Mediascape: A Web-Based Media Space

Nicolas Roussel
Université Paris-Sud

Mediascape provides a Web-based media space—an audio, video, and computing environment that supports distributed groups. Using Web standards and protocols it supports creating a software infrastructure that incorporates three important design principles: integrability, flexibility, and privacy. The Videoserver component handles digital video communication. This Web-based environment offers benefits to users and developers.

In the mid-1980s Stults coined the term "media space" at Xerox PARC.[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]

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. 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). Some recently developed media spaces rely on a digital audio/video network such as Integrated Services Digital Network (ISDN), local area network (LAN), or asynchronous transfer mode (ATM).

Media spaces provide users with different kinds of services, including

- *Awareness view*:[4] 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.

Different approaches ensure users' privacy in these augmented environments. Methods range from a strict reciprocity rule to explicit negotiation or user-defined rules, for access control and notification.

Since 1993 my group at the Laboratory for Computer Science (LRI) has worked in a media space that connects our offices with analog audio/video links. We have developed several software prototypes to control this analog network, from low-level switching to abstract services, collaborative session management with associated shared applications, and access control and notification architectures.

This article focuses on the design of a media space software infrastructure and user interface, based on lessons learned from previous use of these environments. I introduce three key principles for the success of a media space and show how to use existing Web standards and protocols to create software media space components consistent with these principles. Finally, I present Mediascape, our Web-based media space, and Videoserver, a component for interpersonal and interprocess digital video communication.

Learning from existing media spaces

Gaver's analysis of the affordances of media spaces shows that devices' physical characteristics modify the way users perceive and act in such spaces.[5] Following this approach and building on previous experience using these environments, I have identified three important principles for media space design:

- *Integrability*. A media space is integrable to the degree it supports existing practices and tools, rather than imposing new ones on the user.
- *Flexibility*. A media space is flexible to the degree its components can be repurposed to create new uses, with little effort.
- *Privacy*. A media space supports privacy when

1070-986X/99/$10.00 © 1999 IEEE

users can easily understand, operate, and trust mechanisms that control information available about them and how by other users can access it.

Integrability

Bly et al.[2] emphasized 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 into a dedicated room, explicitly switching between personal and group activities. On the contrary, media spaces 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, media spaces are related to ubiquitous computing and augmented reality, two research directions that aim to integrate computers into the real world.

Grudin (see Further Reading sidebar on the next page) suggested integrating groupware features with features that support individual activities and, if possible, adding them to already successful applications. In traditional desktop videoconferencing applications, which resemble videoconference rooms, users switch between the video application and other applications until they find the optimal layout for 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 media space isn't a primary activity in itself, which is how they support spontaneous interactions. However, this also makes media space interfaces difficult to design, because the frequency and variety of uses is hardly compatible with a task-specific application.

Several media space systems manage existing applications or documents. However, they tend to integrate them into their own framework, contrary to the idea of interacting with the media space in the background of other activities. A media space interface should integrate into the software environment in the same way its physical devices do in the real world.

I think that the ubiquitous/augmented approach can apply here. Interactions with the media space should not go through a single workstation and/or application. Instead, we should integrate the digital media and the interface in any existing document or application, whether individual or collective. When designing the software infrastructure of a media space, we must think in terms of components to integrate into existing practices, not in terms of new applications.

Figure 1. Typical configuration for an analog media space node.

Flexibility

The affordances of analog media spaces depend not only on the physical properties of the devices, but also on how people can use them. As Suchman pointed out,[6] people commonly improvise and repurpose their actions. Media space hardware configurations can easily be tailored by moving devices, adjusting them (for example, the volume or the 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 setups.

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

Bentley and Dourish[7] showed that the notion of medium, as opposed to mechanisms, arises from systems openness and flexibility. I think that open protocols and component architectures can serve to implement a set of basic media space software parts for developers and users. Each of these

Further Reading

For information on analog media spaces:

C. Cool et al., "Iterative Design of Video Communication Systems," *Proc. Conf. on Computer-Supported Cooperative Work (CSCW 92),* ACM Press, New York, 1992, pp. 25-32.

