Two Approaches to Casual Interaction over Computer and Video Networks

Alan Borning
Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, WA 98195
borning@cs.washington.edu
206-543-6678

Michael Travers
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
mt@media-lab.media.mit.edu
617-253-0608

Abstract

1 Introduction

Despite the ongoing telecommunications revolution, physical proximity is still extremely important to cooperative work. The everyday life of people in organizations depends upon the chance encounters that depend on people sharing physical space. Our project aims at creating the same atmosphere of casual awareness and informal interaction between people at sites that might be physically separate.

We are interested in supporting shared awareness: distribution of general information about the environment, both physical and social. Such information includes who's here, what they are doing (if they want this to be known), whether they are available for interactions, and what's happening in the common areas. We also want to support informal interactions: the sorts of conversations that occur around coffee machines, mailboxes, in the hallways, and the like. In such interactions, the purpose, duration, and degree of involvement of the interaction is not planned in advance (in contrast to e.g. a scheduled meeting), but is negotiated in a subtle and dynamic fashion during the course of the interaction. We believe that support for these two activities should be intertwined, since a prerequisite for informal interactions is awareness, and since simple awareness without the ability to act on it is not terribly useful.

We have constructed two systems to explore these issues, both of which make use of the extensive collection of AV gear at EuroPARC. The tools we have at our disposal for this project include video cameras and monitors, microphones and speakers, and a framegrabber, as well as workstations with bitmap displays. The video and audio equipment is all interconnectable via a computer-controlled switching network [?], and the workstations are all on an Ethernet computer network. Every researcher's office at EuroPARC is equipped with video and audio input and output devices, as are many of the common areas, so we have a unique opportunity in the laboratory to experiment with ubiquitous audio-video.

The first of the systems we constructed, *Polyscope*, provides images of people and common rooms in the form of video stills. It allows an observer to monitor a large number of video sources conveniently and simultaneously, by providing an array of frame-grabbed video images on a workstation display. A Polyscope observer window also provides a quick and casual interface to other functions in the audio-video system, such as making video phone calls or glances (described below).

The second system, Vrooms, deals with some of the social and interface issues raised by Polyscope by employing a strong spatial metaphor. Users can enter and leave virtual rooms (vrooms). Once in a vroom, users can see and be seen by all the other occupants, and have access to other communication tools for full video links, text dialogs, and the like.

2 Privacy Concerns

Ubiquitous video raises numerous concerns about privacy. How will people feel about having cameras in their offices and in public areas? What sorts of controls are felt to be necessary? Or will most people simply not want cameras in their offices at all? To a considerable extent we and other researchers at EuroPARC have been attempting to answer these questions empirically, by putting systems in place and gathering peoples' reactions. However, early in the course of this project (before any implementation was done), we proposed two abstract design principles for video systems: *control* and *symmetry*.

I should be able to *control* whether or not others can see me, and under what conditions. Thus, I should be able to choose whether I allow a frame-grabbed image to be sampled and when and with what sort of notification, whether or not I allow full video connections, whether perhaps I just electronically post a notice that says "do not disturb," and so forth.

Symmetry means that I can see you if you can see me.[1] The symmetry principle doesn't mean that I *must* see you if you can see me, only that I have that option if I should choose to exercise it. Further, the kinds of information should be comparable: if you can see full live video of me, I should be able to see full live video of you; if you can see frame-grabbed images of me, I should be able to see frame-grabbed images of you; and so forth.

How well these principles worked out in the light of implementation and actual use is discussed later in this paper, after the description of the systems.

3 Polyscope

+++ redundant paragraph – merge with other stuff or delete: Polyscope allows an observer to monitor a large number of video sources conveniently and simultaneously, by providing an array of frame-grabbed video images on a workstation display. The observer can select which of the available sources to monitor; these images are then updated periodically at an interval determined by the observer. On the source side, potential sources of video (e.g. researchers with video cameras in their offices) can decide whether or not to make video available. A Polyscope observer window also is an interface to other functions in the audio-video system, such as video glance and video phone.

+++ somewhere: compare with glance and sweep

There are two types of Polyscope windows: Observers and Sources. A Source window controls the availability of a particular video channel, while the Observer lets the user view all or some of the information that the sources are making available.

[1] We have christened this design principle with its acronym ICSYIYCSM, pronounced icksyism, in analogy with communism, capitalism, and other ideologies.

