Beyond Webcams and Videoconferencing: Informal Video Communication on the Web

Nicolas Roussel
Laboratoire de Recherche en Informatique - CNRS
LRI - Bâtiment 490 - Université Paris-Sud
91405 Orsay Cedex, France
roussel@lri.fr

ABSTRACT

The Web is changing. It is becoming more dynamic, interactive, and tailorable. But can it be used for interpersonal communication? This paper addresses the use of the Web for image-based synchronous communication between distant users. It does not focus on transmission quality, latency, or packet losses. Instead, it shows how the Web can be used to support informal video communication and to address accessibility and privacy issues.

Keywords: Informal video communication, accessibility, privacy, Web.

INTRODUCTION

The Web is changing from a static information repository to more dynamic forms. However, it is still mainly used as a mass media to broadcast mostly textual information to many people. My work is focused on computer support for distributed groups and particularly on the affordances of video as a technology to support informal communication. This form of communication can be described by the following properties [1]: frequent, brief, unscheduled, often dyadic, frequently supported by shared objects, intermittent, lacking formal openings or closings. During the last ten years, experiences with media spaces [2] [3] have shown that these properties are essential for distant people to coordinate and develop relationships despite the lack of physical proximity.

What does the Web offer that can be used for real-time informal video? Traditional videoconferencing tools have performance requirements that make them hardly compatible with today's Web languages and protocols. The phone paradigm used by these tools is also too long and intrusive for informal video communication: it implies bi-directional audio and video connections with explicit call acceptance. The only real-time informal video we can find on the Web comes from webcams, cameras that point at public places. But webcams provide very little and somewhat useless information, e.g. whether it is raining in London (Figure 1).

Figure 1. Sample webcams (Paris, London, New York)

This paper shows that existing Web languages and protocols can be used to provide informal video communication to distributed people without threatening their privacy. It begins by presenting some general requirements for informal video communication support. It then describes several technical characteristics of HTTP and HTML that meet these requirements, and then it presents *videoServer*, a Web-based tool for video communication. Finally it describes some of the problems or limits encountered while developing this tool.

WHAT DO WE NEED?

Fish et al [4] describe three key competing factors for informal video communication: accessibility, the ability for people to have easy access to others, privacy, the ability to control the information about oneself that is available to others, and solitude, the ability to control others' intrusion into one's space or consumption of one's time.

Accessibility requires easy access over time and space. This is why the Web is interesting: it is globally accessible, platform independent, and it is becoming a central access point for applications and services. Ideally, we would like to be able to communicate through the Web without having to install any plug-in or helper application. Using the standard languages and protocols we can build on the document-centered approach that characterizes the Web, by seamlessly integrating coordination and communication channels with shared artifacts (i.e. documents).

Today, publishing documents on the Web is like putting them in the middle of the street and hoping that someone will see them. But it is hard to tell if anyone sees them and if so, who they are. If the documents are to contain pictures or live video of people, the important issues of privacy and solitude must be addressed. The users need to know who is retrieving the documents they publish and possibly to adapt the contents of the document according to whom is retrieving it. For example, colleagues and friends can get a live video of me while others get a pre-recorded static image. These two important notions have been described as *notification and control* mechanisms in many mediaspace systems [5].

WHAT DO WE HAVE?

Digitizing live video is now possible on most platforms. The main problems lie with digital video transmissions and user interface. HTTP is an application-level protocol for raw data transfer of resources across the Internet, a resource being "a network data object or service" [6]. HTML is a simple data format used to create hypertext documents [7]. Together, these two standards contributed to create the Web and constitute the essential basis of any Web browser. How can we use them for digital video transmissions?

Since its origin, HTML has supported still images in documents. Unfortunately current standards do not support stream transfer: when a server receives a request, it sends back the corresponding resource and then closes the connection. However, a mechanism known as "server push" can hold a connection open over multiple responses. The server can send more data when it is available, and every new piece of data replaces the previous one [8]. This mechanism makes it possible for example to send a series of images instead of a single one, which makes it possible to insert a video stream into any HTML document without requiring any special plug-in or helper application. In addition, any image (still or animated) can be used as a button by including it in an HTML anchor pointing to another resource. JavaScript code can also be used to dynamically change images. This is commonly used to highlight menu items when the mouse passes over them or when the user clicks on an interface component.

