CoShare: a Multi-Pointer Collaborative Screen Sharing Tool

Martina Emmert
University of Regensburg
Regensburg, Germany
martina.emmert@stud.uni-regensburg.de

Andreas Schmid
University of Regensburg
Regensburg, Germany
andreas.schmid@ur.de

Raphael Wimmer
University of Regensburg
Regensburg, Germany
raphael.wimmer@ur.de

Niels Henze
University of Regensburg
Regensburg, Germany
niels.henze@ur.de

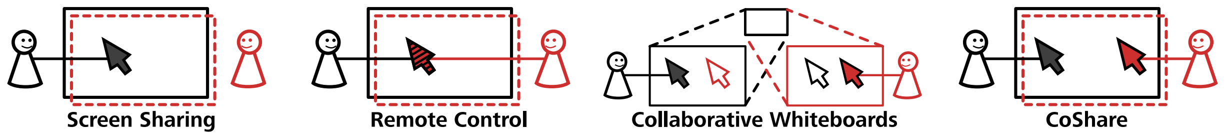

Figure 1: CoShare allows users to share their screen with others and give viewers their own mouse pointer. Unlike collaborative whiteboards, CoShare is application-agnostic and does not rely on third-party servers.

ABSTRACT

Existing tools for screen sharing and remote control only allow a single user to interact with a system while others are watching. Collaborative editors and whiteboards allow multiple users to work simultaneously, but only offer a limited set of tools. With *CoShare*, we combine both concepts into a screen sharing tool that gives remote viewers a mouse pointer and a text cursor so that they can seamlessly collaborate within the same desktop environment. We have developed a proof-of-concept implementation that leverages Linux' multi-pointer support so users can control applications in parallel. It also allows limited sharing of clipboard and dragging files from the remote viewer's desktop into the video-streamed desktop. In focus groups we gathered user requirements regarding privacy, control, and communication. A qualitative lab study identified further areas for improvement and demonstrated CoShare's utility.

ACM Reference Format:
Martina Emmert, Andreas Schmid, Raphael Wimmer, and Niels Henze. 2023. CoShare: a Multi-Pointer Collaborative Screen Sharing Tool. In *Mensch und Computer 2023 (MuC '23), September 03–06, 2023, Rapperswil, Switzerland.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3603555.3608524

1 INTRODUCTION

Remote collaboration has become ever more common in recent years. Most applications for synchronous remote collaboration can be grouped into two categories: screen sharing and collaborative editing. Most video conferencing software allow people to stream their screen's contents to other participants in a meeting. However, there is always a clear distinction between presenter and viewers. Viewers may view, comment, or sometimes annotate shown content, but they cannot interact directly. In contrast, web-based collaborative editors, such as Google Docs or Miro, provide a document or canvas in which multiple people can edit content simultaneously with the same editing powers and tools. However, all collaboration happens on a remote server within a single document and using the provided set of tools.

With *CoShare*, we explore a different type of remote collaboration: multi-pointer interactive screen sharing. The hosting user not only streams their screen to visitors but also allows them to interact with the screen contents as if it were their own computer. Each visitor controls their own pointer and text cursor. To learn about the technical, interactional, and social aspects of such a interaction model, we developed a proof-of-concept prototype for sharing (part of) a Linux desktop with a single remote visitor. In the following, we first give a short overview of the design space and related work. Afterwards, we describe design process and implementation of the *CoShare* prototype. In focus groups and a qualitative user study we identified several important considerations such applications. We conclude by discussing limitations and planned improvements.

MuC '23, September 03–06, 2023, Rapperswil, Switzerland
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0771-1/23/09.
https://doi.org/10.1145/3603555.3608524

2 RELATED WORK

Nowadays, remote collaboration takes place predominantly by using dedicated tools. There are cloud services that provide online file storage and tools to edit those files collaboratively in real-time. Additionally, almost all video conference programs come with a screen sharing function. *TeamViewer* allows for controlling a PC remotely, for example, to let experts solve a problem. However, first visions for collaborative work in the digital age already emerged in the early 1990s, including concepts for an electronic meeting system [16], or early concepts for collaborative editors [4]. With *Virtual Network Computing* (VNC) [19], remotely controlling graphical user interfaces became possible.

