SAGEBoard: a Whiteboard for Large Multitouch Displays

BY

FILIPPO PELLOLIO
B.S, Politecnico di Milano, Milan, Italy, 2014

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2016

Chicago, Illinois

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

IWB	Interactive WhiteBoard
SAGE2	Scalable Adaptive Graphics Environment 2
UIC	University of Illinois at Chicago
WIMP	Windows, Icons, Menus, and Pointing

SUMMARY

Thanks to technological advances, low-priced tiled touch displays have the potential to radically change the way we work together in collaborative environments, such as a classroom or a meeting. Whiteboards are an example device that could be replaced by tiled touch displays. Whiteboards are commonly used in classroom environments to communicate content, improve the teaching experience, and sketch diagrams or illustrations to better explain complex concepts. They are also frequently used in meetings or brainstorming sessions, in order to share ideas among a group of people. In recent years, whiteboards have been increasingly replaced by projectors or very large non-interactive displays. These new technologies bear specific advantages, such as better readability through the use of prepared slides, and improved online access to teaching materials. However, they also introduce a certain rigidity to the teaching process: when a student poses an unexpected question, a quick answer or diagram cannot be promptly sketched.

This thesis proposes an electronic whiteboard for tiled touch displays. An electronic whiteboard can combine the advantages of an electronic device with the fast paced interaction provided by a regular whiteboard, enhancing both solutions with new functionalities. In this application, users can perform classic whiteboard actions such as drawing or writing free-handed on the screen using their finger or a stylus. They can use brushes to paint, and they can erase content. However, beyond the capabilities of traditional whiteboards, users can also relocate content on the screen, insert figures and drawings, and save or load previous sessions. Moreover, content can be shared among all users, and several people can interact simultaneously.

Creating an electronic whiteboard application poses a number of problems that will be analyzed and solved in the following sections, contributing to the literature in the human-computer interaction, computer graphics and software engineering fields. The main graphics and software problems analyzed are: the networked architecture of the application, which needs to run on multiple displays; the integration of the whiteboard application with other display-enabled applications; and the management of multiple simultaneous inputs. We also deal with the problem of rendering the same graphic elements on differently sized displays, and keep the solution lightweight enough to be sent over a network without creating a major latency problem. The interaction problems are the interface design and the analysis of how people behave when collaborating on a shared touch enabled display. We further explore remotely controlled collaboration; the whole application can be shared among both co-located and remote screens. In this context, we evaluate remote interaction using mobile touch displays, and we conduct a small scale user study to analyze the advantages of multi-touch displays over conventional keyboard and mouse configurations.

CHAPTER 1

INTRODUCTION

Nobody can deny the power and ease of use of a whiteboard for sharing ideas.

On the other hand, whiteboards also have shortcomings: handwritings can be difficult to read, especially from a certain distance; long explanations cannot fit the size of the whiteboard, so previous parts must be erased in order to continue; writing long chunks of text takes time, and transcribing them is painfully boring for the audience.

Many of these flaws are not present in electronic devices: text and images can be scaled, so that they can be read from far away. Windows can be hidden and shown again multiple times, without having to recreate anything from scratch. Files can be saved and reopened whenever they are needed, and online material can be incorporated, which is becoming more and more important nowadays.

However, these advantages come at a price: an electronic device such as a computer and projector combination gives the presenter less freedom. Unexpected questions or ideas cannot be easily answered or explained if no material was previously prepared: writing with a keyboard in a text editor is more cumbersome than a marker on a whiteboard. Drawing sketches on a computer is a ridiculously hard task in the absence of dedicated and potentially expensive devices, such as graphic tablets.

The application described in this thesis tries to combine the best features of whiteboard and electronic solutions through the use of wall-sized multi-touch displays.

The application does not aim to replicate the feeling of writing with a marker. Instead, we propose a set of software innovations and enhancements to address the challenges associated with electronic whiteboards while taking advantage of the technological opportunities offered by wall-sized displays.

CHAPTER 2

STATE OF THE ART

2.1 Large Displays

Recent research in large displays has shown that large displays can be helpful in many tasks, providing improved user experience and more accurate results.

Tan et al.[1] [2] notice an improvement in the understanding and navigation of 3D environments when the user has access to a very large display. The improvement is attributed to the wider field of view provided by the larger screen. This wider field of view helps the user gain improved spatial awareness, but also creates "optical flow cues". When navigating in the 3D environment using the large displays, the broader field of view lets the users see the environment flow at the sides of the screen while they move around, activating the sensitive motion sensors in the brain that allow people to perceive not only the structure of the environment but also their movement within it.

Andrews et al.[3] show how large displays can play a fundamental role in sense-making, as they provide more space that can be used as a source of information. Humans use space to encode relationship in everyday life: post-it notes sticked to the side of the monitor, soap placed next to the sink, and keys left on a counter are all situations where we use space to represent a relationship, or to remind us something. Common monitor screen real estate is however scarce, and therefore a valuable resource, so it cannot be used to encode relationships. Large displays, however, give space the meaning that it has in real life, so something can be placed in a corner if the user thinks it might be needed some time later, related applications can be placed side by side, and windows can be placed on top of another to encode relative importance.

Baudish et al.[4] demonstrate through several user studies that large displays can help the users to solve complicated tasks that require high cognitive load in a more efficient way. Thanks to their size, large displays can inherently show focus and context together in the same view, while regularly sized screens need to rely on software based solutions that fail to deliver the same level of naturalness. Being able to see details and an overview of a problem at the same time makes the users faster and more accurate at solving that problem.

Czerwinski et al.[5] talk about the benefits that large displays bring, but also about specific new problems that arise when dealing with large displays. The usual interaction style used in all common desktop applications, called Windows, Icons, Menus, and Pointing (WIMP), is not effective with large displays and creates several issues:

- Losing Track of the Cursor. When the screen size grows, users try to compensate for the growth by increasing the mouse acceleration, but this solution makes it harder to keep track of the cursor position.
- Distal access to information. As the screen size increases, reaching icons or windows that are placed far from the user's position becomes difficult.
- Window management problems. When a notification or a window needs to be created on a large display, its placement becomes a problem: elements should not appear at unexpected positions.

- Task management problems. Greater screen size creates problems under the hood too, with more windows open at the same time, the computational load on a machine increases.
- Failure to leverage the periphery. Large displays create a peripheral zone at their sides, that could be used for notifications or other features supporting a regular user activity. Periphery is wasted if used as a regular portion of the screen because, being too distant from the user, it would never be used.

Moreland [6] also talks about the problem of the WIMP paradigm being wrongly used in large displays and warns developers to "Let the application drive the display, not the other way around". He notices that too often desktop applications are simply ported to a large display, and that does not work.

This research suggests that we should think of large display applications in a completely different way from desktop ones, the same way we do for smartphone or tablet applications.

Ni et al.[7] provide an extensive overview of research in the large displays field, showing that research for large displays has been very active in the past decade, leading to great technological advancements. The report starts by listing the possible hardware configurations of large displays:

- CAVE and Derivatives. CAVE Systems are used mostly in virtual reality and they try to create an immersive environment around the user. They are typically formed by three or more projection-screens arranged to form a cube. A new solution for CAVE systems is to use tiled screens arranged in a circular manner, as in Febretti et al. [8]. This solution avoids the problems associated with projection systems, such as mis-calibration and low brightness.

- Multi-Monitor Desktop. Also called MultiMon, this is a configuration that is becoming more popular in both home and offices, and it consists of connecting multiple monitors to a single computer. However, the fact that the monitors are connected to a single desktop machine limits the number of monitors and consequently the overall size of the composite screen.
- Projector Arrays. Arrays of projectors are the most flexible configuration in terms of geometry control, but they are limited in terms of size and image brightness. One of their biggest advantages is that they create a single huge screen without borders, with seamless connection between two tiles. However, they are expensive, and they deteriorate after a while. In time, differences in colors between the projectors arise, and the seamless screen impression disappears.
- Tiled LCD Panels. This configuration consists of a variable number of LCD displays arranged in a two dimensional array. The screens can be positioned vertically as a wall display, flat like a table, or even in a curved configuration. They are cheaper than projector arrays, and they provide better colors. Their greatest drawback is that the screens have bezels separating them. This is bad for displaying text; however, it is less noticeable when the screens are showing images.

Ni et al.[7] describe the most important applications for large screen displays:

- Command and Control. Control centers, where a great amount of data needs to be shown at the same time with a high level of detail, commonly use large high-resolution displays.
- Vehicle Design. Immersive environments such as CAVEs have been widely used by automotive design industry to display and interact with vehicle models at 1:1 scale.
- Geospatial Imagery and Video. Large high-resolution displays are the perfect media to show accurate geospatial images and large film-quality video applications. The ability to show realistic terrain representation, zoom across scales while still providing a good overview, and the creation of fly-through animations make large screens unbeatable when it comes to geospatial imagery.
- Scientific Visualization. Large high-resolution displays are perfect for scientific visualization because they can both show data at a real life scale and show a great amount of data simultaneously thanks to their increased number of pixels.
- Collaboration. This is the field most pertinent to the application described in this thesis. Since the basic concept of collaboration is having a shared field where ideas can be presented or exchanged, large displays enhance the collaboration by providing a bigger shared field.
- Education and Training. Large high-resolution displays are commonly used for education and training in many fields, such as astronomy, bioinformatics, medical imaging, urban planning, and geographic information. The instructor and the student share the large display in order to present educational material . Thanks to its size the screen can fit multiple pages of material.

- Public Information Displays. Large high-resolution displays are becoming so cheap that they are starting to replace billboards, providing more dynamic images with even better quality.

Finally, Ni et al.[7] propose a list of major challenges that are still unresolved and should be addressed in future research.We here describe the challenges that are related to the application discussed in this thesis:

- Truly seamless tiled displays. LCD tiled displays have the already mentioned bezel problem, while arrays of projectors have problems with variations of colors and luminosity. We need to find a way to build tiled displays in a way that makes them really look as a single giant display. We do not address this challenge.
- Easily reconfigurable large high-resolution displays. Currently used display configurations are very difficult to change, they require hours of realignment. An easier way of arranging and rearranging reconfigurable configurations should be developed. We do not address this challenge.
- Design and evaluate large high-resolution display groupware. Large displays are appealing technologies when it comes to collaboration, but only few applications have been developed natively for them. Most of the applications used for collaborative interaction used on large displays are simply an enlarged version of their desktop counterpart. For this reason they suffer from the scalability issues discussed before. Users perceive large high-resolution displays in radically different ways due to their form factors, and this affects the interaction

behaviors. We further note that large displays create a semi-public environment, which may affect the privacy of the users and their interactions.

- Perceptually valid ways of presenting information on the large displays. When dealing with large displays perception paradigms are different from what we are used to in regular desktop configurations. New features gain significance, such as the field of view, the brightness and the resolution. Since these features heavily affect the user's experience, the developers should take particular care of how they use such features. In this work, we take into account these constraints.
- Integrating large high-resolution displays into a seamless computing environment. Portable computing devices such as tablets or smartphones are now part of everyday life, everybody has one. Large display applications should find a way to leverage this abundance of personalized devices to create a more immersive environment around the user. A monolithic application that cannot connect to a network is undesirable. In this work, we present a tablet extension of the application.

2.2 Whiteboards in Large Multi-touch Displays

The previous section was dedicated to a broad overview of applications and techniques used in large displays. This section looks in greater detail at large multi-touch displays, focusing on whiteboard style applications.

Thanks to their resolution and raw screen space large vertical displays can be used by multiple users at once, offering different collaborative opportunities. Jakobsen and Hornbk [9], [10] show that people who collaborate using a vertical display with multi-touch can obtain improved results in terms of the distribution of display area used, the willingness to share parts of the screen, and the amount of movement performed by users. Different conclusions were obtained by studies directed at tabletop or horizontal large displays[11] [12]. Even though both of these latter categories are large displays, their positioning and orientation in space influences the device interaction.

The Xerox Liveboard [13] is one of the first whiteboard applications to be developed for taking notes during meetings, it is based on a back-projected display and people can draw on it using a special pen. The application, anyway, has some problems with the image quality and the pen accuracy. Pen accuracy is, in fact, a current problem in whiteboard applications for large vertical displays.

Tivoli [14] is another whiteboard application developed for the Liveboard that provides a more accurate user interface, with more functionalities, and a basic multi-user interaction using multiple pens. The number of user is still limited by the number of pens, restricting the freedom of the application. Moreover, the application cannot be shared between multiple sites, so it can only be used for meetings where everybody is in the same room.