When an HTTP server receives a request from a client, it can get information about that client in several ways. First, the client itself generally sends information describing it, such as its name and version number, the kind of system on which it runs, the data types it understands. In addition, since HTTP runs on top of the TCP protocol, the server can use standard services to get the address or name of the remote machine and even the login name of the remote user if an authentication server [9] is running. Using the request and information about the person who made it, the server can then choose between the many response semantics offered by HTTP to reply to the client: accept and execute the request, redirect the client to another resource, ask for a login/password authentication, etc.

WHAT WE HAVE DONE: *videoServer*

VideoServer is a custom HTTP server that encodes live or pre-recorded video as a series of JPEG images and then sends them using the server-push mechanism. It was first prototyped as a simple extension of our Web server (a CGI script) and soon evolved into a standalone custom server that can run efficiently on any of our workstations. The current version allows us to send digital video to anyone connected to the Internet with a frame rate that depends on the available bandwidth: a typical 240x162 JPEG image is about 6 Kb.

VideoServer offers three classes of resources corresponding to the following services: live pictures, live video streams and pre-recorded video streams. The quality and resolution of live video pictures and streams can be controlled by additional query string parameters that specify the compression ratio and zoom factor. Other parameters allow clients to specify the number of images requested for a live stream and the time to wait between two subsequent images. Picture and video streams can be inserted in any HTML document with markup such as this:

```
<img src="http://videoserver:5555/photo?zoom=4&cratio=20.0">
<img src="http://videoserver:5555/video?zoom=3&length=6&pause=10">
<img src="http://videoserver:5555/file/emptyOffice">
```

Of course, more complex code can be used. The following JavaScript code, for example, inserts a snapshot that is an active link to the author's Web page: it turns into live video when the mouse is over it and turns back to a snapshot when the mouse leaves it:

```
<A
HREF="http://www-ihm.lri.fr/~roussel/"
onMouseOver='document.img1.src="http://videoServer:5555/video"'
onMouseOut='document.img1.src="http://videoServer:5555/photo"'>
<IMG NAME="img1" SRC="http://videoServer:5555/photo" >
</A>
```

This code can be used to create an awareness view on distant people similar to Portholes [10] by putting together a set of views from different *videoServers* (Figure 2). Note that this view differs from existing image-based awareness systems in two ways. First, the set of images can be freely modified by the users and is not restricted to a list of registered people, since the system is the entire Web. Second, the JavaScript code shown before provides people with a lightweight interface to refine their judgement about other users' availability (e.g. to distinguish between someone coming in and going out of a room).

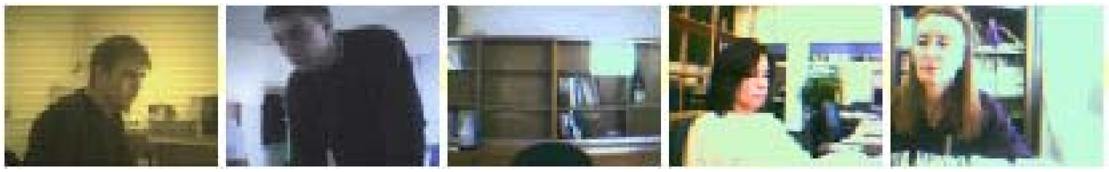

Figure 2. Images from several *videoServers* around Europe

Notification and control are achieved by a customizable program called the *notifier*. *VideoServer* executes the notifier for every request it receives, passing it arguments indicating the name of the client's machine, the login name of the person who sent the request (if available), the service requested and the values of the query string arguments. The notifier sends back the description of the service to execute, which can be different from the requested one. By assuming that the person in front of a camera is also logged on a workstation associated with this camera, the notifier makes it possible for every user to define his or her own personal access rules. One can for example associate different notifications to different categories of people or services, e.g. text display, auditory icons, or reciprocal connection. Since the service to be executed can be redefined, it is also possible to send pre-recorded video instead of a live stream, or to change the resolution, the quality, the number of images or the refresh rate of the stream. For example, users can decrease image quality by changing the compression factor (Figure 3) in order to hide details of their activity to people they don't know.

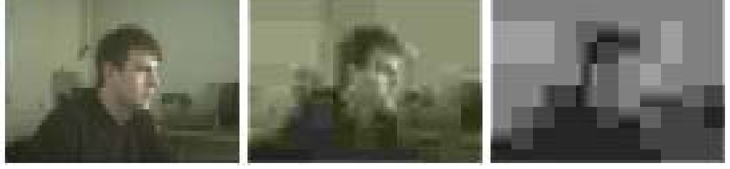

Figure 3. Degrading image quality by increasing compression

WHAT ARE THE PROBLEMS?