2.1 Co-located Collaboration

In 1999, Stewart et al. [23] introduced the concept of *Single Display Groupware* (SDG), which allows co-located users to simultaneously control software on a shared screen, for example, in meetings. *Mighty Mouse* [5] lets multiple co-located users control one PC. Users can switch between contexts by moving the mouse cursor through display boundaries, similar to transitioning between multiple connected displays. Even though they did not implement this feature for *Mighty Mouse*, authors envisioned adding a shard clipboard to SDG applications to allow for easy data transfer between devices. *Lacome* [14] allows for sharing application windows on a large screen. It builds upon VNC and a custom communication protocol. As it features input redirection, *Lacome* allows users to work concurrently, as well as control each other's applications. Steinert et al. [22] present a prototypical implementation of collaborative application for extreme programming. It supports co-located users interacting on a single screen, as well as remote collaboration.

Similar to Single Display Groupware, there are several research projects that investigate how co-located users could interact in multi-display environments. Many of those projects allow for easily distributing application windows among screens in the room, or add multi-pointer support to applications on those screens. *PointRight* [13] is a cross-platform environment that allows for connecting multiple screens to one PC, as well as multiple PCs to one screen by redirecting the PC's inputs. Designed for rooms with smart boards and interactive tables, the system uses a geometrical model of the room for more intuitive context switching. *ARIS* [3] allows for relocating applications windows to different screens in a room and supports multiple platforms such as PC and PDAs. Wallace et al. [24] used a similar concept for a multi-cursor window manager that can be used in control rooms where multi-display applications and context switching are commonplace. *Swordfish* [25] allows users to customize their workspace in multi-display environments.

2.2 Multi-pointer interaction

As buying additional input devices is significantly cheaper than purchasing desktop or laptop computers, collaborative multi-pointer applications are an opportunity for broader access to computers in third world countries. Pawar et al. [17, 18] evaluated single-display multi-mouse applications with children. They observed significantly less fighting for the mouse in a multi-mouse setting, but in competitive situations, older children had a clear advantage. The authors concluded that simultaneous interaction has a positive effect on the value of a PC for education, but users should be granted their own personal space within multi-user applications. Scott et al. [21] compared teams of users solving a collaborative task in three different settings: sharing a single mouse and display, using two independent mice on a shared display, and solving the task together on paper. When sharing a single mouse, users collaborated less and interacted strictly sequentially opposed to parallel interaction when working with two independent mice or on paper.

With the *Multi-Pointer X Server* (MPX), Hutterer and Bruce added multi-pointer support to the *X Window System*, the prevalent graphics subsystem for Linux [11, 12]. This low-level extension allows multiple independent input devices to be used to control GUI applications with the window manager solving cursor ambiguity. With MPX, each connected mouse/keyboard pair controls its own pointer, allowing for simultaneous interaction – even within the same window and with support for legacy applications. In 2009, Chris Ball extended the *Vino* VNC server to support MPX and thus allow multi-pointer support during remote access [1]. Neither Windows nor macOS natively support multiple independent pointers. Therefore, workarounds are required. TIDL [10] achieves multi-pointer interaction by using Java's *GlassPane* as a screen overlay that renders multiple cursors. However, interaction techniques such as drag and drop had to be re-implemented to work with TIDL. *Metamouse* [7] adds multi-mouse support to Windows applications by rendering each user's cursor independently using Westergaard's *CPNmouse* API [26]. *Metamouse* does not allow for simultaneous input but displays an additional meta cursor between all user's cursors. Clicks are only propagated to an application if all users agree by positioning their cursors close together. *TWICE* [6, 20] is a JavaScript toolkit for creating web-based multi-pointer applications that can be accessed via URL with a normal web browser. Even though authors provide an in-depth explanation of their design decisions and implementation, they do not show any applications developed with their toolkit, or results from possible user studies.