W.W. Gaver et al., "Realizing a Video Environment: EuroPARC's Rave System," *Proc. Conf. on Human Factors in Computing Systems (CHI 92)*, ACM Press, New York, 1992, pp. 27-35.

M.M. Mantei et al., "Experiences in the Use of a Media Space," *Proc. Conf. on Human Factors in Computing Systems (CHI 91)*, ACM Press, New York, 1991, pp. 203-208.

To explore digital media spaces:

J.C. Tang and M. Rua, "Montage: Providing Teleproximity for Distributed Groups," *Proc. Conf. on Human Factors in Computing Systems (CHI 94)*, ACM Press, New York, 1994 pp. 37-43.

H. Gajewska et al., "Argo: A System for Distributed Collaboration," *Proc. of Multimedia 94*, ACM Press, New York, 1994, pp. 433-440.

K. Watabe et al., "Distributed Multiparty Desktop Conferencing System: Mermaid," *Proc. Conf. on Computer-Supported Cooperative Work (CSCW 90)*, ACM Press, New York, 1990, pp. 27-38.

For augmented reality information:

W. Buxton, "Living in Augmented Reality: Ubiquitous Media and Reactive Environments," *Video Mediated Communication,* K. Finn, A. Sellen, and S. Wilber, eds., Lawrence Erlbaum Associates, New Jersey, 1997.

M. Beaudouin-Lafon, "Beyond the Workstation, Media Spaces and Augmented Reality," *People and Computers IX*, Cambridge University Press, Cambridge, England, 1994, pp. 9-18. Opening plenary session at HCI 94.

For more information on groupware architecture:

J. Grudin, "Groupware and Social Dynamics: Eight Challenges for Developers," *Comm. ACM*, Vol. 37, No. 1, Jan. 1994, pp. 92-105.

M. Roseman and S. Greenberg, "Building Flexible Groupware Through Open Protocols," *Proc. ACM Conf. on Organizational Computing Systems,* ACM Press, New York, 1993, pp. 279-288.

O. Stiemerling, "Supporting Tailorability in Groupware Through Component Architectures," *Proc. Workshop on Object Oriented Groupware Platforms (ECSCW 97),* Available at http://www.trc.nl/events/ecscw97oogp/papers.htm, Sept. 1997 pp. 53-57.

G.Henri ter Hofte, "Working Apart Together: Foundations for Component Groupware," *Telematica Institute Fundamental Research Series*, Number 001, Telematica Instituut, Enschede, The Netherlands, 1998, pp. 288.

parts would correspond to a well-defined system requirement: connection management, access control and notification, session management, or digital media services. 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 nonprogramming users as well.

Privacy

Asymmetric connections such as awareness views and glances are essential for providing nonobtrusive awareness of the presence and activity of other media space 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 media space is important because of the highly dynamic nature of access control to live media. Whereas specifying access rights on documents usually proves simple to specify by using read/write permissions granted by the owner, access rights on live video or audio sources proves more complex because of the multiple uses they 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. Media space software should provide users with notification mechanisms that help determine the identity and intention of the remote person. It should also give users 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 refuse the service) but should be able to describe complex behavior such as request modification before execution.

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.[8]

Integrability, flexibility, and privacy using HTTP and HTML

This section investigates the use of HTTP and HTML standards (see the sidebar) to support implementation of a media space 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 just introduced.

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`), the two basic actions performed when Web browsing. Status codes allow servers to express 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 don't retrieve data from the server. This can control the switching of audio and video connections, or the movement of a remote camera. Therefore access to the media space results from including links such as `<a href="http://mediascape/connect_X_to_Y"> X-Y </a>` in any HTML document.

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" (see http://home.netscape.com/assist/net_sites/pushpull.html) 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 makes it possible to send a series of images instead of just one. With this feature, HTTP can transmit live video to existing clients without modifying them or adding a plug-in. This makes digital video sources available to nearly everyone connected to the Internet.

