two questions:

—What system behaviors must be brought to the user's attention?
—What aspects of the system do users need and want control over?

The answers to these questions were based upon our users' experiences with the original Coda deployment and greatly influenced the design of the interface presented in Section 4.

Certain system behaviors introduce confusion. For example, when the state of network connectivity changes, the system adapts by changing its behavior. This adaptation is completely transparent to the user in most situations. Under certain conditions, however, the transparency of this transition is problematic. In particular, if the change in network connectivity is not apparent to the user and if the user is operating in an environment in which he or she can detect the change in system behavior (such as in the example concerning the delayed propagation of file changes given in Section 2.1), then the user can become confused by these behaviors. Errors can also cause confusion. For example, when users' tokens[3] expire, users no longer have the authority to read or write their own files. Because the file system has no direct interaction with the user, it must return a UNIX-compatible error message to the application (in this case, permission denied). If the application simply passes that error message through to the user, it will mislead the user to search for incorrectly set permissions rather than expired tokens. Coda needs a better way to bring error conditions such as these directly to the user's attention. We concluded that an interface that brings error conditions to the user's attention and that sets user expectations appropriately could greatly improve the usability of Coda and of similar systems.

The second question, regarding giving users control over aspects of the system, posed more difficulty because users of the original system had little control over the system so we had less data for what was difficult or desired. Our approach to this question was to examine resources scarce enough to warrant users spending the time and effort necessary to control them. For the mobile user, these resources include cache space (the only aspect of the original system users had any control over), network bandwidth, and energy consumption, although we focused only on the first two. With respect to cache space, our conclusions were that users required assistance preparing their caches for disconnected or weakly connected operation and that they needed feedback regarding the availability of their data (to prevent the need for practice disconnections, as described in Section 2.1). With respect to network resources, our conclusions were that users required control over whether missing files are fetched during weakly connected operation (either on demand or via a hoard walk) and

[3]A *token* authenticates the user to the file servers (similar to Kerberos tickets [Steiner et al. 1988]). Tokens are obtained in a process similar to logging in and are valid only for a limited period (by default, 25 hours). After this time, users must reauthenticate to gain continued access to the system.

that they needed control over whether reintegration[4] takes place and, if so, how much.

3.2 What Are the Risks?

Exposing the details of caching to the user is a risky proposition. Exposing too much detail could require the user to spend so much time managing the cache that it detracts from the user's productivity—defeating the entire purpose of the system.

The primary threat to usability for a mobile client comes from *critical cache misses*, requests for files that are critical to the user's work, but that are missing from the cache. Because critical cache misses present such a large problem to the usability of a highly available file system, the system must minimize (or, even better, eliminate) them.

Secondary threats result from demands the system places on the user's valuable time. The system demands the user's time in three ways. Users must invest time in learning how to use the system and its interfaces. They must wait for requests to be serviced, not necessarily insignificant in the face of weak connectivity. Finally, they invest time offering advice to the system in hopes of experiencing better availability.

3.3 What Are the Design Principles?

HCI researchers have studied usability for many years. In designing an interface for translucent caching, we must remember to apply the results of these efforts (e.g., Nielsen [1993] and Cooper [1995]). To minimize the risks inherent in translucent caching, the design of the interface pays particular attention to the following principles.

—*Do no harm*. Translucence should solve more problems than it introduces. Although obvious, this point is important to keep in mind.
—*Do not assume an interactive user*. As a logical part of a multiuser operating system, Coda cannot assume an interactive user. The user may start a long-running process (e.g., a system build) and then leave the office to run an errand, expecting the process to have finished in the meantime.
—*Be unobtrusive*. The user's goal is to work, not to monitor the system. The number of disruptions to the user's work must be minimized.
—*Balance costs with benefits*. The cost of using a system that can cope with disconnected and weakly connected operation must not outweigh the benefits. Relevant costs include the time required to give the system advice and the time required to prepare for the possibility of degraded connectivity.

4. DESIGN

The previous section explored the requirements placed upon our system as well as the design principles we followed. From this analysis, we identified aspects

[4]*Reintegration* is the process of propagating updates made during disconnected or weakly connected operation back to the server(s).

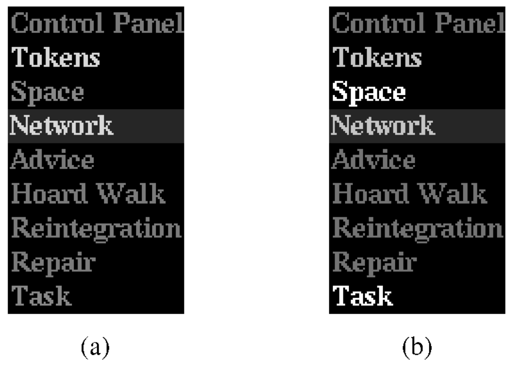


Fig. 1. Indicator lights give users a peripheral awareness of system state: (a) default green–yellow–red color scheme; (b) monochrome scheme. In both views, the tokens and network indicators are shown with a warning urgency level and the space and task indicators are shown with a critical urgency level. All other indicators are normal.

of the system about which users need to be made aware and over which users need to be given control. We now turn to the design of an interface that makes caching translucent. It exposes aspects of caching and network usage to the user while balancing the burdens placed upon them with the benefits they receive. In this section, we give an overview of this interface.

4.1 Indicator Lights Metaphor

The primary window of the CodaConsole interface, shown in Figure 1, presents a small number of indicator lights to the user. These indicators introduce a point of shared focus between the user and the file system. Each indicator light is labeled with a term familiar to Coda users, such as tokens, repair,[5] and reintegration. It is worth noting that our interface assumes users have a minimal understanding of how Coda works and a familiarity with Coda terminology, which we supply through training; it is not intended as a walk-up-and-use interface.

Figure 1 shows two views of the indicator window: one in the default green–yellow–red color scheme and one in a monochrome scheme. The window itself is very small, only about $1'' \times 2''$, so we expect that users will not find it too imposing and will be willing to keep it visible at all times (*Be unobtrusive*). Like the indicator lights shown on the dashboard of some cars, our indicator lights use color to encode severity. Red indicates a critical problem; yellow-orange indicates a warning status; green means all is well. Unlike a car, however, the colors used by our interface can be customized to accommodate colorblind individuals (*Do no harm*). Also unlike a car's indicators, our indicators are dynamic. To display a more detailed explanation of the problem, the user need only doubleclick on the indicator light (*Be unobtrusive*). The *Help* button provides further explanations and possible user-initiated solutions.