In summary, previous work investigated different models of co-located and remote collaboration. With common screen sharing and remote access tools, there is a clear difference in agency as only a single user is in control at a time - others can just watch and comment. There is some evidence that synchronous multi-pointer interaction gives users a feeling of agency and may improve team performance. While cloud-based collaborative editors allow for simultaneous interaction with documents and canvases, the tool range is limited, and the experience is controlled by the cloud service.

3 DESIGN AND IMPLEMENTATION

With *CoShare*, our goal is to explore how screen-sharing can be turned from a broadcasting medium to an interactive medium. Our interaction concept is inspired by physically co-located collaborative work where *visitors* are invited into the *host*'s home or office, given access to facilities and tools, and equally contribute to the task at hand.

Using a *design science* approach [8, 9], we iteratively developed a prototype of a collaborative screen-sharing application. First we gathered requirements and attitudes in two focus groups. Based on the requirement analysis, we developed a proof-of-concept prototype and evaluated its utility and limitations in a small user study.

3.1 Requirement Analysis

To learn about needs of users in remote collaborative work, we conducted two remote focus groups with four respectively six participants via Zoom. All participants were local media informatics students (21–26 years, 7m, 3f) and had experience with a variety of screen-sharing tools, for example for collaborating remotely with others to work on university projects. The selected group is representative of the typical audience we envision for CoShare.

Both focus groups had the same structure. First, groups of two participants were asked to consolidate the advantages and disadvantages of existing screen-sharing tools and collaborative work

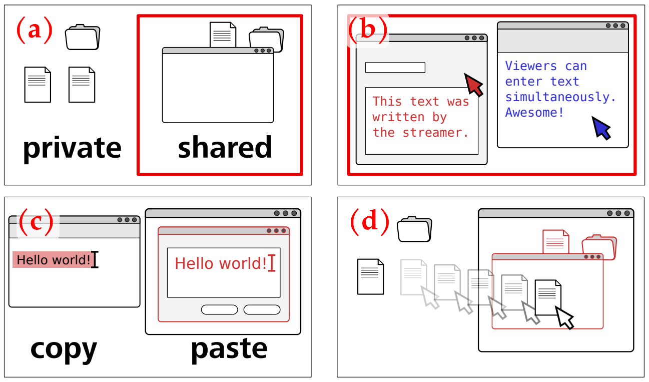
(a)
private
shared
(b)
This text was written by the streamer.
Viewers can enter text simultaneously. Awesome!
(c)
Hello world!
Hello world!
copy
paste
(d)

Figure 2: Interaction concepts for *CoShare*: Streamers can share any rectangular screen region while parts of their desktop remain private (a). All collaborators can work simultaneously with independent mouse and text cursors (b). Viewers can paste their clipboard content to the shared workspace (c). Viewers can transfer files from their PC to the shared workspace using drag and drop (d).

solutions. After that, we presented initial concepts for an interactive screen-sharing application, such as sharing a partial region of the host's screen and redirecting the visitor's mouse and keyboard input to the host's desktop. Participants then discussed the concepts and made suggestions for the features' implementation.

By consolidating the focus groups' transcripts, we identified following general topics (participant ID in parentheses):

To **preserve privacy**, participants appreciated the opportunity to share arbitrary regions instead of a whole window or screen (6). The shared region should be highlighted at all times (3) and re-scaling should be possible (3). To **interact with the shared content**, mouse and keyboard input within the shared region should be propagated through the stream (5). **Parallel interaction** requires simultaneous and independent inputs (4). Visual clues should help to distinguish cursors (3). The application should include a direct way of **data transfer** between visitor and host, avoiding the need for a separate file sharing platform (4). The system should be as **simple to use** as possible, e.g., by leveraging and supporting established interaction techniques, such as copy/paste or drag/drop (5). **Performance** should be good enough not to distract users and latency should be minimized (5). Furthermore, participants suggested integrating these features into an existing cross-platform video conference software (5), an option for the host to restrict the others' interaction for security reasons (5), and the option to open shared files with their own preferred tools (3).

Interaction concepts derived from the requirement analysis are depicted in Figure 2.

