Chapter 9

WEB-BASED MEDIASPACE

Nicolas Roussel

Abstract

This chapter addresses the use of the Web as a platform for developing *mediaspaces*; environments that combine audio, video and computing to support distributed groups in their daily tasks. It does not focus on transmission quality, latency, or packet losses. Instead, it shows how Web standards and protocols can be used to create a software infrastructure and user interfaces that offer accessibility as well as privacy. Building on previous work on the social and technical issues of mediaspaces, we introduce three important design principles: integrability, flexibility and privacy. We show how Web standards and protocols can be used to create a software platform consistent with these principles. We present Mediascape, a Web-controlled analog audio/video mediaspace and videoServer, a Web server dedicated to digital video communication. Finally, we introduce videoSpace, a toolkit that allows building Web-based video applications.

8.1. INTRODUCTION

Stults coined the term "Media Space" at Xerox PARC in the mid 1980's [1]. In order to link two related laboratories located in Palo Alto, California and Portland, Oregon, Stults and his colleagues developed one of the first systems that combined audio and video with computers to support coordination, communication and collaboration among distributed groups. For more than 10 years, research has explored the social and technical issues raised by these environments, including privacy concerns, long-term use, and appliance and service design [2, 3].

Many systems have focused on using video to support formal distributed meetings among groups of people. Experiences with mediaspaces have demonstrated the ability of these environments to support informal communication. This form of communication has proved to be essential for distant people to coordinate and develop relationships despite the lack of physical proximity. It can be described by the following properties [4]: frequent, brief, unscheduled, often dyadic, frequently supported by shared objects, intermittent and lacking formal openings or closings.

The Web is constantly evolving, becoming more dynamic, more interactive and more tailorable. It has already changed many of our habits. But although we might use it sometimes to communicate with distant people by exchanging text or images, the Web is still mostly used as a ubiquitous access point to information, as a link between people and knowledge.

This chapter focuses on the combined use of audio, video and the Web to link people with other people. Building on previous work on the social and technical issues of mediaspaces, we identify three important design principles for such systems: *integrability*, *flexibility* and *privacy*. We show how Web standards and protocols can be used to create a software infrastructure and user interfaces consistent with these principles. We present Mediascape, a Web-controlled analog audio/video mediaspace. We then present videoServer, a Web server dedicated to digital video communication. Finally, we introduce videoSpace, a toolkit that allows building Web-based video applications.

8.2. LEARNING FROM PREVIOUS MEDIASPACES

Technically, a media space consists of a group of offices and public spaces connected through an audio/video network. Early media spaces used an analog network: standard cameras and monitors connected to a computer-controlled crossbar switch [5, 6, 7, 8]. A typical office "node" had a video camera with a microphone, a monitor with speakers, and a workstation to run the connection management software (Figure 1). These early systems were developed on analog infrastructures because computers and digital networks could not handle live video in real-time with an adequate image quality. Although digital video is still pushing the limits of the technology, this is less and less true, and more recent mediaspaces rely on a digital network such as Integrated Services Digital Network (ISDN), local area network (LAN), or asynchronous transfer mode (ATM) [9, 10, 11].

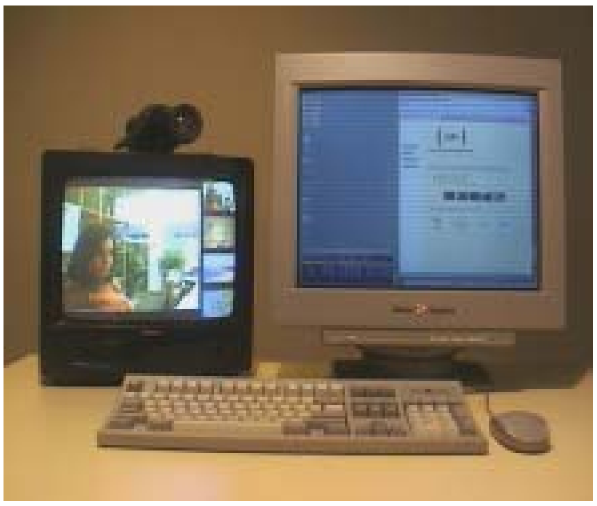


Figure 1. Typical configuration for an analog mediaspace node

Mediaspaces provide users with different kinds of services, including:

- *Awareness view* [12]: a window displaying a series of small digitized images of the different nodes, grabbed at regular intervals;
- *Background connection*: a public source, instead of having a black screen when there is no connection (for example, a view of the campus or a TV channel);
- *Office share*: a background connection used to "share" an office with someone for a long period of time;
- *Videophone*: an audio and video link between two nodes, used like a traditional phone call;
- *Glance*: a one-way video connection lasting a few seconds, to see if someone is there.

The introduction and use of a mediaspace raise a lot of issues concerning the protection of individual privacy and access control. In all the existing systems, different technical solutions have been designed, along with the services, from a strict reciprocity rule ("I can see you if you can see me") to explicit negotiation or dynamic user-defined mechanisms for access control and notification. The cultural context around the mediaspace is also important. In addition to the technological solutions, social protocols and a sense of culture emerges when "living" in a mediaspace [13].

Gaver's analysis of the affordances of mediaspaces shows that the physical characteristics of devices modify the way users perceive and act in such spaces [14]. Following this approach and building on our own experience using these environments, we have identified three important principles for the design of a mediaspace:

- *Integrability*: the integrability of a mediaspace is the degree to which it supports existing practices and tools, rather than impose new ones on the user;
- *Flexibility*: the flexibility of a mediaspace is the degree to which its components can be repurposed to support new uses with little effort;
- *Privacy*: a mediaspace supports privacy when users can easily understand, operate and trust the mechanisms that control information available about them and others access to it.

In this section, we detail these three principles and explain how they can be applied to the design of a mediaspace software infrastructure and user interfaces.

8.2.1. INTEGRABILITY

Bly et al. [2] emphasize the importance of placement and physical access to communication devices (such as cameras and monitors) and their integration into work practices. For example, videoconference rooms require users to go in a dedicated room, explicitly switching between personal and group activities. On the contrary, mediaspaces tend to augment physical space by integrating devices in the real world and making them instantly and permanently accessible. This approach eliminates the notion of a call, with a beginning and an end, and fosters smooth transitions between peripheral awareness (hearing and seeing) and more focused communication (listening and looking). In this sense, mediaspaces are related to Ubiquitous Computing [15] and Augmented Reality [16], two research directions that aim to integrate computers into the real world.

Grudin [17] suggests integrating groupware features with features that support individual activities and, if possible, adding them to already successful applications. In traditional desktop videoconferencing, which resemble videoconference rooms, users have to switch between the video application and the other ones until they find the optimal layout for the multiple windows on the screen. The telephone model implemented by these applications requires explicit actions for placing and accepting calls, making the interface even more cumbersome. In contrast, using a mediaspace is not a primary activity in itself. It is a peripheral activity that supports spontaneous interactions. However, this also makes mediaspace interfaces difficult to design, because the frequency and variety of uses are hardly compatible with a task-centered design approach.

