This paper presents a Tcl/Tk recording/playback architecture and implementation that records, plays back and executes a Tcl/Tk collaborative Internet-based desktop. Specifically, the desktop brings together distributed data, application workflows, and teams into collaborative sessions in which the control of the desktop editing and execution is shared. A typical workflow invokes distributed tools and data to support the design of microelectronic systems.

We argue that recording and playback of collaborative user interactions can have a wide-range of applications, such as: 'keeping minutes' of interactive discussions, clicks of menu-specific commands associated with different tools on the shared desktop, user-entered data and control inputs, user-queried data outputs, support for automated software documentation, tutorials, collaborative playback of tutorials and solutions recorded earlier, etc.

The summary of 540 Internet-based experiments, each relying on RecordTaker and PlaybackMaker to record, playback, and execute ReubenDesktop configurations from local, cross-state, and cross-country servers, demonstrates the effectiveness of the proposed concepts and implementation.
Recording and Playback of Collaborative Desktops on the Internet

Amit Khetawat  Hemang Lavana  Franc Br̆glez

Technical Report 1997-TR@CBL-06-Khetawat
October 1997
© 1997 CBL
All Rights Reserved

"Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of CBL. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee."

If you choose to cite this report, please add the following entry to your bibliography database:

@techreport{
1997-TR@CBL-06-Khetawat,
author = "A. Khetawat and H. Lavana and F. Br̆glez",
title = "Recording and Playback of Collaborative Desktops on the Internet",
institution = "CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695",
class = "technical report",
month = "Oct",
year = "1997",
ote = "Also available at http://www.cbl.ncsu.edu/publications"
}

To contact the Collaborative Benchmarking Laboratory via Internet, you may consider:

WWW : http://www.cbl.ncsu.edu/
For an auto-reply: benchmarks@cbl.ncsu.edu
To deposit a file at CBL: ftp cbl.ncsu.edu
cd /pub/Incoming
put new_benchmark.tar.Z
Abstract - This paper presents a Tcl/Tk recording/playback architecture and implementation that records, plays back and executes a Tcl/Tk collaborative Internet-based desktop. Specifically, the desktop brings together distributed data, application workflows, and teams into collaborative sessions in which the control of the desktop editing and execution is shared. A typical workflow invokes distributed tools and data to support the design of microelectronic systems.

We argue that recording and playback of collaborative user interactions can have a wide-range of applications, such as: 'keeping minutes' of interactive discussions, clicks of menu-specific commands associated with different tools on the shared desktop, user-entered data and control inputs, user-queried data outputs, support for automated software documentation, tutorials, collaborative playback of tutorials and solutions recorded earlier, etc.

The summary of 540 Internet-based experiments, each relying on RecordTaker and PlaybackMaker to record, playback, and execute ReubenDesktop configurations from local, cross-state, and cross-country servers, demonstrates the effectiveness of the proposed concepts and implementation.

Keywords: recording, playback, desktop, collaborative, workflow, Internet.

I. INTRODUCTION

The Internet and the on-going evolution of the world-wide web is expected to evolve into a network without technological, geographic or time barriers - a network over which partners, customers and employees can collaborate at any time, from anywhere, with anyone. Even before the emergence of the Internet, the design of microelectronic systems increasingly relied on globally distributed databases, tools, and design teams. The challenge of the Internet is how to make this process more user-friendly, efficient, and effective - at a cost that is transparent to end-users.

Customization, coordination, and repeated execution of a collaborative Internet-based desktop environment for a specific design project is a non-trivial task, especially for a complex project involving a large number of distributed data, tools, and team members. To support such efforts, we have developed two utilities: RecordTaker and PlaybackMaker. Since this work started before the advent of JAVA [1], the current prototypes are written in Tcl/Tk [2]. Both can record, playback, and execute the collaborative Internet-based ReubenDesktop environment described in [3, 4]. We argue that recording and playback of collaborative user interactions can be seen as 'keeping minutes', not only of the interactive discussions but also of the menu-specific commands associated with different tools on the shared desktop, of user-entered data inputs, and of user-queried data outputs. There are other benefits of recording, such as

1. support for automated software documentation and tutorials, capturing the dynamics of software interactions for playback and review at a later time;
2. study of activities and feedback on how teams actually collaborate, to improve the effectiveness and efficiency of collaborative environments;
3. remote assistance, by selecting and playing back effective solutions recorded earlier.