3.2 Design Decisions and Implementation

Based on the requirement analysis, as well as on consensus from related work, we made several design decisions for *CoShare*. For the first design iteration, we focused on basic interaction concepts, data transfer, and privacy features. Therefore, we postponed cross-platform support and security features, and only support a single visitor.

In the current prototype, the host can select an arbitrary rectangular screen region to be streamed (Fig. 3b). Parts of application windows in the selected region are shared while everything else remains private. On the host's desktop, the shared area is indicated by a red border (Fig. 3c). On the visitor's desktop, the stream is displayed within an application window (Fig. 3d). This way, all collaborators can have their own private area, similar to the suggestions of Pawar et al. or Barry et al. [2, 18]. Independent cursors for each user allow for pointing, clicking, and typing simultaneously at different positions on the screen without interfering with each other's input during parallel work. Each participant's cursor is marked with a distinctly colored dot. Each participant's keyboard is linked to their mouse pointer, so the window/widget focused by the mouse pointer receives keyboard input. This allows users to work in different applications on the host's desktop in parallel. Applications that explicitly support multi-pointer interaction may also be used simultaneously by multiple participants at once. Clients can copy and paste text through the shared screen using the usual shortcuts *CTRL+C* and *CTRL+V* without affecting the host's clipboard. Visitors can transmit files to the host by dragging a file on their own computer into the application window. Dragging files from the host's desktop to the visitors is not possible with the current prototype.

We developed *CoShare* for GNU/Linux as it is the only major operating system that with native multipointer support and provides easy access to input events and clipboard. The application is written in Python 3.8 with PyQt5 as UI toolkit. A few components are written in C. Visitors can join the stream via a tray menu (Fig. 3a). The host then receives a registration request and starts streaming the shared screen region which is displayed in a window on the viewer's system. All real-time communication between host (server) and visitor (client) happens via UDP. For transferring files, we use a *Flask* HTTP server running on the host's system. For the video stream, we use *gstreamer*[1]. We use the MPX [11] to allow for parallel work with multiple mouse cursors and virtual keyboards. The visitor's keyboard input is captured using *evdev*[2], and mouse events are retrieved via *pynput*[3]. All captured input events within the visitor's application window are sent to the host as evdev events. On the host's side, received events are assigned to a virtual input device and input events are emitted using *uinput*. The keyboard shortcuts CTRL+C/V are not forwarded to the host. Instead, when copying from the stream window, the host's clipboard content is transferred to the visitor's local clipboard. If they paste into the stream window, the visitor's clipboard is sent to the host and pasted at the position of the visitor's cursor. To allow for file transfer via drag and drop, we capture drop events on the stream window with PyQt5 and send the dropped file to the HTTP server running on the host's machine where a drop event is simulated.

[1]https://gstreamer.freedesktop.org/
[2]https://python-evdev.readthedocs.io/
[3]https://pypi.org/project/pynput/

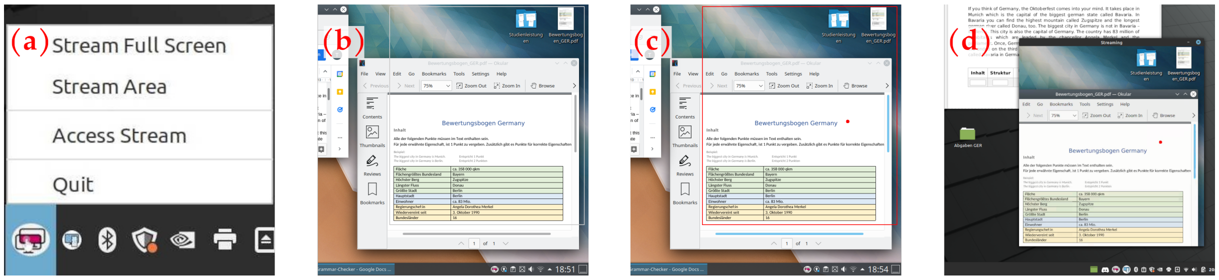

Figure 3: Different states of *CoShare*. The application can be controlled via a tray menu (a). The shared area is highlighted by a border. The border is initially gray (b) and turns red once a viewer joins (c). The viewer's cursor position is marked by a red dot (c). On the viewer's side, a window displays the shared area of the host's screen with which the viewer can interact (d).

