ELSEVIER

Computer Networks and ISDN Systems 28 (1995) 209–219

COMPUTER NETWORKS and ISDN SYSTEMS

Beyond the Web: manipulating the real world

Ken Goldberg [a,*], Michael Mascha [b], Steven Gentner [a], Jürgen Rossman [c], Nick Rothenberg [d], Carl Sutter [e], Jeff Wiegley [a]

[a] *Department of Computer Science, University of Southern California, Los Angeles, CA 90089, USA*
[b] *Department of Anthropology, University of Southern California, Los Angeles, CA 90089, USA*
[c] *University of Dortmund, Dortmund, Germany*
[d] *Department of Visual Anthropology, University of Southern California, Los Angeles, CA 90089, USA*
[e] *Center for Scholarly Technology, University of Southern California, Los Angeles, CA 90089, USA*

Abstract

This paper describes a WWW site that allows users to "leave the Web" and interact with the real world. An interdisciplinary team of anthropologists, computer scientists and electrical engineers collaborated on the project, desigining a system which consists of a robot arm fitted with a CCD camera and a pneumatic system. By clicking on an ISMAP control panel image, the operator of the robot directs the camera to move vertically or horizontally in order to obtain a desired position and image. The robot is located over a dry-earth surface allowing users to direct short bursts of compressed air onto the surface using the pneumatic system. Thus robot operators can "excavate" regions within the environment by positioning the arm, delivering a burst of air, and viewing the image of the newly cleared region. This paper describes the system in detail, addressing critical issues such as robot interface, security measures, user authentication, and interface design. We see this project as a feasibility study for a broad range of WWW applications.

Keywords: Robot; Remote control; Mercury Project; USC; Random token; Telerobotics

1. Goals of the project

The WWW provides a multi-media interface that spans all major platforms. Thousands of sites have been set up in the past year. Our goal with this project was to provide public access to a teleoperated robot, thus allowing users to reach beyond the digital boundaries of the WWW.

Such a system should be *robust* as it must operate 24 hours a day and it should be *low in cost* (we had an extremely limited budget). It is worth noting that the manufacturing industry uses the same criteria to evaluate robots for production. Thus our experience with RISC robotics (see below) proved helpful.

Our secondary goal was to create an evolving WWW site that would encourage repeat visits by users to collectively solve a puzzle. This paper focuses on the details of the implementation. The Mercury Project is archived with supporting examples and statistics at: http://www.usc.edu/dept/raiders/

[*] Corresponding author. E-mail: goldberg@ieor.berkeley.edu.http://www.usc.edu/users/goldberg. This work was supported in part by NSF Awards IRI-9123747 and IRI-9457523.

0169-7552/95/$09.50 © 1995 Published by Elsevier Science B.V. All rights reserved
SSDI 0169-7552(95)00094-1

2. Related work

The first "teleoperated robots" were developed over 40 years ago. The basic objective has always been to develop systems capable of working in inhospitable environments (such as radiation sites). Teleoperation began with very simple mock-ups in nuclear power plants [6], progressing to more versatile setups for teleoperation of robots in space [7]. Over the last 20 years, the development of intuitively operable teleoperation tools has continued to play an important role in the development of robotics in general. The basic objectives have remained the same, even though the methods and technical limitations have changed. See also [4].

Today, sophisticated "Telerobot Operator Control Stations" [3] are equipped with stereoimage-displays, "force reflecting hand controllers" and comprehensive video graphics support. The development of teleoperation stations is currently being pushed further with the help of latest graphics workstations to provide so-called "telepresence". Modern telepresence systems, considered to be pushing the frontier of research in this field, are defined as follows [1]: "At the worksite, the manipulator has the dexterity to allow the operator to perform normal human functions. At the control station, the operator receives sufficient quantity and quality of sensory feedback to provide a feeling of actual presence at theworksite".