Today, the basic desktop environment of a computer display is largely determined by the windowing/operating system of the host, e.g. MacOS and WindowsNT. The Common Desktop Environment (CDE) that makes applications running on UNIX systems portable and easy to use is a relatively recent commercial development [5]. Alternatively, there is TkDesk [6], a public-domain desktop and file manager for Unix and X written in Tcl/Tk. Prototypes of environments that provide user-configurable GUI capabilities for collaborative Internet-based desktop computing, with data and applications distributed on different hosts, have been demonstrated only recently [3, 4, 7, 8].

Much of the research on issues addressed in this paper predates the challenges and opportunities that have arisen with the Internet. For example, an overview of research issues related to sharing applications is presented in [9, 10, 11]. Some of the existing systems which provide a recording mechanism include [12, 13, 14, 15]. In most of the systems listed above, the implementation has been done using X protocols [16, 17]. A notable exception is the TkReplay [12], which provides an extension to Tcl/Tk.

The paper is organized into the following sections: (2) background and motivation, to define a collaborative environment and illustrate collaborative remote assistance using playback; (3) recording and playback architecture; (4) recording and playback implementation; (5) summary of 540 Internet-based experiments, and (6) conclusions.

II. BACKGROUND AND MOTIVATION

The ReubenDesktop, described in this paper as recordable and executable upon playback, satisfies the following properties as a collaborative desktop environment [4, 7]:

P1: desktop is shared and multi-cast, so that each participant can observe desktop actions of the others;
P2: desktop supports a shared and segmented 'talk window', so each participant can type messages to all others in his/her own window segment;
P3: the shared and segmented ‘talk window’ supports a token passing mechanism, so that at any time, only a single user controls the desktop, but can pass the token to any other user when requested.

An example of a ReubenDesktop satisfying properties P1-P3 is shown in Figure 1(a). The instance of the particular desktop has been multi-cast by student Amit to his instructor Hemang with a request for on-line assistance. In the case shown, the desktop consists of two windows: (1) a sample workflow that is not executing, hence the problem, and (2) a FlowSynchronizer window that allows Amit and Hemang to ‘talk’ and describe the problem and a solution.

Here, instructor Hemang could have requested and received permission from Amit to edit the workflow and thus show a solution. Instead, Hemang remembers that earlier, he recorded a solution to a similar problem for another student. Subsequently, he decides to playback the pre-recorded solution, shown in Figure 1(b). By passing control to Amit (the respective FlowSynchronizer window is not shown), Amit can now study the solution by re-executing the PlaybackMaker.

It is clear that the paradigm described in this example applies to a number of situations, including design reviews, with high potential to reduce design errors or catch them early in the process, thereby significantly enhancing the productivity of the team effort.

III. ARCHITECTURE

Recording and playback essentially involves capturing all events that are generated during a session, and reproducing those events in exactly the same sequence as they were generated. Event is an occurrence of an interaction between the user and the windowing system. The windowing system constitutes the local display, the keyboard, and the mouse.

In order to distinguish between the events occurring during recording and playback, we categorize the events into two types:

Window events are generated by the windowing system during run time of an application, in response to the interaction of the user with the application.

Synthesized events are invoked internally by the application using Tcl/Tk commands and not in response to user input. The Tcl/Tk interpreter arranges for the synthesized event to be processed just as if it were a part of the user input from the window system.

Every event consists of at least one primitive component. It may also contain additional secondary components for details. Examples of primitive components, which occur when the user interacts with an application on the local windowing system include: ButtonPress, ButtonRelease, MouseMotion, KeyPress. The secondary component associated with each event describes details such as the x-y coordinates of the mouse on the screen, the key which was pressed, the mouse button number which was clicked, etc.

(a) Block diagram of recording session
(b) Block diagram of playback session

Recording Session Architecture. Figure 2(a) shows the block diagram for the recording session. During the recording mode, the Tcl/Tk code passes through a Recording Interpreter which records the user interactions with the application and generates the Run Time Trace Data. The recording session also provides a facility to segment the entire playback session into several frames. The user can also insert a description about each frame which will be replayed during the playback session.