4 EVALUATION

We evaluated *CoShare* in a small formative user study. We recruited twelve participants (21–25 years, 9m / 3f) with and without a technical background to cover different perspectives on the system and different approaches to solving the task. We designed a scenario which represents a realistic use case for collaboration: a teacher (host) and a tutor (visitor) work together to grade assignments of a language course and record student's grades in a summary sheet. The assignments, short texts about the USA and Germany, were stored on the tutor's computer. The teacher had an assessment sheet with a list of requirements for the assignments, the summary sheet, and a language tool to help find grammar and spelling mistakes. For each group, both participants were seated at separate desks in our laboratory. On each desk, there was a PC running *CoShare*. The experimenter observed the study from a distance and took notes.

First, we informed participants about the study's procedure and asked them for demographic data with a questionnaire. Then, we introduced participants to the prototype and explained the task. Additionally, we provided a printed sheet with a task description and information on where to find required documents and tools on their computers. Each group graded eight texts in total, switching roles after the first half. As we intend *CoShare* to be used together with voice chat or video conference software, participants were allowed to talk to each other during the study, for example to discuss their strategy to solve the tasks. The study was followed by a semi-structured interview to reflect participants' usage of *CoShare*. We asked them for a comparison between the host's and the visitor's part, their opinion on provided features, if they missed any features, and which problems occurred or might have occurred using the system. We then asked them for advantages and disadvantages of *CoShare* compared to existing screen-sharing tools, and during which use cases they would benefit from an interactive screen share.

5 RESULTS AND DISCUSSION

We transcribed and paraphrased all interviews and used inductive category formation [15] to group results.

Streaming and Viewing: According to eight participants, sharing a selected region makes sense to preserve privacy and hide irrelevant areas. However, new windows might open within the shared region. Five participants found *CoShare* most useful when using a separate screen as shared space. Five participants wished to be able to re-scale the streamed region during runtime. Sometimes, compression artifacts degraded the stream and there was no indication for the viewer when the streamer stopped the stream.

Overall, **Interacting with the Stream** was appreciated by the participants. Four participants noted that most applications do not support simultaneous input. Two participants found the mouse cursor indicator not clear enough.

Text and Data Transfer: Due to a bug with the shared clipboard, *CoShare* occasionally pasted the streamer's clipboard content instead of the viewer's when they pressed *CTRL+V*. For four participants, the clipboard worked flawlessly, and two participants did not use it. Four participants found the drag and drop feature simple and useful. Two participants had safety concerns, as the streamer can not opt out of receiving files at the current state of development.

Feature Requests: All participants found *CoShare* useful for collaborating, but agreed that some kind of voice chat is necessary when working remotely. Six participants requested the option for streamers to bind the stream to an application window, so it can be moved or minimized without affecting the stream. Four participants wished to use an independent system to initialize the stream, so all users are equal. Six participants requested bi-directional data transfer. Annotations on the streamer's screen, as well as stream recording and rewinding were mentioned by one participant each.

Advantages vs. Screen Sharing: Seven participants saw *CoShare*'s biggest advantage in the simple data transfer. Seven participants appreciated increased efficiency as they could do things themselves instead of describing them to the streamer. Two participants found *CoShare*'s context independence to be an advantage over cloud-based tools.

Use Cases: Participants identified several use cases for *CoShare*: teaching (3), group projects (4), pair programming (2), collaborative drawing (1), helping/guiding someone else (1), checking on a colleague's progress without disturbing them (1), and remote meetings with friends (1) or colleagues (1).

Overall, participants liked concept and implementation. Focus groups and study highlight important aspects that need to be considered when developing multi-pointer screen-sharing tools: privacy and security concerns need to be addressed. These mechanisms should be based on existing mental models. Making only a part of the screen accessible to visitors is a useful feature. Sharing clipboard contents and files is an important feature that needs to be fleshed out.