The usage of the special pen to draw in the case of the Xerox Liveboard [13] was forced by the technologies available at the time, but it is the start of a discussion about the advantages and disadvantages of using a specialized tool to interact with the display. The major problems of relegating the user interactions to dedicated tools are:

- Number of users. The specialized tools needed to interact with the display are usually a scarce resource; there might be just a couple of controllers. This limits the number of users that can use the application at the same time, the collaboration.
- Ease of Use. Even the simple task of taking the pen to write on the screen discourages the user in using the whiteboard. Constricting the user to use a specialized tool will affect the naturalness of the application.

However, using dedicated tools has advantages too:

- User Recognition. The special object can provide a form of identification to the application, so that the experience can be personalized. A simple example of this behavior is having pens that represent different colors.
- Precision. If the application knows the size and the shape of the object used for the interaction, it can react more precisely to each user input, making assumptions that could not be done without this additional information. For instance the inclination of a touch event could be calculated if the pen tip size at different angles was known.

It is important to distinguish pen inputs from specialized pen inputs: on a display that recognizes every type of touch a pen could, and probably will, still be used to enhance precision.

Research in multi-touch large displays has been performed also in the educational field . Agostini and Di Biase [15] show that collaboration in classrooms using large multi-touch displays bring good results, engaging the students in the learning process. They conducted a user study inside an elementary school, using an Interactive WhiteBoard (IWB). Their work shows how digital native kids are attracted to this kind of technology and that they would like to use it in all their classes. The collaboration environment created by the IWB can shift the usual teacher-centric paradigm to a different approach, where the kids can feel more involved and learn more.

Ashdown and Robinson [16] show that wall displays with writing capabilities can enhance speaker presentations, especially when interaction with the audience is needed. They use a projector based large display, as described in Section 2.1, while the multi-touch input is simulated by infrared pointers that can be pointed at the screen from anywhere inside the room. The use of pointers to interact with the wall provides an interesting approach that increases the distance at which the screen can be interacted with, but it also significantly decreases freedom, since pointers are a finite resource and they are mandatory to interact with the screen. Another problem with this approach comes up when the screen is in a crowded room: pointers need an unblocked view from the user to the screen, otherwise their raycast would be blocked. The study further shows that the quality of the answer to an audience question can be improved by the presence of a large multi-touch display, and that the question itself can be better explained using the interactive wall by the person who posed it.

Work on whiteboards on large touch displays has also been done in the medical field. France et al. [17] study how electronic whiteboards can improve efficiency and communication in the Emergency Department of an hospital. The study conducted in the field points out that the large touch display helps the Emergency Department doctors to understand the current situation and consequently to come up with a better workload distribution.

Most large multi-touch displays and whiteboard applications do not handle well concurrent interaction. In fact, the majority of the touch detection systems employed by large displays are infrared overlays that provide a context which comprises an identifier, an event type and the coordinates of the touch event. As long as applications for large display remain simple, this information suffices for interaction. However, there are situations in which a user executes an action through a sequence of two touch-events. In a single-user setting, each user has their own separate context, and no problems occur. However, in a shared context, the touch of a different user could unwittingly or erroneously complete the first user's action. As this simple scenario suggests, multi-user and shared environments pose significant challenges to large multi-touch display applications.

2.3 Collaborative Tools

With the "cloud" becoming more and more important in the past years, many tools for online collaboration made their way into the life of every internet user. Some examples are Google Docs, CoderPad, or the Adobe Creative Cloud.

This section compares several of these tools.

2.3.1 Text Based Collaboration

Google Docs: Google Docs [18] is the collaborative tool developed by Google that allows multiple remote users to work together on a single text file. All the users will see the changes made to the shared document in real time. Every change is saved using a versioning system that will allow the user to revert some changes at every point during the creative process.

Microsoft Office: All the Microsoft Office[19] suite applications have a built-in sharing mechanism. However, the changes made by other people cannot be seen in real time: the document needs to be synchronized manually by the user, who is then in charge of resolving conflicts and accepting revisions.

2.3.2 Sketch Based Collaboration

CoSketch: CoSketch offers the user a white canvas where he can draw with his mouse, along with basic drawing functions. Every sketch has a link and it can be shared with other users, so that they can see and modify the same sketch in real time.

Twiddla: Twiddla is an online shared whiteboard application that can be used both from a pc using the mouse or on a touch enabled device emulating the mouse (single touch only). It also offers the user the possibility of drawing over uploaded files, documents, and even web pages.

2.3.3 Whiteboard Applications

Tivoli Xerox Whiteboard [14]: The multiuser extension of the Xerox Whiteboard [13] allows the users to sketch on a blank canvas or annotate over documents, but only on one document at a time. The collaboration task is enabled only for in-presence collaboration. The whiteboard cannot be mirrored and interacted with remotely.

SMART kapp iQ: SMART kapp iQ[20] is an interactive whiteboard to be used in a classroom. It comes with a maximum size of 65" and it allows both in-place and remote collaboration using mobile devices. The main screen can be interacted using a pen, limiting the interaction with the screen to one person at a time. Documents can be opened on the whiteboard using their proprietary software.

2.3.4 Critique

Each of this tools has its strengths and weaknesses; SAGEBoard tries to combine all their best functionalities without incurring in their flaws. Table I shows how SAGEBoard is the only tool that, thanks to large multi-touch displays, offers all the collaboration paradigms (In-place, Real Time, and Remote). Moreover, SAGEBoard offers both a Web Browser interface and a dedicated mobile interface, in order to leverage different devices capabilities. The completely touch-based interactions offer more precision than mouse-based ones, without constraining the users to dedicated input devices.

2.3.5 Comparative Table

TABLE I: COMPARATIVE TABLE OF COLLABORATIVE TOOLS.

Solution	**Device**	**Input**	**Canvas**	**Collaboration**
Google Docs	Web Browser	Mouse, Keyboard	Shared document	Real-time, Only remote
Microsoft Office	Computer	Mouse, Keyboard	Shared document	Revisions, Only remote
CoSketch	Web Browser	Mouse	Blank	Real-time, Only Remote
Twiddla	Web Browser	Mouse	Blank, Shared Document	Real-time, Only Remote
Tivoli Liveboard	Dedicated Hardware	Dedicated Pen	Blank, Document	Real-time, Only In-Place
SMART iQ	Dedicated Hardware, Mobile	Dedicated Pen, Remote	Blank, Shared document	Real-time, Only Remote
SAGEBoard	Web Browser, Mobile	Multi-touch, Remote	Blank, Shared document	Real-time, Remote, In-Place

CHAPTER 3

BACKGROUND: SAGE2

In order to solve specific problems such as the inter-application communication, the management of different sized displays, and the networking architecture behind all the client-server requests, our whiteboard application is built inside the second version of the Scalable Adaptive Graphics Environment (SAGE2) [21].

SAGE2 is an open-source middleware that manages different configurations of displays and creates an environment that combines them into one single space that can be used by the user as one single screen. SAGE2 offers a multi-user environment, both collocated and remote, where users can share documents, images, or even a capture of their screen.

SAGE2 can be accessed through a simple web browser just by navigating to the server page, both as an input client or a display client.

SAGE2 provides the opportunity of building custom applications that can be instantiated and interacted with in the environment, providing space for the development of custom visualizations or the enhancement of the system with new functionalities such as maps, timers or clocks.

SAGE2 has been successfully used in class environments, lectures, and meetings. Its ease of use makes it accessible even to first time users. The most common scenario in which SAGE2 is used involves a single big display shared between the users in a room, who can connect to the server, control and interact with the applications currently displayed, open new applications, or share personal files from their devices to the server. However, SAGE2 is not limited to in-place collaboration: users can connect from remote locations, both controlling the environment or mirroring the display. In this chapter we discuss the SAGE2 features in more detail.

3.1 Architecture

SAGE2 is a server-client application, where multiple clients connect to a single server. The clients can be input clients, controlling what is being displayed on the screen, or display clients, that just replicate the shared screen or part of it.

Large displays can run both as a single giant display or as a number of tiled displays that present just a part of the canvas. This is particularly important, not only because it enables the use of tiled displays running on different machines, which lowers the price of the infrastructure, but, also in the whiteboard case, because users can replicate the shared screen on remote devices, enabling a different degree of collaboration.

SAGE2 is responsible for synchronizing all the clients in order to replicate the feeling of having a single shared canvas.

Input clients can move or launch applications, share files, and in general control the whole environment. SAGE2 receives the inputs from the input clients and sends the changes made to all the connected display clients, creating a seamless collaboration experience.

A special kind of display client can also be very useful and will be extensively used in this thesis: the overview display client. SAGE2 is developed to work on different resolutions. More often than not, when working with large displays, it is useful to have an overview of the whole environment on a regular screen, since not every connected client might be a large display. In the overview client the whole SAGE2 canvas is scaled to fit the user browser window, but the system is still completely functional and reactive even in this view.

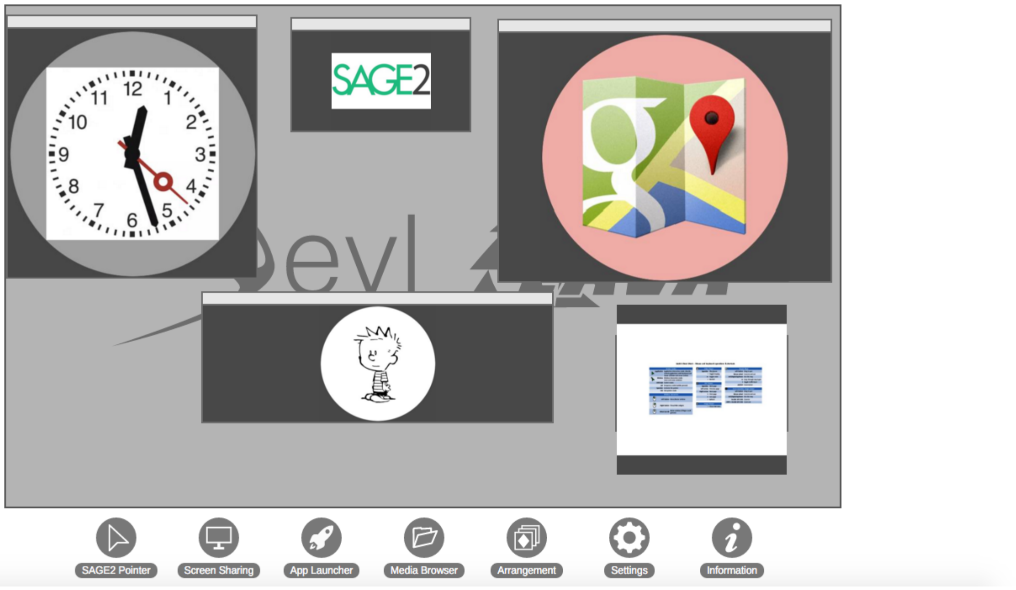


Figure 1: A SAGE2 Input Client.

3.2 Applications

Users can create applications for SAGE2 with minimal effort: the applications are written in javascript and they are represented by a ¡div¿ element inside the shared canvas. The user is free to manipulate the application ¡div¿ using javascript, and she can analyze server inputs such as clicks and keyboard events using the SAGE2 APIs.

Inputs cannot be managed in the default javascript mode: listening for mouse or keyboard events triggered on the window object. This is because no events are being performed on the display clients: they originate from remote input clients. The SAGE2 server is in charge of analyzing the remote input and communicating it to the proper application on all the display clients.

Users can upload their applications also to already running servers, by uploading them as a regular file from the input clients.

The SAGE2 server comes with many applications already installed, in order to perform basic tasks. We list here the applications relevant to this thesis:

- PDF Viewer: This application is important to the purpose of this thesis. It is a plain application that displays PDF documents uploaded by the user. This application is one of the few that cannot be launched from the application list, it is opened by the system when the users uploads a PDF document.
- Image Viewer: Another default application opened by the system when the user uploads an image, it displays the image and allows the user to scale it any way he wants.
- Google Maps: An application that allows the user to navigate Google Maps on a SAGE2 server, allowing collaboration and sharing of geospatial informations.

In Figure 2 we can see several applications in action: a cronograph in the top-left corner of the view, an image uploaded from the user displayed with Image Viewer in the top-center, the Google Maps application in the top-right, a user custom application that displays the Calvin and Hobbes strip of the day in the bottom center, and finally a PDF document uploaded by the user displayed thanks to the PDF Viewer in the bottom-right corner.