[5]In a system that supports optimistic replications, file replicas occasionally become in conflict. These conflicts are corrected through a process Coda calls *repair*.

In the sections that follow, we describe the behavior of each indicator light. Due to space limitations, we cannot show the details of every interface screen. Interested readers are referred elsewhere [Ebling 1998].

4.2 Control Panel

In actual fact, the control panel is not an indicator light. It is always green. Like the control panel in the Windows® environment, it allows the user to configure the system. One configuration panel allows users to change the colors used in the indicator lights. Figure 1 shows two possible choices: monochrome and red–green–yellow. In addition, users can specify how they should be alerted to the various events. For example, when their tokens expire, should the user be notified? If so, what is the urgency of this event? Should the system simply change the indicator's color? Should it beep? Should it flash the indicator? Should it pop up the dialog window? By giving users control over whether and how they are alerted to each event, the interface can be unobtrusive (*Be unobtrusive*) and avoids annoying the user with irritating alerts and uninteresting events (*Do no harm*).

4.3 Tokens

The tokens indicator keeps users informed about the state of their authentication. If the tokens indicator is green, then the user is authenticated to the system. Yellow indicates that the user's authentication has expired and red indicates that some activity, such as a file request, is waiting for the user to obtain a token. This indicator addresses a problem described in Section 3.1.

4.4 Space

The space indicator alerts users to various space problems. If the user were to doubleclick on the space indicator light, a window similar to the one shown in Figure 2 would appear. This window provides three gauges that show the state of different areas of space important to the cache manager's operation. For example, this indicator can show users when they have requested that more data be hoarded than can fit in their cache.

4.5 Network

The network indicator light alerts users to the state of the network connection. In effect, it tells the user whether the system is operating strongly connected, weakly connected, or disconnected to any servers. This indicator presents some difficulties because a client can be operating strongly connected with respect to one server, weakly connected with respect to another, and disconnected with respect to a third. We handled this problem conservatively. If the system is operating disconnected with respect to one or more Coda servers, then the indicator shows red. If it is operating weakly connected with respect to one or more servers but strongly connected to all others, then it shows yellow. Otherwise, it shows green.

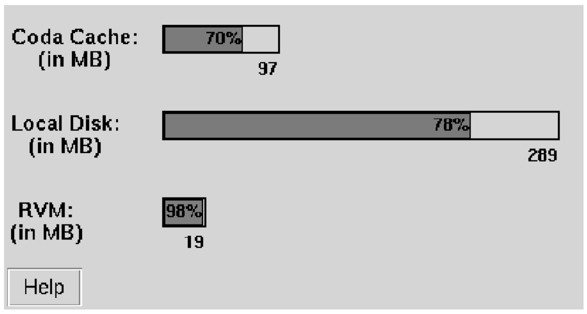


Fig. 2. This space information window might appear after the user doubleclicks on the space indicator shown in Figure 1. The gauges shown in this window indicate the current state of three separate space areas important to the cache manager. The top gauge shows the state of the Coda cache (e.g., 70% of the cache contains hoarded data). The middle gauge shows the state of the local disk partition containing the cache. The bottom meter shows the state of RVM space, a critical data structure internal to the cache manager. The user can get help in understanding the problem and identifying solutions by clicking on the Help button in the lower left-hand corner.

4.6 Advice

The system asks for user advice on occasion. When such a request arrives, the advice indicator changes color to show whether a user's request is waiting for the advice result. The indicator changes to red if the advice request is delaying a request made by the user and to yellow if the advice request is delaying a background request made by the system. Because the user is frequently waiting for the completion of such requests, the default configuration for advice requests is to pop up the dialog box on the user's screen.

These requests allow users control over the use of network resources, the need for which was discussed in Section 3.1. For example, Coda requests advice from the user when a requested file is missing from the cache during a period of weak connectivity. In most cases, a user would rather be asked whether a file is important before being made to suffer a long fetch delay. However, simply popping up a dialog box on each cache miss is unsatisfactory for numerous reasons.

Ideally, the system could distinguish those files for which users are willing to wait from those for which they are not. We hypothesize that we can model user patience and that this model depends upon just two things: the importance of the file and the expected fetch time [Mummert et al. 1995]. On a cache miss, the cache manager first invokes the model with the file name, current bandwidth, and other system information as parameters. If the model suggests that no user interaction is needed, the miss is serviced transparently. Otherwise, the system requests advice from the user. If no response is received with a timeout period (*Do not assume an interactive user*), the system proceeds with servicing the miss. If such a model can predict the user's willingness to wait with reasonable confidence, it could reduce the frequency with which it requests advice during weak connectivity, thereby reducing the burden of translucent caching and balancing its costs with its benefits.

Although research in the area of response time delays (e.g., Thadhani [1981], Barber and Lucas [1983], Lambert [1984], Martin and Corl [1986], Teal and

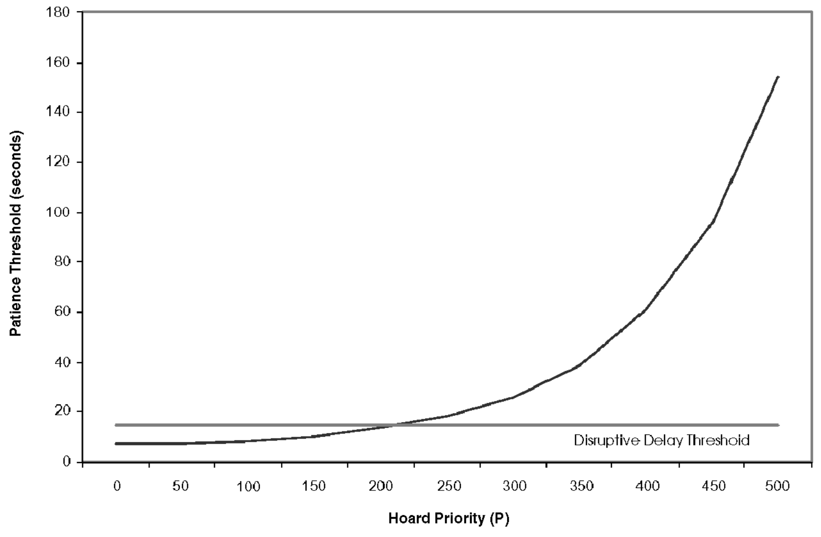
Rudnicky [1992], and O'Donnell and Draper [1996]) would seem to be relevant to this problem; in fact it is not. These studies typically examine the effect of response times of all commands issued by the user and involve short delays (typically under 3 seconds). In contrast, cache misses should be an infrequent occurrence in Coda (or it would be unusable in a disconnected environment), but they may be quite lengthy (proportional to the size of the missing file) and unpredictable (if the network is intermittent). Thus we are left with the difficulty of determining the threshold above which a delay due to a cache miss should be brought to the user's attention. Shneiderman [1998], in his well-known textbook, states that delays beyond approximately 15 seconds are clearly disruptive, but this result does not take into account the importance of the file. Johnson's [1995] work in applying the concept of marginal utility to the problem of retrieval delays on the Web seemed particularly promising until we realized that the utility of a Coda file is often known whereas the utility of a Web search engine hit is frequently unknown. We therefore define an initial model of user patience.