HTTP servers usually respond to each client independently of previous requests by the same client. A state management mechanism known as cookies[9] 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 allows us to simplify the naming of the media space's resources (services). Since the HTTP server can deduce the caller's identity and location from the cookie, we can create a unique resource `/connectWith.Y` instead of the multiple resources `/connect_X_to_Y` mentioned previously. This lets different users use the same HTML document containing the appropriate URLs as an interface to the media space services.

By implementing media space software as HTTP servers or clients, we can make services available to any existing Web-aware application. By embedding HTML commands (that is, URLs pointing to custom servers), we can also make them accessible from any existing Web document. These are two important steps towards the integration of the media space interface into existing work environments.

Flexibility using HTTP and HTML

An HTTP URL has the following form: `http://host:port/path?querystring`. The optional query string specifies parameters that control how the server handles the request. We can use it to specify duration of a connection or request an image with a given resolution by specifying a zoom factor (`http://videoserver/photo?zoom=4`).

HTTP and HTML Defined

HTTP is an application-level protocol for transferring resources across the Internet. A resource is "a network data object or service" (see http://www.w3.org/Protocols) and is specified by a URL. HTML is a simple data format used to create hypertext documents (see http://www.w3.org/TR/REC-html40/). Together, these standards contributed to create the Web: a globally accessible and platform-independent hypermedia information system. The Web has become one of the most successful systems for communication between people. In many ways, it's becoming a central access point to applications and services. More and more applications and programming toolkits can use HTML for content description or HTTP as a transfer protocol.

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 toward 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 the system. They can compose them to create new services, available like core services (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 pass through a security firewall.

The interface of EuroPARC's Rave (Xerox Research Centre Europe's Ravenscroft Audio Video Environment) media space used Buttons,[10] a system based on end-user tailorable objects. Some of this systems characteristics arise from 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. The current architecture of the Web encourages such forms of reuse, where anyone can view the source of an HTML document and copy and paste parts without understanding the details of how it works. Current browsers' tolerant use of HTML tags support this reuse. HTML-based interfaces support a tailoring culture based on existing skills and work practices, rather than development of new ones. Thus, we have interfaces 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 both the developer and the user. Although perhaps insufficient in the long term, it allows quick creation of easily tested and modified prototypes.

Privacy using HTTP and HTML

Despite what many people believe about the Internet—reflected in the common catch-phrase "On the Internet, nobody knows you're a dog"—some information about the client usually goes along with an HTTP request, often without the user's consent or knowledge. Types of information include the name of the client software (possibly operating system name and version), types of media or encoding it can handle, and the URL of the resource that led to this server. Additional information can be obtained through the network interface: the remote host Internet address and possibly its name, and sometimes the user login name (through the Ident daemon[11]).

The client also might send back a cookie obtained from a previous request. With all this information, it's possible to identify the person or process that made the request, or at least know if it comes from a known source. In addition, HTTP provides several challenge-response authentication mechanisms that a server can use to challenge a client request. The client can then provide authentication information.

These mechanisms can build access control policies into every component, ranging from open access to restricted communities based on IP addresses, login names and passwords, or cookies.

Mediascape

Over the past five years, we have developed Mediascape, an experimental media space to explore and develop the approach described in the previous sections. Mediascape consists of an analog audio/video network connecting six workspaces (or nodes), several public spaces, a VCR, and a workstation used to digitize analog images (Figure 2).

Mediascape offers the following services to our group:

- `Register`, to inform the system of the user's current location
- `Glance`, a bidirectional analog video only connection lasting a few seconds
- `Call`, a bidirectional 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, three 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. Available in different sizes, from 80 × 60 to 640 × 480 pixels, they can be gamma corrected (specifying these parameters with a query string). PostIt message 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, and can be used to call back (Figure 3).