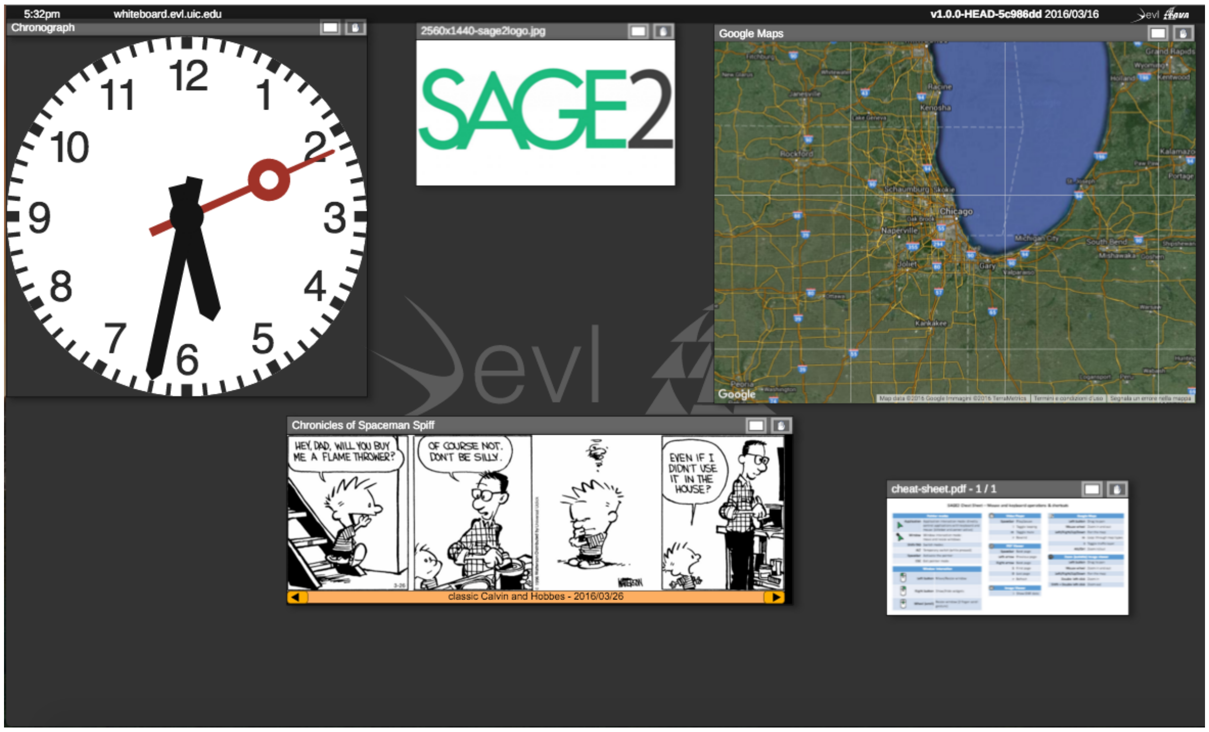

5:32pm
whiteboard.evl.uic.edu
Chronograph
2560x1440-sage2logo.jpg
SAGE2
Google Maps
v1.0.0-HEAD-5c986dd 2016/03/16
Chronicles of Spaceman Spiff
HEY, DAD, WILL YOU BUY ME A FLAME THROWER?
OF COURSE NOT. DON'T BE SILLY.
EVEN IF I DIDN'T USE IT IN THE HOUSE?
classic Calvin and Hobbes - 2016/03/26
cheat-sheet.pdf - 1 / 1

Figure 2: The SAGE2 Overview Client associated with Figure 1.

3.3 Touch in SAGE2

Touch interactions is supported in SAGE2, but it is a rarely used experimental feature. The touch interactions are not really managed for what they are, instead they are used to simulate pointer events, as commonly seen in online applications that can be used by touch based devices such as tablets.

The advantages of this approach are its ease of implementation and its abstraction of the input device. However, considering the touch event as a more abstract type of input, we lose many possible information about it that could be useful. An example of this loss of information can be found in the size of the touch event: pointer interactions do not have a size, so the abstracted touch event loses this potentially useful feature. Another problem is given by the completely different nature of the inputs. Consider the common right button click on a mouse: what does it correspond to in touch interactions? A simple answer could be a two finger touch, but now two touch events are mapped to only one input. This solution creates a plethora of new problems: touch events cannot be mapped directly to mouse events, they need to be checked every time for simultaneous events in their surroundings, and there is not a one to one relationship within them.

The abstraction approach offers anyway too many advantages in a very input dependent system such as SAGE2, so that is the road the SAGE2 team decided to take. The solution developed, in order to spare the server from dealing with all the problems mentioned earlier, was to defer the touch interaction analysis to the sender. The client sending the touch events is in charge of understanding what the touch event means, and to communicate it to the server using a dedicated protocol. This solution asks the clients for a lot of work, and this is what makes it still an experimental feature.

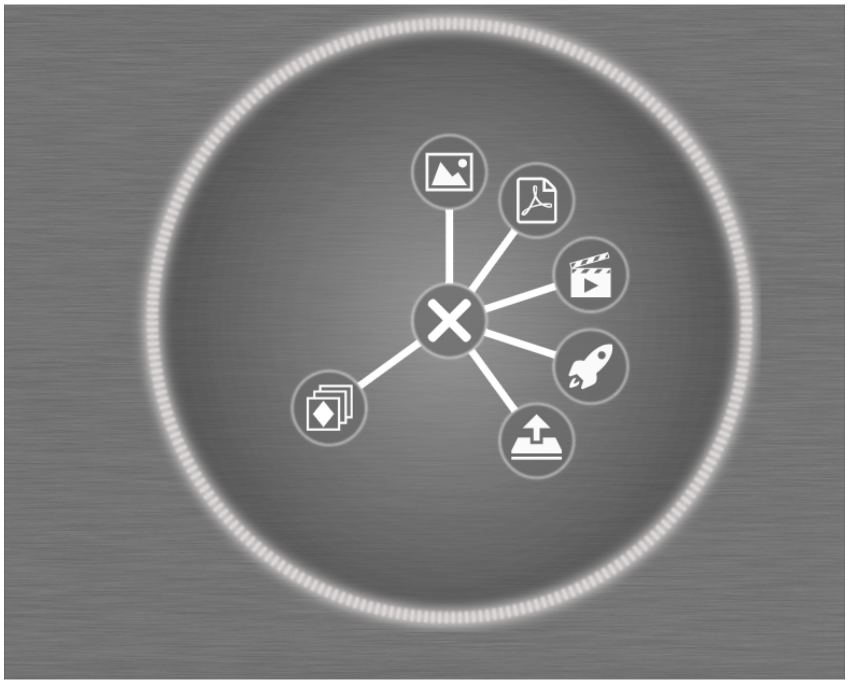


Figure 3: SAGE2 Radial Menu.

Common scenarios of touch interaction in SAGE2 are the single touch interaction, where the touch and drag on the display is interpreted as a click and drag with a mouse pointer, and the three-touch gesture, used to emulate a right click on a mouse. Figure 3 shows what happens when a user performs either a three finger touch or a right mouse click on a free portion of the canvas: the radial menu opens. Through the radial menu the user can open applications or documents, giving a touch display client all the functionalities of an input client, in this way the user does not need to step away from the touch screen to perform any operation.

In conclusion, touch interactions in SAGE2 are possible and perfectly functional, but they need to be preprocessed by the client before being sent to the server, and they are still processed as pointer events by SAGE2. That is why we had to completely rethink touch interactions in our application starting from the raw input. Drawing would have been impossible using the emulated mouse interaction.

CHAPTER 4

METHODS

SAGEBoard was developed over a period of 4 months through a close collaboration with several wall-display and networking experts. The development followed a Human-Centered-Design paradigm, while paying particular attention to user tasks and nonfunctional requirements. We implemented this paradigm through an iterative process where we met with experts and regular users to confirm requirements, explore prototypes, refine the design, test the software, and verify that requirements were being satisfied.

4.1 Requirements Analysis

Requirement gathering started with a semi-structured interview of the experts, followed by several observation sessions of instructors and students interacting with both wall displays and with regular whiteboards. The requirements we gathered include the following functional and nonfunctional elements:

- provide both a canvas and a control panel (palette or marker and eraser station)
- large scale and multi-user
- high-fidelity and responsive when drawing and writing
- touch-enabled, with fluid use of touch gestures (e.g., some users erase with the back of their hands)
- be able to annotate over other applications (e.g., pdf docs)

- undo, save, reload and clear-all capabilities
- resize and relocate content
- always float in front of other applications

4.2 Design

We started by creating a series of low-fidelity and high-fidelity prototypes, including storyboards, to illustrate potential visual layouts and interaction flows. Our first prototypes pictured SAGEBoard as a common SAGE2 application, it consisted in an opaque canvas that the user could scale and move among all the screen, with the actions palette attached to one of its borders like it is done in most drawing softwares. The application would have handled touch as every other application in SAGE2: as a pointer input simulated by the server through the process described in Section 3.3. We already described why and how this process brings to a loss of precision, but we thought we could give it a try.

So we developed our first high-fidelity prototype: a functional drawing application for SAGE2. We quickly realized that this solution was not the best one, for a lot of reasons:

- The opaque canvas occluded the other applications, preventing the user from annotating over documents or applications.
- The canvas was being occluded as well by other applications, while we wanted the drawings to always stay in front of everything.
- The reduced canvas size meant a reduced multi-user interaction, limiting the space for collaboration.

- The canvas, if increased in size in order to collaborate, wasted a lot of precious space, that could have been used by other applications.
- The simulated touch system was completely unreliable, lines were staggered and the overall drawing experience was awful.

After these findings we got back to prototyping and considering all these elements we decided that SAGEBoard should have been embedded inside the SAGE2 server, it should have taken the touch data directly from the source, and that the canvas should have been transparent and as big as the whole screen. We describe in the following sections the resulting architecture of our second high-fidelity prototype .

4.2.1 Application Architecture

To build an application integrated with the sharing middleware, we need to take into account the following SAGE2 components:

- The SAGE2 Server, which represents the SAGE core
- The Touch overlay: the system dedicated to touch input detection.
- The Display: the client window of SAGE.

Our application consists of two components built on top of SAGE2:

- a Node-drawing module: the software representing the SAGEBoard application on the server.
- a Palette module: the client application used for the application interface;

These modules are described in detail further below in Section 4.2.3.

We briefly describe here the two main workflows characteristic to the application: the Drawing interaction and the Palette interaction (Figure 4). For simplification purposes, the touch input flow is shown as going directly from the touch overlay to the Node-drawing module, even though (as described below) the touch input is processed by a secondary machine. In both cases the interaction starts with a user touch that is captured by the touch overlay, processed by the underlying components and then sent to the Node-drawing module.

4.2.2 Drawing interaction

The Drawing Interaction workflow (Figure 4.a) starts with a user touching directly the display (A) with the intention to draw or edit a drawing, without accessing the palette. When the touch occurs, it is sent to the Node-drawing module (B). Since the module is aware of the palette position, it recognizes that the touch occurred without accessing the palette. The touch is thus allowed to update the drawing state. The Node-drawing module then identifies the connected clients that are observing the wall-display section affected by the touch, and sends a message to the server (C) summarizing the differences between the previous drawing state and each affected client. Once it receives the message, the server notifies all the involved clients of the new drawing state (D).

