Collaborative Distributed Cognition Using A Seamless Desktop Infrastructure

Thomas Geymayer* Dieter Schmalstieg†

Graz University of Technology

Figure 1: The tiled display in our visualization laboratory is build from 24 monitors (50" diameter, $1920x1080$ resolution.)

Abstract

Inexpensive displays make large, tiled displays attractive for visual analysis and collaborative investigation. Especially in multi-user environments, the increased space allows to better organize the findings and results and, therefore, helps to improve collaboration. One important requirement is that all users can navigate seamlessly on the whole display space, while employing the standard software they are familiar with. In this paper, we present a seamless desktop infrastructure for distributed cognition and collaboration. Our infrastructure only uses standard hardware and software. By choosing a minimally invasive, web-centric approach, we can integrate existing web applications and visual analysis software with little or no effort into our system. We can even leverage existing synchronization mechanisms built into many web applications today.

Keywords: Distributed Cognition, Single Display Groupware, Collaboration, Minimally invasive infrastructure.

Index Terms: H.5.2 [Information Interfaces and Presentation]: User Interfaces—Graphical User Interfaces

*e-mail: geymayer@icg.tugraz.at

†e-mail: schmalstieg@icg.tugraz.at

1 Introduction

Large tiled displays can now be built at a reasonable cost. Consequently, they have become popular for visual analysis, since they provide *space to think* [2]: Digital information can be arranged spatially on a large interactive surface. Relying on the layout of visual information items to externalize one's thought processes is considered an important part of *distributed cognition* [12]. Moreover, a large display area naturally lends itself to collaborative work, since it provides space for concurrent viewing by multiple users.

These considerations have led to a variety of frameworks for collaborative analysis [6, 32]. However, these frameworks have had relatively little impact on visual analysis practice in the wild, since they were designed for research and often lack many mundane features which are important for everyday use. It is much more likely that a typical user of professional analysis software employs two or three large monitors, but always connected to a standard workstation running a standard graphical user interface (GUI). Neither the application software nor the underlying hardware/software platform supports large tiled displays or is designed for collaborative use.

With a GUI infrastructure that is able to span one *seamless desktop* across displays belonging to multiple computers, we could use existing, unmodified applications for distributed cognition. However, existing seamless desktop solutions based on remote desktop streaming, such as VNC [22] or Microsoft RDP, do not scale well. And even if we could use these solutions, a seamless desktop per se does not provide sufficient collaboration support, since only one

user can be active at a given time.

Clearly, we need an alternative approach to build a collaborative seamless desktop. We build on the observation that visual analysis software, like most types of information systems, is moving towards a *web-centric architecture*. Today, an analyst or information worker wishing to conduct everyday work entirely using web applications can do so with relatively few restrictions. There are even dedicated platforms for this form of computing, such as Chromebooks. A desirable side-effect of the move towards web applications is that the client-server model makes support for collaboration significantly easier. As a consequence, many web applications, such as Google Docs, automatically synchronize application state between multiple users in real time.

In this paper, we describe how the combination of a seamless desktop with web-centric applications yields an infrastructure for collaborative distributed cognition. We build this infrastructure on top of a standard GUI and a standard web-browser using a *minimally invasive* approach: Installing a lightweight custom software package on every involved computer is sufficient to add the new capabilities. Existing applications do not require any modification and continue to work normally, even if they are disconnected from our infrastructure. To our knowledge, our system is the first to let multiple users operate with multiple unmodified applications on a seamless desktop.

2 Related work

Several special-purpose applications have been developed specifically with collaboration support in mind. These information systems include topics such as web search [1, 28], text analysis [15], collaborative picture galleries [20] and network visualization [14].

Tools supporting co-located, synchronous user interactions do not need to be built from scratch. Forlines et al. [10] built a wrapper for Google Earth to support viewport synchronization, multi-user interaction, and annotations, without changing the applications core implementation. Isenberg et al. [14] are following a similar approach, by, for example, adding multiple color-coded mouse cursors within an existing applications. Both tools still need modifications to support collaboration and therefore can not easily be combined with other groupware applications.