Several mediaspace systems are able to manage existing applications or documents [9, 10, 11]. However, they tend to integrate them into their own framework, contrary to the idea of interacting with the mediaspace in the background of other activities. A mediaspace interface should be integrated into the software environment in the same way its physical devices are integrated in the real world. The ubiquitous/augmented approach should be used here: interactions with the mediaspace should not be available through a single workstation and/or application. Instead, we should integrate the digital media and the interface into any existing document or application, whether individual or collective. When designing the software infrastructure of a mediaspace, we must think in terms of components to integrate into existing practices, not in terms of new applications.

8.2.2. FLEXIBILITY

The affordances of analog mediaspaces depend not only on the physical properties of the devices, but also on how people can use them. Suchman points out [18] that people commonly improvise and repurpose their actions. Mediaspace hardware configurations can easily be tailored by moving devices, adjusting them (for example, the sound volume or the image brightness), replacing them, or combining them (adding a wide-angle lens or an audio mixer). Again, this contrasts with dedicated videoconference rooms where users cannot change complex preset setups.

A large variety of services can be created by changing the duration (very short, intermittent, or persistent) and the nature of media (small digitized image, video only or audio/video) of a connection. These services support different activities, from formal to informal and from scheduled to spontaneous. Like hardware configurations, mediaspace software should be tailorable. Instead of providing users with rigid predefined communication services, we should try to create a "medium" that users can adapt to suit their needs.

Bentley and Dourish [19] show that the notion of medium, as opposed to mechanisms, arises from systems openness and flexibility. Open protocols [20] and component architectures [21, 22] can serve to implement a set of basic mediaspace software parts for developers and users. Each of these parts would correspond to a well-defined set of system requirements: connection management, access control and notification, session management, digital media services, etc. Users could then define their own policies or patterns of use by configuring, replacing or combining these parts, like they do with physical devices. In many existing systems, this flexibility tends to be accessible only to developers or expert users. Care must be taken to bring tailorability to the large majority of non-programming users as well.

8.2.3. PRIVACY

Asymmetric connections such as awareness views and glances are essential for providing nonobtrusive awareness of the presence and activity of other mediaspace users. Since these services break the strict reciprocity rule of the real world, mechanisms become necessary to ensure users' privacy and, at the same time, keep the system as open and accessible as possible.

Privacy in a mediaspace is important because of the highly dynamic nature of access control to live media. Whereas access rights on documents are usually simple to specify by using read/write permissions granted by the owner, access rights on live video or audio sources are much more complex because of the multiple uses these media might serve. A short glance into an office, a slowly updated view or a live video feed correspond to different intentions by the caller although they have the same basic requirement (a video link). The identity of the person requesting live media is also important: relatives, friends, colleagues or strangers should not have the same access to a user's camera and microphone. Mediaspace software should provide users with notification mechanisms that help determine the identity and intention of the remote person. It should also provide users with simple mechanisms to control available information about them, based on this knowledge.

Another important issue is the extent to which users trust the system. When users turn off or unplug a physical device, they know the effects on the device and the system as a whole, they can easily check these effects, and they know they can always go back to the previous state. Ideally, the same should be true of software access control. Notification and control mechanisms should be simple so that people can trust them like physical mechanisms. Flexibility again offers the key to a successful compromise between unobtrusive spontaneous interactions and explicit access control. Users should not be restricted to binary choices (that is, accept or deny the service) but should be able to describe complex behavior such as changing the request before execution (for example, switching to a lower resolution). Graphical or auditory notification should also be available before, during and after the execution of a request, so that the state of the system is always known. Every modification of the access policy should be easily checked and reversible. All these elements contribute to Dourish's notion of selective accessibility [13].

8.3. WEB SUPPORT FOR MEDIASPACES

HTTP is an application-level protocol for transferring resources across the Internet. A resource is "a network data object or service" [23] and is specified by a URL (Uniform Resource Locator). HTML is a simple data format used to create hypertext documents [24]. Together, these standards contributed to create the Web: a globally accessible and platform-independent hypermedia information system. The Web is one of the most successful systems for communication between people and in many ways, it is becoming a central access point to applications and services: more and more applications and programming toolkits are able to use HTML for content description or HTTP as a transfer protocol.

This section investigates the use of these standards to support the implementation of a mediaspace software infrastructure. It presents several features of HTTP that make it suitable for connection management and digital video streaming services. It also shows how HTTP clients and HTML documents help create interfaces to these services consistent with the three principles introduced above.

8.3.1. INTEGRABILITY USING HTTP AND HTML

HTTP is a request/response protocol between a client and a server. It provides both parties with a set of methods and status codes to express different semantics. Client methods include retrieval and posting of data (`GET` and `POST`), which are the two basic actions performed when browsing the Web. Status codes allow servers to describe the success of a request (for example, `200 OK`, followed by the requested document, image or sound), and also authorization or payment requirements, redirection to another resource, and server or client errors (including the famous `404 Not Found`). Another code, rarely used in existing applications, indicates that the request succeeded but didn't generate any output for the client (`204 No content`). This feature of HTTP makes it possible to issue commands that do not retrieve data from the server. In a mediaspace server, these commands can control the switching of analog audio and video connections, or the movement of a remote camera. Access to the mediaspace is then achieved by including links such as `<a href="http://mediascape/connect_X_to_Y"> X-Y </a>` in any HTML document.

HTTP servers usually respond to each client independently of previous requests by the same client. A state management mechanism known as cookies [25] lets clients and servers place requests and responses within a larger context. A cookie set by the server in a response contains information that the client should transmit back in subsequent requests to that server. Generally, this information is a unique ID that lets the server restore the client's context. This notion of context simplifies the naming of the mediaspace's resources (services). A mediaspace server based on HTTP can use cookies to store the identity and location of users. This way, a unique resource `/connectWith.Y` can be used instead of multiple `/connect_X_to_Y` as mentioned previously. This lets different users use the same HTML document containing the appropriate URLs as an interface to the mediaspace services.

When an HTTP server receives a `GET` request, it usually sends back the corresponding resource and then closes the connection. A mechanism known as "server push" [26] can take advantage of a connection held open over multiple responses, so the server can send more data when available, every new piece of data replacing the previous one. This mechanism can be used to send a series of images instead of a single image. With this feature, HTTP can transmit live pictures and video streams from a mediaspace server to existing clients without modifying them or adding a plug-in. This makes digital video sources available to nearly everyone connected to the Internet.

The integrability of a mediaspace requires easy access to its services over time and space. By implementing the mediaspace software as HTTP servers, we can make analog connection management and digital video services available to any existing Web-aware application. By embedding HTML commands —URLs pointing to the servers—, we can make these services accessible from any Web document. These are two important steps towards the integration of the mediaspace interface into existing work environments.