4.2.3 Palette interaction

The Palette Interaction workflow (Figure 4.b) starts with a user touching the display (A) with the intent to invoke a Palette function (and therefore pressing a palette button). The touch is sent to the Node-drawing module (B), which is aware that the touch has occurred

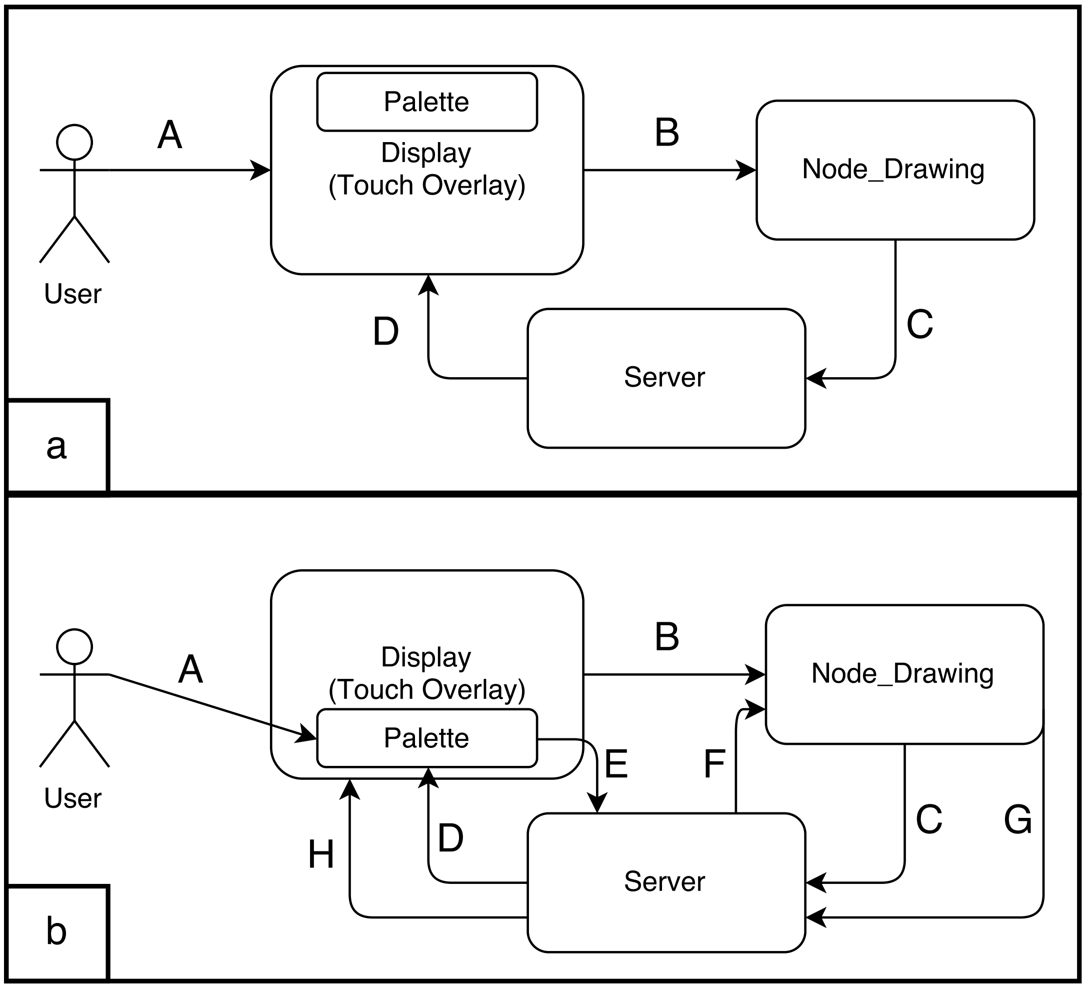


Figure 4: Two workflows in SAGEBoard.

within the palette area. Consequently it sends to the server (C) a message containing the touch coordinates and the involved client. The server communicates to the client that a palette touch occurred (D). The client interprets the touch event and figures out which palette button is pressed, then generates a visual feedback for the user. Each palette button is connected to a specific server call (E). Once the server receives the message, it sends a specific message back to the Node-drawing module (F). The module then changes the drawing state and communicates the corresponding action to the clients.

4.2.4 Node-drawing Module

From a hardware point of view, the touch overlay is mounted around the wall display. When the touch overlay mode is activated, the SAGE2 server connects to both the large vertical display and, through a secondary machine, to the touch overlay. This secondary machine has the role to fetch and prepare the raw touch data for the server. This means that even though the touch overlay is mounted on the display, these two components are connected to two different machines and must be handled as separate entities.

From a software point of view, the SAGEBoard application runs on the server through its Node-drawing module. Multiple clients aside from the Whiteboard application can connect to the SAGE2 server and, through it, to specific displays in the tiled wall. We handle in the Node-drawing module the connect/disconnect requests from clients. We further record which client is connected to what display, in order to send client updates only if its display part is involved. The Node-drawing module also stores the current drawing state. We implement this configuration using an observer pattern: the subject being observed is represented by the

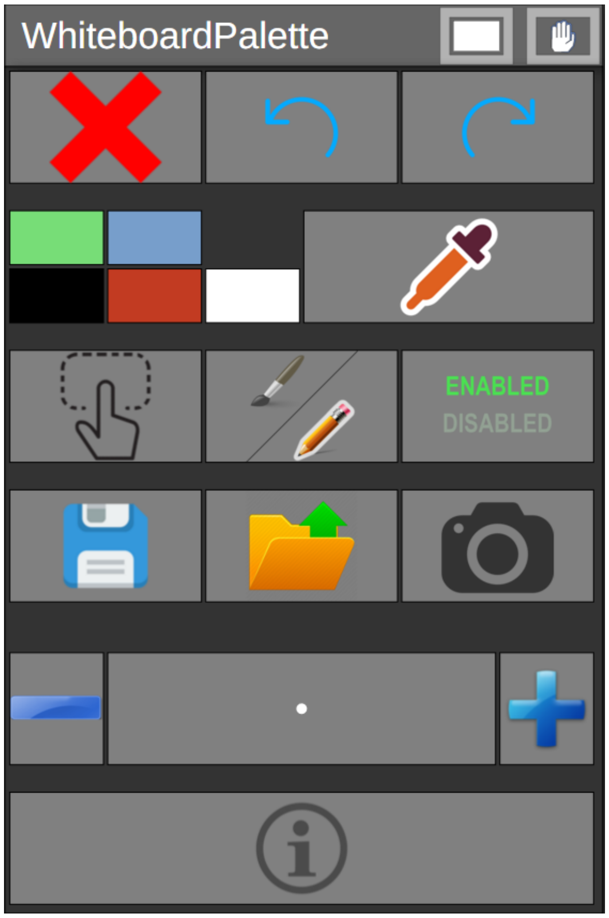


Figure 5: The Palette Interface.

drawing state contained into Node-drawing and the observers are represented by the display clients.

4.2.5 Palette Module

Users can interact with the application using the palette interface shown in Figure 5. The palette is an application built for the SAGE2 server. The palette is designed with scalability and ease of use in mind: thanks to its SVG-based interface it can be scaled without any loss of precision, allowing it to work both on very large displays or on smaller ones, such as tablet displays. All the buttons in the interface are rectangles, in order to be very easy to press even on displays with not enough touch precision to deal with circular buttons.

Upon activating the palette, an SVG canvas appears on the screen along with a recall bar in the lower part of the screen. From that moment on the detected touches get redirected to the Node-drawing Module, and the users are allowed to draw on the screen. The drawing functionality can be deactivated either by closing the palette or by pressing the Enable/Disable button on it. The Enable/Disable switch allows the users to switch back and forth between the drawing mode and opening other applications or documents, which the users can then annotate. This functionality was not present in the first release of the application, but it was introduced later due to users' demand, proving once again how a design cycle involving frequent testing sessions helps developing complete applications. Once the user has opened the document she wants to annotate, she can revert to drawing by pressing the palette.

The palette can also be moved around on the screen by the users just by performing a dragging gesture starting from its title bar. This was at first the only way to move the palette, but after the first testing session we noticed that often the users ended up drawing very far away from the palette and they had to walk across all the screen in order to move it. While this problem would not arise with standard size displays, in very large displays it can not be overlooked, so we decided to introduce the recall bar. The recall bar is located in the lower section of the screen, and when a user touches it, the palette moves directly to the horizontal coordinate of the touch event, allowing faster interactions.

4.2.6 Annotations

SAGEBoard allows the users to annotate over documents, that can be shared by any of the user using an input client, or the touch interactions provided by SAGE2. Thanks to this functionality the application allows a new degree of collaboration, that is not possible using physical whiteboards.

Once a document is displayed on the screen using one of the functionalities explained in Section 3, the user is completely free to draw anything over it. This is possible because the SVG canvas containing all the drawings floats above all other applications, except for the SAGE2 pointer and the palette. Thanks to its transparent background the canvas does not hide the applications lying under it, allowing the user to see what they are drawing over.

After the user draws something on the screen, the drawing is analyzed searching for applications lying under it. If an application is found to be directly beneath a point contained in the drawing path, then that drawing gets associated to that application by the system. This behavior allows the annotations to follow the document they are drawn over when the document is moved around the screen.

4.2.7 Extended Capabilities: Beyond Drawing

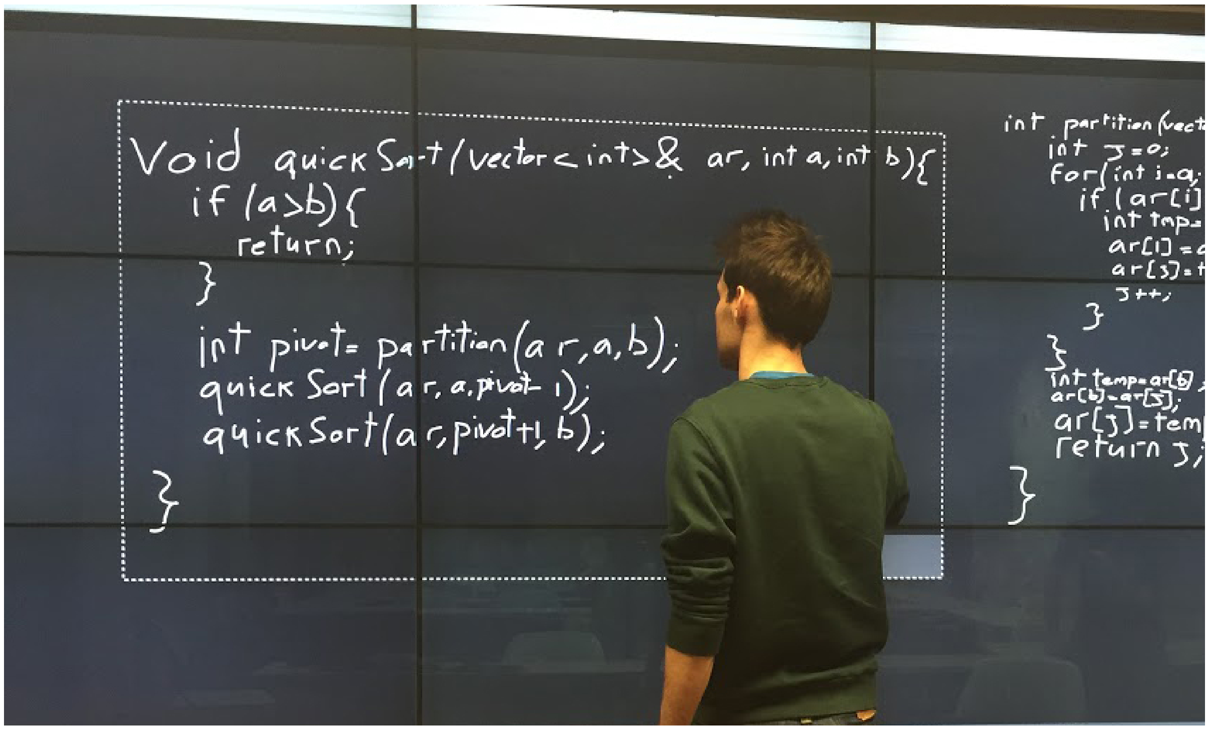
The application can be used in two different modes: the whiteboard mode and the painting mode. The main difference between these two modes is the result of large-size touch events. In the whiteboard mode a large-size touch is interpreted as an eraser, so that an imaginary user can easily erase a spelling error using her whole hand or sleeve. In painting mode, on the other hand, the stroke size of the line drawn on the screen is the same as the touch size, so a large-size touch is simply interpreted as a large stroke line.

The user can toggle between the two modes by pressing the switch mode button on the palette panel. The switch mode button is represented by an image of a brush, which represents the painting mode, and a pencil, which stands for the whiteboard mode. The icon relative to the currently active mode is highlighted by a white outline, in order to make the user always aware of the current application mode.

Beyond basic drawing, the functions available to the user are as follows:

- Clear Screen: Allows the user to erase all the drawings currently shown on the screen.
- Undo: Lets the user erase the last line drawn on the screen. The user is allowed to undo a virtually infinite number of drawings: the system keeps track of the order of all the drawings made.
- Redo: Allows the user to redraw items previously erased using the undo function.

- Change Stroke Color: The user can select a color from the list of colors shown on the palette or pick one using the color picker. The marks drawn on the screen from that moment on will be of the selected color.
- Selection: The first touch interaction after the user presses the selection button inside the palette is interpreted as a selection creation. The user can create a rectangular dashed box surrounding the drawings she wants to select. The selection box will persist on the screen until a new drawing is drawn on the screen; at that point the selection box will be deleted. An example of this function is shown in Figure 6.
- Relocating a Selection: When a touch event occurs inside a selection box, it is interpreted as a moving selection interaction. All the drawings inside the selection box will be shifted.
- Resizing a Selection: Inside the selection box, at the right bottom corner, we provide a smaller box, called the resizing box. If the user starts a touch movement inside the resizing box, the box will follow the movement, scaling the selection box and all its contents accordingly.
- Save Session: The current drawing state gets saved on the server, in a file labeled with the timestamp of the interaction.
- Load Session: The user is presented with the list of saved drawing sessions on the server. By selecting one of them the system will load that drawing state, and discard the current one.



```
Void quickSort(vector<int>& ar, int a, int b){
   if (a>b){
      return;
   }
   int pivot= partition(ar,a,b);
   quickSort(ar,a,pivot-1);
   quickSort(ar,pivot+1,b);
}
```

```
int partition(vect
   int j=0;
   for(int i=a;
      if (ar[i]
        int tmp=
        ar[i]=a
        ar[j]=t
        j++;
      }
   }
   int temp=ar[b]
   ar[b]=ar[j];
   ar[j]=tem
   return j;
}
```

Figure 6: Single user using SAGEBoard.