To overcome this limitations in conventional windowing systems, Hutterer and Thomas [13] created multi-pointer X (MPX), an extension to the X Window System for multi-pointer interaction. This is in line with the observations of Lauwers and Lantz [18] which have already suggested using existing single-user applications in any windowing system with collaboration support. Though, this approach does not support any more advanced collaboration features, such as interaction histories, access control mechanisms or any automated translation or communication between applications.

Adding multi-input support to a single, shared display is commonly referred to as a single-display groupware (SDG) [25]. Typical examples for SDG systems are wall displays [5, 14] or tabletops [15, 20, 27, 28]. One problem reported commonly is the mutual distraction of users while performing individual work. For example cursor movements by other users [31] and changes of the spatial display layout [9] are often causes for disruption. As a result multiple users often use their personal territories on a high-resolution display to visualize a lot of information simultaneously [14, 28]. On large displays common problems include locating the cursor and items of interest within the whole information space [26] and interacting with distant display locations [23]. Spotlights [17] have been introduced to guide the users attention to individual regions on large screens. VisLink [7] and visual links across applications [29] use visual line connections to guide users to related elements (eg. text or map locations) inside distinct application windows as well as outside the users visible field of view. Visual links have also been shown useful in multi-user SDG systems to show relations between information in personal windows and shared information [30].

Currently many visual analytics applications are adopting web technologies. To handle multiple multiple devices and users, several frameworks allow application developers distributing their web applications across multiple displays. An early example is Multibrowsing [16] which allows moving information to different devices connected to the system. More recent systems, like Panelrama [33] and PolyChrome [3], allow splitting the user interface across multiple devices, but require a deeper integration with the individual applications. SAGE2 [19] also makes heavy use of modern web technologies, and provides a multi-user interaction capable windowing system running in the browser.

3 System Overview

Instead of building a monolithic framework, we *stitch* together individual desktops of multiple computers such that the users have the illusion of working on a seamless desktop. This provides three important advantages: First, the engineering effort for the stitching is much smaller than development of a new monolithic framework, which would have to duplicate many existing GUI functions. Second, existing GUI functions, such as window management, do not have to be duplicated. Third, existing application software does not have to be re-written or ported to the new framework.

3.1 Requirements

What is required to successfully convey the illusion that users are interacting together on a seamless desktop? The first group of requirements concerns the users' input devices. We assume one mouse and one keyboard per user. Users should be able to use their input devices as usual, without having to consider display boundaries or coordinate with other users. This implies the following requirements:

- R1 Multiple computers should be operated with one set of input devices.
- R2 Multiple sets of input devices should be usable on one computer, including multiple mouse cursors.
- R3 Multiple application foci should be provided, so that multiple users can interact one-on-one with applications on a single computer.

The second group of requirements concerns the applications:

- R4 Multiple applications should run concurrently, either on the same or on different computers.
- R5 Multiple clones of one application window can be created and linked to the state (e. e., viewport scrolling) of the master window.
- R6 Multiple instances of a document, which are being modified by multiple users, should be automatically synchronized.

We begin with an overview of the software architecture, and then look at the important system components in the following sections.

3.2 Overall architecture

Before we discuss in detail how the requirements set forth in section 3.1 are addressed, we give an overview of the overall software infrastructure. It is intended for a cluster of computer nodes connected over a local area network. The cluster driving our tiled display consists of eight nodes. Each node is a standard computer with three displays (50", 1920×1080 resolution, arranged vertically), a