8.3.2. FLEXIBILITY USING HTTP AND HTML

An HTTP URL usually has the following form:

```
http://host:port/path?querystring#fragment
```

The optional query string specifies parameters that control how the server handles the request, whereas the optional fragment consists of additional reference information to be interpreted by the client after the retrieval action has been successfully completed. For example, the query string parameters can control the duration of a connection (`http://mediascape/glance.nicolas?duration=3`) or request an image with a given resolution by specifying a zoom factor (`http://mediascape/glance.nicolas?zoom=4`).

In addition to letting users specialize requests, HTTP offers three types of intermediaries between a server and a client for composing a request/response chain:

- *Proxies* are forwarding agents. They receive requests, rewrite all or part of the message, and forward the reformatted request to the original server.
- *Tunnels* act as a relay point between two connections without changing the messages.
- *Gateways* act as a layer above some other server, translating the requests to the underlying server's protocol.

These intermediaries let developers customize an HTTP-based system. They can compose them to create new services from existing ones, for example adding a proxy to implement an access policy or a gateway to an ISDN videoconferencing system. They can configure or replace each element at any time without affecting the others. This lets developers share their experience by exchanging these elements. They can also benefit from existing clients and servers, and overcome possible incompatibilities by inserting intermediaries, for example, creating a tunnel to go through a security firewall.

The interface of EuroPARC's RAVE mediaspace used Buttons [27], a system based on end-user tailorable objects. Some of this systems characteristics are found in the way people create and edit HTML documents: different classes of users (worker, tinkerer, handyman, programmer) share their experience by begging, borrowing or stealing pieces of HTML from others. The current architecture of the Web encourages such forms of reuse, since anyone can view the source of an HTML document and copy and paste parts of it without understanding the details of how it works. The tolerant use of HTML tags by current browsers supports this reuse. The use of HTML-based interfaces to mediaspace services supports a tailoring culture based on existing skills and work practices, rather than the development of new ones. Thus, we can have interfaces that are easier to start, learn, operate and customize.

Using HTTP as a transfer and command protocol and HTML to describe the interface offers some level of tailorability to the users. Although perhaps insufficient in the long term, it allows quick creation of easily tested and modified prototypes.

8.3.3. PRIVACY USING HTTP AND HTML

Publishing information on the Web is like putting documents in the middle of the street, hoping that someone will see them. But it is hard to tell if anyone sees them and if so, who they are. Users need two things to communicate efficiently on the Web without exposing their privacy: they need to know who is retrieving the information they publish and to be able to deny access or adapt the information according to that knowledge.

Despite the saying that "On the Internet, nobody knows you're a dog", some information about the client always goes along with an HTTP request, often without the user's consent or knowledge. HTTP communication usually takes place over TCP/IP connections. Even before the client sends the first byte, an HTTP server already knows the name or IP address of the remote host from the underlying TCP connection. It can use this name or address to query the login name of the remote user (for example, through the Ident [28] daemon). This login name can then be used to inquire further details about the user, such as his or her real name and email address, through SMTP [29] or the Finger [30] protocol (Figure 2).

```
Remote host     : sgi5.lri.fr
Remote TCP port : 7832

Login name      : roussel

Real name       : Nicolas Roussel
Email address   : roussel@sgi5.lri.fr

Finger info     :

  Login name: roussel         In real life: Nicolas Roussel
  Directory: /u/roussel       Shell: /bin/sh

  On since Sep  7 10:12:11 on ttyq0 from :0.0
  4 minutes 3 seconds Idle Time

  No unread mail

  Plan:
    See my web page (http://www-ihm.lri.fr/~roussel/)
```

Figure 2. Sample information gathered from a TCP connection

The HTTP server also gets information from the request sent by the client. This includes the name of the client software (possibly including operating system name and version), the types of media or encoding it can handle, the URL of the last resource accessed by the client, a cookie obtained from a previous request, or authentication information (Figure 3).

```
User-Agent : Mozilla/4.61C-SGI [en] (X11; I; IRIX64 6.5 IP30)
Accept : image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png, */*
Accept-Encoding : gzip
Accept-Charset : iso-8859-1,*,utf-8
Accept-Language : en
Referer : http://www-ihm.lri.fr/
Cookie : LOCATION="office228"
Authorization: Basic bWJsOndlbmR4
```

Figure 3. Sample information sent by an HTTP client

Most of the Web servers employed today do not use this information. At best, they record it into log files. Users usually don't know where these log files are stored, and getting the useful data out of them often requires running scripts. However, having all or part of this information, it is usually easy to guess the identity of the person or process that made the request, or at least to know whether it comes from a known source. An HTTP server specifically designed to ensure users' privacy could trigger notification and control mechanisms before handling requests. These mechanisms would let users decide of the reply, based on the identity or location of the requester and the description of the request itself.

8.4. MEDIASCAPE

Since 1993, our group at LRI Université Paris-Sud has been working in a mediaspace that connects our offices with audio/video links. Over these years, we have developed several software prototypes for low-level analog switching, digital video transmission, abstract services and collaborative session management, access control and notification. Mediascape is one of the prototypes we have developed to explore the ideas described in the previous section. It is a mediaspace based on an analog audio/video network connecting six workspaces (or nodes), several public spaces, a VCR and one of our workstations used to digitize analog images.

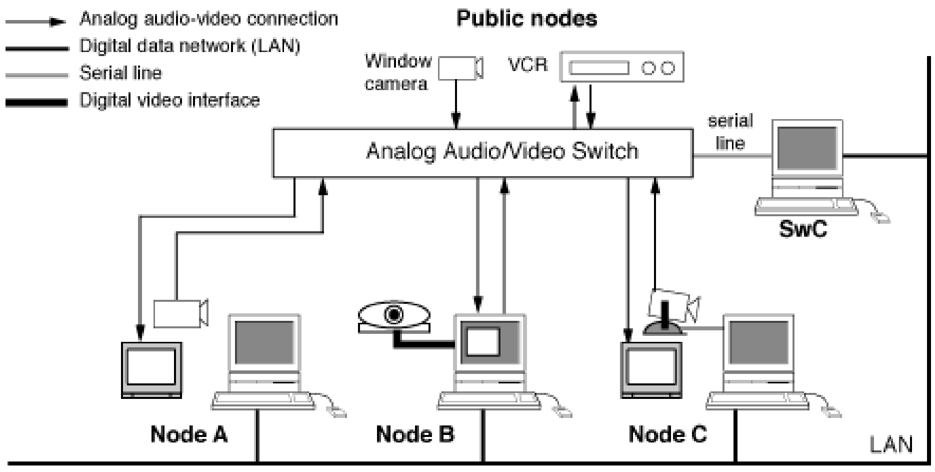


Figure 4. Mediascape hardware configuration

Figure 4 shows Mediascape hardware configuration. For simplicity, it shows only three nodes and does not include audio equipment. Node A is an analog node with the camera and monitor connected to the switch. Node B, a digital node, uses a digital camera and displays video on the computer screen. It also sends and receives analog video through a video digitizing board. Node C is an analog node where the workstation can control the orientation and zoom of the camera through a serial line. The public nodes consist of a window camera and a VCR. SwC marks the workstation that controls the switch through a serial line. It runs the connection server used by other nodes to establish connections.