- Take Screenshot: When the user presses the screenshot button, a PNG file representing the current state of the screen is saved on the server.
- Change Stroke Size: This functionality is available only in whiteboard mode. Pressing on the plus and minus buttons next to the stroke indicator on the palette will change the stroke size.

Figure 6 shows a user writing and then performing a selection operation on SAGEBoard. As you can see the writing seamlessly spans multiple tiled displays.

4.2.8 Gestures

As discussed in Section 2.2, one of the greatest problems when dealing with multi-user and multi-touch whiteboards is the management of gestures composed by more than one touch. How can we understand if a touch is part of a user gesture or if it is simply part of another user drawing? The simple answer is that we cannot. The more complex one is that we could, using some sort of user recognition device, such as cameras tracking the users and their positions, or using dedicated devices as inputs that are associated with each other: a user could be forced to use only pens tagged with a color that the system knows are associated.

We already discussed the disadvantages of using dedicated devices as the only form of input, while the camera tracking system would increase the complexity of the system by far, without even helping with the most difficult situations: when the users are too close to each other to be recognized.

Another possible solution to the problem is a heuristic: assuming that two touch events are generated by the same user if they are very close. This approach automatically rules out gestures made with two hands, but that is not a great limitations: one hand still provides plenty of possible gestures.

In the first implementation of SAGEBoard we followed the heuristic road: before assigning the action to a touch event the system waited for a fixed amount of time, and, if another touch event occurred at a distance lower than an arbitrary threshold before the timeout, the two identifiers would have been associated in the system as part of the same gesture and their state machines would have transitioned to the gesture state. The system was theoretically sound and working, but the first testing session revealed an unexpected problem.

The problem was related to the touch overlay: since the screen we use is very big, the touch overly needs to be as big, and this unusual size makes it not very precise. The touch overlay detects touches way before the user actually touches the screen; this causes a lot of accidental touches, especially when the user is interacting with the screen with his hand: the fingers that are not used to touch the screen might be detected as well, since they are very close to it. If an accidental touch is detected as a drawing touch, it could be annoying, and it would affect the drawing quality, but it could still be acceptable. If an accidental touch is detected as part of a gesture, instead, it makes the user interaction impossible: a user trying to draw would continuously trigger the behavior associated to the gesture instead of drawing.

This problem, however, was only associated with a faulty hardware, so a possible solution could have been to only activate the gesture functionality on systems precise enough to handle it. However a second testing session on a less faulty hardware highlighted another problem in the system. When trying the application without the wrong gesture detection we noticed that the timeout needed to recognize the gesture, summed with the inevitable latency provided by the network, created an unacceptable delay between the actual touch interaction and the system feedback. We noticed that it is almost impossible to write properly even with a very little delay.

All these problems convinced us to drop the multi-touch gestures completely, since they were creating a lot of problems without providing a comparable amount of advantages. We kept anyway the gestures based on the size of the touch, such as the eraser touch, and we introduced gestures based on the location of the touch, such as the recall bar, in order to get back some of the advantages that multi-touch gestures gave to the system.

4.3 Implementation

4.3.1 Input Data

The user data input comes from an infrared touch overlay. This overlay is connected to a Windows machine that sends the data to the server running SAGE2, through the network. The data coming from the overlay is processed on the Windows machine by a C++ program, that converts it to the format used by the server.

The touch data is sent using UDP messages structured in a predefined protocol, that could also be used to simulate touch events coming from non-touch devices. This simple protocol definitions allows the application to be input independent: data coming from any kind of touch input could be converted to this simple structure, hiding the real complexity of the data from the server.

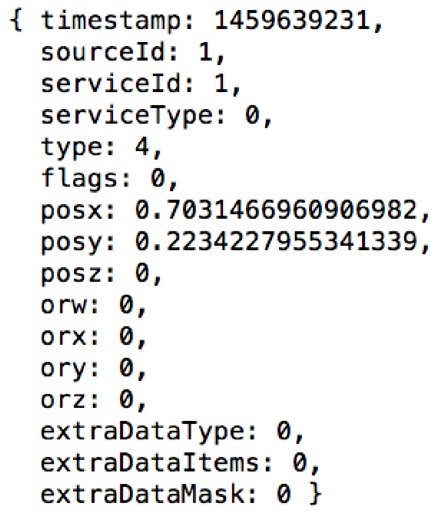
Thanks to the great speed of the UDP protocol the server gets very quickly the location of all the touches occurring on the screen. Each touch event is characterized by a source identifier, a location, and a type. The source identifier is common to all the touch events contained in a single touch interaction: if the user draws a line with her finger on the screen, all the touch events generated by this interaction will have the same source id. The location is composed by the x and y coordinates of the touch, normalized with respect to the touch input size. The normalization is needed because in many occasions the touch input device and the screen on which it is mounted on have different resolutions. The type of the touch event can assume three different values:

- Touch Down: It represents the first touch event of an interaction, and it is sent when the user first touches the screen. There is only one touch down event associated with each source identifier.
- Touch Move: It represents a movement of a given touch interaction, and it is sent when the user moves his finger on the screen. There can be many touch move events associated with each source identifier.
- Touch Down: It represents the last touch event of an interaction, and it is sent when the user detaches the finger from the screen. There is only one touch down event associated with each source identifier.

The structure of the UDP message can be observed in Figure 7. As you can see the position is normalized before being sent. The touch event type shown in this particular message (4) represents a Touch Move.

4.3.2 Drawing Data

The state of what is drawn on the screen is stored on the server in a compact format, designed to minimize the network load, and consequently latency. Because the input data could be streaming at high rates, rendering without processing would have disadvantages: the size of the drawing data stored on the server would be massive, the network would be really stressed, since all data saved on the server needs to be sent to all the clients, and lastly the



```
{ timestamp: 1459639231,
  sourceId: 1,
  serviceId: 1,
  serviceType: 0,
  type: 4,
  flags: 0,
  posx: 0.7031466960906982,
  posy: 0.2234227955341339,
  posz: 0,
  orw: 0,
  orx: 0,
  ory: 0,
  orz: 0,
  extraDataType: 0,
  extraDataItems: 0,
  extraDataMask: 0 }
```

Figure 7: The structure of one touch event sent via UDP.

resulting drawings would look staggered, because of the low precision passed by the touch overlay.

For these reasons, we sample the input data and keep only the points needed to render fairly smooth and faithful lines, the sampling interval has been determined empirically. The points are interpolated on the client side by a D3.js function that connects them to produce an SVG path. Each drawing is then associated with a drawing identifier which allows us to later modify or delete the drawing, and a style object, which defines how the drawing will be rendered on the client side.

Thanks to the determinism of the interpolation function the drawings are identically represented on each display, regardless of the browser used. Figure 8 shows how a line is really

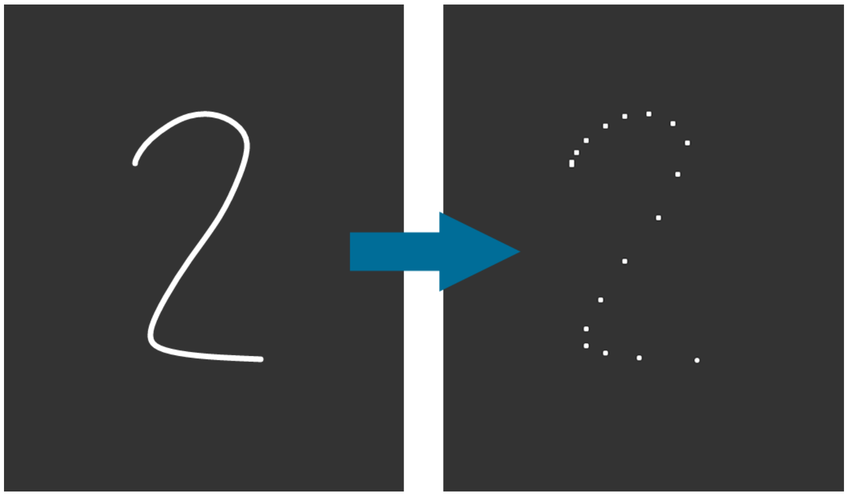


Figure 8: The way a line is encoded in the server.

encoded in the server: the left image shows the line as it is presented on the display clients, while the right image shows the points associated to that line saved on the server.

4.3.3 Multiple Touch Events

Our approach in handling multiple touch events consists in a state machine pattern: associated to every touch identifier there is a state, and the state determines what actions should be taken when a new event with that identifier is received. In this way we can manage multiple, coexisting touch events originating from different users.

The state machine is encoded in the system as a dictionary where the key is a touch identifier, while the value associated to it is a string representing the current state, or action. We also keep track of all the drawings that a touch identifier generates using a dictionary, so that undo actions can be easily performed.

However, a problem arises when identifiers are no more unique: some systems, including our touch overlay, recycle identifiers in a cyclic manner. This behavior could cause conflicts in our system, and touch events could be misinterpreted. Our application deals with this problem by clearing all the informations relative to a touch identifier when its touch release event is received, so, as long as two simultaneous touch events always have different identifiers, no conflict can arise.

Thanks to this approach, at any time SAGEBoard deals with an event, it is fully aware of its state and simultaneous touch events cannot interfere with each other. Moreover, thanks to the state machine pattern, it is very easy to take appropriate actions when a touch is received.

But multi-user interactions produce another problem: conflicting interactions. An example of this problem is when two users are simultaneously trying to move the palette in different directions, what should happen in that case? Our solution to this problem is to treat elements where possible conflicts could arise as scarce resources, and serve them on a first come first served basis. Whenever a user wants to access one of these resources there are two possible scenarios: either the resource is free, meaning that nobody else is using it, or it is already taken. If the resource is free, then we can simply let the touch event use it, and mark it as busy until the owner event is released. If the resource is already taken the touch event transitions to a sink state: the Ignore state. If the state machine associated with a touch identifier is in the ignore state, all the events bearing that identifier are completely ignored, apart from the release event, that erases the identifier from the state machines dictionary.

4.3.4 Processing the input data

When the server receives input data, it sends them to the Node-drawing Module, which interprets the touch input, determines what action the user wants to perform, and changes the state of the server to correctly respond to the user interaction. An action is associated with each source identifier using a dictionary, as explained in Section 4.3.3. The action is determined when the touch-down event of that particular identifier is received. The decision is based on a multi-layer, four-level priority system, from highest to lowest:

- First level: Interaction with the palette. It happens when the touch is located inside the palette title bar, the palette window, or the recalling bar on the lower part of the screen
- Second level: Selections. It happens when the user first touches the screen after pressing the selection button on the palette, or when the touch is detected inside of a selection window already present on the screen.
- Third level: Eraser. It is detected when the user is in whiteboard mode and the size of the touch is bigger than a predefined size.
- Fourth level: Drawing. It is the default action assigned to each touch that does not fall in one of the preceding categories.

Another possible action assigned to a touch identifier is the Ignore action. This means that the system will ignore all the following touch events related to that particular identifier.

In general, touch events cannot change their effect during their lifespan, with the exception of the Eraser touch. A touch that started as a drawing can become an eraser if its size grows

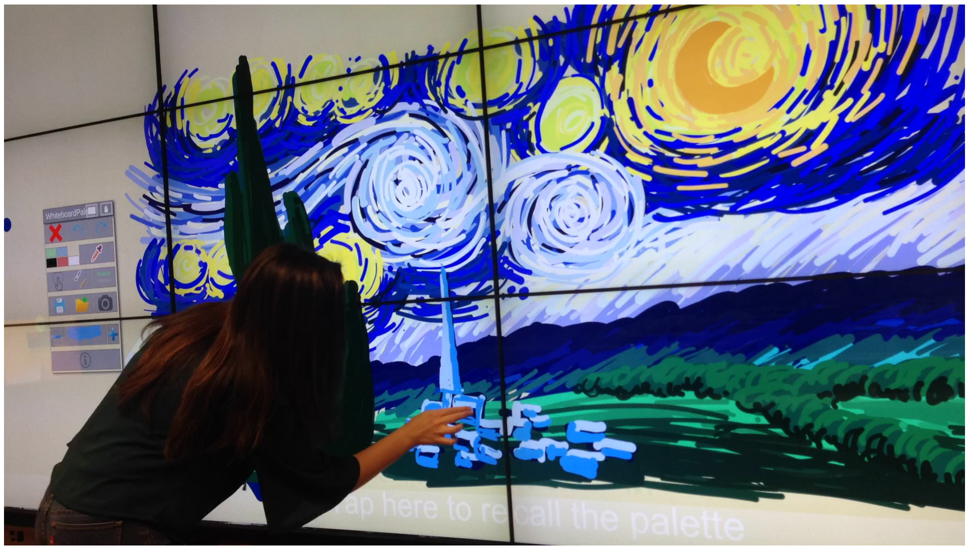


Figure 9: An experienced user drawing using SAGEBoard.

bigger than the predefined eraser threshold. This decision is driven by the fact that when a big object touches the screen, the first touch is rarely due to the whole surface touching the display: the first part to reach the screen is usually a corner. Such a behavior would result in the mistaken detection of an eraser-sized object at the touch-down event.