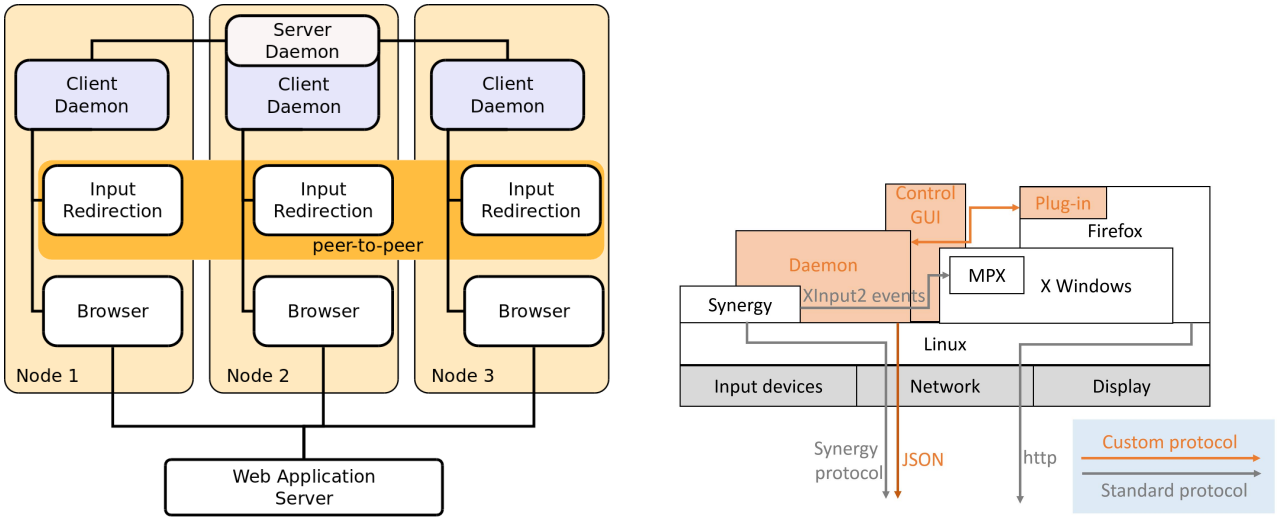

Figure 2: (left) The network architecture combines hierarchical communication for system management and web applications with peer-to-peer components for input redirection. (right) Standard software components (white) are extended with a minimal set of custom software components (orange). The custom software extends, but does not restrict, standard desktop use.

keyboard and a mouse. Additional computers, such as notebooks, can be added via Wifi at any time, even in the middle of a session.

Each node runs a standard GUI (Ubuntu Linux with Unity Desktop) and a standard browser (Mozilla Firefox) with custom extensions. The extensions use only the standard API provided by the operating system and browser. This approach differs from previous seamless desktop infrastructures, which wrap all the physical computing resources into a monolithic software framework, replacing the standard API with a proprietary one.

Figure 2 illustrates the software architecture, with standard software components shown in white. The standard components provide a conventional, self-contained GUI. By installing the proprietary software components (orange), we add additional capabilities.

On each node, a daemon must be started, which can work as only as a client or – additionally – take on the function of a server. Exactly one daemon in the local network must be configured as the server. It advertises its address via periodic broadcasts. Client daemons listen to the broadcasts and connect to the advertised server address. After establishing the connections, the server brokers communication among clients, using a protocol consisting of simple JSON strings.

The server also maintains a list of connected clients. To facilitate a seamless desktop, the arrangement of the displays on each client node is recorded in a spatial model maintained by the server. A regular lattice of displays, such as in our laboratory, is trivially recorded by hand. However, we have developed automatic tools for detecting irregular display configurations and even movable displays such as notebooks [21]. In addition to the spatial model, the server maintains a list of user records, indicating the node where the user's input devices are connected and assigning a distinct cursor color for each user. A control GUI allows interactive setting of these parameters.

Every daemon, including the server, manages activities on the local node:

1. The daemon starts a local instance of the input redirection software (see section 4) and controls it via a local connection.

2. The daemon accepts local connections from web-browser instances or other applications that have been extended to communicate with the daemon. The web-browser communicates via a WebSocket opened by a plug-in written in JavaScript. This approach effectively establishes a two-tier communication architecture, with daemons forming the first level and web-browsers forming the second level.

3. The daemon opens a *glass-sheet*, a transparent window covering the entire desktop, used to display visualizations and user interface elements for collaboration.

4 Input Device Management

We address basic input redirection (R1) with the open-source software *Synergy* [1]. Events from the physical input devices are intercepted and injected into the event manager of the destination computer. On the destination computer, the input events appear to be coming from virtual input devices, while the events are actually received over the network.