The Observer window will display some number of screen images in a rectangular grid. When it first comes up, the observer window will show images from all the currently available sources, updated more or less once a minute. The user has some control over the appearance of the display, such as the ability to choose a subset of images, to control how often they are updated and whether or not they are animated, and whether new images are displayed as they become available.

In addition, each image in an observer window can act as an entry point to the audio-visual system. Clicking on an image will pop up a menu of options that allow the user to initiate various types of connections. These include a glance (a short, temporary one-way connection) and a video phone call (a long-term, two-way, negotiated connection). At this level, privacy and permissions are controlled by users "agents", separate processes running on their workstations that handle negotiation [?].

A Source window controls a particular video source, typically a camera on a desktop pointing at a user's head. Source windows give users own control over the distribution of images from their cameras.

The Mode line controls what information is being made available. The user can choose between no information, a short text message, or manual or automatic video. Video consists of small frame-grabbed images. In manual video mode, an image is grabbed only at the request of the owner of the source window. In automatic video mode, images are grabbed under control of the observers. Observers typically grab images periodically (say, once a minute).

The system can grab frames at rates fast enough to do crude animation. The Animation Frames line specifies how many images are taken at a time. This field is normally 1.

he Grab Now button is for use when Manual Video mode is selected. It will cause a new image to be recorded (or images if the animation frames field is greater than 1).

Save Images saves the current frame-grabbed images in a file. The user is prompted for a file name. There are actually several files written. If the file name the user gives is "elvis",and there are two animation frames, then files elvis.nframes, elvis.frame0, and elvis.frame1 will be written, containing respectively 2 (the number of frames), and the two actual frames.

Restore Images prompts for a file name, and restores the images from those files. (For example, typing "elvis" would load in the two elvis images.) Restore Images can only be used if the mode is set to Manual.

The Feedback line specifies whether feedback is desired, and if so what kind. The three choices are:

Off No feedback (this is the mode when the source window is created) Names Only List names of all observers watching this source. This list is updated at regular intervals. Video Show video images of all observers watching this source if they have made such images available. If video isn't available, the Text from the source is shown, or if no text, then a statement that the source is Off or unavailalbe is shown. Again, the images are regularly updated.

If Names Only or Video is chosen, an attached feedback window will be opened under the source menu to display the requested information. This window has a middle button menu with items to change the update frequency, and to request an immediate update of the feedback information.

Finally, the Video Symmetry Required line specifies whether or not symmetry is required. If no, then any observer can view this source, irrespective of whether or not that observer in turn makes video information available. If yes, then to see video from this source, an observer must in turn make video available. Note that this is independent of the feedback window. We can require video symmetry without actually looking at the video, or we can ask for video feedback without requiring symmetry.

More precisely, if video symmetry is required, and the source is providing Auto Video, then any observer of this source must provide Auto Video as well. If the symmetry requirements aren't met, the picture of the source will be deleted from the observer's window, and an explanatory message will appear in the observer's Prompt Window.

If the source window is closed, the source files will continue to be available. If a new source window is opened, its state will be initialized from the files.

4 Experience with Polyscope

custom animations – entertaining, but interferes with goal of awareness of what's going on now

One problem with Polyscope was the many options presented by its interface. The intent of such features as the symmetry mechanism and the feedback mechanism was to enforce a basic property of face-to-face interaction, namely, the mutual accessibility and visibility of participants. This goal was satisfied, but at the cost of forcing users to make explicit decisions about their accessibility and visibility. Although some social theorists believe that such decisions and operations are part of normal face-to-face interaction [?], such matters are extremely personal, private, and subtle. It feels very clunky and somewhat invasive to have to make such matters explicit on a computer screen, which has the effect of making public and explicit what is usually private and implicit.

5 Vrooms

5.1 Motivations

Vrooms was designed to answer some of the problems experienced in the use of Polyscope. Dialog boxes with explicit options had proved awkward. Since people already have a great familiarity with physical space, and use this familiarity with great subtlety in social interaction, our thoughts turned towards using a spatial metaphor or "virtual reality" interface.

Our metaphor was one of virtual rooms (hence the name Vrooms). A virtual room is a place for social interaction and conversation. In Vrooms, people are aware of each other when they share the same virtual location. Although this virtual space may be elaborated further, for now let us think of it as made up of rooms, linked to each other by doorways. A room is presented as a window containing a collection of images, as in Polyscope, together with some other items described below.