4.3.5 Drawing quality

The resulting quality of the drawings on SAGEBoard is heavily dependent on the quality of the input hardware. We tested SAGEBoard on a small display with a highly precise touch overlay and the drawings were flawless. However, the large screen used to test the application in its full scale has an imprecise overlay, and this may result in some discomfort for the first time users. Anyway, after a short adaptation period the users are able to write, and especially draw, more accurately. Figure 6 shows how a frequent user can write on the wall: the quality is not very high, but it is definitely acceptable, considering that we are not used to write on vertical surfaces. Drawing is instead easier, Figure 9 shows how an experienced user could reproduce Van Gogh's famous painting, Starry Night, using only her fingers and a brush.

CHAPTER 5

EXTENSION: IPAD APPLICATION

As shown by our experience with SAGEBoard, touch overlays on large displays are often imprecise. We tested the application on smaller displays and the accuracy was improved, enabling precise drawing and writing. However the smaller size limited the collaboration, due to the inevitable space occlusion.

We also noticed that, in general, it is better to write on horizontally placed displays than vertical ones, probably because of the way we are used to write in real life.

For all these reasons we decided to develop a mobile application that allows remote interactions with SAGEBoard using a tablet. In this chapter we will discuss the challenges we encountered developing this extension and how we solved them.

5.1 Native Application

At first we thought about developing the extension as a web application, so that many different devices could access it just using a web browser, but after a brief research we decided that it was not possible. Web browsers, even on tablet, use the touch events to simulate a mouse pointer, losing all the valuable informations contained in the raw touch data.

A native application, instead, can leverage all the informations related to the touch events, allowing the interaction to be smoother and more accurate. Lastly, but still very important, simulating a pointer event on a tablet means losing the multi-touch capability: there cannot be more than one simultaneous pointer interaction. In a native application all the touch events are reported to the system, and multi-touch can be managed in the ways we will explain in the following sections.

The choice of the iPad was dictated by availability: we had an iPad available to use and we decided to develop the extension for it.

5.2 Design

The application needs two different connections to the server:

- Data Connection. Needed to send the touch input data to the server.
- Display Connection. Needed to show what is currently visible on the collaborative canvas of the selected server, in order for the user to see where he is drawing.

Surprisingly, the data connection issue turned out to be the simplest one. Using a Swift library called SwiftSocket [22], we open a UDP connection with the server and send the touch data analyzed on the tablet using the same protocol we describe in Section 4.3.1. As you will see, instead, the display connection is more challenging to implement.

The display visualization is not strictly necessary in the application, but after the first tests we noticed that writing, and even more annotating, was difficult without directly seeing what we were writing over. However, the implementation of this feature presented many problems, such as how to display a very large screen on a pretty small display. Another problem was given by the intrinsic tileable nature of SAGE2: which display should we connect to? If we connected to an arbitrary display, drawing over its borders would have been impossible, so we connected to the overview display.

The overview display gives a scaled version of the canvas that can fit the client browser size, as we explained is Section 3.1. At first we tried to use the overview client directly, so the drawings made on the tablet were scaled back to the original size of the canvas, but, especially with very large displays this method turned out to be completely unfeasible. It is difficult to write if an error of one pixel is translated to 20 pixels on the original screen.

Using the overview client as it is was not a good idea, so we thought about what would have been the better way to write on a big screen using a smaller device. The solution we found was a simple one: we can navigate the big screen using a window having the same size of the device and write on that, so that our drawings can be reproduced with a one to one scale on the screen.

In some cases most of the display might be empty, making it difficult for the user to understand where the device screen is located with respect to the whole canvas. To overcome this problem we show the user his position on the display using a colored box. This solution also creates a visual feedback for the other users, allowing them to see how many people are currently connected with a remote device and where they are planning to draw something. This approach is explained in Figure 10.

5.3 Implementation

5.3.1 Data Connection

As already mentioned in the previous section, we use an open source Swift library called SwiftSocket [22] in order to send UDP messages to the server.

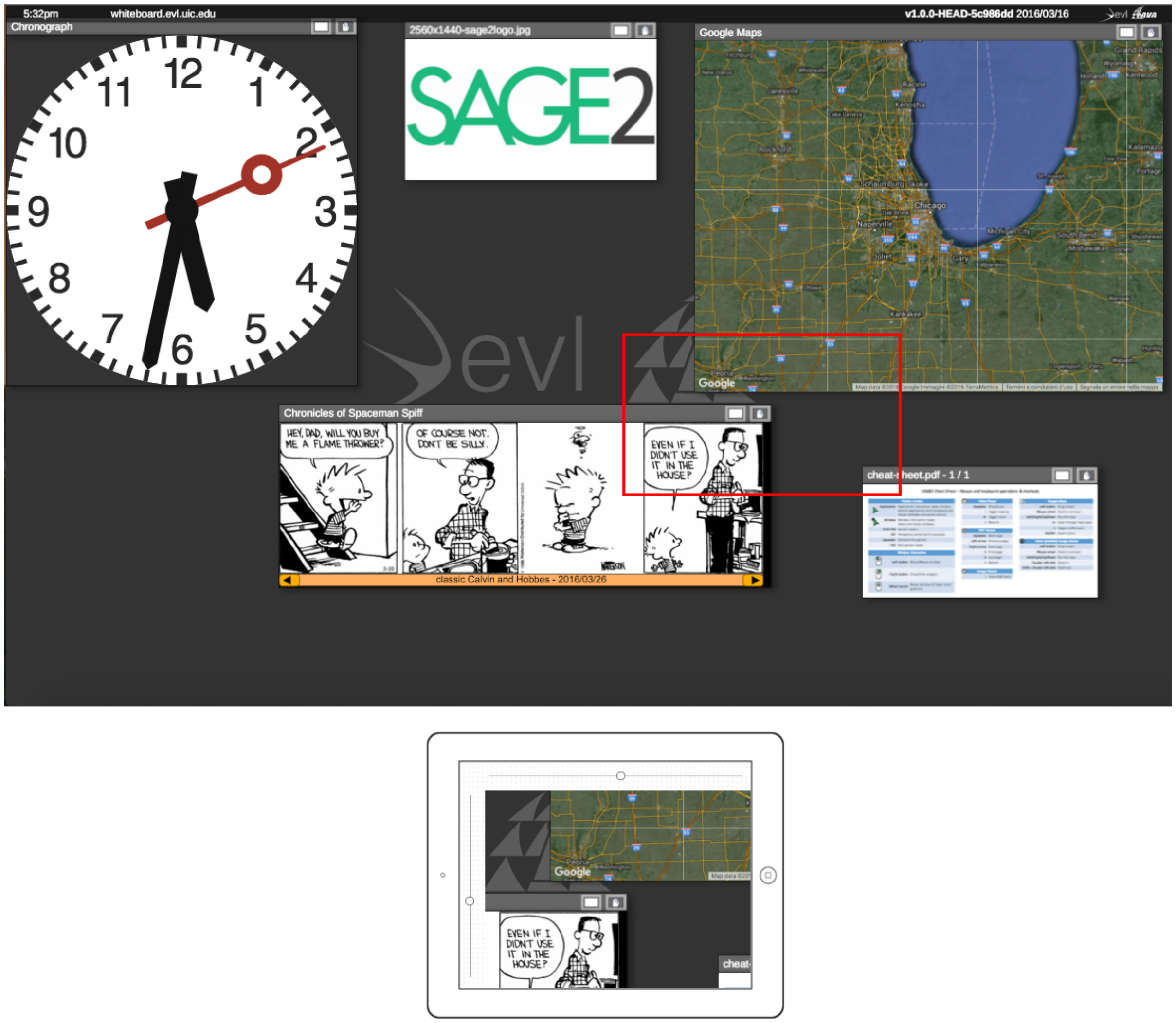

5:32pm
whiteboard.evl.uic.edu
v1.0.0-HEAD-5c986dd 2016/03/16
Chronograph
2560x1440-sage2logo.jpg
SAGE2
Google Maps
Chronicles of Spaceman Spiff
HEY, DAD, WILL YOU BUY ME A FLAME THROWER?
OF COURSE NOT. DON'T BE SILLY.
EVEN IF I DIDN'T USE IT IN THE HOUSE?
classic Calvin and Hobbes - 2016/03/26
cheat-sheet.pdf - 1 / 1
evl
EVEN IF I DIDN'T USE IT IN THE HOUSE?
cheat-

Figure 10: An iPad connected to the server, the red box represents the position of the iPad focus window inside the whole screen.

It is technically wrong to call the Data Connection a connection, since the UDP protocol strength is the connectionless approach. However, in this case we communicate to the server when we are connecting and when we are disconnecting, so we use this term to refer to our hybrid connection protocol.

The core of the data connection is: once the touch events are recorded by the application, they are given an incremental id in order to distinguish them, they are translated in the data format that the server expects, and finally they are sent directly to the server, that deals with the raw data.

The protocol described in Section 4.3.1 has been extended for this application, in order to deal with the increased complexity. The UDP message contains an input device field and an additional data field, so that the server can recognize that the touch data is coming from the remote device. Some examples of additional data can be the offset of the device window with respect to the whole display, the screen in which the device is currently located, or even some informations about the user.

5.3.2 Display Connection

As we explained in Section 5.2, we use a focus window paradigm in the remote application.

First of all we have a WebView inside our application, that connects as an overview client to the server. But the overview client is scaled to fit the WebView, and that is not the behavior we want. So, using some javascript injection in the WebView, we change the scale of the overview canvas to 1, resulting in a one to one representation of the whole server display.

Once the canvas is scaled back to normal, what we can see on the tablet is just the top-left corner of the display, but at its natural scale. Now the problem is how to let the user navigate inside the whole display with the focus window. The solution we developed consists in a pair of slider, one arranged vertically and one horizontally, that represent the WebView position inside the bigger canvas.

When one of the sliders is moved, the application injects Javascript code inside the WebView, translating the whole canvas div in order to bring the correct portion of the display inside the top-left corner of the page. The application also sends a special UDP message to the server, communicating the new position of the focus windows, so that the box representing the connection can be moved on the display.

5.3.3 Additional Features

Thanks to the additional fields in the UDP message we can send virtually any kind of information to the server, so we decided to use these fields in order to create a better application.

The first problem we noticed was that, if the user wanted to change the color or size of the stroke, he had to navigate all the way to the palette using the sliders. Since in a very large display this task takes a considerable amount of time, without even considering the trip back, we decided to insert a palette bar in the remote application to make the user life easier.

Using the palette bar the user can access all the functions available on the palette described in Section 4.2.5. When the user touches one of the palette bar buttons a message is sent to the server, using the additional fields to specify what action should be taken and possibly additional informations such as the color of the new stroke.

Thanks to this approach users can customize their experience: different users could draw with different colors at the same time, as long as they are using different input devices. This is possible because the messages sent by the remote application can contain a user, or device, identifier, providing what was not possible in the original SAGEBoard application: a form of user recognition.

CHAPTER 6

USER STUDY

In this chapter we will discuss a small scale user study that was conducted using our application.

6.1 Goal

The goal of this user study is to understand how the collaborative behaviors change between using an electronic whiteboard such as our application and a more common single mouse input.

6.2 Hypothesis

A whiteboard application could help users in creating an optimal path on a city map through its annotative functions. The user could circle the interesting points and visually try to create a feasible solution in order to visit the interesting points.

We hypothesize the user will spend more time discussing possible solutions and be more satisfied with the outcome than when performing the same task with a simple mouse.

6.3 Procedure

The user study consists of six variations of the same task, and it is performed by two users.