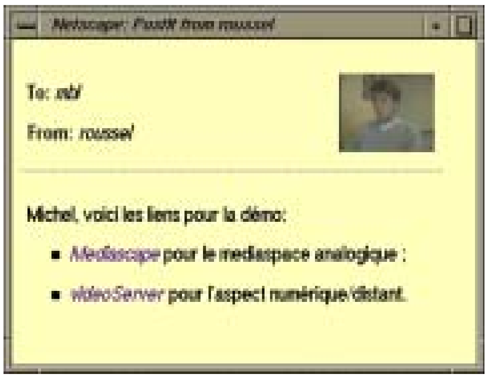
Mediascape offers the following services to our group:

- `register`, to inform the system of the user's current location;
- `glance`, a bi-directional analog video-only connection lasting a few seconds;
- `call`, a bi-directional analog audio/video connection, lasting an unknown duration (like video phone);
- `authlevel`, to choose between three levels of accessibility (everything, glance only, or nothing).

In addition to these services available to local people only, two other services are publicly available:

- `postit`, to allow other users to leave messages on our computer screens;
- `grab`, to get a frame-grabbed still image from one of our nodes.

The current implementation of Mediascape consists of two custom HTTP servers: one for analog connection management and image digitizing, the other for driving a computer-controlled video camera. Digital images are captured on a per-request basis. They are available in different sizes, from 80x60 to 640x480 pixels, and can be gamma corrected (specifying these with a query string). PostIt messages composition employs an HTML form sent by the connection server and can contain HTML code. Commands automatically added to the message include the sender's name and a snapshot that can be used to call back the sender (Figure 5).



To: *mbl*

From: *roussel*

Michel, voici les liens pour la démo:

- *Mediascape* pour le mediaspace analogique ;
- *videoServer* pour l'aspect numérique/distant.

Figure 5. PostIt sample

8.4.1. MEDIASCAPE INTEGRABILITY

The connection server uses resource names containing only the service and the callee names (for example, `/call.nicolas`). People must identify themselves with their name and location the first time they use the mediaspace. The server stores their location and sets a cookie in their client containing the user's name, which accompanies any subsequent request. Thus, when the server receives a request for `/call.nicolas`, it also contains a cookie such as `NAME=Paul`, telling who wants to call Nicolas. If the request does not contain the cookie, the server returns a user identification form to the client. Users can change their location by issuing a `register` command (such as `/register.office228`). The connection server also permits requesting several resources at once: `/glance.paul/glance.michel`.

An HTML document can contain references to other resources in a number of ways, such as images, hypertext anchors or embedded objects. We can use existing Web browsers to build a mediaspace interface by including the appropriate code in an HTML document. For example:

- Include a snapshot grabbed upon retrieval of the document

  ```
  <img src="http://mediascape/grab.nicolas">
  ```

- Add links to allow people to call me or my officemate

  ```
  Call <a href="http://mediascape/call.nicolas"> me </a>
  or <a href="http://mediascape/call.paul"> Paul </a>
  to know where is office 228
  ```

- Combine the two previous examples to include a snapshot that people can click on to call me

  ```
  <a href="http://mediascape/call.nicolas">
  <img src="http://mediascape/grab.nicolas">
  </a>
  ```

- Add JavaScript code to a text link that will execute a glance when the mouse passes over it

```
<a
href=http://www-ihm.lri.fr/~roussel/
onMouseOver="window.location='http://mediascape/glance.nicolas"
> N. Roussel </a>
```

- Add a link to compose a postit for Paul

```
<a href="/postit.paul"> Note for Paul </a>
```

- Add a command that executes several glances in sequence

```
<embed
src="http://mediascape/glance.michel/glance.stephane">
```

The default interface to Mediascape is an HTML document that displays still images of the different users (Figure 6). The server alters these images to reflect users' accessibility. Three icons represent door states, as in CAVECAT [8]. Each door state (open, ajar or closed) corresponds to an authorization level (`authlevel` service) controlled by the user. A user can glance at others by moving the mouse over their name or call them by clicking on the snapshot. Two other icons provide access to the PostIt service and a mail gateway. The document also contains metainformation that tells the browser to reload it after a few minutes to keep it up to date. As explained previously, the use of cookies in the connection server and the naming of resources (for example `/call.nicolas` instead of `/X.call.nicolas`) allow everyone to use the same document.

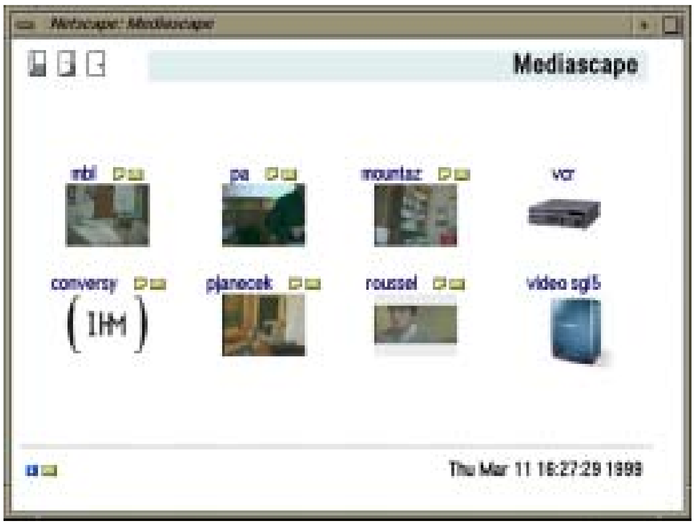
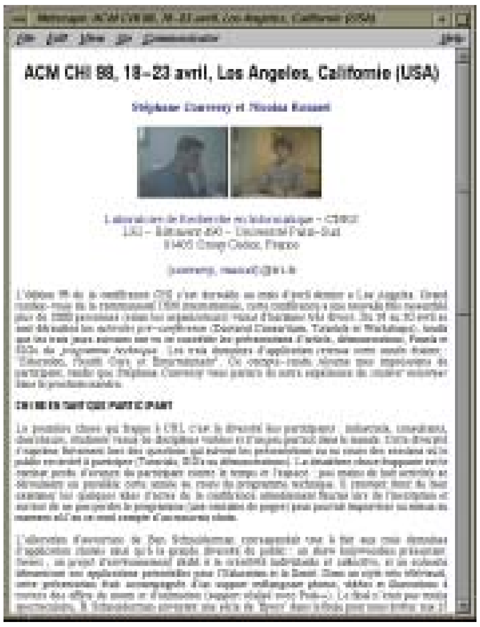


Figure 6. Basic interface to Mediascape.
Conversy has closed his door and Roussel's is ajar.
The two rightmost icons give access to public nodes (a VCR and a digital video source)

8.4.2. MEDIASCAPE FLEXIBILITY

Since Mediascape's basic interface is an ordinary HTML document, users familiar with HTML authoring can save a copy and modify it to create their own personal interface. Users could for example select a subset of the users, remove icons to save screen space, or add icons that allow panning, tilting and zooming with the remote camera. This type of adaptation —described as surface customization of the interface [19]— lets users choose between a number of predefined options, determined by the capabilities of HTML and other related languages such as JavaScript or VRML.