The server push mechanism was introduced by Netscape in 1995. Although it is the simplest solution for streaming data over HTTP connections, it has not been implemented in every Web client (e.g. it hardly works with Microsoft Explorer). The

World Wide Web Consortium is working on dynamic documents and audio or video integration. This work will probably lead to usable standards, but at the same time it is likely these standards will be much too complicated when compared with the simple requirements presented in this paper.

By giving *videoServer* to people outside our lab, we can quickly have bi-directional video connections with anyone with a Web browser and a video digitizing board. We use the system on a regular basis with third party applications such as text chat and audio broadcasting or with regular phone calls to support daily group work. Although the Web interface works fine, we felt the need for specialized lightweight clients to reduce screen and memory usage or to implement different display policies, e.g. resizing the images to fit the window. Writing our own HTTP servers and clients made us realize that HTTP has many features that are yet unexploited by traditional servers and clients. The redirection semantic could be used by *videoServer* to negotiate with the client instead of unilaterally redefining the request. Unfortunately generic Web clients do not negotiate: they follow the redirection because they don't know anything about the semantics of the resources. By developing custom clients and servers that follow the HTTP protocol, it is possible to build a complex communication system that will remain compatible with existing applications. *VideoServer* is a simple example of this approach.

HTTP is based on the TCP protocol, which is inefficient for high quality and high frame rate videoconferencing. Other protocols such as UDP or RTP give better performance. *VideoServer* can send each JPEG image as a UDP datagram instead of using the server push mechanism. The HTTP connection remains open and is used as a signaling channel. The custom client mentioned above displays images with this new protocol, leading to better frame rates. However the request still conforms to HTTP.

CONCLUSION

VideoServer is a video gate that helps the members of distributed groups to be accessible to each other. Its notification and control mechanisms provide a good balance between accessibility, privacy and solitude, as described in [4]. This makes *videoServer* a tool for informal video communication rather than another videoconferencing tool or a new Big Brother. In this paper, we have shown how such a tool can be implemented with existing Web languages and protocols. We have also described some limitations imposed by these standards and how to overcome them.

VideoServer is implemented on SGI workstations and is freely available from http://www-ihm.lri.fr/~roussel/Mediascape/videoserver.html. It has been in use for over a year by our group and other users around the world. Any help to port it to other platforms is welcome.

ACKNOWLEDGMENTS

This work is partially supported by CNET-France Telecom under project number 961B222 (Telemedia).

REFERENCES

[1] E. Isaacs, S. Whittaker, D. Frohlich, and B. O'Conaill. “Informal communication re-examined: New functions for video in supporting opportunistic encounters”. In K. Finn, A. Sellen, and S. Wilbur, editors, *Video-mediated communication*. Lawrence Erlbaum Associates, 1997.

[2] S. Bly, S. Harrison, and S. Irwin. “Mediaspaces: Bringing people together in a video, audio and computing environment”. *Communications of the ACM*, 36(1):28-47, January 1993.

[3] W. Mackay. “Media Spaces: Environments for Informal Multimedia Interaction”. In Michel Beaudouin-Lafon, editor, *Computer-Supported Cooperative Work, Trends in Software Series*. John Wiley & Sons Ltd, 1999, in press.

[4] R. Fish, R. Kraut, R. Root, and R. Rice. “Evaluating Video as a Technology for Informal Communication”. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 37-48. ACM, New York, 1992.

[5] P. Dourish. “Culture and Control in a Media Space”. In G. De Michelis, C. Simone, & K. Schmidt, editor, *Proceedings of European Conference on Computer-Supported Cooperative Work ECSCW'93*, Milano, pages

335-341. Kluwer Academic, September 1993.

[6] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. “Hypertext Transfer Protocol - HTTP/1.1”. Technical report, W3C Architecture Domain, August 1998. http://www.w3.org/Protocols/.

[7] D. Raggett, A. Le Hors, and I. Jacobs. “HyperText Markup Language - HTML/4.0”. Technical report, W3C User Interface Domain, April 1998. http://www.w3.org/TR/REC-html40/.

[8] “An Exploration of Dynamic Documents”. Technical report, Netscape Communications, 1995. http://home.netscape.com/assist/net_sites/pushpull.html.

[9] M. St Johns. “Authentication Server”. RFC 931, IETF Network Working Group, January 1985.

[10] P. Dourish and S. Bly. “Portholes: Supporting Awareness in a Distributed Work Group”. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 541-547. ACM, New York, 1992.