The Mercury Project does not achieve this level of telepresence but provides a limited level of teleoperation. One of our goals was to provide "teleoperation for the masses". Instead of developing a highly sophisticated, multi-million-dollar testbed, we opted for a simple and reliable end-effector on a commercial robot. Combined with an intuitively operable man-machine-interface, the system gives all WWW users access to teleoperation. In the Discussion section, we describe a number of other WWW sites that offer interactive capabilities.

3. User interface and environment design

The interface design for the system was challenging due to the limitations of the HTML/HTTP environment, as well as network traffic considerations. An effective system was created within such limitations by carefully designing the physical environment for the robot, and by fine-tuning the user-machine interface. For example, the initial idea of a live video feed from the camera was dropped in order to maintain compatibility with all visual clients on the Web. (Although we could have implemented some custom clients [1], we decided to stay within the limits of HTML/HTTP to reach as large a user base as possible, making this a truly global system.) In addition, initial simulations using a robot fitted with grippers (simulated in VIRTUS WALKTHROUGH) revealed a high degree of

[1] There are two possible fixes to this problem, One is to release specially modified clients that set up a two-way communication, the second is to use some other software to display the current system on the user's client workstation. Since many clients are used to view the WWW, making modifications would be difficult, especially since they are being updated all the time. Even if source code could be obtained for every major client, changes would have to made to every release of all these be on each release of these applications, The second possibility is to write a separate program to run on the clients' workstation. The problem here is to write a robots client that can be released for enough platforms to be useful, Since this would be an esoteric piece of the system, it is not likely that other sites would customize the software for different systems like is done for the major systems. One technique is to use the X windows protocol to display a client application on the users workstation running an X server (weather, movies). We felt that this would be a limited audience, however. It also may compromise security from the user's point of view. Both these approaches may be attempted in version 2.0 of the system to allow more enhanced use of the system for some users. The HTTPD protocol could be extended to allow these sort of connections, though—maybe we need a new protocol for passing media only back that doesn't have all the hooks into system calls like X Windows and Display PostScript.

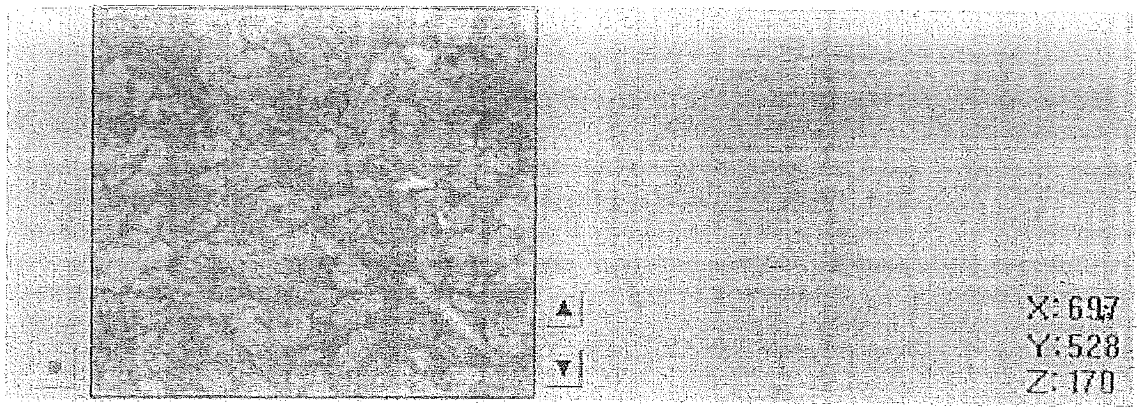


Fig. 1. The control panel.

complexity in control functions [2], not suitable for the anticipated 5–10 seconds per frame page loading time, a 2D Mosaic window and a naive/untrained user.

The team chose instead to use a simple environment which would allow relatively easy control of the robot. Here the analogy taken from real world archaeology—using a dry-earth environment and compressed air bursts—allowed us to simplify the robot control dramatically. Thus users could be quickly trained in the operation of the system, through a simple "Operator's Orientation" and a "Level 1 Clearance Test".