This presentation metaphor allows us to dispense with some of the complexity of the current user interface. Instead of many options, we make use of people's tacit knowledge of what it means to be in a room.

5.2 The social properties of virtual space

Being in a room with others means that you can see them and they can see you, and that both have the option of initiating a conversation, and that this conversation will usually be more casual than one whose initiation involves a purposeful trip by one party to the other party's location.

One point on which virtual rooms differ from physical rooms is that you can be "in" more than one at a time: you can have windows open to more than one virtual room. This means that being in a virtual room involves less commitment than being in a physical room. This has many ramifications. Because of this and the slow refresh rate of video pictures, less information is available about the real activity of the person behind the virtual presence. More positively, it makes it easier for people to "hang out" in rooms that might not have much ongoing activity. This might help solve the "social bootstrapping" problem, in which a room remains empty because nobody wants to be the first one to occupy it.

5.3 The Current Vrooms Implementation

The current implementation uses a client-server architecture. A server program maintains a database of all existing vrooms, their occupants, and other details. A client program runs on the machine of each vrooms user, and exchanges information with the server using the Xerox Courier remote procedure call implementation [?].

A user sees a vroom as a window on his or her workstation. A vroom has small frame-grabbed images of each of its occupants, positioned arbitrarily within the vroom. Each image is animated (as with Polyscope), displaying several frame-grabbed bitmaps in rapid succession, and is labelled with a name and last update time. The vroom can also hold various sorts of shared objects, such as communication devices. In the curent implementation the only such sort of shared object is a dialog box *+++ poor term, we already use that to mean something else above*, in which users can type statements. Discounting communication delays, all users with a window open on the vroom see the same contents. *+++ perhaps they should get an audio mix as well? Clarify use of audio)* If any user makes a change—for example moving his or her image or typing in a statement in a dialog box—the change is sent to the server, which updates its own database and also broadcasts the change to all other users with an open window on that vroom.

Top-level access to vrooms is via several buttons on the user's display: *enter existing vroom*, *create new vroom*, and *delete vroom*. (A button is a pressable icon, which causes some arbitrary piece of Lisp code to be executed when it is pressed [?].)

When the user presses *enter existing vroom*, the client program queries the server for the names of existing vrooms, which are displayed in a menu to the user. If the user picks one, a new window is created on that vroom. The server sends information about the current occupants of the vroom and their locations, and about dialog boxes, and the new window is opened with its current contents displayed. The user is then prompted visually for where to locate his or her image in the vroom (Figure **??**). If the *create new vroom* button is pressed instead,the user is prompted for the name of the new vroom. He or she then specifies its size, and again the initial location of his or her image. A user's image remains in the vroom as long as the corresponding window is open on his or her screen; when the window is closed, the user automatically leaves the vroom. Closing the window doesn't delete the vroom from the server's database, however; others may still be in it. Such a deletion can be performed using the *delete vroom* button.

+++ figure entering-vroom here +++

Thus, in vrooms the symmetry principle is automatically and unobtrusively satisfied. The actions of opening a window on a vroom and entering it (i.e. positioning one's frame-grabbed image in the vroom) are bound together by the user interface, as are those of closing the window and leaving a vroom. Once I'm in a vroom, I can always see all the other occupants, and they can see me.

Once in a vroom, an occupant can move his or her image around in the room, and can create, delete, or move dialog boxes. More importantly, an occupant has several ways of communicating with other occupants. If an occupant moves close to the image of another occupant, a full two-way video and audio connection is immediately set up between the two on a separate monitor (Figure ??). When either person moves away, the connection is broken. The vrooms system provides feedback that a connection will be set up—if an image is moved close enough to another, a thick grey box will be drawn around the two images of the people in conversation. This box appears in the windows of all occupants of the vroom, to let others know that a conversation is taking place. In addition, any user can type text into a shared dialog box (Figure ??). The text appears in the room's dialog box, labelled by the name of the person who typed it.

+++ figure making-connection here +++

+++ figure dialog-box here +++

All windows on a given vroom are constrained to be the same size. The reason for enforcing this same-size constraint is so that we don't want to have something happening in a vroom that some users can see but others can't. (An alternative would be to scale the images depending on the size of the window, but we didn't do this because it was more expensive computationally.)

+++ I don't know about above: having a scrolling window and being able to focus down to your local conversation seems to be a thing you'd like to do (it's like creating a subroom, but more flexible). Do we want to say anything about this?