We have modified Synergy to use a peer-to-peer architecture [8], rather than the standard client-server architecture. The standard version of Synergy installs a a device server on the computer where the physical input devices are attached and device clients on the remote controlled computers. Our modified version installs the same software component on each computer, which combines the capabilities of Synergy server and client (R2): On the one hand, every computer injects events from the attached input devices into the network, tagged with one particular user's identity. On the other hand, every computer receives events from all other computers and determines if they concern the desktop area managed by that computer, based on the spatial model we provide.

Moreover, we install the X-Window System [24] with support for *Multi-Pointer X* (MPX), which allows using multiple sets of input devices on one computer [13]. MPX supports multiple concurrent window foci (R3), as long as the applications running under MPX correctly distinguish events from different input devices. Fortunately, this is – mostly – the case for contemporary Linux software, in particular, the Firefox web-browser, which is used as the main application platform in our system. Since a standard window manager cannot render multiple mouse cursors without flickering, we use the glass-sheet to render multiple cursors in distinct colors.

MPX was originally intended for use with multiple physical input devices connected to the same computer. However, we have

[1] http://synergy-project.org/

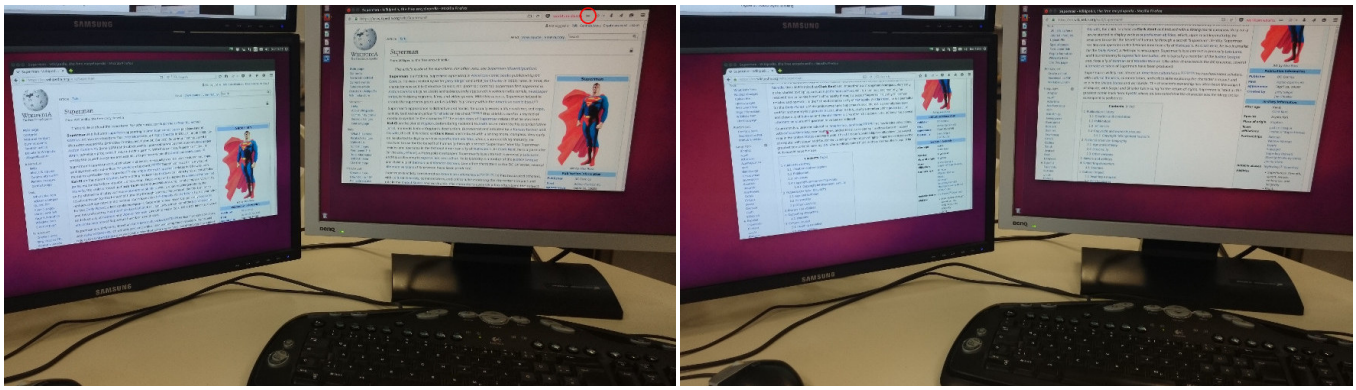

Figure 3: Cloning a web-browser on another desktop. An icon in the browsers toolbar indicates that it is synchronized to another browser (red circle in the upper image). After moving the viewport in the source browser (left), the viewport of the synchronized browser (right) has automatically been updated.

extended it to accept the virtual input devices provided by our input redirection component. Consequently, we arrive at the combined effect of being able to control any application on any display connected to any computer using any input device. Per default, we make all displays accessible. However, if a user desires privacy for a personal display, such as a notebook computer, the users allowed to enter the personal display can be restricted.

Of course, using multiple input devices in the same application instance only works with applications designed for multi-user operation, which is rarely the case. Fortunately, this restriction turns out to be of little practical consequence. Concurrent operation inside one application instance is rarely an important requirement for analysis work. Collaborators mostly take turns between working in parallel with separate documents and discussing common findings, with one user being active at a time [11]. With ample display space, joint inspection and manipulation of the same document can be carried out in two synchronized windows, as we will explain in the next section.

5 Web Application Management

Support for multiple concurrent applications on one computer (R4) is trivially fulfilled by a multi-tasking system such as Linux. Each computer in the cluster runs arbitrary native application, so that users can keep using their favorite tools.