The core of the task is to create an optimal path connecting interesting points in city map under a time constraint. The users are given a list of interesting points, along with a value describing how much they are interested into that point. After receiving the list they are told that they need to create a path, starting from a given start point, connecting as many interesting points as they can, the value of the path will be given by the sum of the interest value given by each attraction visited. Each interesting point has a visit duration time, describing how much time is spent visiting it. The path can only connect points that are connected in the map visualized on the screen and shown in Figure 11. Every transfer from an attraction to another, following one of the lines, takes 15 minutes. The users are asked to create the optimal path with a duration lesser or equal to 8 hours, representing a day from 9AM to 5PM. The path also needs to pass through at least one restaurant, three special points on the map, representing a lunch break during the visit. The restaurant also have different visit time, increasing the problem complexity.

The map not only encodes the connections between the attractions but also the interest points for each attraction, as shown in Figure 11. The shape of the markers encodes the user who has an interest in that attraction: the circle represents User 1, while the star represents User 2. The color of the marker encodes the interest in a particular attraction: red means a low interest, an interest value of 1, yellow means a medium interest, an interest value of 2, and finally green means a high interest value, with a value of 4. The black circle in the center of the map represents the starting point of the path, while the blue markers are the three restaurants.

The task is performed first by both the users independently, one time using a mouse as input device for the whiteboard (drawing with the mouse) and one time using SAGEBoard and its touch based input. The two configuration use maps with different interest values, in order to give the user a new and at the same time comparable problem. The order of the two configurations is randomized, so that the learning time of the interface does not pose a problem during the data analysis. These first two tasks get the users used to the two configurations, preparing them for the last round of tasks.

The task is then performed two times in collaboration between the two users, using the two input configurations with two different maps. In these last tasks the users are asked to create a path representing a good compromise between their different interests, like a couple would do in real life. The task also has a competitive side, but the users are asked to create a good balance between their interest, so the optimal path should score the maximum value of combined interest points while keeping them balanced between the two of them.

6.4 Problem

The problem itself is fairly difficult, what we are asking to the users is in fact to search for an optimal path in an undirected graph given some weights on the nodes. We are leveraging the fact that the problem is common enough in real life to make it easier.

In the individual execution of the tasks, the user just needs to find a path that connects the most interesting points.

In the collaborative execution, instead, the problem gains complexity because the optimal path for one of the two users could be a very scarce path for the other. The best balanced path must be found through the use of diplomacy, and the purpose of our user study is to understand how behaviors change with different input devices. Will the user that keeps the mouse for most of the time be the one with the better score? Will the user that has drawn the least the one with the worst score?

The answers to these questions will be discussed in the following sections.

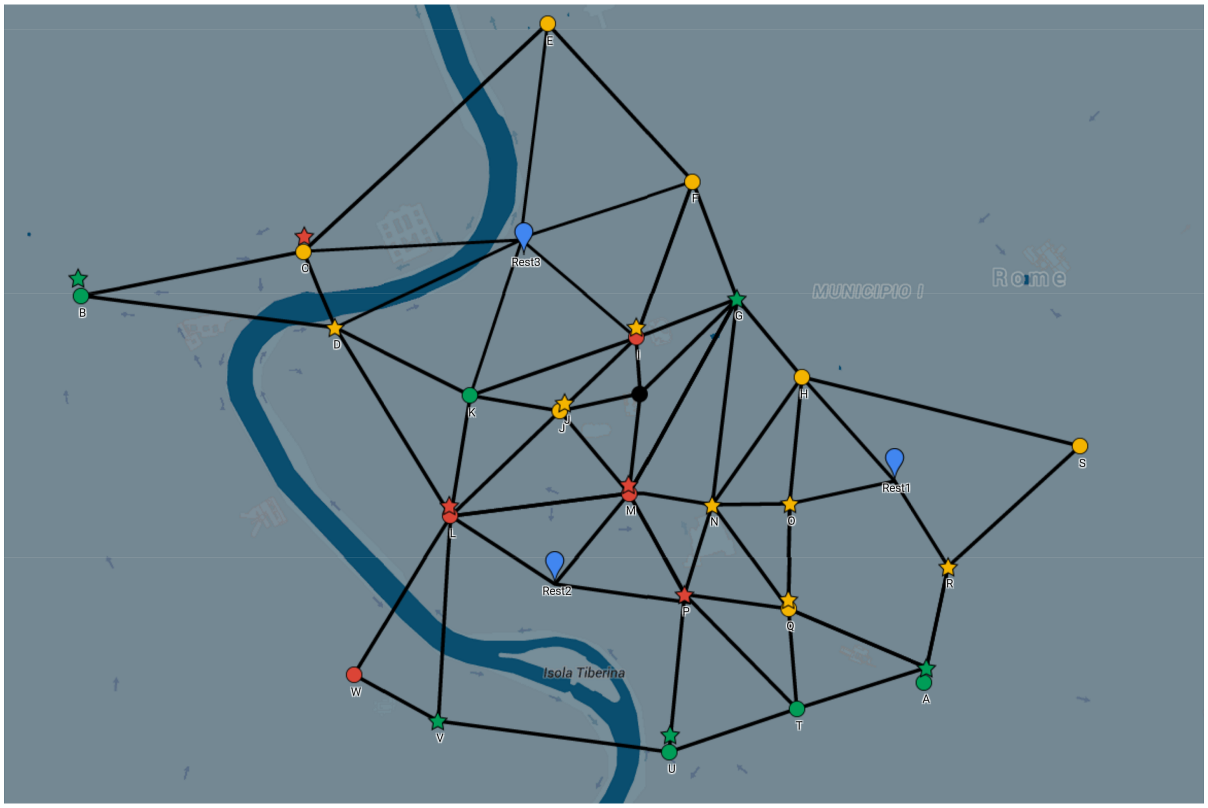


Figure 11: One of the maps used in the user Study.

6.5 Mouse Drawing

In order to make the results between the whiteboard approach and the mouse approach comparable we implemented a new functionality in SAGEBoard: mouse drawing.

If we had observed the user using a laptop in the mouse configuration, we would have invalidated the results, because the differences might have been related to the screen size and not to the input type.

So we decided to let the user draw on the whiteboard using a mouse pointer, in this way the two different input configurations have the exact same interface, and all the differences must be related to something else.

The user can draw a line by clicking the left-hand button and dragging the mouse through the desired path. The user can also interact with the palette in the same way he can with the touch input. Finally, the user can erase what he already wrote just by clicking the right-hand button and use the mouse as an eraser.

The implementation of this feature is fairly simple and does not deserve a dedicated section, we simply emulate a touch event when we receive a pointer event. The only tricky part was the eraser, but we solved it just by emulating a touch slightly bigger than the whiteboard mode eraser threshold when the user clicks the right-hand button of the mouse.

6.6 Setup

In this section we will describe the environment in which the users were observed, in order to give a correct frame to the experiment.

We performed the experiment in the CyberCommon classroom at the University of Illinois at Chicago. SAGEBoard was running on a large tiled vertical display composed by 18 LED screens spanning a total size of 6 meters in width and 1.8 meters in height, with a total resolution of 8200px by 2300px. The screen has an infrared touch overlay to detect touches, it is a prototype version of a commercialized overlay, so its precision is not very high, but it is after all functional. The room is just a tiny bit larger than the screen, so there is not much space for the users to stay at the side of the screens. When the users are being observed in the mouse configuration, they sit at a table located roughly 1 meter away from the screen and they use a high precision bluetooth mouse.

The two maps are shown simultaneously side by side on the screen, in order to save time parallelizing work. During the first two single user experiments the users work in parallel side by side, one of them uses a mouse, while the other one uses the multi-touch input on a different map. In the collaborative experiments the two users first collaborate on one of the two map using a random input configuration, then they move to the other map using the other input. Each task has a 15 minutes time limit, in order to keep a reasonable overall experiment duration.

The experiment was repeated 10 times in order to get a statistically significant result, involving 20 different users.

The setup for the user study is shown in Figure 12.

6.7 Feedback

At the end of the experiment we ask the users to fill in a questionnaire aimed at assessing the SAGEBoard usability and the best input configuration.

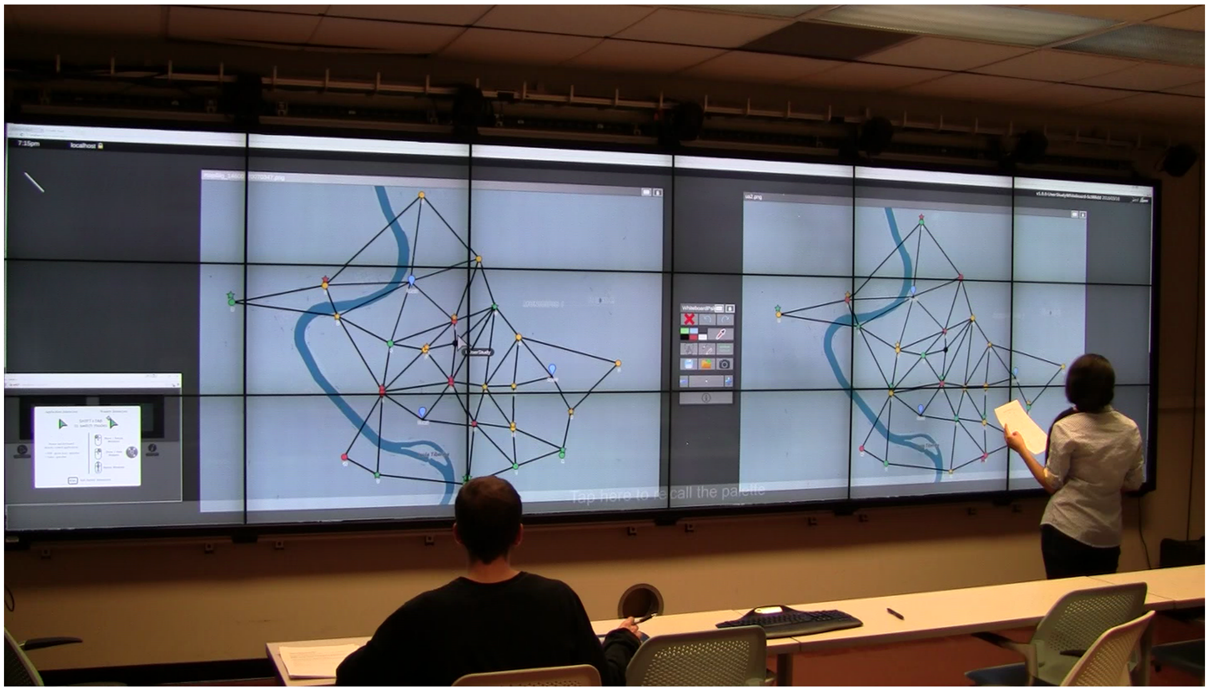


Figure 12: The room setup for the user study.

The first seven questions are focused on SAGEBoard and the multi-touch input. We ask the users if they found the whiteboard application easy to use and also if they have any suggestions to improve it. The last three questions try to understand what is the best configuration between mouse and multi-touch input in the user's opinion, and why.

6.8 Results

In this section we will analyze what we have learned from this user study, and how the users felt about it.

6.8.1 Premise

We will not go into many details about the actual score of the generated paths, because what we are really interested in is how the users collaborate using the different input devices. Moreover, the results are not very different between the two input devices because of the high complexity of the problem. The solution space of this problem, in fact, is immense, and in all our observations we almost never found two equal solutions. This diversity of solutions makes the result distributions for the two input devices very similar.

What we will be very thorough in analyzing is the collaboration pattern: did the users collaborate as equals, or a leader emerged? Was the mouse be shared equally or was it used as the leader's representation of power?

6.8.2 Individual Task

As explained in Section 6.6, the two users are first asked to solve the given task on their own, using the two possible input devices. This part of the experiment should give us an idea of how the input influences the task resolution itself, so that we can take what we discover here into consideration when analyzing the collaborative task.

Figure 13 shows the average results for the two users on the two different maps, where the results related to the mouse input are shown in blue, and the ones related to the multi-touch input in orange. As you can see Map 1 has lower average results, the mean score for the two players is similar, and the input does not seem to play a role in the outcome. Map 2, instead, has higher average results, User 2 has a better mean score, and the multi-touch input seems to increase the scores sensibly.