Recording Interpreter Implementation. Tcl/Tk applications have an event-driven control flow, just as with most window system toolkits. An event is handled by associating a Tcl/Tk command to the event with the bind command. Each Tk widget has default bindings for some of the events which provides the basic functionality of that event with the widget, e.g. the event Enter inside a button widget highlights the button. Event bindings are structured into a simple hierarchy of global bindings, class bindings, and instance bindings. Tcl/Tk provides the default behavior of buttons as bindings on the Button class.

We introduce a new class called RecordClass, create new bindings for each event we want to record, and associate these bindings with the RecordClass. This RecordClass is attached to each widget of the application to be recorded. The attachment is done when the widget is created on the screen by using the bindtags command.

The Trace Data Structure, used to store the information
about the intercepted events, is implemented using Tcl/Tk’s associative arrays. This data format makes it easier to analyze and create commands which would replay those events.

We also store the timings for each event. Timing information associated with each event is very critical, and is useful for synchronizing the synthesized event during the playback session. Various terms related to a recording session are as follows:

- \( E_i \) The \( i^{th} \) event in a session.
- \( t_{ri} \) The time at which event \( E_i \) occurs during a recording session.
- \( t_{ri+1} - t_{ri} \) The time difference between the occurrence of the event \( E_{i+1} \) and the event \( E_i \).
- \( n \) The total number of events for a session.

Figure 3(a) shows a timing diagram illustrating the relationship between various events and their recording times. Figure 3(b) shows a part of the trace data, which is a list of events and their corresponding recording times.

### Playback Session Architecture

Figure 2(b) shows the block diagram for the playback session. During the playback mode, the Trace Data Processor reads the trace data and creates commands to synthesize the recorded events. These synthesized events are then scheduled by using event timings to create the playback session. The playback session can be controlled and tailored at the user’s convenience.

### Trace Data Processor Implementation

Tcl/Tk provides a command event generate to synthesize the recorded window events. The Trace Data Processor creates the synthesis commands for each of the recorded events with every detail about that particular event. The event generate command has the following format:

```plaintext
event generate window event [options]
```

The window is the widget in which the event is to be synthesized. The options are used to specify the details which are specific to each particular event. In addition to the basic event synthesis command, Trace Data Processor also creates the dynamic timing information for that event. This dynamic timing event allows the user to playback in a user-friendly manner. Some of the terminologies related to the playback session are as follows:

- \( t_{ri} \) The time at which event \( E_i \) will be played back.
- \( s \) Constant scale factor. This scaling factor remains constant for the entire playback session of all \( n \) events and is pre-computed at the start of a playback session.
- \( s_i \) The dynamic scaling factor for the \( i^{th} \) event. This scaling factor may change anytime during the playback session.

The two schemes we considered to implement the timing details are given in Figure 4. Both the schemes use the after command provided by Tcl/Tk to schedule an event at a later time. Figure 4(a) shows the static scheduling of events in which all the \( n \) events are scheduled at the start of a playback session. The time, for which the event \( E_i \) is scheduled to execute, is computed by multiplying \( t_{ri} \) with the constant scale factor \( s \). This approach has several limitations which include the inability to schedule events dynamically during the playback session. This limits the user’s ability to pause or vary execution speed between consecutive events.

This limitation can be overcome by using a dynamic approach, as depicted in Figure 4(b). In this approach, the event \( E_{i+1} \) is scheduled at the start of execution of event \( E_i \). The scaling factor used for scheduling event \( E_{i+1} \) is computed not at the start of playback session but at the start of execution of event \( E_i \). This gives the user flexibility to pause during playback, or dynamically scale down or scale up the playback speed. A comparison between the approaches is shown in Figure 4(c).

### IV. RECORDING AND PLAYBACK TOOLS

We use a simple application Print Hello button in Figure 5 to illustrate the main ideas used to implement the recording and playback mechanism.

The left side of the figure shows the trace data, and the right side of the figure shows the Tcl/Tk commands used for synthesis of the recorded events and the user views as each event is synthesized.

We now describe the steps illustrated in the Figure 5 to synthesize the events like Enter, ButtonPress, etc.