Even with a simplified system (see Fig. 1), users are still able to choose between fine and gross movements of the arm. Fine pitch movements are executed by clicking in the camera image, with the robot moving to center the arm over the X,Y-coordinates of the click-point. Crude navigation is provided by clicking on a schematic picture of the robot and it's workspace, with the robot moving to center the arm over the click-point. Two buttons allow navigation in the Z-axis (between "up" and "down" positions), with a button to blow air only active when at the $Z = O$ (i.e., "down") position.

[2] 3D control of a robot needs: 3 dimensions of spatial movement, 3 dimensions of orientation and 1 to 3 dimensions of gripper control.

Other features of the system were designed to balance functionality with user needs. All HTML documents sent to the clients are carefully designed to minimize network traffic in order to get a high refresh rate. For example, control panel functions are clearly distinguished from text-based information documents. The "Operator's log" was implemented to create a forum for collaborative efforts to solve the puzzle/problem regarding the underlying logic which links the artifacts. (The "Operator's log" is readable throughout the system but only writeable after completing an operating session.) A **second entry path** [3] was also created to the system, which provides a "back-story" explaining the project while also hinting at possible "real world" uses of the system.

4. Access to the robot

Most of the HTML documents seen by the user on our site are generated by a script running on the WWW server. Using a random token scheme described below, the system tracks each user as he or she proceeds through the interface and generates appropriate HTML documents. This allows the system to discriminate between

[3] http://www.usc.edu/dept/raiders/story/mercury-story.html

"observers" and "operators" so that it presents only accessible options to each.

To operate the robot, the user must read the information on how to use the control panel, and then complete a level-1 clearance test to get a password. Since only one person can operate the robot at a time, the system maintains a queue of pending operators. A typical user will enter his/her password, and then add him/herself to the queue. Each time the update button is clicked, the system updates the queue and returns a current status page. When the user's turn arrives, the screen returned is the live operators' control screen.

5. System architecture

Fig. 2 shows a block diagram for the system. We start with an overview that necessarily glosses over many interesting details.

At one end are WWW clients from around the world; at the other end is a robot arm combined with a camera. The robot and camera provide an updated image of the environment, which is combined with a schematic of the robot arm/workspace and control buttons to produce the final GIF image that is send to users.

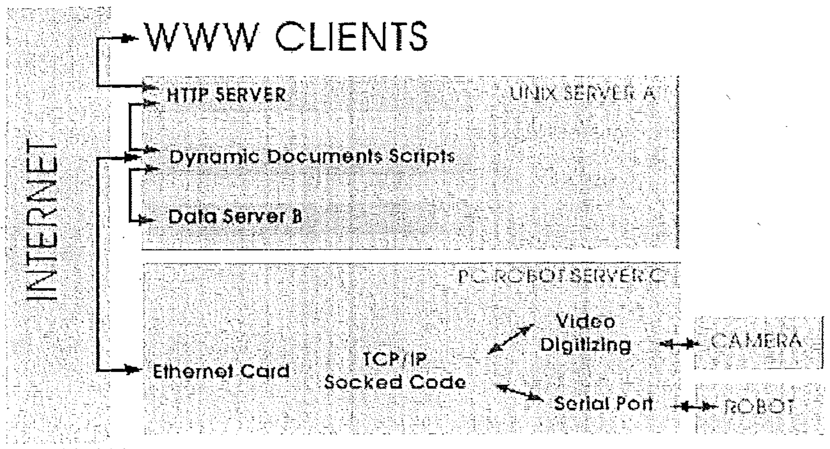
At any given time there may be dozens of clients interacting with the system. Since there can only be one operator at a time, one challenge is to keep track of which client is the operator.