Our initial model of user patience is based on the conjecture that patience is similar to other human processes such as vision and hearing, whose sensitivity is logarithmic [Cornsweet 1970]. Our model accounts for the time necessary for the user to respond to the advice request (through the use of a keystroke-level model, a form of GOMS analysis [Card et al. 1983]). Because we hope future research will offer more accurate models, we encapsulated our model into a single module to allow our simple model to be replaced easily with a more advanced one. Designing and validating a user patience model is a challenging and open-ended problem.

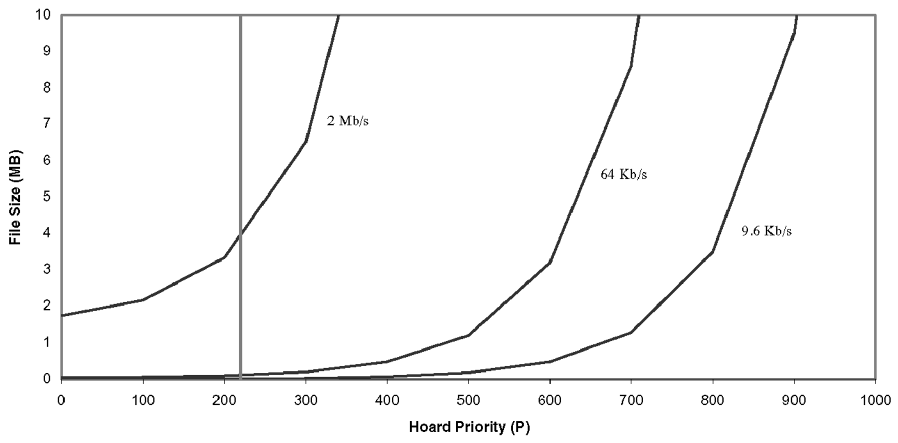
Figure 3 illustrates two different views of our user patience model. In View (a), we show the amount of time τ a user would be willing to wait for files of various hoard priorities according to our model. For illustrative purposes, we show only priorities up to 500, although they may actually be as high as 1000. (Note that, with the CodaConsole, users never see hoard priorities.) To make the model more concrete, View (b) shows τ expressed in terms of the size of the largest file that can be fetched at a given bandwidth. For example, 60 seconds at a bandwidth of 64 Kb/s yields a maximum file size of 480 KB. Each curve in this figure shows τ as a function of P for a given bandwidth. For both views, in the region below the curves, cache misses are transparently handled and preapproval is granted automatically during hoard walks. In the region above this curve, the system pops up a dialog box asking the user if she wants the system to fetch the file, but approval is assumed if the user does not respond.

The curves shown in Figure 3 predict that users would be willing to wait indefinitely for sufficiently important files. Clearly, this is not true. Recalling Shneiderman's statement about delays, we decided to present any delay greater than 15 seconds to the user. This maximum delay threshold is also shown in the figures. Of course, this value is a parameter and easily changed as we learn more about user patience and as we develop better models. Thus the function employed by the cache manager is

$$\tau = \min(15, \alpha + \beta e^{\gamma P}).$$



(a)



(b)

Fig. 3. User patience model. In both views, the hoard priority is shown on the X-axis. (a) Maximum hoard priority shown is 500 and the Y-axis shows the number of seconds the model predicts the user would be willing to wait. Any files that can be fetched in fewer seconds would automatically be fetched. (b) Maximum hoard priority is 1000 and the Y-axis shows the maximum size file the model predicts the system should automatically fetch. Each curve in this view represents a different network bandwidth. The area below the curve represents those objects that would be automatically fetched; the area above the curve represents those that require user intervention. (Parameter values for both views were $\alpha = 6$, $\beta = 1$, $\gamma = 0.01$.)

Asking users for permission before fetching a file differs from the standard HCI technique of giving users a button to cancel an operation if they decide it is taking too long. Although the more typical approach is appropriate in many, if not most, situations, it is not appropriate in all situations. For example, if a user is charged for network usage on a per-packet basis, the user may want to approve network usage before costs are incurred. Similarly, military scenarios may dictate that ground personnel behind enemy lines approve all communication. Our approach explores a different part of the design space and does not preclude other approaches.

4.7 Hoard Walk

During strongly and weakly connected operation, the system periodically performs a hoard walk. Hoard walks ensure that the current cache contents match the current hoarding requirements. For example, a request to view a large (but unhoarded) file could displace a hoarded file. The hoard walk indicator turns yellow to show that a hoard walk is in progress, but turns red to show that a hoard walk has stalled pending user advice. The advice indicator would also turn red to show the impending advice request. The hoard walk advice request allows users to control how much data actually get fetched during a hoard walk. As with all advice requests, the request will timeout after a period of time, allowing the hoard walk to continue in the absence of an interactive user.

4.8 Reintegration

After a period of disconnected operation, the Coda client must perform a reintegration. This process updates any changes made locally with the data stored on the servers. The indicator turns yellow when a reintegration is in progress. It turns red when a reintegration is pending but cannot proceed without the user's tokens (in this case, the tokens indicator would also turn red). This indicator light addresses certain usability problems that we observed (Sections 2.1 and 3.1). The first occurred when users failed to authenticate after a period of disconnection, preventing their updates from being reintegrated as was expected. The second occurred when a weakly connected client delayed updates, storing them locally pending reintegration. Although this was the proper behavior, when it occurred while the user was connected on an overloaded LAN, the user might have noticed the updates missing on the servers and become concerned. By indicating that the client is operating weakly connected and that a reintegration is pending, users are better able to understand the system's behavior. The design of the interface [Ebling 1998] also specifies how the user could be given control over the extent of reintegration as discussed in Section 3.1, but this aspect was never implemented.