However, our main application platform is the web-browser. It puts an additional level of indirection between applications and the operating system. For example, applications and data in web applications are loaded on demand, avoiding the need to install applications before use. This makes it easy to migrate application state across computers.

Conventional static web pages and server-side technologies such as REST encode the user's document view in the URL. Feeding this URL into a web-browser on another computer results in displaying a clone of the same web page, i. e., a document view with the same content.

We exploit this fact by letting the user invoke a cloned web-browser instance with the same URL on any computer in the cluster. The clone can be coupled to the master instance (Figure 3), so that any viewport manipulation (scrolling, resizing, scaling) or page change of the master instance is reflected by the clone (R5). This is useful for "teleporting" information across large display areas to make them better readable for a collaborating user [4]. We also allow to reserve the roles of master and clone or enable bi-directional synchronization, as long as only one user at a time manipulates the browser.

By combining input redirection and web-browser cloning, the illusion of a seamless desktop, which behaves like a conventional desktop, is achieved, include a *migrate operation*: Assume that the user wants to drag a web-browser window from a location on the source node's desktop to a location on the target node's desktop.

While traversing the source node's desktop, normal dragging is carried out by the local window manager. As soon as the mouse moves to the target node' desktop, an event in the input redirection component is triggered. A preview picture of the web-browser window is rendered and transmitted to the daemon on the target node, along with the current URL of the dragged web-browser (Figure 4).

At the target node, the preview picture is attached to the mouse cursor and rendered on the glass-sheet. When the user drops the web-browser in the target location, the daemon on the target note creates a new web-browser instance with the URL specified in the input redirection event.

In a final step, the source web-browser is deleted to finish the illusion that the web-browser has been migrated to the new location. Alternatively, a *copy operation* retains both web-browser instances and synchronizes their behavior.

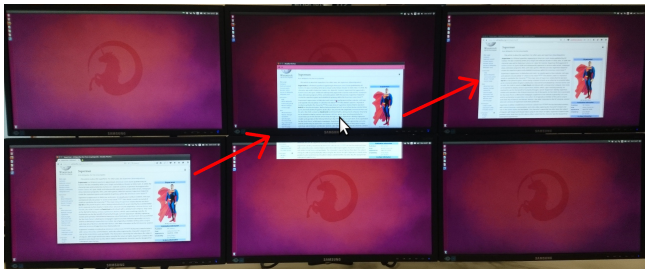

Figure 4: Migrating a browser window to another desktop. The source window is shown on the left, a preview while dragging in the middle, and the target window on the right.

Since we allow arbitrary web applications, document manipulations across browser instances are not automatically synchronized. However, many web applications are able to synchronize with their web server in real-time. Changes reported to the web server are forwarded to other users viewing the same document, effectively allowing shared work (R6).

One could serialize the full internal state of a master web-browser and duplicate it in a clone. However, for the analysis scenarios investigated so far, we found it sufficient to rely on cloning the URL and using shared web applications such as Google Docs or Wikipedia.