So far, we have only seen *dedicated* HTML documents used to interface our mediaspace. However, HTML code requesting Mediascape resources can also be added to *existing* documents. When collaborating on a project, people can add snapshots and Mediascape commands to the project description. This allows any project member to know who is around and easily make connections every time he checks the current state of the project. This assists coordination: in a co-authoring situation, the authors can include a connect command in the document (Figure 7). Each time one of the authors reads or works on the document, the connection with the other author is automatically established. As an increasing number of email applications understand HTML and HTTP, mediaspace commands can also be included in a message and executed when the receiver reads it, again providing implicit coordination between users.



ACM CHI 98, 18–23 avril, Los Angeles, Californie (USA)

Stéphane Conversy et Nicolas Roussel

Laboratoire de Recherche en Informatique – CNRS
LRI – Bâtiment 490 – Université Paris-Sud
91405 Orsay Cedex, France

{conversy, roussel}@lri.fr

[illegible]

Figure 7. Coordination between co-authors through a Mediascape-aware document. Here, we can see that both authors are present at the same time

8.4.3. MEDIASCAPE PRIVACY

The Mediascape servers log all requests. Users can check the log files to see if someone called in their absence. They can also use applications that monitor these files and deliver feedback when someone requests a connection or an image from the Web. For example, they could use nonspeech audio to differentiate between callers.

For privacy purposes, a request for a still image grabbed by the video digitizer is treated like a glance, because these requests usually come from distant users who do not have access to our analog audio/video network. Whereas analog services are based on a strict reciprocity rule, image digitizing uses a relaxed version of this relationship because remote users can see us but we can't see them. We use the information gathered from the HTTP connection to identify the caller and forward this information to the callee. If someone repeatedly asks for images, it is usually easy to let them know we are aware of their presence by gesturing, showing a message on a piece of paper, or sending an email.

8.5. VIDEOSERVER

Mediascape supports local audio/video connections as well as remote access to live pictures, but not remote live video or audio. To support video communication over long distance with colleagues, friends or relatives, we have designed and implemented videoServer, a software component for both interpersonal and interprocess digital video transmission.

The first videoServer prototype was an extension of our Web server (a CGI script). The current implementation is a custom HTTP server written in C++ that can run efficiently on SGI workstations —ports are underway on Macintosh, MS-Windows and Linux platforms. It encodes live or prerecorded video as a series of JPEG images (also known as Motion-JPEG or MJPEG) and sends them using the server push mechanism described earlier.

VideoServer offers the following services:

- `/photo` captures a single live image;
- `/video` produces a series of live images;
- `/file/test` sends a video file named "test" in the directory from which videoServer was launched.

8.5.1. VIDEOSERVER INTEGRABILITY

Since most Web browsers can display a server-pushed sequence of images in place of an ordinary image without any plug-in, videoServer allows a large number of users to get live video. The frame rate depends on the available bandwidth. A typical 160x120 JPEG image is about 3 Kbytes and we routinely get 10 to 15 frames per seconds, even over long distances. A snapshot image, live or recorded video can be included in place of any image in an HTML document simply by using links of the following types:

```
<img src="http://videoserver/photo">
<img src="http://videoserver/video">
<img src="http://videoserver/file/test">
```

We have distributed videoServer outside the lab to experiment with bi-directional connections with friends and colleagues. We often use these long distance digital video links in conjunction with third-party applications such as text chat or audio broadcasting, or with regular phone calls.

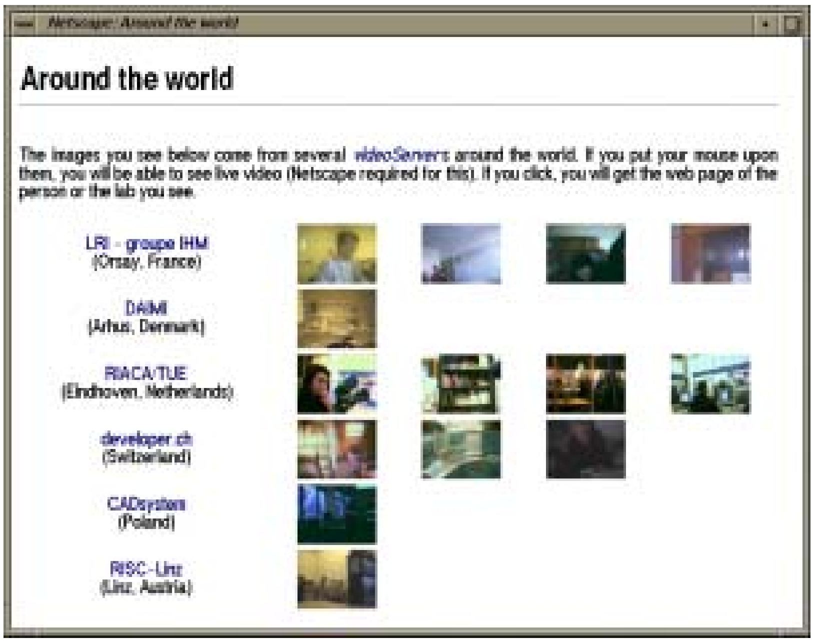
8.5.2. VIDEOSERVER FLEXIBILITY

Clients can use a query string to specify the compression ratio and zoom factor of live images (for example `/photo?zoom=4&cratio=20.0`). These two parameters let clients adapt their requests to the available bandwidth by reducing the resolution or augmenting the compression ratio. For live video, two extra parameters are available to specify the number of images requested and the time to wait between two subsequent images. The former can be used to create a glance connection, the latter to create awareness views updated at a slow rate.

More sophisticated HTML code can be used. For example, the following code uses JavaScript to insert a snapshot that turns into live video when the cursor is over it and turns back to a snapshot when the cursor leaves it:

```
<a
href=http://www-ihm.lri.fr/
onMouseOver='document.img1.src="http://videoserver/video"'
onMouseOut='document.img1.src="http://videoserver/photo"'>
<img name="img1" src="http://videoserver/photo" >
</a>
```

This code can be used to create an awareness view from several videoServers (Figure 8). This document differs from traditional awareness views in two ways. First, users can freely modify the set of images, which is not restricted to a list of registered people. Second, the JavaScript code allows two-degree awareness by switching between still and live images.


Netscape: Around the world

Around the world

The images you see below come from several videoServers around the world. If you put your mouse upon them, you will be able to see live video (Netscape required for this). If you click, you will get the web page of the person or the lab you see.