6 CONCLUSION AND FUTURE WORK

We presented *CoShare*, a proof-of-concept implementation for a novel approach to computer-supported remote collaboration. We could show that interaction through a shared screen is possible with only a few low-level features such as input forwarding or data transfer. In contrast to traditional screen sharing, agency is distributed evenly between participants as all have simultaneous access to files and applications in the shared area. Furthermore, *CoShare* allows users to solve tasks with tools and workflows they are familiar with, opposed to restrictive applications targeted at specific tasks that come with current approaches to collaborative work. While most applications are not built with multi-pointer support in mind, *CoShare* allows for working in parallel on separate applications. For an ideal solution, true multi-pointer support would include features such as multiple cursors in a single text input widget. Solving this problem would require major modifications of existing software, for example by integrating multi-pointer support into established UI frameworks. Future development of *CoShare* will focus on multi-platform support, improved data transfer between participants, and integration of a video/voice channel. After implementing these features, we will be able to compare *CoShare* to established systems for remote collaborative work. Additionally, we plan to evaluate how to visually distinguish the different cursors effectively without distracting the users.

Source code of *CoShare* is publicly available under an open source license[4].

REFERENCES

[1] Chris Ball. 2009. Multi-pointer Remote Desktop. https://blog.printf.net/articles/2009/01/26/multi-pointer-remote-desktop/

[2] Lior Berry, Lyn Bartram, and Kellogg S. Booth. 2005. Role-based control of shared application views. In *Proceedings of the 18th annual ACM symposium on User interface software and technology (UIST '05)*. Association for Computing Machinery, New York, NY, USA, 23–32. https://doi.org/10.1145/1095034.1095039

[3] Jacob T. Biehl and Brian P. Bailey. 2004. ARIS: an interface for application relocation in an interactive space. In *Proceedings of Graphics Interface 2004 (GI '04)*. Canadian Human-Computer Communications Society, Waterloo, CAN, 107–116.

[4] Eric A. Bier, Steve Freeman, and Ken Pier. 1992. MMM: The multi-device multi-user multi-editor. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*. ACM Press, Monterey, California, United States, 645–646. https://doi.org/10.1145/142750.143065

[5] Kellogg S. Booth, Brian D. Fisher, Chi Jui Raymond Lin, and Ritchie Argue. 2002. The "mighty mouse" multi-screen collaboration tool. In *Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST '02)*. Association for Computing Machinery, New York, NY, USA, 209–212. https://doi.org/10.1145/571985.572016

[6] Muriel Bowie, Oliver Schmid, Agnes Lisowska Masson, and Béat Hirsbrunner. 2011. Web-based multipointer interaction on shared displays. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work (CSCW '11)*. Association for Computing Machinery, New York, NY, USA, 609–612. https://doi.org/10.1145/1958824.1958926

[7] Kurtis Heimerl, Divya Ramachandran, Joyojeet Pal, Eric Brewer, and Tapan Parikh. 2009. Metamouse: Multiple Mice for Legacy Applications. (2009), 6.

[8] Alan Hevner and Samir Chatterjee. 2010. Design Science Research in Information Systems. In *Design Research in Information Systems*. Integrated Series in Information Systems, Vol. 22. Springer US, Boston, MA, 9–22. http://link.springer.com/10.1007/978-1-4419-5653-8_2 https://doi.org/10.1007/978-1-4419-5653-8_2.

[9] Alan Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. Design Science in Information System Research. *MIS quarterly* 28, 1 (March 2004), 75–105.

[10] Peter Hutterer, Benjamin S. Close, and Bruce H. Thomas. 2006. TIDL: mixed presence groupware support for legacy and custom applications. In *Proceedings of the 7th Australasian User interface conference - Volume 50 (AUIC '06)*. Australian Computer Society, Inc., AUS, 117–124.

[11] Peter Hutterer and Bruce H. Thomas. 2007. Groupware support in the windowing system. In *Proceedings of the eight Australasian conference on User interface - Volume 64 (AUIC '07)*. Australian Computer Society, Inc., AUS, 39–46. https://doi.org/10.5555/1273714.1273721

[12] Peter Hutterer and Bruce H Thomas. 2008. Enabling Co-located Ad-hoc Collaboration on Shared Displays. Wollongong, NSW, Australia, 8.