4.9 Repair

Coda's use of optimistic replication[6] implies that occasionally file updates will conflict with each other. When this happens, the file is said to be inconsistent. Sometimes the system is able to resolve these conflicts automatically, but

[6]Optimistic replication is a term used to describe a number of replication techniques. Each of these techniques trades the possibility of inconsistencies for increased availability by allowing the possibility that updates could be made concurrently, resulting in conflicts that could require manual intervention to resolve. Optimistic replication is commonly used in mobile computing, where more pessimistic alternatives are perceived as sacrificing too much availability.

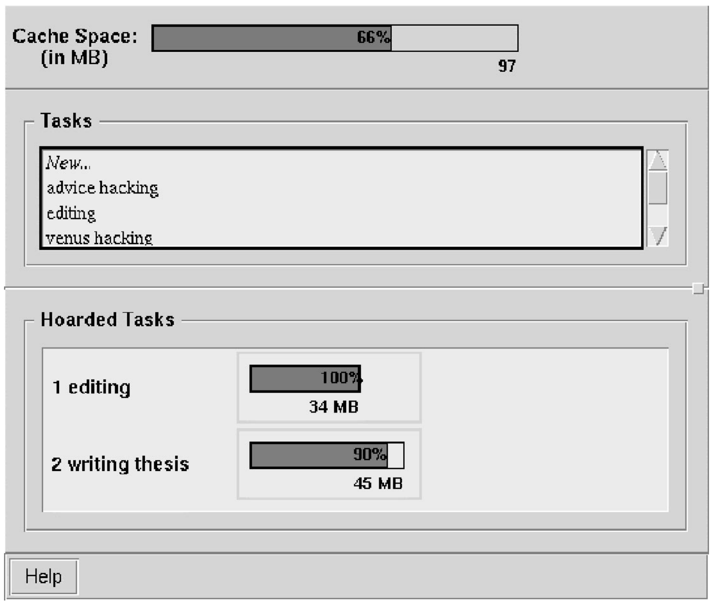


Fig. 4. This task information window might appear if the user were to doubleclick on the task indicator shown in Figure 1. The top section of this window shows the current state of the cache space. The middle section presents a list of all tasks the user has defined. The bottom section shows which tasks the user has hoarded. For each hoarded task, the window shows the current priority (1 being most important) as well as its current availability. Note that task priorities are automatically converted to hoard priorities used internally by the cache manager and in the user patience model discussed previously. The availability of a task is presented in the form of a gauge showing the percentage available; the color of the gauge indicates if it is currently available (green) or not (red).

occasionally the system requires the user to manually repair these conflicts [Kumar 1994]. When such a repair is required, the user is alerted using the repair indicator light (e.g., it turns red). The design of the interface [Ebling 1998] specifies how Kumar's repair tool could be incorporated into the CodaConsole, but this aspect was never implemented.

4.10 Task

The CodaConsole allows users to define and hoard tasks. A task is a group of related files and directories that the user needs to perform some unit of useful work. For example, to write a paper, the user might need the directory containing the paper as well as programs associated with the editor and formatter he plans to use.

The task indicator gives users feedback about what tasks they can perform locally. If the task indicator is green, then all the user's hoarded tasks are available locally. If it is red, at least one is unavailable. If the user were to doubleclick on this indicator, the window shown in Figure 4 might appear. This window shows a list of all tasks the user has defined as well as a list of hoarded tasks with meters that indicate availability.

Table I. Indicator Functional Overview

Indicator	Communicates State	Offers Control
Tokens	×	
Space	×	
Network	×	
Advice	×	×
Hoard Walk	×	×
Reintegration	×	× (future)
Repair	×	× (future)
Task	×	×

The task indicator also acts as the entry to managing the defined tasks, including adding new tasks as well as modifying and deleting existing ones. Doubleclicking on a task name (or on the name *New*) brings up another series of windows that allow the user to modify (or define) a task by typing in the names of the necessary component data (files and directories) and programs. In addition, tasks can include previously defined tasks that the user can select from a list.

The use of tasks addresses another usability problem observed in the original Coda system (Section 2.1), the practice disconnection. By allowing users to define tasks and then showing the state of those tasks, the CodaConsole obviates the need for this time-consuming process. Tasks also serve to raise the interface to a level more meaningful to users. Now, rather than dealing with individual files and directories, users communicate their needs in terms of tasks.

Although defining tasks represents a slight cost to the user, it delivers great benefits (*Balance costs with benefits*). Many existing Coda users already organize their hoard profiles[7] by task. By making this organization explicit in the process, users can see what tasks are available for use. Explicit support for tasks removes the need for users to practice disconnections, thus reducing the burden placed on users.

4.11 Summary

In Section 3.1, we discussed the system behaviors and errors that need to be brought to the user's attention and the aspects of the system over which users need to have control. This section has provided an overview of the CodaConsole interface, illustrating how it brings information to the user's attention and offers the user control over the system. Table I summarizes the purpose of each indicator light and indicates whether each indicator communicates the state to the user or offers the user control over the system's behavior (or both).

5. EVALUATION

Now that we have described the basic features of the interface, we turn our attention to the evaluation. Our evaluation of the CodaConsole had both

[7]A *hoard profile* is a file containing a list of file and directory names. Users utilize these profiles to organize the data they use frequently.

summative and formative goals. Our summative goals were intended to determine whether we met our design objectives. Our formative goals were used to determine ways in which we could improve the interface.

The first objective of the test was to gauge whether the interface was meeting our learning and performance goals, as described in Section 3. In particular, the test focused on three major issues: the ease of mastering the interface, users' ability to understand the feedback provided, and their ability to operate the interface. Although our goals were not formalized in a written requirement, our objective was a rough "80/20 rule," where even novice Coda users could use Coda with an accuracy of at least 80% with only 20% of the previous effort. The original Coda interface was never formally benchmarked because training was by apprenticeship with new users asking experienced users for help, often over a period of weeks or months. Therefore, we stated our objectives for a think-aloud usability test as follows, and operationalize these further in Section 5.1.4.

Ease of mastering the interface:

—Can all users complete a tutorial in approximately one hour and still maintain 80% accuracy on the exercises?

Ability to understand feedback:

—Are all users able to understand the information provided by the interface with an accuracy of 80%?

Ability to operate:

—Are all users able to operate the interface with an accuracy of 80%?