5.4 The Full Vrooms Design

The previous section describes our current prototype implementation of vrooms. It was done under severe time constraints by Borning in the waning weeks of his year-long sabbatical at EuroPARC. Our full design is for a more ambitious system, and we thought it worthwhile to note some of its additional properties.

The current implementation only allows two-way full video connections. Our plan is to allow up to four people to converse at once, using video mixers that can combine four or more video images into a single one. If a third person wanted to join an existing conversation, he or she would move near the participants. The grey conversation box would expand to surround all three images, the video mixer would be switched in, and all participants would be able to see each other through a split-screen image. Such a transition from a two-way to a multi-way video connection has already been used at EuroPARC, although Vrooms does not yet make use of it. Audio mixing would be done in a similar fashion.

We imagine having other sorts of shared objects available, in addition to dialog boxes. (Our implementation uses object-oriented techniques, so as long as these other objects obey the correct protocol they can be accomodated.) Other possibilites include shared text editors, shared drawing programs, and posted announcements or pictures (in analogy with physical bulletin boards).

Finally, we envision a "door" mechanism that would allow moving among vrooms, and that would also serve to control access to vrooms (if that was desired). A door to another vroom could be placed in a vroom; when the user clicked on it, he or she would leave the current vroom (thus deleting its window from the screen), and enter the new one. One could optionally remain in the existing vroom as well as entering the new one. (A feature of being in virtual rather than physical space is that one can be in several places at once.) Doors could also be used to control access to some vrooms, for example by checking visitors against a list of allowed occupants, by asking permission to enter of those already in the vroom, by locking a vroom from inside, and so forth.

Each person's office would have a corresponding vroom as well, with doors to these offices could be placed in a corridor vroom. (Note that the layout of the corridor need not correspond to physical space—the offices on a corridor could be scattered over several sites.) Individuals would of course control access to their own offices: we envision this being done by altering the door icon to be closed, open a crack, half-open, fully open, and so forth. (Again, we are trying to tap into the rich set of conventions people have now for using physical doors.) For an open door, one could see a much-reduced image of what was in the vroom.

5.5 Timing Considerations and Concurrency Control in Vrooms

+++ this section is too detailed, particularly since it hasn't been completely implemented +++

As noted above, our goal is that all occupants of a vroom should see the same thing in the windows on their displays. Network and computational overheads make this impossible to achieve completely, although performance even in the current implementation is acceptable. One place we compromised was in moving an image or shared object. For example, if a user moves his or her image, the about the movement isn't tracked in real time in each window. Rather, information the movement remains local until the user releases the mouse button and positions the image. At that point, the server is told about the movement, and the server in turn broadcasts the change to all other clients. Each client shows the movement by displaying a little animation of the image moving to its new position, rather than simply erasing the old image and displaying the new. (This technique proved to be both efficient and entertaining to watch.)

Concurrency control involves more knotty issues. When there are several occupants in a vroom, there are a number of ways in which nearly-simultaneous changes made by different occupants could interfere with each other. Three general approaches to concurrency control are:

1. to lock shared resources while one person is making changes (so that others must wait to make changes)
2. to take an optimistic viewpoint, allowing simultaneous activity and fixing up any problems later)
3. when possible, to design the system in such a way that certain conflicts don't occur.

We use a combination of the second and third methods, since we felt that locking would damage the casual, interactive feel of the system. (For our prototype implementation, we had time to program only the dialog box mechanism and some of the movement mechanism; the others are paper designs at this point.)

For dialog boxes, one could imagine bits of interspersed text from different writers. To prevent this, we buffer up entries. When a person is done typing, the entire entry is sent to the server, which broadcasts it to all clients with an open window on that vroom. Arrival order at the server determines the order in which the entries are displayed. Another solution would be to give every user an independent box to type in, but this would lose the t

Movement within a vroom and/or resizing the vroom presents less trivial problems. For example, one person might be moving his icon at the same time that another is resizing the vroom (with the result that the first person's icon would end up outside of the resized vroom), or two people could be trying to move a dialog box simultaneously, or two people could be trying to resize a vroom at the same time.

A situation where two people are trying to move an image at the same time is simply disallowed—you can't push other people around; you can only move yourself. However, anyone is allowed to move a shared object such as a dialog box. In this case, the server would receive the move requests in some order (the remote procedure call mechanism automatically serializes them), and would broadcast first one move and then the other.