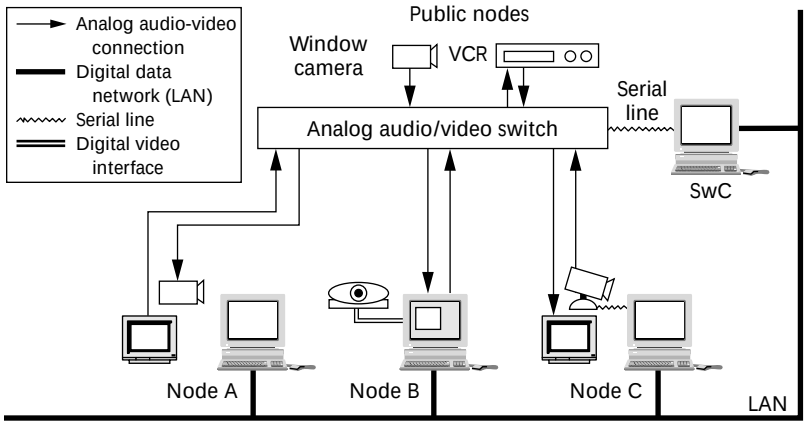

Figure 2. Mediascape hardware configuration. For simplicity, it shows only three nodes and doesn't include audio equipment. Node A is an analog node with the camera and monitor connected to the switch. Node B, a digital node, uses a digital camera and displays video on the computer screen. It also sends 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 integrability

The connection server uses resource names containing only the service and the callee names (for example, `/call.nicolas`). People must identify themselves by name and a location the first time they use the media space. The server stores their location and sets a cookie in the 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 the media space 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">
```

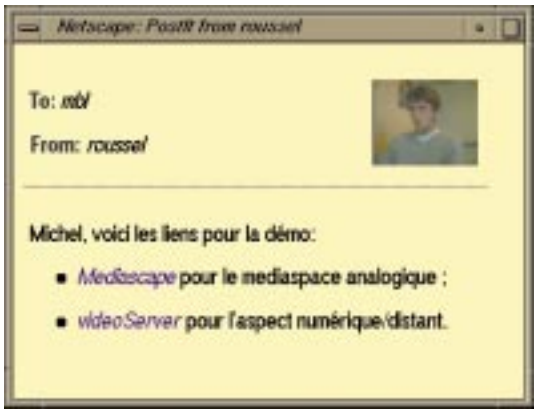

Figure 3. PostIt sample.

- 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/
```

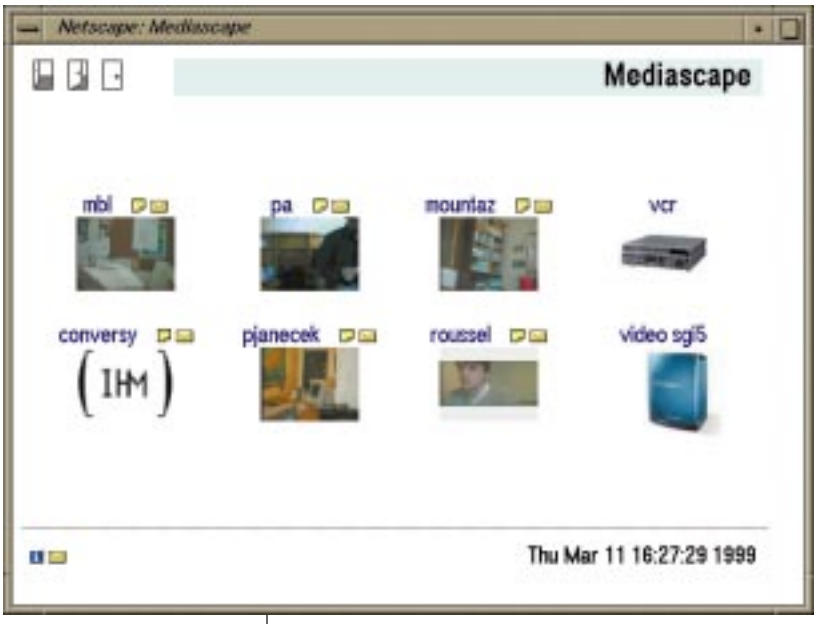

Figure 4. Basic interface to Mediascape. Conversy has closed his door, while Roussel's is ajar. The two rightmost icons give access to public nodes (a VCR and a digital video source).

```
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/"
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