In answering these questions, our goal was to obtain an indication of how well people with backgrounds similar to those of Coda users would be able to perform activities similar to those required of Coda users. Providing conclusive answers to these questions would obviously require a study of actual Coda users interacting with the production-quality interface and is beyond the scope of this work.

The second objective of our usability test was to identify areas of the interface that caused confusion. Our goal was to create a list of usability problems, categorized by their scope and severity [Dumas and Redish 1993]. We identified these problems through observation and analysis.

5.1 Method

To answer these questions, we performed a think-aloud usability test in which we observed a total of fourteen people interacting with the interface. The first five participants were pilot users. The interface, materials, and/or procedures used for these users differ somewhat from those used for the remaining nine participants. We used these five pilot users to identify problems in our test procedures and materials; in addition, we discovered a few problems in the interface that we corrected prior to the start of the test. The remaining nine participants all used the same interface, materials, and procedures. In this section, we discuss the demographics of the participants, the procedure used during the test, and the measurements we collected.

5.1.1 *Participants.* The users for our test were recruited from among graduate students in computer science at Carnegie Mellon. Although this user population has more experience with computers than the general public, this population experienced problems during the original Coda deployment. Unless we can make Coda easily accessible to this population, broader deployments are unrealistic. Because Coda ran only under UNIX-based operating systems at the time, all participants were required to have substantial UNIX experience. All participants were further required to have substantial experience with the Andrew File System [Howard et al. 1988], a broadly deployed predecessor to Coda.

We defined a *novice* user as a person who knew little about Coda and had no direct experience with it. We defined an *experienced* user as one who had a Coda laptop and had operated disconnected for substantial periods of time over the course of at least a year.

We observed a total of fourteen users: five pilot users, three novices (N1, N2, and N4), three experienced users (E1, E2, and E3), and three who could not be classified as either an experienced or novice user.[8] The quantitative data collected from the three novice Coda users and the three experienced Coda users were used in our summative evaluation and are presented here. However, because qualitative data are useful regardless of origin, all such data were used in determining our usability findings.

5.1.2 *Procedure.* After obtaining appropriate permission, the experimenter asked the participant to fill out a brief background questionnaire, providing some basic demographic information. Once that was completed, the experimenter gave instructions and demonstrated a think-aloud protocol using a computer game (Klondike). The participant then played Othello briefly to practice thinking aloud. After a few minutes of practice, the participant began a detailed interactive tutorial of the CodaConsole interface. This tutorial was expected to require approximately one hour to complete. After a break, the participant performed a number of exercises ranging from identifying problems to preparing for an extended disconnected session. Users were expected to spend approximately one hour completing the exercises, after which the participant was to answer a brief survey evaluating the interface. Finally, the experimenter debriefed the participant, allowing the participant the opportunity to ask questions. All materials used in this test can be found in Ebling [1998].

5.1.3 *Exercise Tasks.* The exercise segment consisted of 24 individual exercises typical of those required to use the existing deployment of Coda or expected to be required of CodaConsole users. They were proposed by the first author and discussed with two other experienced Coda users. Each exercise

[8]These three unclassified users knew too much about the concepts of Coda to be novices, but had no experience using it and therefore could not be classified as experienced. Two of these users needed to learn more about Coda and were willing to give us feedback on the system and tutorial. The third was originally classified as a novice, but knew more about Coda than he thought (probably because he was the officemate of a Coda user).

consisted of one or more questions. The answer (or response) to each question took one of a number of forms, including a short written answer, a change to a configuration file, or a change to an interface object (such as a task definition). Each exercise fell into one of three categories that reflected important aspects of using Coda and/or the CodaConsole.

The first type of exercise asked participants to use the CodaConsole to answer basic questions about the state of the system and to perform basic operations with the interface. One question asked participants to determine which tasks were defined, which ones were hoarded and their priorities, and which tasks were available. Another question asked participants to describe the configuration of an event and then change that configuration slightly. These exercises allowed us to gauge each participant's ability to understand and operate the interface. These exercises covered nearly the entire interface. They used all the features described in Section 4.1 except repair and reintegration because they were not available.

The second type of exercise asked participants to define and prioritize several tasks. The first was a debugging task; the second was for hoarding generally useful C header files; and the third was for working on a curriculum vitae. When the participants prioritized their defined tasks, they were also asked to describe the reasoning behind their chosen priorities. These exercises allowed us to gauge each participant's ability to define and hoard realistic tasks. Each of the specific tasks chosen for the exercises was similar to one that the first author had defined under the previous Coda interface and representative of those defined by other experienced Coda users. None of the participants who were experienced Coda users commented that they felt the tasks were too easy or misrepresented the difficulties involved in defining real tasks.

The third type of exercise asked participants to edit a file while reacting to problems indicated by the interface. Approximately every minute, the support code sent an event notification to the interface. The interface then indicated the event to the user. The participant's job was to fill out an event report whenever she noticed a problem had occurred. These exercises allowed us to gauge each participant's ability to understand the feedback provided by the interface as well as whether the interface was able to catch his peripheral attention. Each of the events indicated to the user represented an event that actual Coda users experience and together they covered nearly all the events supported by the CodaConsole interface. The few remaining events were similar to other events. All the supported events known to have caused confusion in the past were indicated during one of these exercises.

Together, the exercises covered nearly all the functionality supported by the CodaConsole. Furthermore, wherever appropriate, the exercises were based upon actual use of the original Coda deployment. The least realistic aspect of the exercises was the frequency with which the events were indicated to the participants; if Coda indicated events at a frequency of one minute in practice, users would quickly stop using the system so that they could actually get work done.

Table II. Summary of Learnability Results

	Learnability (Time on Task, in Minutes)	
User ID	Tutorial	Exercises
N1	61	62
N2	63	62
N4	42	66
E1	59	70
E2	59	56
E3	43	62
Avg (StdDev)	54 (±9)	63 (±4)

5.1.4 *Measurements.* The first measurement we collected was the time (in seconds) required to teach users about the details of the CodaConsole interface. We measured these times for each screen of the online tutorial.

The second measurement we collected was the time (in seconds) required for the user to complete each exercise. We measured the time for each complete exercise, not at the granularity of individual questions within an exercise.

The third measurement addressed the accuracy of the user's response to each exercise, which we recorded. A response could be a written answer to a question or a change to a configuration file. Prior to the start of the test, the first author created a key for the exercises. After the test, each participant's responses were compared to this key according to strict scoring criteria. Each correct response scored a point.