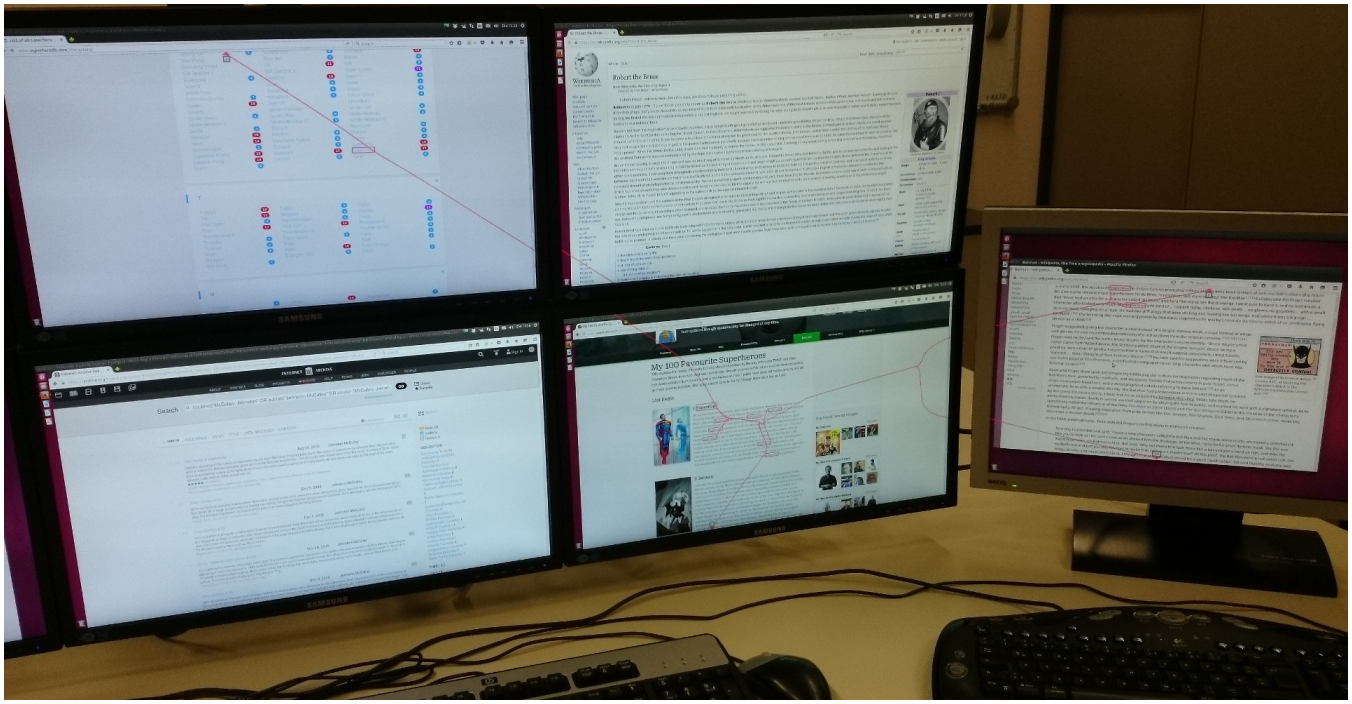

Figure 5: Visual links can span keyword instances across multiple physical desktops.

6 Visual links

Visual links have been used to guide the users attention to important information on large screen areas [29]. The same visualization technique can also be used on our seamless desktop to increase the level of collaboration (Figure 5).

Users can highlight a keyword or section in a web-browser and invoke visual links to guide other users to other occurrences of the search term. Occurrences are identified by searching the web-browsers document object model (DOM). Since most web applications store its data in the DOM (including alt tags for images), this approach yields good results without knowing details about the applications in advance. The search term is sent to the server daemon and forwarded via client daemons to the web-browser instances. Matches are returned to the server daemon, which computes the global routing of the visual links. The routes are forwarded to the client daemons, which display them by drawing colored curves on the glass-sheets.

Extending web applications to provide support for other data sources is rather straight forward using a small amount of custom JavaScript that answers requests in JSON via the WebSocket interface provided by the browser plug-in. For example, we have created a mashup using the Google Maps API [2] to show geographic locations corresponding to a textual identifier, such as an address.

7 Discussion

The conceptually closest work to ours is SAGE2 [19], which shares many of the design objectives with our framework. The creators of SAGE2 state that among their key objectives are (1) integration of multi-user applications, (2) enhanced real-time distance collaboration, and (3) a reduced barrier to entry. Like us, they choose web technology to address the need for easy entry and for re-use of legacy (web) applications.

However, their approach differs in key design decisions. SAGE2 turns the canvas of a browser displayed full-screen as a replacement for the desktop. By allowing the browser to monopolize all input and output resources, any restrictions of the underlying operating system concerning multi-user operation, screen size etc. are effectively removed.

This comes at a price: SAGE2 must duplicate a substantial amount of operating system functions, such as window management, cursor and focus management, and encapsulation of web page instances inside a browser window or browser tab. SAGE2 requires that web applications must be customized to fit inside a JavaScript container class, and that animations can be synchronized in lock-step by the server. Native legacy applications and web applications that cannot be modified must fall back to pixel streaming from an application server.