**Step 1.** Invoke the button application with the command pack [button .b -text "Print Hello]

**Step 2.** Synthesize the event ‘Enter’ in the window ‘.b’ with the command event generate .b <Enter>

**Step 3.** Synthesize the event ‘ButtonPress’ in the window ‘.b’ with the command event generate .b <ButtonPress> -button 1

The option -button 1 specifies the Mouse button 1.

**Recording and Playback Tools.** We have implemented a RecordTaker and a PlaybackMaker. These tools assist users to
recording a session. Thus a session may be broken up into sequence of events during the playback. We introduce the to each step. These descriptions may be needed to explain the number of steps. It also facilitates the addition of descriptions. The provides a facility to record a session in a RecordTaker back as described below. Figure 6 shows the GUI of create customized recordings and to provide convenient play- back as described below. Figure 6 shows the GUI of RecordTaker, which allows the users to customize their recordings. The RecordTaker provides a facility to record a session in a number of steps. It also facilitates the addition of descriptions to each step. These descriptions may be needed to explain the sequence of events during the playback. We introduce the concept of frames in this context. Each step is called a frame. The frame is essentially a breakpoint, which is inserted while recording a session. Thus a session may be broken up into several frames or it could be a single frame. Each frame itself constitutes several events. The RecordTaker interface consists of the following components:

File. This is a menu button, which allows the user to save the recordings, import a particular frame description file, and exit the recording mode.

Next Frame. This button inserts a marker for the current frame. The marker indicates the end of the current frame and the beginning of a new frame. This marker is used during playback session to automatically pause after the set of events in that frame have been played back, and wait for the user to continue.

Current Frame. This is a text label to indicate to the user the frame number of the current frame. The frame number increases as each frame is recorded.

Edit Frame. This button allows the user to go back and edit the description for a particular frame.

Frame #. This is the number of the frame whose description is to be edited.

FrameDescription. This is a text box in which the description of the steps involved in creating a frame, can be recorded.

Fig. 5. Details of a recording and playback session.

Event Synthesis during Playback Session

pack [button .b -text "Print Hello"]

Print Hello

Window : .b
Event : Enter
Time : \( t_0 \)

event generate .b <Enter>

Print Hello

Window : .b
Event : ButtonPress
Time : \( t_1 \)
Mouse button : 1

This button restarts the playback session.

Rewind.

Pause.

Continue. This button continues the execution of the active frame. It puts a marker on the next step within the active frame.

Exit. This button exits the playback session.

FrameSpeed. This slider is used to vary the playback speed within a frame. This slider provides granularity of scheduling events within a single frame.

V. EXPERIMENTS

The prototype of an environment that records, plays back and executes a Tcl/Tk collaborative Internet-based desktop, will be put to the test as an integral part of a national-level collaborative and distributed design project involving teams at 8 sites (http://www.cbl.ncsu.edu/vela/). Specifically, the desktop brings together distributed data, application workflows, and teams into collaborative sessions that share the control of the desktop editing and execution. A typical workflow, such as the one shown in Figure 7, invokes distributed tools and data to support a major phase in the design of microelectronic systems. A detailed description is available in [3, 4].

We argue that recording and playback of collaborative user interactions can have a wide-range of applications, such as: 'keeping minutes' of interactive discussions, clicks of menuspecific commands associated with different tools on the shared desktop, user-entered data and control inputs, user-query data outputs, support for automated software documentation, tutorials, collaborative playback of tutorials and solutions recorded earlier, etc. The 540 experiments, summarized in this section, are the initial part of the Internet desktop environment performance and functionality evaluation, conducted before its release to Vela Project participants and others.

Each of these experiments relies on interactive user inputs. To maintain consistency of user inputs during the repeated trial executions across the Internet (with variable quality-of-service), we first record a single reference instance of each test case on the local server (without relying on the network) and then move these recordings to cross-state and cross-country servers on the Internet. Each server has an executable version of ReubenDesktop, OmniBrowser, RecordTaker, and PlaybackMaker. The experiments are initiated with a playback that executes recorded instances of test cases, multi-casting them to 1, 2, or 3 workstation displays at CBL. Additional details

Frame #. This button displays the number of the current frame being played.

Print Hello
about these tools are available in [3, 7, 8]. Experiments reported in this section support a conjecture that will be the subject of more detailed experimentation later:

Task-specific performance of a single/multiple client-server ReubenDesktop execution can be predicted, as produced by the Unix utility time. The 'real-time' corresponds to the 'stopwatch.time' that could

In other words, to assess the performance of pre-recorded task-execution can be predicted, under comparable server and network loading, by measuring the performance of pre-recorded task-specific experiments that are executed and multi-cast by the server to one/multiple client displays.