The final measurement addressed the efficiency of the user's response to each exercise. We counted the number of steps taken to complete each correctly answered exercise. A step corresponded to doubleclicking an indicator light, clicking a button, or typing a pathname, as well as similar activities. The baseline measurement, which we call *par* (as in the game of golf), was defined as the number of steps required of an interface expert who had prior knowledge of both the questions and the expected answers.

All but the first participant were videotaped during their session. From transcripts of these tapes, we identified comments made by each user and categorized them based upon content.

5.2 Results

Overall, the usability test indicates that the CodaConsole interface met all its design goals. The average time required to complete the tutorial was 54 minutes. In approximately one hour, participants were able to understand and operate the interface with an average accuracy of over 90%. Furthermore, the novice Coda users were able to use the system almost as well as the experienced Coda users. Tables II, III, and IV provide summaries of the quantitative data collected from our users. We should note, however, that despite the participants' highly accurate performance, the usability test also suggested numerous areas in which the interface could be improved. In this section, we discuss the original objectives of the test, examining whether users were able

Table III. Summary of Understandability Results

	Understandability	
User ID	Accuracy (% Correct)	Efficiency (Score/Par)
N1	90	1.7
N2	98	1.7
N4	98	2.0
E1	97	1.8
E2	98	1.4
E3	98	2.0
Avg (StdDev)	97 (±3)	1.8 (±0.2)

Table IV. Summary of Operability Results

	Operability	
User ID	Accuracy (% Correct)	Efficiency (Score/Par)
N1	90	1.5
N2	94	1.8
N4	84	1.9
E1	97	2.4
E2	94	1.8
E3	90	2.9
Avg (StdDev)	92 (±4)	2.1 (±0.5)

to master, understand, and operate the interface. We also summarize some of the lessons we learned regarding the usability of mobile systems.

5.2.1 *Learnability.* The first set of exercise questions relates to the ease with which users learned to use the interface. Our first objective was to determine whether users could learn to use Coda and the interface in approximately one hour. Indeed, they could, as shown in Table II. The experienced and novice participants in the test required 55 (±9) and 54 (±8) minutes on average, respectively. The fastest user, N4, took just under 42 minutes; the slowest, N2, took just under 63 minutes. Furthermore, our other measures show that these users learned the skills necessary to understand and operate the interface.

The qualitative data collected as part of the questionnaire administered after completion of the exercises supports these conclusions. From these data, we learned that users felt the tutorial was basically complete. On a scale of 1 to 5 (where 5 was "very complete" and 1 was "very incomplete"), the tutorial received an average of 3.8. Users thought the tutorial was easy to understand (average of 4.2 out of 5, where 5 was "very easy" and 1 was "very difficult"). Users unanimously felt that the tutorial allowed them to grasp the scope of the interface. Finally, five out of six users felt it was presented at an appropriate level of detail. The sixth user, one with Coda experience, felt there was too much detail, specifically too much Coda background. Because the tutorial was written for a novice audience, this is not surprising.

Inasmuch as the tutorial was intended for use in future Coda training, we also wanted to identify the segments of the tutorial that participants found difficult to understand. The basic idea behind our analysis was to examine how much time users spent understanding each screen of the tutorial relative to other screens. If a screen required an unexpectedly large amount of time, then we wanted to examine it to see if it could be made more clear. We had to account for some complicating factors. First, each screen of the tutorial required users to perform different actions with a mouse and/or keyboard. We addressed this problem by estimating the amount of time needed to perform the actions required by a given screen and then subtracting this time from the total time spent on that screen. Second, users read at different rates. We addressed this issue by estimating each user's reading rate and then comparing the user's reading rate on each screen with her average reading rate over all screens.[9]

Based upon the quantitative and qualitative data we collected, the tutorial achieved its objectives of teaching users about Coda and the interface in about an hour. Furthermore, no serious difficulties solely attributable to the tutorial were uncovered.

5.2.2 *Understandability.* A subset of the exercises explored the users' ability to understand the information presented by the interface. Eighteen of the twenty-four exercises address this issue. These exercises include those that asked users to describe the current network connectivity, to identify which tasks are currently hoarded, and similar questions as well as the exercises that asked users to explain events notified via the indicator lights. Isolating the users' scores on these exercises as shown in Table III, we find that users did extremely well. Five out of six users scored 97% correct or better, and the sixth user scored 90% correct.

We examined the 12 event notification exercises in more detail. By doing this, we were able to explore the users' ability to understand the feedback provided during event notifications. For each event, users were asked to fill out an *incident report*, consisting of the urgency of the event, a description of the problem, and a description of any actions that might resolve the problem. Five out of six users explained each of the 12 incidents. The sixth user explained 11 of the 12—this user failed to report an event that transitioned from a "bad" state to a "good" one. Of the 71 incident reports submitted by these six users, only three contained incorrect information. Users clearly understood the problems indicated and what, if any, steps were possible to resolve those problems, silencing any concerns over whether users could understand the feedback provided during event notifications.

We also examined the remaining six exercises in more detail. We found that, with the exception of just one of these, five out of six users answered accurately. The remaining exercise appears to have been poorly worded because, although three users performed one or more parts of this exercise incorrectly, they all did so differently. Our reasoning was that if the interface had been leading people astray, we would have expected to see participants making similar errors, but if the question were unclear or ambiguous, then we would have expected them to interpret the question differently and, consequently, respond with different answers. In fact, when we examined the participants' answers in detail, this is

[9]The reading rate portion of this technique was inspired by a similar, but undocumented, analysis done by Frank Lee [1999] as part of his thesis research.

the behavior we found.[10] We conclude that the incorrect answers were related to the way we asked the question and not to the interface.

5.2.3 *Operability.* The third set of questions related to the users' ability to operate the interface: in particular, whether users would be able to define and hoard tasks, provide advice to the system, and configure events. Eight exercises addressed this issue. These exercises included three that asked users to configure events and provide advice as well as all the exercises that asked users to define and prioritize three tasks. As shown in Table IV, five out of six users scored 90% or better, and the sixth user scored 84%. Furthermore, the detailed data showed that, for six out of these eight exercises, no two users answered the same exercise incorrectly. Although these scores are very encouraging, users clearly had more difficulty operating the interface than they did understanding the feedback it provided.

The most disturbing trend we observed was the fact that five out of six users failed to hoard at least one program needed to complete a given task. Each of these five participants hoarded a compressed file, but failed to hoard the decompression utility needed to read that file. Two of them also failed to hoard editors as part of a task that required modifying a file. This observation highlights the need for additional help from the system in identifying programs that are required to complete a task, but that are not salient to the user. This lesson is one that will likely have parallels in other systems. For example, users might also forget the plugins that support their Web browsers or word processors.