LRI - groupe IHM (Orsay, France)

DAIMI (Arhus, Denmark)

RIACA/TUE (Eindhoven, Netherlands)

developer.ch (Switzerland)

CADsystem (Poland)

RISC-Linz (Linz, Austria)

Figure 8. HTML document presenting images from several videoServers

8.5.3. VIDEOSERVER PRIVACY

For every request it receives, videoServer executes an external notifier program with arguments indicating the name of the client's machine, possibly the sender's login name, the requested service and the values of the query string arguments. The notifier sends back the description of the service to execute, which can differ from the one the client requested. This simple mechanism lets people control the available information about themselves and how other users can access it.

The default notifier, a UNIX shell script, allows users to easily define different access policies. For example, different sounds can reflect the requested service or the address of the remote client, implementing a form of caller ID. The notifier program can also control image access. Since the notifier can redefine the service to execute, it lets users change the resolution, quality, number of images and refresh rate. In particular, a very high compression factor generates a highly degraded image that still provides useful information, such as the number of people present (Figure 9). Such degraded images can be used for example for requests issued by unidentified users. Another interesting use of service redefinition for privacy is the ability to send a prerecorded sequence instead of live video, for example, a short clip showing that the person is away or busy.

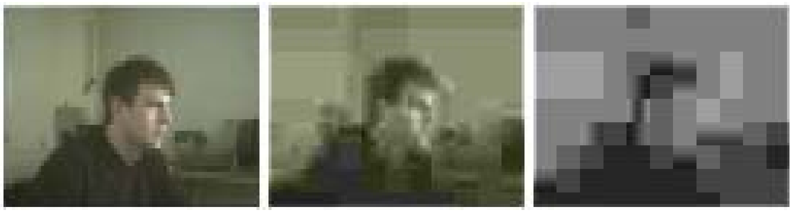


Figure 9. Degrading image quality by increasing compression

When the request does not specify the number of images for live video, videoServer limits it to 5000 (that is, up to three minutes). This ensures that constant monitoring cannot take place without periodically asking permission. Thus, when every service has an associated auditory notification, repeated requests quickly gain notice, as if someone stood watching through the window while leaving a finger on a doorbell.

8.6. VIDEOSPACE

To access Mediascape and videoServer, end-users can create dedicated HTML documents or modify existing ones. This approach supports quick prototyping of high-level services by cutting and pasting HTML code or exchanging files and emails. However, browsers' capabilities and the description languages they use limit this document-centric approach. In some cases, we may want to use complex data and input/output techniques —for example, touchpad, sound or shaped windows— not available in HTML browsers. We may also want to access or control the mediaspace from within other existing applications, such as a shared editor.

In order to extend the accessibility of mediaspace services, we have developed videoSpace, a software toolkit designed to facilitate the integration of live video into existing or new applications for the purpose of supporting collaborative activities. The first goal of videoSpace is to promote the development of collaborative environments where communication facilities are embedded in the applications rather than provided as separate applications. The second goal of videoSpace is to support novel uses of video by supporting real-time filtering of live video images.

8.6.1. TOOLKIT OVERVIEW

The videoSpace toolkit is implemented in C++, with Tcl/Tk [31] and Python [32] bindings also available. The design of the toolkit has been driven by the principle that simple things should be simple and complex things should be possible. Creating a video connection requires a few lines of code; managing multiple sources and including video processing is not much more complicated. For a detailed description of the toolkit, see [33].

VideoSpace is organized around the following basic concepts:
- Images: rectangular pixmaps in various formats;
- Image sources: objects that generate images;
- Image filters: algorithms that transform or analyze images;
- Image sinks: visible representations of images;
- Multiplexers: control objects that can wait on several sources and sinks simultaneously to multiplex several streams.

Two kinds of image sources are available: local sources (local files and digitizing hardware) and network sources. Network sources are servers generating series of JPEG images. Naturally, videoServers are the most common network sources we use. However, we also use anonymous WebCams available on the Internet that also use the server push protocol.

In addition to the server push streaming mechanism, two alternative protocols have been added to videoServer. Instead of sending the images over the TCP connection associated with the HTTP request, videoServer can send them as unicast or multicast UDP datagrams, using the TCP connection as a signaling channel. UDP can be more efficient than TCP over long distance and is well adapted to live video: lost packets are not retransmitted, therefore achieving the best possible frame rate and lag according to the available network bandwidth. However, these two UDP-based protocols are available only to videoSpace applications, not to traditional Web browsers.

Image sources are described by an encoding —a pixel format— and a URL. Image source objects are created from their URL by a factory object. When a new source type is added to the toolkit (for example, if a new protocol is implemented in videoServer), the factory object is modified to handle the corresponding URLs and all applications benefit from it without any other change. In addition to standard HTTP URLs used to access server push sources, two new URL schemes have been introduced for UDP unicast and multicast protocols (`vstp` and `vsmp`). When given one of these URLs, the image source factory is responsible for sending the appropriate HTTP request to the videoServer, adding information to describe the UDP protocol to use as well as the host (or group) address and port number. For example, the URL

```
vstp://sgi5.lri.fr/video?zoom=2
```

accessed from the machine `sgi3` would result in an HTTP request like

```
GET /video/udp?zoom=2&host=sgi3.lri.fr&port=8965 HTTP/1.0
```

sent to the videoServer running on machine `sgi5`.

Every videoSpace application can be seen as a custom HTTP client able to retrieve video images from several network servers and local sources at the same time and display or process these images in an arbitrary way. These applications supplement the document-centric approach described in the previous two sections in a consistent way: the same HTTP URLs can be used in HTML documents and videoSpace applications to access videoServers, which will use the same notification and control mechanisms, regardless of the origin of the request.

8.6.2. SAMPLE APPLICATIONS

Several applications have been developed with videoSpace to allow users to display video streams outside HTML documents. The simplest client, videoClient, displays a single stream in a window on the computer screen that the user can resize. This application can be used for focused forms of communication, for example accompanying a phone call (Figure 10).

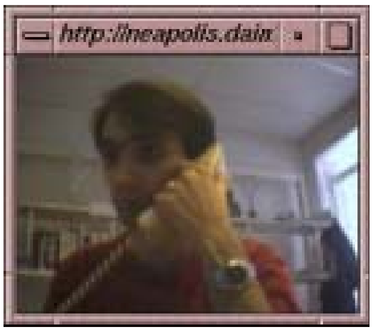


Figure 10. videoClient

Several other applications take advantage of the analog video output capability of some workstations to display one or more video streams on a TV monitor. One such application has been used to install a mediaspace between two common spaces at Aarhus University (Figure 11).