In other words, to assess the performance of interactive distributed sessions that involve one or more participants, we have verified that the experiments, as reported in this section, can be extrapolated by measuring the performance of single- and multi-cast executions that are based on playback of pre-recorded experiments on a reference server. The benefits of not requiring a number of individuals to sit through repeated session experiments are obvious. Specifics about the tested configurations, test cases considered, and tabulated results follow.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Reference Server</th>
<th>CBL Server</th>
<th>UCB Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Time* CPU Timeb</td>
<td>1-client</td>
<td>2-clients</td>
<td>3-clients</td>
</tr>
<tr>
<td>Co-editing-1</td>
<td>119.4</td>
<td>122.4</td>
<td>125.2</td>
</tr>
<tr>
<td>Co-editing-2</td>
<td>153.1</td>
<td>157.5</td>
<td>165.3</td>
</tr>
<tr>
<td>Co-editing-3</td>
<td>223.3</td>
<td>248.1</td>
<td>258.4</td>
</tr>
<tr>
<td>Co-browsing-1</td>
<td>13.7</td>
<td>131.2</td>
<td>134.3</td>
</tr>
<tr>
<td>Co-browsing-2</td>
<td>32.2</td>
<td>158.6</td>
<td>167.2</td>
</tr>
<tr>
<td>Co-execution-1</td>
<td>305.6</td>
<td>337.7</td>
<td>357.3</td>
</tr>
</tbody>
</table>

Table I Summary of 540 experiments performed on the Internet among three sites.

*Both minimum and maximum values of 'real.time' are reported.
*Only average values of 'user.time' and 'system.time' are reported.

Testbed Configurations. In order to approximate typical instances of a distributed multi-site collaborative desktop environment, we have created:

1. **Local environment** by installing the desktop software on a CBL server\(^2\) which is multi-casting its desktop to one or more CBL client hosts;
2. **Cross-state environment** by installing the desktop software on a server\(^3\) at Duke University in Durham, NC, which is multi-casting its desktop to one or more CBL client hosts;
3. **Cross-country environment** by installing the desktop software on a server\(^4\) at the University of California in Berkeley, CA, which is multi-casting its desktop to one or more CBL client hosts.

Test Cases. We have created and recorded, directly on the CBL server under negligible loading conditions, six test cases of collaborative sessions with useful attributes that demonstrate typical user-invoked tasks. The brief description that follows includes the reports of real.time, user.time and system.time as produced by the Unix utility time. The 'real.time' corresponds to the 'stopwatch.time' that could have been obtained by the user monitoring the task. The 'user.time' is the time required by the CPU to complete the task. The 'system.time' is the CPU time required by the system on behalf of the task. A brief description of all test cases engaging two participants, that were recorded for the experiment, follows.

1. **Co-editing-1** (real.time=119.4s, user.time=31.1s, system.time=1.5s): Using ReubenDesktop, we open, and edit, a single 4-node, 3-arc workflow by selecting, opening, and closing a single data file node-configuration window.
2. **Co-editing-2** (real.time=153.1s, user.time=44.0s, system.time=1.9s): Using ReubenDesktop, we open, and edit, the same 4-node, 3-arc workflow by selecting, opening, and closing a single data file node-configuration window and a single program node-configuration window.
3. **Co-editing-3** (real.time=223.8s, user.time=67.5s, system.time=2.5s): Using ReubenDesktop, we open, and edit, the 17 node, 22 arc workflow by selecting, opening, and closing 3 data files and a single program node-configuration windows.
4. **Co-browsing-1** (real.time=136.7s, user.time=56.1s, system.time=2.1s): Using OmniBrowser, we traverse a directory structure, located on the server's local file system, across 3-levels, with up to 141 items in each directory. The directory structures of all the three servers were made exactly the same for uniform comparison.
5. **Co-browsing-2** (real.time=159.2s, user.time=97.5s, system.time=5.0s): Using OmniBrowser, we select, open, and scroll, from start to end, the same copy of a text file of about 1000 pages (2.2Mb), located on each server.