A different metric, which also gauges the users' ability to operate the interface, compares the number of actions users needed to perform each exercise they answered correctly with the number required by the interface designer (par). The data shown in Table IV include this metric, called *efficiency*. In general, users needed approximately twice as many actions as defined by par. They needed a bit more than twice as many when they were required to operate the interface and a bit less than twice as many when they simply had to understand the interface. This trend is not surprising because par did not include any problem-solving or information search, being defined on an expert who knew the questions and answers prior to task execution. Participants confronted with these questions would naturally engage in problem-solving and information search to answer them, both processes that are known to be inefficient.

[10]The first question of this exercise was, "Please describe the current space status." Two users appear to have misinterpreted this question; they answered that the cache was two-thirds full. Although this was a true statement, it did not answer the question completely; the predefined answer in our key was "All space areas are within normal parameters." The second question of this exercise was, "How much disk space is dedicated to the Coda cache?" Two users also answered this question incorrectly. We were looking for an absolute measurement: "97 MB." One of these users answered with the percentage of the cache that was currently occupied. Although this answer was, in some sense, more correct, according to our strict scoring criteria, it was incorrect. The other user gave the size of a different component. A review of the verbal protocol confirms that at least one of these users was confused by the interface, saying "I'm not sure if that's what's dedicated or what's used."

5.2.4 *Summary.* Because the detailed examination of user data focuses on negative results, we state again: our design met or exceeded all of its user performance goals. All users completed the tutorial in about an hour and achieved greater than 90% accuracy in understanding the interface and greater than 80% accuracy in operating it. Furthermore, the test results show that the interface allowed these novice Coda users to employ Coda with approximately the proficiency of the experienced Coda users. These novices scored 90% or better on exercises typical of those required of Coda users. They successfully diagnosed problems and possible solutions, defined tasks, and provided advice to the system about hoard walks and demand fetch requests. The interface did not hinder experienced users; in fact, their responses indicated an excitement about the release of the interface. The interface clearly met its objective of helping novice users while not unduly hampering experienced users.

5.3 Usability Findings

Despite its obvious successes, like every interface, the CodaConsole could be improved. The test provided a wealth of information about how users interacted with the interface and where they encountered problems, even if they eventually overcame them. Combining all the data from this usability test (the quantitative analysis presented above along with an analysis of the verbal protocol and an evaluation questionnaire from all participants) resulted in 87 usability findings [Ebling 1998; Ebling and John 2000]. We classified each finding according to its scope and severity, as defined in Dumas and Redish [1993]. Due to space limitations, we cannot present each of these findings here.

Of the most severe findings, many can be easily remedied, but others are more troublesome. Some of the more interesting lessons we learned include the following.

—We observed that users were occasionally confused by event notifications. For example, at one point during the exercises, a task became unavailable. The *Task* indicator light blinked and beeped to alert the participants, confusing a number of them in the process. This confusion was not limited to the duration of the tutorial, so it is not simply a learning curve issue. This observation suggests the need for a mechanism through which users can be given a written explanation of the most recent events, a sort of log or history of recently alerted events. This lesson would apply equally well to other applications of indicator lights, not just Coda or mobile systems.

—As described earlier, we observed that five out of six participants failed to include necessary programs (e.g., a decompression utility) as part of their task definitions. We also found that, even if users defined the need for a program, they occasionally forgot to actually hoard it at an appropriate level. For example, two users failed to include their editing task as a subtask of the debugging task or even as a high priority top-level task. This observation suggests the need to help users identify required programs. Again, this lesson applies beyond Coda to any system supporting programs running disconnected from their server.

—We observed that one user did not recognize that the hoard walk advice request offered details in a hierarchical tree control widget. This observation is particularly troublesome given the recent popularity of these widgets. This problem might be solved with more explicit training, but there may well be underlying interface issues here. Further study of these widgets would be required to differentiate these two possible conclusions.

As indicated, each of these lessons applies beyond Coda.

6. CONCLUSIONS

The usability test we performed on the CodaConsole interface resulted in highly positive feedback. The three users we tested, who had no previous Coda experience, were using Coda and the interface with nearly the proficiency of the three experienced Coda users, after just one hour of training. The strategy of making caching translucent to users was successful in this instance and may help designers make other systems supporting mobility more accessible to users. Not surprisingly, our test also provided a wealth of information that can be used to improve the interface.

Although this research was undertaken to address the needs of the Coda user community, we learned a number of lessons that apply more broadly to mobile systems. Systems that support mobile computing encounter a wider range of operating (and failure) conditions than systems that do not. Users of these systems need more information regarding the current state of the system. They also need more control over the ways in which precious resources, such as network bandwidth and power, get used. This study lends credence to Dix's approach of making users aware of the state of network connectivity. In addition, a metalesson for systems designers to take from our work is that careful analysis of user needs, diligent attention to design principles, and thorough user testing do indeed produce a learnable and usable interface.

ACKNOWLEDGMENTS

We would like to thank the reviewers and editors for their many helpful comments and suggestions. We would also like to thank the people who have been involved with Coda through the years, including the many developers and users. Your insights and feedback have been invaluable to this research. The people who volunteered to participate in this user study deserve special thanks for donating their valuable time for the benefit of our research. We would also like to thank the administrators of Carnegie Mellon University's User Studies Lab for the use of their facilities, especially Sandy Esch for her dedication to making the lab run smoothly. We also thank IBM for allowing the first author the time necessary to complete this publication of her PhD research. Windows is a registered trademark of the Microsoft Corporation in the USA and/or other countries.

REFERENCES

ACMDL 2002. The importance of translucence in mobile computing systems: Coda overview. Available at http://portal.acm.org/tochi/archive/.

Alonso, R., Barbará, D., and Cova, L. 1990. Using stashing to increase node autonomy in distributed file systems. In *Proceedings of the Ninth IEEE Symposium on Reliable Distributed Systems* (Huntsville, Ala., Oct.), IEEE Computer Society, Los Alamitos, Calif., 12–21.

Barber, R. E. and Lucas, H. C., Jr. 1983. System response time operator productivity, and job satisfaction. *Commun. ACM 26*, 11, 973–986.

Card, S., Moran, T., and Newell, A. 1983. *The Psychology of Human-Computer Interaction*. Lawrence, Erlbaum, Hillsdale, N.J.