```
<embedsrc="http://mediascape/
glance.michel/glance.stephane/
glance.patrick/glance.paul">
```

The default interface to Mediascape is an HTML document that displays still images of the users (Figure 4). The server alters these images to reflect users' accessibility. Three icons represent door states, as in Cavecat (Computer Audio Video Enhanced Collaboration and Telepresence, University of Toronto). Each door state (open, ajar, or closed) corresponds to an authorization level (authlevel service) controlled by users. 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 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.

Mediascape flexibility for the user

Since Mediascape's basic interface employs 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 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[7]—lets users choose between a number of predefined options, determined by the capabilities of HTML and other related languages such as JavaScript or Virtual Reality Markup Language (VRML).

So far, we have seen dedicated HTML documents used to interface our media space. However, HTML code requesting Mediascape resources can also be added to existing documents. When collaborating with others 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 make connections.

Here the users define the notion of group, which relates to a particular activity, without having to explicitly declare it to the system. This assists coordination: in a co-authoring situation, the authors can include a connect command in the document (Figure 5). Each time one author reads or works on the document, the connection with the other author is automatically established. As an increasing number of e-mail applications understand HTML and HTTP, media space commands can be included in a message and executed when the receiver reads it, again providing implicit coordination between users.

Mediascape flexibility for the developer

To access media space services, end users can employ HTML to create dedicated documents or modify existing ones. This approach lets us quickly prototype high-level services and share experiences by cutting and pasting HTML code or exchanging

files and e-mails. 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 control the media space from within other existing applications, such as a shared editor.

We have developed a series of applications based on a set of APIs (application programming interface) that communicate with Mediascape servers in Tcl, Python, and C++. One such application provides a lightweight interface to the media space consisting only of a title bar that reveals snapshots of various users when the cursor passes over it (Figure 6). A single click on an image glances at that user while a double click asks for a videophone call. A pop-up menu lets users monitor and change their access rights. This example shows how a developer can benefit from the openness of Mediascape to integrate existing services into nonbrowser applications.

Mediascape privacy

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

For privacy, 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 don't have access to our analog audio/video network. Whereas analog services are based on a strict reciprocity rule ("I can see you if you can see me"), 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's usually easy to let them know we're aware of their presence by gesturing, showing a message on a piece of paper, or sending an e-mail.

Figure 5. Coordination between co-authors through a Mediascape-aware document. Here, we can see that both authors are present at the same time.

Videoserver

Two years ago, we started using digital video links to support informal communication over the

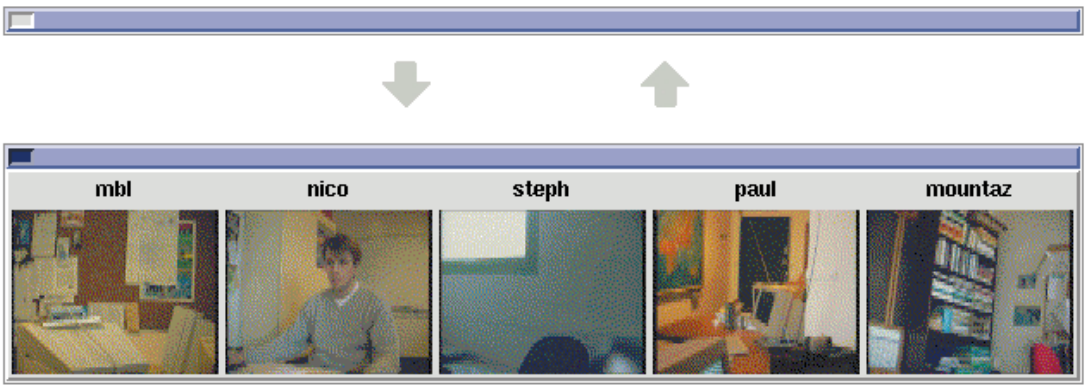

Figure 6. Lightweight custom client for Mediascape.

Internet between our offices and distant colleagues. The Mediascape connection server could only digitize images from one camera at a time, but most of our workstations had a frame grabber and camera. We decided to design and implement Videoserver, a software component for both interpersonal and interprocess digital video transmission.

The first Videoserver prototype was an extension of our Web server (a Common Gateway Interface, or CGI, script). The current implementation is a custom HTTP server, which runs efficiently on any of our SGI workstations. 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 protocol described earlier.

Videoserver offers the following services:

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

Videoserver integrability

Since most Web browsers can display a server-pushed sequence in place of an ordinary image without using a plug-in, Videoserver gives live video access to a large number of users. The frame rate depends on the available bandwidth. A typical 160 × 120 JPEG image is about 3 Kbytes and we routinely get 10 to 15 frames per second, even over long distances. A snapshot image or live and recorded video can be included in place of any image in an HTML document simply by using the following types of links:

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

We have distributed Videoserver outside the lab to obtain bidirectional 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. In addition to using Web browsers as video clients, we have developed a C++ API that lets us display images coming from Videoservers in custom applications. One application consists of a window containing only the video stream, which users can resize to zoom in or out. Another application lets us broadcast the digital images on our analog video network, making remote people accessible on the TV monitors of our media space nodes, like local users.

Videoserver flexibility

Clients can use a query string to specify the compression ratio and zoom factor of live images. These two parameters let clients adapt their requests to the available bandwidth by reducing the resolution or augmenting the compression ratio. For video, two extra parameters specify the number of images requested and the time to wait between two subsequent images. The former can help create a glance, the latter can help 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 moves over it and turns back to a snapshot when the the cursor leaves it:

```
<a href="http://www-ihm.lri.fr/"
onMouseOver=
   'document.i1.src=
      "http://videoserver/video"'
onMouseOut=
   'document.i1.src=
      "http://videoserver/photo"'>
<img name="i1"

src="http://videoserver/photo">
</a>
```

This code can help create an awareness view from several Videoservers (Figure 7). 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.

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 resolution, quality, number of images, and refresh rate. In particular, a very high compression factor generates a highly degraded image that still provides some useful information, such as the number of people present. 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 the person is absent or busy.

When the request doesn't specify number of images for live video, Videoserver limits the number 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.

Future work

One advantage of using digital video rather than analog video is the ability to process images. Previous work on media spaces shows that primitive image processing can enhance some services. Examples include remote camera control by local movements,[12] addition of recent activity representation to digitized images,[13] and to support privacy, substitution of a shadow for a person,[14] or eigen-space filtering.[15]

We're investigating several techniques based on simple image difference and partition to extend Mediascape services. These extensions include context capture—knowing whether a user is present or absent, motion detection, and image segmentation to support natural annotation and gesturing. We're implementing these applications as Videoserver clients. Consequently, users are always notified and can control access to their image in a single, unified way.

We're also prototyping a shared graphical editor where live video images can be manipulated as first class objects. When sharing a window with another user, you can create a live video image of that user as a new object on the drawing surface. This object lets you gesture or control Mediascape services. For example, it might let you make a call when the mouse cursor nears a video object.

Figure 7. HTML document presenting images from several Videoservers around the world.

Today, Mediascape and Videoserver support point-to-point connections between users. In earlier work, we designed a more generic connection and session management model[16] in which users with different sets of audiovisual resources can freely create, enter, and leave sessions, bringing documents or applications with them. We intend to pursue our work on media spaces to combine this model with the software infrastructure described in this article, ultimately leading to a toolkit for building media spaces.

Videoserver is available for download at http://www-ihm.lri.fr/~roussel/Mediascape/. The current implementation works only on SGI machines, but should port easily to other platforms. MM

Acknowledgments

I would like to thank Michel Beaudouin-Lafon, Wendy Mackay, the other members of our group, and the anonymous reviewers for providing helpful comments on this work. I also want to acknowledge the different people around the world who use Videoserver and thank them for their enthusiasm and collaboration.

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

References