\(^2\)SUN SPARC 20 (chip=60MHz memory=64Mb swap=732Mb)
\(^3\)SUN SPARC Ultra 1 (chip=167MHz memory=256Mb swap=288Mb)
\(^4\)SUN SPARC 20 (chip=60MHz memory=96Mb swap=365Mb)
Co-execution-1 (real_time=123.9s, user_time=90.0s, system_time=3.8s): Using ReubenDesktop, we open, and execute, the hierarchical workflow in Figure 7. As shown, the workflow has 22 nodes and 28 arcs; during execution, the node labeled as optimizer expands into a sub-workflow with 14 nodes and 15 arcs.

All test cases involved two participants working collaboratively and consisted of exchanges of several dialogs via the FlowSynchronizer between the two, during each recording session.

Evaluation Method. All software and the files of six test cases, recorded directly on the CBL server, have been replicated on the server at Duke U. and the server at UCB. Scripts have been invoked, during the night when both servers and the network were least loaded, to execute the 540 experiments as follows:

From each of the three servers, execute and multi-cast 10-times, with interval of 30 seconds between each execution:

1. successively to one, two, and three client hosts at CBL, recordings of co-editing-1, co-editing-2, co-editing-3;
2. successively to one, two, and three client hosts at CBL, recordings of co-browsing-1, co-browsing-2;
3. successively to one, two, and three client hosts at CBL, recording of co-execution-1.

A log file, generated by `time` (real_time, user_time, system_time) command, archives timing data for each experiment. Similarly, a log file, generated by `sar` (system activity report) command, archives the load on each of the three servers during the execution of these experiments. The log file generated by `sar` provided the information whether or not both the load on the server and the network was sufficiently stable to accept the `real_time` and `user_time` results for tabulation.

Table I summarizes results of these experiments as follows:

1. The first column lists all the six test cases.
2. The second column reports the time required to record the example on the reference server.
3. Each cell in the remaining columns contains four values. The top two entries report the minimum and maximum values of `real_time` and the bottom two entries report the average values of `user_time` and `system_time` for each experiment.

Summary of Results. The data presented in Table IV allows us to evaluate the performance of Internet-based desktop environments.

1. The `real_time` for playback to a single-client on the reference server is approximately the same as the time required to record the test cases.
2. The `real_time` for playback from other servers varies, depending on the distance between the host server and its clients and the characteristics of the host server. Specifically, for single-client playback, Duke server consistently reported least execution times, followed by CBL server and UCB server. This is attributed to the higher performance server at Duke. However, for multi-clients, the execution times increased with distance in the order CBL, Duke, and UCB.
3. When the experiment is multi-cast to 2-clients or 3-clients, it takes slightly more time, of the order of few seconds, for execution than the time required for single client execution. The negligible increase in the playback time for multi-client execution is due to the fact that the exchange of dialog among participants is computationally least intensive.
4. The variations in minimum and maximum values of `real_time` for each experiment are negligible since the experiments were performed during the night.

We have proposed a Tcl/Tk recording/playback architecture and an implementation that records, plays back and executes a Tcl/Tk collaborative Internet-based desktop. Both tools, RecordTaker and PlaybackMaker, can be used as stand-alone Tcl/Tk applications or as a part of a larger system such as ReubenDesktop.

We envision that a number of collaborative user interactions and Internet users will find useful application of the proposed recording and playback mechanisms. Specifically, considerable resources would be required to conduct the feasibility of collaborative remote user-interactions, sharing of tools, and desktops to accumulate as much information as we tabulated on the 540 Internet-based experiments in this paper. Without the RecordTaker and PlaybackMaker, we would require a number of participants over an extended period of time.

There are a number of new features that will extend the applications and the utility of RecordTaker and PlaybackMaker. These include:

1. an environment in which several recordings can be spliced together to create a new recording.
2. extending the recording and playback collaborative environment to the World Wide Web (WWW). Such an environment can be seen as a new service, available from the WWW.

ACKNOWLEDGMENTS. We could not have reported as comprehensively on the results of our Internet Desktop experiments without getting generous user accounts on two remote servers, facilitated by Dr. Richard Newton at UC Berkeley and Dr. Gershon Kedem at Duke U. We thank them and their support staff for this privilege.
REFERENCES