In contrast, our framework does not monopolize input and output resources. We extend native system components, such as the window manager, web browser instances, mouse cursor and input focus. All features and behaviors of these system components, including enhancements from third-party desktop tools, are retained. This improves the chances that users can continue with their established work practices and adopt the new collaborative features as an enhancement rather than as a replacement of existing features. We believe that this deep integration with the underlying GUI is important for minimizing disruptive changes in the user experience and ensuring maximum productivity of the users. We plan on conducting a user study to verify these hypothesis.

8 Conclusion

We have presented the first infrastructure that spans a seamless desktop across multiple computers without wrapping everything into a monolithic software architecture without any real applications. Instead, we rely on existing, mature web applications to manage the content and provide content-level synchronization across multiple computers and users. We achieve the illusion of a seamless desktop by the combination of software components for input redirection and output ”redirection”; the latter in the form of web-browser instances that are manipulated by a custom software

[2] https://developers.google.com/maps/documentation/javascript/

component. Our infrastructure is easy to set up and maintain, and leaves room for exploring new collaborative analysis strategies via arbitrary web application software. We are currently preparing a series of experiments on such collaborative work strategies.

ACKNOWLEDGEMENTS

The authors wish to thank Mark Dokter, Manuela Waldner, Alex Lex, Marc Streit and Markus Steinberger.

REFERENCES

[1] S. Amershi and M. R. Morris. CoSearch: A System for Co-located Collaborative Web Search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1647–1656, New York, NY, USA, 2008. ACM.

[2] C. Andrews, A. Endert, and C. North. Space to Think: Large High-resolution Displays for Sensemaking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 55–64, New York, NY, USA, 2010. ACM.

[3] S. K. Badam and N. Elmqvist. PolyChrome: A Cross-Device Framework for Collaborative Web Visualization. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, ITS '14, pages 109–118, New York, NY, USA, 2014. ACM.

[4] A. Bezerianos and R. Balakrishnan. View and space management on large displays. *Computer Graphics and Applications, IEEE*, 25(4):34–43, July 2005.

[5] O. Chapuis, A. Bezerianos, and S. Frantzeskakis. Smarties: An Input System for Wall Display Development. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 2763–2772, New York, NY, USA, 2014. ACM.

[6] M. Chuah and S. Roth. Visualizing common ground. In *Seventh International Conference on Information Visualization, 2003. IV 2003. Proceedings*, pages 365–372, July 2003.

[7] C. Collins and S. Carpendale. VisLink: Revealing Relationships Amongst Visualizations. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '07)*, 13(6):1192–1199, 2007.

[8] M. Dokter. *Synergy+ MPX: Towards Multi-User Interaction in Multi-Display Environments*. Bachelor Thesis, Graz University of Technology, Graz, 2010.

[9] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: Some Issues and Experiences. *Commun. ACM*, 34(1):39–58, Jan. 1991.

[10] C. Forlines, A. Esenther, C. Shen, D. Wigdor, and K. Ryall. Multi-user, Multi-display Interaction with a Single-user, Single-display Geospatial Application. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, pages 273–276, New York, NY, USA, 2006. ACM.

[11] C. Gutwin and S. Greenberg. The mechanics of collaboration: Developing low cost usability evaluation methods for shared workspaces. In *Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE '00, pages 98–103, Washington, DC, USA, 2000. IEEE Computer Society.

[12] J. Hollan, E. Hutchins, and D. Kirsh. Distributed cognition: Toward a new foundation for human-computer interaction research. *ACM Trans. Comput.-Hum. Interact.*, 7(2):174–196, June 2000.

[13] P. Hutterer and B. H. Thomas. Enabling Co-located Ad-hoc Collaboration on Shared Displays. In *Proceedings of the Ninth Conference on Australasian User Interface - Volume 76*, AUIC '08, pages 43–50, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[14] P. Isenberg, S. Carpendale, A. Bezerianos, N. Henry, and J. Fekete. CoCoNutTrix: Collaborative Retrofitting for Information Visualization. *IEEE Computer Graphics and Applications*, 29(5):44–57, Sept. 2009.