1. R. Stults, "Media Space," Technical Report, Xerox PARC (Palo Alto Research Center)), Palo Alto, Calif., 1986.
2. S.A. Bly, S.R. Harrison, and S. Irwin, "Mediaspaces: Bringing People Together in a Video, Audio and Computing Environment," *Comm. ACM*, Vol. 36, No. 1, Jan. 1993, pp. 28-47.
3. W.E. Mackay, *Media Spaces: Environments for Informal Multimedia Interaction*, M. Beaudouin-Lafon, ed., Computer-Supported Cooperative Work, Trends in Software Series, John Wiley & Sons Ltd, Chichester, England, 1999.
4. P. Dourish and S. Bly, "Portholes: Supporting Awareness in a Distributed Work Group," *Proc. Conf. on Human Factors in Computing Systems (CHI 92)*, ACM Press, New York, 1992, pp. 541-547.
5. W.W. Gaver, "The Affordances of Media Spaces for Collaboration," *Proc. Conf. on Computer-Supported Cooperative Work (CSCW 92),* ACM Press, New York, 1992, pp. 17-24.
6. L. Suchman, "Office Procedures as Practical Action: Models of Work and System Design," *ACM Trans. on Office Information Systems*, Vol. 1, 1983, pp. 320-328.
7. R. Bentley and P. Dourish, "Medium versus Mechanism: Supporting Collaboration Through Customization," *Proc. European Conf. on Computer-Supported Cooperative Work (ECSCW 95),* Kluwer Academic, Dordrecht, The Netherlands, 1995, pp. 133-148.
8. P. Dourish, "Culture and Control in a Media Space," G. De Michelis, C. Simone, and K. Schmidt, eds., *Proc. European Conf. on Computer-Supported Cooperative Work (ECSCW 93),* Kluwer Academic, Dordrecht, The Netherlands, 1993, pp. 335-341.
9. D. Kristol and L. Montulli, "HTTP State Management Mechanism," Proposed Standard, RFC 2109, available at http://www.ietf.org/rfc/rfc2109.txt, IETF Network Working Group, 1997.
10. A. Maclean et al., "User Tailorable Systems: Pressing the Issues with Buttons," *Proc. Conf. on Human Factors in Computing Systems (CHI 90)*, ACM Press, New York, 1990, pp. 175-182.
11. M. St Johns, "Authentication Server," RFC 931, IETF Network Working Group, 1985.
12. W.W. Gaver, G. Smets, and K. Overbeeke, "A Virtual Window On Media Space," *Proc. Conf. on Human Factors in Computing Systems (CHI 95)*, ACM Press, New York, 1995, pp. 257-264.
13. A. Lee, A. Girgensohn, and K. Schlueter, "NYNEX Portholes: Initial User Reactions and Redesign Implications," *Proc. GROUP 97*, ACM Press, New York, 1997, pp. 385-394.
14. S. E. Hudson and I. Smith, "Techniques for Addressing Fundamental Privacy and Disruption Tradeoffs in Awareness Support Systems," *Proc. Conf. on Computer-Supported Cooperative Work (CSCW 96)*, ACM Press, New York, 1996. pp. 248-257.
15. J. Coutaz et al., "Early Experience with the Mediaspace CoMedi," *IFIP Working Conf. on Engineering for Human-Computer Interaction (EHCI 98),* available at ftp://ftp.imag.fr/imag/IIHM/ENGLISH/publications/1998/EHCI98_CoMedi.pdf, 1998.
16. N. Roussel, "Beyond the Media Space: A Model for Mediated Collaboration," *Proceedings of IHM 97* (in French), Futuroscope, Cépadués, Toulouse, France, 1997.

Nicolas Roussel is a PhD student in computer science at the Université Paris-Sud. His research concerns the design of environments to support remote communication, coordination, and collaboration. He is currently exploring video as a first-class object in user interfaces and web-based media spaces.

Readers may contact Roussel at the Université Paris-Sud, Laboratory for Computer Science (LRI), UMR 8623 CNRS, Bat. 490, 91405 Orsay Cedex, France, e-mail roussel@lri.fr.