A related problem occurs if two people are moving two different images (or shared objects) around, and they end up in overlapping positions. This isn't allowed; the server will accept the first move request that arrives, and will modify the second by moving the image or object away enough to remove the overlap. (The server also respects video connection requests, repositioning the object close enough to make a video connection if that was intended, or far enough away so that there is no video connection if that was not intended. It is easy to determine whether or not a video connection was intended, based on where the occupant moved relative to the positions of the images in his or her window.)

+++ above is not entirely comprehensible, but the idea is right, i hope

+++ I think it's very important that people can move their icons simultaneously, for heightened social realism, following someone around, etc. I don't see what's wrong with overlapping, either.

+++ How about a force model (spring constraints) when two people are both trying to move the same object?

+++ Early versions of Playground had shared object manipulation. Don't know exactly how they worked, will try and find reference.

Similarly, nearly simultaneous attempts to resize a vroom would be serialized and broadcast. (As noted above, we constrain all windows on a given vroom to be the same size.) However, we do insist that all occupants of a vroom be visible, so in case a move request from another user arrived while the vroom was being resized, before the server will broadcase a resize request, it is checked to see that all occupants and shared objects are within the new vroom dimensions. If not, the dimensions are expanded to encompass all occupants and shared objects. *+++also hard to understand...*

6 Implementation and System Architecture

Both Polyscope and Vrooms were programmed in Xerox Common Lisp running on Sun 4's, Sparcstations, and Xerox Daybreaks. We also had in place a large amount of video and switching equipment [?]. Every office at EuroPARC is equipped with a TV camera, monitor, microphone, mixer, and speakers. These are all connected to a computer-controlled switch, which is easily accessible from Lisp, allowing connections to be set up as desired. In addition, the video network included a frame grabber, which we used for the images displayed in both Polyscope and Vrooms windows, and a quad picture-in-picture box.

Polyscope sources and observers communicate exclusively through the shared network file system. A Polyscope directory contains a subdirectory for each source. Each subdirectory in turn contains a status file giving information about the source, files with the actual frame-grabbed images, and other files containing the names of observers, in case feedback is desired. There is no computational expense in being a source, except the small cost of writing out a new status file when a change is made. Each observer periodically updates the images it displays. To do this, it checks the file write date of the frame-grabbed image. If it is new enough, the existing image is grabbed; if not, a new one is grabbed. (Thus, multiple observers share frame-grabbed images; otherwise, our single frame grabber would be overloaded.)

We chose this design for Polyscope, rather than using the Courier remote procedure call mechanism, because we believed it would be simpler for our first implementation. In retrospect, this was a bad decision: first, since Courier proved to be very easy to use for Vrooms; and second, because we spent a lot of time trying to get the network file system to handle various concurrency problems properly.

As previously described, Vrooms uses a client-server architecture. A server program, written in Lisp, maintains a database of all existing vrooms, their occupants, and other details. A client program runs on the machine of each vrooms user, and exchanges information with the server using the Xerox Courier remote procedure call implementation [?]. In general, a client will make some change and will send the information to the server. The server will then update its own database, and will broadcast the change to the other clients. In addition, the server takes care of updating frame-grabbed images. The server includes an update process, which makes sure images aren't too old. This process snaps a new frame-grabbed image when needed, and informs all clients that use that image that a new one is available.

7 Future Work

[how does this relate to the "full vrooms design" section?]

Portholes – connecting PARC and EuroPARC

The full benefit of these systems will become apparent when they are used to connect physically remote sites. At the time of writing, we have full video only within the single EuroPARC site. We would like to be able to share awareness with PARC in California. The eight-hour time difference makes this problematic.

Vrooms is based on a spatial metaphors, and as such would like to be a full three-dimensional virtual reality. We can't, however, maintain our goal of providing casual communication with the current generation VR technology, which requires donning various items of clothing. Myron Kreuger's earlier approach to VR, using image processing to detect body movements, may prove more useful [?]. Other approaches to generating 3D data non-invasivly include the range camera [?] and various techniques from computer vision such as stero-pair matching [?].