[15] P. Isenberg and D. Fisher. Collaborative Brushing and Linking for Co-located Visual Analytics of Document Collections. *Computer Graphics Forum*, 28(3):1031–1038, June 2009.

[16] B. Johanson, S. Ponnekanti, C. Sengupta, and A. Fox. Multibrowsing: Moving Web Content across Multiple Displays. In G. D. Abowd, B. Brumitt, and S. Shafer, editors, *Ubicomp 2001: Ubiquitous Computing*, number 2201 in Lecture Notes in Computer Science, pages 346–353. Springer Berlin Heidelberg, Sept. 2001. DOI: 10.1007/3-540-45427-6_29.

[17] A. Khan, J. Matejka, G. Fitzmaurice, and G. Kurtenbach. Spotlight: directing users' attention on large displays. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*, pages 791–798. ACM, 2005.

[18] J. C. Lauwers and K. A. Lantz. Collaboration awareness in support of collaboration transparency: requirements for the next generation of shared window systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90)*, pages 303–311. ACM Press, 1990.

[19] T. Marrinan, J. Aurisano, A. Nishimoto, K. Bharadwaj, V. Mateevitsi, L. Renambot, L. Long, A. Johnson, and J. Leigh. SAGE2: A new approach for data intensive collaboration using Scalable Resolution Shared Displays. In *2014 International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 177–186, Oct. 2014.

[20] M. Morris, A. Paepcke, and T. Winograd. TeamSearch: comparing techniques for co-present collaborative search of digital media. In *First IEEE International Workshop on Horizontal Interactive Human-Computer Systems, 2006. TableTop 2006*, pages 8 pp.–, Jan. 2006.

[21] C. Pirchheim, M. Waldner, and D. Schmalstieg. Deskotheque: Improved Spatial Awareness in Multi-Display Environments. In *Proceedings of the IEEE Conference on Virtual Reality (VR '09)*, 2009.

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

[23] Y. Rogers and S. Lindley. Collaborating around vertical and horizontal large interactive displays: which way is best? *Interacting with Computers*, 16(6):1133–1152, Dec. 2004.

[24] R. W. Scheifler and J. Gettys. The X Window System. *ACM Trans. Graph.*, 5(2):79–109, Apr. 1986.

[25] J. Stewart, B. B. Bederson, and A. Druin. Single Display Groupware: A Model for Co-present Collaboration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 286–293, New York, NY, USA, 1999. ACM.

[26] K. Swaminathan and S. Sato. Interaction Design for Large Displays. *interactions*, 4(1):15–24, Jan. 1997.

[27] M. Tobiasz, P. Isenberg, and S. Carpendale. Lark: Coordinating Co-located Collaboration with Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1065–1072, Nov. 2009.

[28] P. Tuddenham, I. Davies, and P. Robinson. WebSurface: An Interface for Co-located Collaborative Information Gathering. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, pages 181–188, New York, NY, USA, 2009. ACM.

[29] M. Waldner, W. Puff, A. Lex, M. Streit, and D. Schmalstieg. Visual Links Across Applications. In *Proceedings of the Conference on Graphics Interface (GI '10)*, pages 129–136. Canadian Human-Computer Communications Society, 2010.

[30] M. Waldner and D. Schmalstieg. Collaborative Information Linking: Bridging Knowledge Gaps between Users by Linking across Applications. In *Proceeding of the IEEE Symposium on Pacific Visualization (PacificVis '11)*, pages 115–122. IEEE, 2011.

[31] J. R. Wallace, S. D. Scott, T. Stutz, T. Enns, and K. Inkpen. Investigating teamwork and taskwork in single- and multi-display groupware systems. *Personal and Ubiquitous Computing*, 13(8):569–581, June 2009.

[32] D. Wigdor, H. Jiang, C. Forlines, M. Borkin, and C. Shen. WeSpace: The Design Development and Deployment of a Walk-up and Share Multi-surface Visual Collaboration System. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1237–1246, New York, NY, USA, 2009. ACM.

[33] J. Yang and D. Wigdor. Panelrama: Enabling Easy Specification of Cross-device Web Applications. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 2783–2792, New York, NY, USA, 2014. ACM.