Cheverst, K., Blair, G., Davies, N., and Friday, A. 1999. Supporting collaboration in mobile-aware groupware. *Pers. Technol. 3*, 1.

Cooper, A. 1995. *About Face*. IDG, Braintree, Mass.

Cornsweet, T. 1970. *Visual Perception*. Academic, New York.

Davies, N., Blair, G., Friday, A., Raven, P., and Cross, A. 1996. Mobile open systems technology for the utilities industries. In *Remote Cooperation: CSCW Issues for Mobile and Teleworkers*, A. J. Dix and R. Beale, Eds., Springer, New York, 145–166.

Dix, A. 1995. Cooperation without (reliable) communication: Interfaces for mobile applications: *Distrib. Syst. Eng. J. 2*, 3, 171–181.

Dix, A. and Beale, R. 1996. Information requirements of distributed workers. In *Remote Cooperation: CSCW Issues for Mobile and Teleworkers*, A. J. Dix and R. Beale, Eds., Springer, New York, 113–144.

Dumas, J. and Redish, J. 1993. A *Practical Guide to Usability Testing*. Ablex, Norwood, N.J.

Ebling, M. R. 1998. Translucent cache management for mobile computing. PhD Dissertation, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Penn.

Ebling, M. R. and John, B. E. 2000. On the contributions of different empirical data in usability testing. In *Proceedings of Designing Interactive Systems (DIS2000)* (New York, August), ACM, New York.

Friday, A., Davies, N., Blair, G. S., Cheverst, K. W. J. 1999. Developing adaptive applications: The MOST experience. *J. Integ. Comput. Aided Eng. 6*, 2, 143–157.

Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and West, M. 1988. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst. 6*, 1, 51–81.

Huston, L. and Honeyman, P. 1993. Disconnected operation for AFS. In *Proceedings of the USENIX Mobile and Location-Independent Computing Symposium* (Cambridge, Mass., August), USENIX Association, Berkeley, Calif., 1–10.

Huston, L. and Honeyman, P. 1995. Partially connected operation. In *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing* (Ann Arbor, Mich., April), USENIX Association, Berkeley, Calif., 91–97.

Johnson, C. 1995. The impact of retrieval delays on the value of distributed information. Available at http://www.dcs.gla.ac.uk/~johnson/papers/value.html.

Kistler, J. and Satyanarayanan, M. 1992. Disconnected operation in the Coda file system. *ACM Trans. Comput. Syst. 10*, 1, 3–25.

Kuenning, G. and Popek, G. 1997. Automated hoarding for mobile computers. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles* (Saint-Malo, France, Oct.), 264–275.

Kumar, P. 1994. Mitigating the effects of optimistic replication in a distributed file system. PhD Dissertation, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Penn.

Lambert, G. N. 1984. A comparative study of system response time on program developer productivity. *IBM Syst. J. 23*, 1, 36–43.

Lee, F. 1999. Does learning of a complex task have to be complex? PhD Dissertation, Carnegie Mellon University, Pittsburgh, Penn.

Martin, G. L. and Corl, K. G. 1986. System response time effects on user productivity. *Behav. Inf. Technol. 5*, 1, 3–13.

Monk, A. 1986. Mode errors: A user-centred analysis and some preventative measures using keying-contingent sound. *Int. J. Man-Mach. Stud. 24*, 4, 313–327.

Mummert, L., Ebling, M., and Satyanarayanan, M. 1995. Exploiting weak connectivity for mobile file access. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (Copper Mountain Resort, Colo., Dec.), 143–155.

Nielsen, J. 1993. *Usability Engineering*. AP Professional, Boston.

O'Donnell and Draper. 1996. Temporal aspects of usability: How machine delays change user strategies. *SIGCHI Bull*. *28*, 2.

Pascoe, J., Morse, D., and Ryan, N. 1998a. Developing personal technology for the field. *Pers. Technol. 2*, 1, 28–36.

Pascoe, J., Ryan, N., and Morse, D. 1998b. Human-computer-giraffe interaction: HCI in the field. In *Proceedings of the First Workshop on Human Computer Interaction with Mobile Devices* (Glasgow, UK, May). Available at: http://www.dcs.gla.ac.uk/~johnson/papers/mobile/HCIMD1.html.

Rodden, T., Chervest, K., Davies, N., and Dix, A. 1998. Exploiting context in HCI design for mobile systems. In *Proceedings of the First Workshop on Human Computer Interaction with Mobile Devices* (Glasgow, UK, May). Available at: http://www.dcs.gla.ac.uk/~johnson/papers/mobile/HCIMD1.html.

Satyanarayanan, M., Kistler, J. J., Kumar, P., Okasaki, M. E., Siegel, E. H., and Steere, D. C. 1990. Coda: A highly available file system for a distributed workstation environment. *IEEE Trans. Comput. 39*, 4, 447–459.

Satyanarayanan, M. 1996. Fundamental challenges in mobile computing. In *Proceedings of the Symposium on Principles of Distributed Computing* (Philadelphia, May) 1–7.

Shneiderman, B. 1998. *Designing the User Interface*: *Strategies for Effective Human-Computer Interaction*, Third ed., Addison-Wesley Longman, Reading, Mass.

Steiner, J. G., Neuman, C., and Schiller, J. I. 1988. Kerberos: An authentication service for open network systems. *USENIX Conference Proceedings* (Dallas, Winter), 191–202.

Tait, C., Lei, H., Acharya, S., and Chang, H. 1995. Intelligent file hoarding for mobile computers. In *Proceedings of the First Annual International Conference on Mobile Computing and Networking* (*MobiCom '95*) (Berkeley, Calif., Nov.), 119–125.

Teal, S. L. and Rudnicky, A. I. 1992. A performance model of system delay and user strategy selection. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems* (Monterey, Calif., Addison-Wesley, Reading, Mass., 295–305.

Tesler, L. 1981. The Smalltalk environment. *Byte 6*, 8, 90–147.

Thadhani, A. J. 1981. Interactive user productivity. *IBM Syst. J. 20*, 4, 407–423.

Walker, W. F. 1998. Rapid prototyping of awareness services using a shared information server. *ACM SIGCHI Bull. 30*, 2, 95–101.

Wax, T. 1996. Red light, green light: Using peripheral awareness of availability to improve the timing of spontaneous communication. In *Proceedings Short Papers CSCW '96*.

Received May 1999; revised July 2001; accepted October 2001