+++ redundant with concurrency control section ... fix

One issue concerns access and modification controls. We could have an anarchic system, in which anyone could enter any room, or modify the control settings on any room. This is probably a bit too anarchic—but we don't want to get bogged down by bureaucratic mechanisms. Some possible controls include: who could enter a given room at any time, who can enter a room if others are already using it, who can construct doors off a given room, and who can modify the control settings. Some rooms could "lock from the inside." In place of restrictions on who constructs doors or modifies controls, we could also have a simple log mechanism for recording who made such modifications, which would at least make it easier to find someone to yell at. What controls should exist needs to be investigated experimentally.

Another issue concerns how dynamic the creation and deletion of rooms should be. At one extreme we could have a collection of rooms that changed slowly over time; at another extreme, the collection of rooms could change rapidly, with new rooms being created and removed as conversations and other interactions begin and end. Again, the pattern of use should to be discovered rather than mandated; but this usage pattern will impact the user interface for creating, removing, and modifying rooms. *+++ huh?*

Finally, there is a huge sea of social issues surrounding how well Vrooms would support our goal of encouraging informal interaction. People have very ingrained long-term accumulations of routines for initiating and terminating conversations. The more realistic our virtual world becomes, the more it will cause people to see interactions within the system in terms of their normal day-to-day interactions. But because the system cannot offer the full richness of physical interaction (for instance, you can't redirect your gaze or observe the gaze of other participants), we may expect false expectations to be generated. We have no particular position on these social issues, but we wish to recognize their existence so that our design and experiments will take them into account.

More advanced interface technology such as head-mounted displays [?], body suits [?], and eye tracking [?] could of course provide a richer interactional vocabulary. However, since our goals are to provide casual interaction, a technology that requires changes of clothing is not suitable. Of course, someday the whole work environment might be in virtual reality. Vrooms demonstrates that virtual space metaphors may be used for social interaction *without the full panopoly of three-dimensional graphics and body interface technology.*

8 Related Work

8.1 Spatial User Interface Metaphors

The metaphor of rooms has been used in other interfaces. The Rooms system [?] used rooms as an extension of the desktop metaphor. Rather than having objects and windows appear in a single workspace, they could be distributed among rooms, which were essentially separate desktops. Rooms could be linked through door-shaped icons. +++ more, consequences

More generally, spatial and architectural metaphors have been proposed as the basis for interfaces. The idea here is to exploit the user's existing abilities to navigate in space. (cite SDMS, what else?) However, this approach is problematic because the experience of navigating through simulated space is quite different from travelling through physical space. In simulated space, the body hardly moves, and the gaze remains fixed on a screen which presents a changing image. Recent developments in head-mounted displays have made it possible to actually control movement in virtual space by means of normal body motion, and to present a coherent viewpoint through the use of 3D graphics [?].

The use of virtual rooms as places to communicate was inspired by the multi-user adventure game TinyMUD (Applegate et al. 90), as well as earlier programs such as the Plato system's Talkomatic (ref?).

The notion of virtual rooms and spaces also appears in early work of ours [?, ?]. The emphasis in this previous work was on navigation through complex structures by individuals, rather than how to manage shared space.

8.2 Shared manipulation spaces

Colab?

Playground

Must be others...

8.3 The Sociology of Face-to-face Interaction

We are essentialy trying to engineer new forms of communication that mimic face-to-face interaction as closely as possible. What does such interaction involve? Sociologists exploring this problem have revealed a multiude of interesting problems in this everyday domain. Most of these proceed from the assumption that the participants have varying purposes, beliefs, and attitudces, and interaction is a process in which these various individual properties play themselves out through communicative mechanisms.

Goffman [?] views interaction as a matter of mutual performance, where participants exert conscious control over their appearances. The nature of such "face-work" varies over different contexts (for instance, their are "back-stage" regions where constraints on action might be loosened). His approach might be described as ethnographic, in that he mainly relies on example and anecdotes. [?] attempts to formalize some of the face phenomena studied by Goffman in terms of speech-act theory. Conversation analysis [?] looks at the ways in which individuals use conversation and other interactional techniques to create, mantain, and account for social organization.

Shared virtual spaces might benefit from an awareness of this research. They can also serve as useful testbeds for further social research, since they make it easy to record the details of complex social interactions

"The human personality is a sacred thing; one does not violate it nor infringe its bounds, while at the same time its greatest good is in communion with others" – Durkheim, Sociology and Philosophy, trans. D. F. Pocock (Lodon: Cohen West, 1953), p. 37 (from Goffman).

Acknowledgements

thanks to europarc, in particular paul dourish and lennart lovstrand, etc. thanks to Marc Davis for helpful commentary on draft version