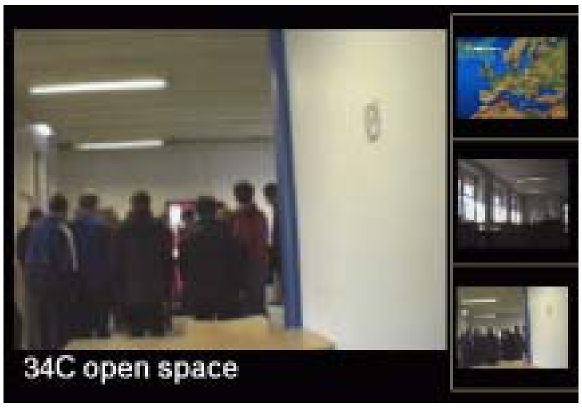
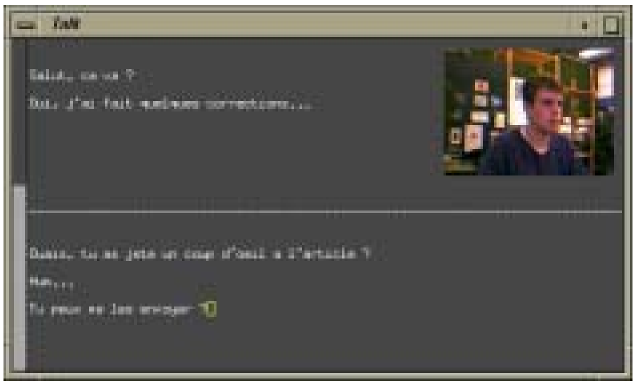


Figure 11. Snapshot of an analog video output showing a weather map updated from CNN's web site and live video from two common spaces

VideoClient can take advantage of the architecture of the X Window system to display video in a new subwindow of a running application, rather than in a new window. With a few lines of Unix shell commands, it is possible to add video to any running application. Figure 12 shows an *xterm* window running the Unix *talk* program with an embedded videoClient. Since the video window is a child of the xterm window, it is moved, raised, lowered and iconified with it. In addition, interaction with the video window (for example, clicking in it) is handled by videoClient, not the host application.



```
Salut, ça va ?
Oui, j'ai fait quelques corrections...

Ouais, tu as jeté un coup d'oeil à l'article ?
Hum...
Tu peux me les envoyer ?
```

Figure 12. Vtalk, a video "augmented" talk

This approach can also be used in conjunction with user interface toolkits that explicitly support widgets that host external applications. For example, the frame widget of the Tcl/Tk toolkit can host a separate application by setting the widget's 'container' property to true. We have developed a series of simple Tcl/Tk applications based on that feature. One such application provides a lightweight awareness view showing local Mediascape users and remote videoServers. It consists only of a title bar that reveals recent snapshots of the users when the cursor passes over it (Figure 13). We also have integrated videoClient with GroupKit [34], a Tcl/Tk-based toolkit for developing groupware applications. With a few lines of code, any GroupKit application such as a shared drawing editor can be "augmented" with video links between the participants.

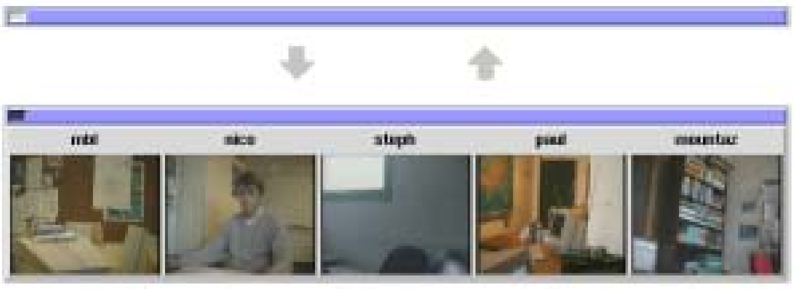


Figure 13. Lightweight awareness view

Developers can also use the C++ API of videoSpace to implement their own video-enabled client applications. In particular, with the advent of digital video and the increasing CPU power of computers, the images of a video stream can be analyzed in real-time to extract useful information. VideoSpace supports video filters that can transform or analyze the video frames as they go through the application. Previous work on mediaspaces shows that such video filters can enhance their services. Examples include remote camera control by local movement detection [35], addition of recent activity representation to digitized images [36], and to support privacy, substitution of a shadow for a person [37], or eigen-space filtering [38]. VideoSpace supports several techniques based on simple image difference and partition algorithms to extend collaborative applications (including Mediascape). These extensions include context capture —knowing whether a user is present or absent—, motion detection, and image segmentation to support natural annotation and gesturing [33].

VideoServer is an HTTP server. The videoSpace applications presented so far are HTTP clients. Other applications can be implemented that correspond to other roles in the HTTP request/response chain. For example, a proxy could compose several image sources and make the resulting composition available as a single video stream. A gateway could translate between the UDP-based protocol and the server-push protocol, so that video displayed in HTML documents would also benefit from the faster frame rate provided by UDP.

8.7. SUMMARY

We have introduced three important principles for the design of a mediaspace: integrability, flexibility and privacy. We have described how to use the Web standards and protocols to implement the software infrastructure and the user interfaces of a mediaspace with respect to these principles. We have presented Mediascape, our Web-controlled analog mediaspace, videoServer, a Web server dedicated to digital video communication, and videoSpace, a toolkit based on videoServer to integrate live video into applications.

We have described how Mediascape supports a document-centric approach through the design and use of different interfaces. Commands for the mediaspace services can be integrated into any HTML document, making it easily accessible, intuitive to use and easy to adapt and customize. We have described how videoServer expands our local mediaspace to distant people, again using a document-centric approach, by taking advantage of the simple server push mechanism to bring live video into Web browsers. We have shown how the notification and control mechanisms of videoServer provide a good balance between this high degree of accessibility and the need for privacy. VideoSpace is an example of an application-centric approach of the Web: by developing applications as a set of custom HTTP clients, servers or intermediaries, it is possible to overcome the limitations of Web browsers but still be compatible with them.

The document-centric approach has proved useful for informal communication between distant people. This is not surprising. The properties we might choose to describe daily Web usage in a workplace would probably be close to the ones that describe informal communication: frequent, brief, intermittent, unscheduled, lacking formal openings or closings. The application-centric approach offered by videoSpace allows to move towards more complex or focused forms of communication, such as traditional videoconferencing tools or collaborative applications.

The notification and control mechanisms implemented in videoServer are rather unusual for an HTTP server. Most traditional servers are designed to bring generic information online, regardless of who is making it available or who is retrieving it. In order to communicate through the Web, people have to make trade-offs: when you put a picture of your house, your dog or your children on the Web, it is accessible to your friends and relatives, but also to perfect strangers. People usually accept this, sometimes naively thinking that if the picture is not referenced anywhere, it will be accessible only to those who have the exact URL. We believe that simple content negotiation and access notification based on the user or process that made the request could lead to more subtle forms of communication between individuals. Our current work investigates the generalization of such mechanisms to other media than video (text documents for example) and to evaluate the impact of these mechanisms on client and server architectures as well as Web protocols.