The Mercury system is comprised of three communicating servers. The first, call it A, is a standard Mosaic server (NCSA httpd v.1.3, currently running on a Sun SPARCserver 1000, with SunOS Release 5.3. When the Site is requested by an observer, the most recent image, which is stored on server A, is simply returned.

The database of registered users is handled by another server, call it B. In our case, server B runs on the same machine as server A. The database server is custom programmed for this project, but performs fairly standard database functions.

When a client request comes in, server A communicates with server B. If that client is an operator, server A must then communicate with a third server, call it C, that controls the robot. Server C runs on a Pentium-based PC and communicates with servers A and B via the Internet. Server A decodes the ISMAP X and Y mouse coordinates, and sends them to server C.

On server C, a custom program decodes these coordinates into a robot command and verifies that the command is legal, e.g., within the robot workspace. If it is, this command is then converted into a robot command format which is sent to the robot over a serial line. Once the robot move is completed server C uses the CCD cam-



System Block Diagram

Fig. 2. System block diagram.

(1). need pw for images
(2). is video vs. image?

era to capture a stable 8 bit 192×165 image of the workspace.

Using a simple set of equations for inverse kinematics server C then generates a schematic view of the robot in its new configuration. This schematic is combined with the camera image, and the up, down, and air control buttons to form a new composite image. Server C then compresses this image into GIF format and returns it to server A, which updates the most recent image and returns it to the operator client.

6. Subsystems

6.1. Random tokens for cache avoidance and user tracking

Following some complex and unwieldy tests, we implemented a random token scheme for tracking users as they use the system. Each time a URL is returned, a large random number is added to the path (which the NCSA HTTPD 1.3 server splits into the PATH_INFO environment variable). This "token" serves two purposes:

website

The first is to prevent the WWW client from caching the robot view. When a document is requested a second time during a session, it is much faster to swap in a local copy of the document rather than going back over the net to retrieve it a second time. Most implementations of Mosaic support such caching at various memory levels. However in our case we want to repeatedly retrieve the URL containing the robot image because it is updated continuously. In brief, we don't want users to cache this URL. The random token makes each request look different and tricks the client into retrieving a fresh version of the document.

The second use for the token is to identify operators. When an operator logs in with a correct password, the system begins tracking him/her as he/she moves from viewing the robot to being on the operators' queue to operating the robot. Since the same script is used for all views, the token allows the system to customize the result for every user depending on his/her position in the system.

6.2. Scripts

The robot view screen is controlled for the most part by one script at the HTTPD server. Each call to the main script has a token attached to the URL. The token is decoded by the WWW server, and placed in the PATH_INFO environment variable. The main script then checks the token with the database server to determine the status of the user. Each check of the database generates a system update to keep the queue moving. The user's status is used to generate the custom system status page.

The robot image itself is only changed by the operator when he or she makes a move. Each image is date and time stamped, so WWW clients that cache the image will only retrieve the image when it changes (since its filename will be different, due to a different time stamp). (2)?

Due to the client-server architecture of the World Wide Web HTTPD protocol, the robot system (server) has no way to contact the client except at the client's prompting. From the user's point of view, once he or she gets the robot view screen, there is no way for the server to keep sending updates automatically as the robot is moved by the operator. The screen updates must be driven by the user. Since the user must trigger each update, we wanted to provide a button for doing so, since each web client handles reloading the page differently. Some sites have a "reload" hypertext link to the same page, but this doesn't work for any client that caches pages. If a page is being viewed, hitting reload will just re-display the page from the cache, thus not obtaining a new view from the system. Asking the user to disable his/her cache is also problematic, since not all clients allow this option.

One attempt was made to use a mini-form, since the submit button always calls a script and is not cached. That scheme was eventually dropped, since passing registered user identification information to the server via hidden fields only worked on some clients. Using the random token allows for an elegant interface.

Since the robot can only be controlled by one person at a time, a registration scheme was implemented to allow the server to track operators

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.