[13] Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. 2002. PointRight: experience with flexible input redirection in interactive workspaces. In *Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST '02)*. Association for Computing Machinery, New York, NY, USA, 227–234. https://doi.org/10.1145/571985.572019

[14] Zhangbo Liu. 2007. *Lacome: a cross-platform multi-user collaboration system for a shared large display*. Ph. D. Dissertation. University of British Columbia. https://doi.org/10.14288/1.0051402

[15] Philipp Mayring. 2015. Qualitative Content Analysis: Theoretical Background and Procedures. In *Approaches to Qualitative Research in Mathematics Education: Examples of Methodology and Methods*, Angelika Bikner-Ahsbahs, Christine Knipping, and Norma Presmeg (Eds.). Springer Netherlands, Dordrecht, 365–380. https://doi.org/10.1007/978-94-017-9181-6_13

[16] J. F. Nunamaker, Alan R. Dennis, Joseph S. Valacich, Douglas Vogel, and Joey F. George. 1991. Electronic meeting systems. *Commun. ACM* 34, 7 (July 1991), 40–61. https://doi.org/10.1145/105783.105793

[17] Udai Singh Pawar, Joyojeet Pal, Rahul Gupta, and Kentaro Toyama. 2007. Multiple mice for retention tasks in disadvantaged schools. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. Association for Computing Machinery, New York, NY, USA, 1581–1590. https://doi.org/10.1145/1240624.1240864

[18] Udai Singh Pawar, Joyojeet Pal, and Kentaro Toyama. 2006. Multiple Mice for Computers in Education in Developing Countries. In *2006 International Conference on Information and Communication Technologies and Development*. 64–71. https://doi.org/10.1109/ICTD.2006.301840

[19] T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper. 1998. Virtual network computing. *IEEE Internet Computing* 2, 1 (Jan. 1998), 33–38. https://doi.org/10.1109/4236.656066

[20] Oliver Schmid, Agnes Lisowska Masson, and Béat Hirsbrunner. 2014. Real-time collaboration through web applications: an introduction to the Toolkit for Web-based Interactive Collaborative Environments (TWICE). *Personal and Ubiquitous Computing* 18, 5 (June 2014), 1201–1211. https://doi.org/10.1007/s00779-013-0729-0

[21] Stacey D Scott, Garth B D Shoemaker, and Kori M Inkpen. 2000. Towards Seamless Support of Natural Collaborative Interactions. In *Proceedings of the Graphics Interface 2000 Conference*. Montréal Québec Canada, 8.

[22] Bastian Steinert, Michael Grünewald, Stefan Richter, Jens Lincke, and Robert Hirschfeld. 2009. Multi-user multi-account interaction in groupware supporting single-display collaboration. In *2009 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing*. 1–9. https://doi.org/10.4108/ICST.COLLABORATECOM2009.8290

[23] Jason Stewart, Benjamin B. Bederson, and Allison Druin. 1999. Single display groupware: a model for co-present collaboration. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '99)*. Association for Computing Machinery, New York, NY, USA, 286–293. https://doi.org/10.1145/302979.303064

[24] Grant Wallace, Peng Bi, Kai Li, and Otto Anshus. 2004. A Multi-Cursor X Window Manager Supporting Control Room Collaboration. (2004), 7.

[25] Jim Wallace, Vicki Ha, Ryder Ziola, and Kori Inkpen. 2006. Swordfish: user tailored workspaces in multi-display environments. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems (CHI EA '06)*. Association for Computing Machinery, New York, NY, USA, 1487–1492. https://doi.org/10.1145/1125451.1125724

[26] Michael Westergaard. 2002. Supporting Multiple Pointing Devices in Microsoft Windows. In *Proceedings of Microsoft Summer Workshop for Faculty and PhDs*. Cambridge.

[4]https://github.com/PDA-UR/CoShare/