Detailed information about videoSpace and its availability can be obtained from `http://www-ihm.lri.fr/~roussel/videoSpace/`

ACKNOWLEDGMENTS

Thanks are due to Michel Beaudouin-Lafon for the inspiration and motivation for the development of Mediascape, videoServer and videoSpace. I thank Stéphane Conversy for the many discussions we had about the implementation of HTTP clients and servers. I also want to acknowledge the different people around the world who have used videoServer and thank them for their enthusiasm and collaboration. This work was partially funded by CNET-France Telecom.

REFERENCES

1. R. Stults. "Media Space". Technical report, Xerox PARC, 1986.
2. S.A. Bly, S.R. 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. E. Mackay. "Media Spaces: Environments for Informal Multimedia Interaction". In M. Beaudouin-Lafon, editor, Computer-Supported Co-operative Work, Trends in Software Series. John Wiley & Sons Ltd, 1999.
4. E. A. Isaacs, S. Whittaker, D. Frohlich, and B. O'Conaill. "Informal communication re-examined: New functions for video in supporting opportunistic encounters". In K. E. Finn, A. J. Sellen, and S. B. Wilbur, editors, Video-mediated communication. Lawrence Erlbaum Associates, 1997.
5. C. Cool, R.S. Fish, R.E. Kraut, and C.M. Lowery. "Iterative Design of Video Communication Systems". In Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work, Toronto, Ontario, pages 25-32. ACM, New York, November 1992.
6. W. Buxton and T. Moran. EuroPARC's Integrated Interactive Intermedia Facility (IIIF): Early Experiences. In Multi-User Interfaces and Applications, pages 11-34. S. Gibbs and A.A. Verrijn-Stuart, North-Holland, September 1990. Proceedings of IFIP WG8.4 Conference, Heraklion, Greece.
7. W.W. Gaver, T. Moran, A. MacLean, L. Lövstrand, P. Dourish, K. Carter, and W. Buxton. "Realizing a Video Environment: EuroPARC's RAVE System". In

Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems, pages 27-35. ACM, New York, 1992.

8. M.M. Mantei, R.M. Baecker, A.J. Sellen, W.A.S. Buxton, and T. Milligan. "Experiences in the Use of a Media Space". In Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems, pages 203-208. ACM, New York, 1991.
9. K. Watabe, S. Sakata, K. Maeno, H. Fukuoka, and T. Ohmori. "Distributed multiparty desktop conferencing system: MERMAID". In Proceedings of ACM CSCW'90 Conference on Computer-Supported Cooperative Work, pages 27-38. ACM, New York, October 1990.
10. H. Gajewska, J. Kistler, M.S. Manasse, and D.D.Redell. "Argo: A System for Distributed Collaboration". In Proceedings of Multimedia 94, pages 433-440. ACM, New York, October 1994.
11. J.C. Tang and M. Rua. "Montage: Providing Teleproximity for Distributed Groups". In Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems, pages 37-43. ACM, New York, April 1994.
12. 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.
13. 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.
14. W.W. Gaver. "The Affordances of Media Spaces for Collaboration". In Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work, Toronto, Ontario, pages 17-24. ACM, New York, November 1992.
15. W. Buxton. "Living in Augmented Reality: Ubiquitous Media and Reactive Environments". In K. Finn, A. Sellen, and S. Wilber, editors, Video Mediated Communication. Lawrence Erlbaum Associates, 1997.
16. M. Beaudouin-Lafon. "Beyond the Workstation, Media Spaces and Augmented Reality". In People and Computers IX, pages 9-18. Cambridge University Press, August 1994. Opening plenary session at HCI'94 (Glasgow, UK).
17. J. Grudin. "Groupware and social dynamics: Eight challenges for developers". Communications of the ACM, 37(1):92-105, January 1994.
18. L. Suchman. "Office procedures as practical action: Models of work and system design". ACM Transactions on Office Information Systems, 1:320-328, 1983.
19. R. Bentley and P. Dourish. "Medium versus mechanism: Supporting collaboration through customization". In Proceedings of European Conference on Computer-Supported Cooperative Work ECSCW'95, Stockholm, pages 133-148. Kluwer Academic, September 1995.
20. M. Roseman and S. Greenberg. "Building Flexible Groupware Through Open Protocols". In ACM Conference on Organizational Computing Systems, California, pages 279-288. ACM Press, October 1993.
21. Oliver Stiemerling. "Supporting Tailorability in Groupware through Component Architectures". In Proceedings of the ECSCW '97 Workshop on Object Oriented Groupware Platforms, Lancaster, GB, pages 53-57, September 1997.
22. G.H. ter Hofte. "Working apart together : Foundations for component groupware". Number 001 in Telematica Instituut Fundamental Research Series. Telematica Instituut, Enschede, the Netherlands, 1998.
23. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. "Hypertext Transfer Protocol - HTTP/1.1". Standards Track, RFC 2616, IETF Network Working Group, June 1999.
24. 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/.
25. D. Kristol and L. Montulli. "HTTP State Management Mechanism". Proposed Standard, RFC 2109, IETF Network Working Group, February 1997.

26. "An Exploration of Dynamic Documents". Technical report, Netscape Communications, 1995. http://home.netscape.com/assist/net_sites/pushpull.html.
27. A. Maclean, K. Carter, L. Lövstrand, and T. Moran. "User tailorable systems: Pressing the issues with Buttons". In Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems, Seattle, pages 175-182. ACM, New York, April 1990.
28. M. St Johns. "Authentication Server". RFC 931, IETF Network Working Group, January 1985.
29. J. Postel. "Simple Mail Transfer Protocol". RFC 821, IETF Network Working Group, August 1982.
30. D. Zimmerman. "The Finger User Information Protocol". RFC 1288, IETF Network Working Group, December 1991.
31. Tcl/Tk Home Page, Scriptics Corporation. http://www.scriptics.com/
32. Python Language Home Page. http://www.python.org/
33. N. Roussel and M. Beaudouin-Lafon. "VideoSpace: A Toolkit for Building Mediaspaces". Research report 1216, LRI, Université Paris-Sud, France, May 1999.
34. M. Roseman and S. Greenberg. "Building Real Time Groupware with GroupKit, A Groupware Toolkit". ACM Transactions on Computer-Human Interaction, 3(1):66-106, March 1996.
35. W.W. Gaver, G. Smets, and K. Overbeeke. "A Virtual Window On Media Space". In Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems, Denver, pages 257-264. ACM, New York, May 1995.
36. A. Lee, A. Girgensohn, and K. Schlueter. "NYNEX Portholes: Initial User Reactions and Redesign Implications". In Proceedings of GROUP'97, Phoenix, pages 385-394. ACM, 1997.
37. S. E. Hudson and I. Smith. "Techniques for Addressing Fundamental Privacy and Disruption Tradeoffs in Awareness Support Systems". In Proceedings of ACM CSCW'96 Conference on Computer-Supported Cooperative Work, Boston, Mass., pages 248-257. ACM, New York, November 1996.
38. J. Coutaz, F. Bérard, E. Carraux, and J. Crowley. "Early experience with the mediaspace CoMedi". In IFIP Working Conference on Engineering for Human-Computer Interaction, Heraklion, Crete, 1998.