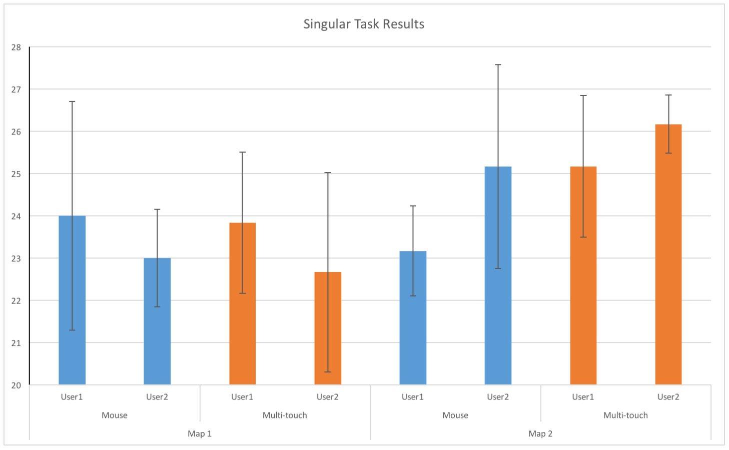


Figure 13: Overview of the Individual task average results in the user study.

Our interpretation of these results, considering also the scores distribution, is that Map 1 is slightly more difficult in terms of score, but there is a collection of similar paths for both the users that is easy to find and gives good results. Map 2, instead, allows higher scores, but there are many good paths that can be taken, and this increased solution space leads to an higher task complexity. The input does not seem to play a major role in the score, small differences in the mean results are probably simple statistical fluctuations. Our assumptions are backed up also by Figure 14, that shows the number of users that reached the time limit when solving the individual task. The graph shows that users found Map 2 more challenging and needed more time to find a satisfactory path, regardless of the input used.

Keeping these findings in mind we will be able to objectively analyze the collaborative results.

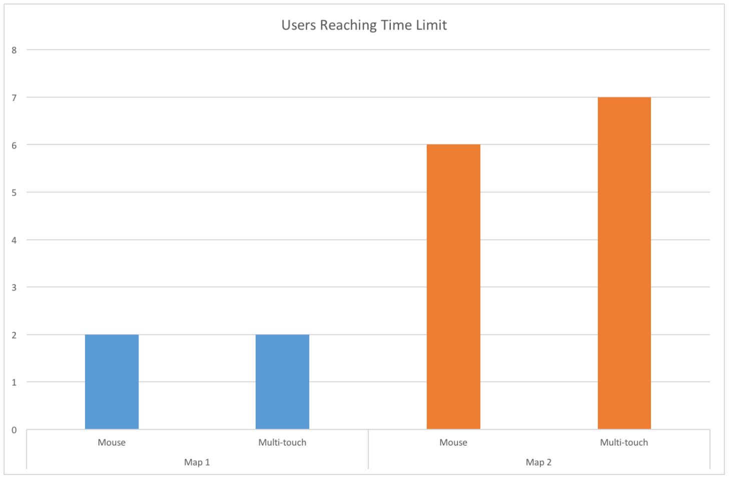


Figure 14: Number of users reaching the time limit in the individual task.

6.8.3 Collaborative Task

Now that we know the two input devices do not influence the task resolution, we can assume that every difference we find in the collaborative task will be associated with how different input devices change the collaborative behavior.

Let's start by analyzing the average scores for the different maps and input configurations: the graph is shown in Figure 15. The first thing we notice is that the scores are now more equally distributed between the two maps, and again in Map 1 the input does not really seem to make a difference. Map 2, instead, shows an interesting pattern: the average score achieved using the multi-touch input is consistently higher than the one scored with the mouse input. This might be somehow correlated with the collaboration behaviors, but we cannot say it for sure because the same pattern is not shown in Map 1.

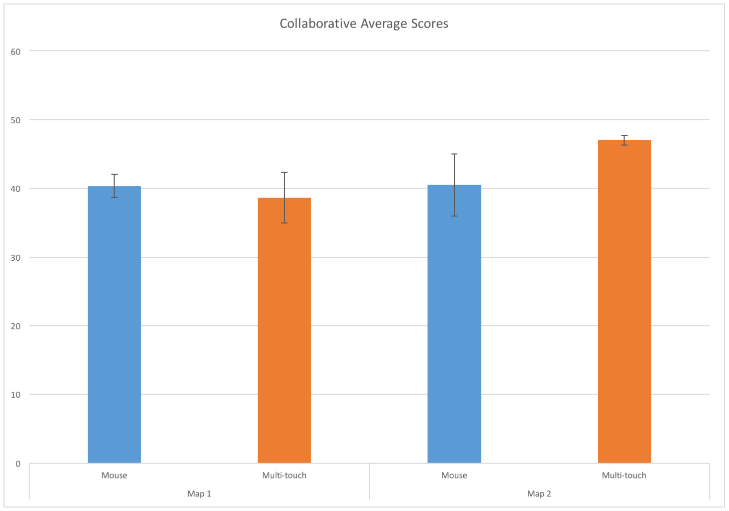


Figure 15: Average scores in the collaborative task.

However, analyzing the data more deeply, we can discover more interesting and consistent patterns. Figure 16 shows the mean difference in the two users individual score during the collaborative task, and we can see that the multi-touch input really lowers this difference. The users were asked to create a common path, but to still keep in mind that they had their own personal tastes, adding a competitive factor to the task. Our interpretation of this difference is that when creating the common path stronger personalities emerge and drive the path towards a better score for themselves. This happens more often with the mouse input because the two users do not perceive themselves as peers, since they usually play different roles.

As a matter of fact, when using the mouse configuration, the users do not share the mouse equally, instead one of them is usually assigned to using the mouse, while the other one checks the feasibility of the path, or sometimes stays in the proximity of the screen, pointing the

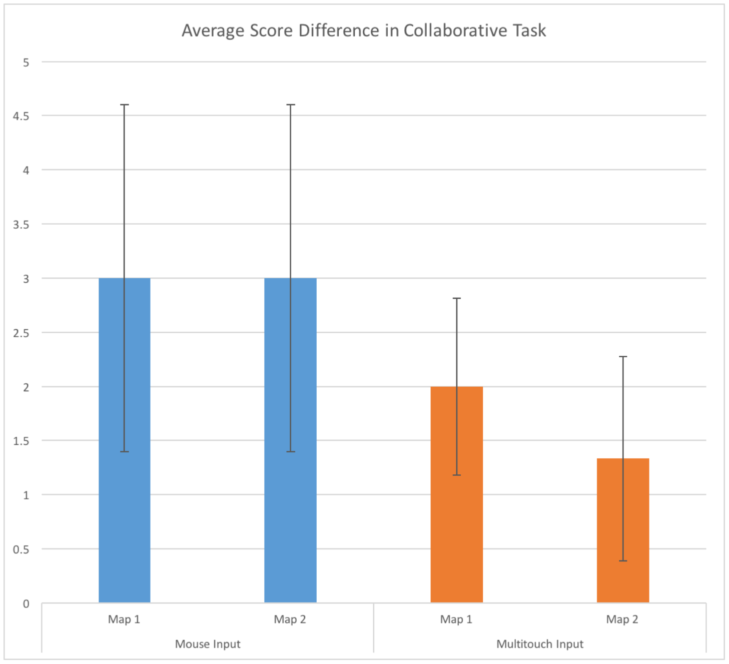


Figure 16: Average score differences in the collaborative task.

path to follow. These two situations are shown in Figure 17, where the left image represents a pointer and draftsman situation, while the other one shows a draftsman and analyst situation. Interestingly enough, the role played does not seem to be related with the leadership of the situation: sometimes the leader is the one drawing the path, while on other occasions the leader was the pointer or the analyst. Another interesting finding is that when a leader emerges, it is rarely on purpose: it seems like strong personalities subconsciously create a better path for themselves. During one of the observations, one user thought of a first draft path and went ahead drawing it, after a while he realized on his own that the path was completely in his favor,

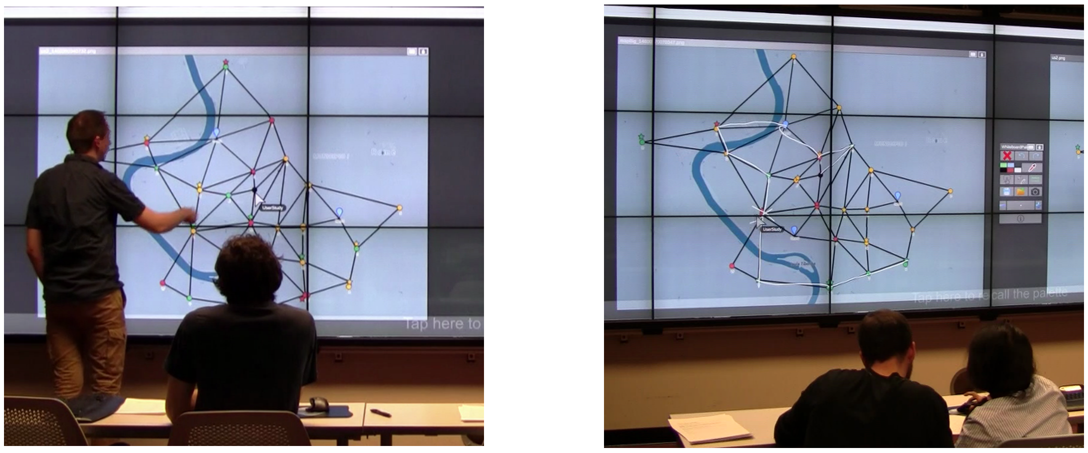
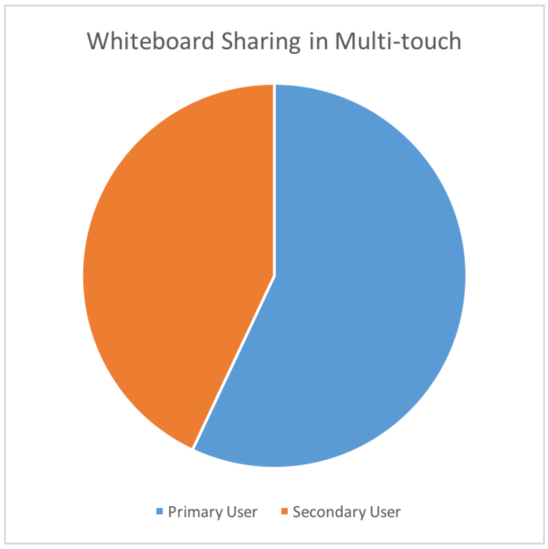


Figure 17: Two possible situations observed when using the mouse input.

and stated: “Wow! That’s remarkable! I did not hit a single one of them that does not have a circle. That can’t possibly be the best common path.”

Not all the experiments showed a leader clearly emerging, sometimes the users collaborated as peers, but even in these occasions the mouse configuration showed a great disparity in terms of whiteboard usage. When collaborating using the multi-touch input, the users draw a roughly equal amount of path in almost every observation, while when in the mouse configuration, the mouse is used mostly by one of the two users, and it is rarely shared at all. Figure 18 shows the ratio between primary and secondary user usage time of the whiteboard application in the two configurations, where the primary user is the one that interacted the most during the experiment. The graph clearly shows how there is an almost even distribution of interaction time when using the multi-touch input, while the scenario changes completely when dealing with a single mouse input.



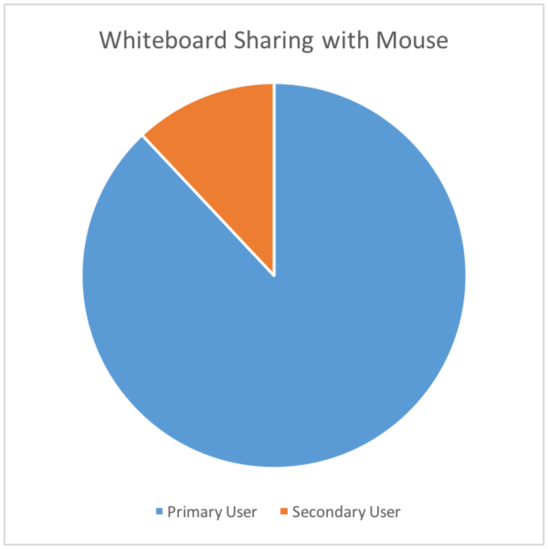


Figure 18: Whiteboard sharing in the two configurations.

So, even in observations where a leader could not be clearly appointed, one of the two users had to step up and be the one in charge of the mouse. Oddly, in these occasions, the user appointed to be the draftsman did not take advantage of the situation, on the contrary most of the times when faced with a decision between two paths, he would willingly lose points. This is probably caused by the fact that, being the one actually drawing the path, the user feels like he has the last word in the decision, and choosing the path that is best for him would make him feel like he is taking advantage of the situation.

6.8.4 Annotations in the User Study

Before the start of each session, we told to the users that, since the problem at hand was fairly complex, they could annotate anything both on the whiteboard or on the sheet of paper we gave them. We provided them both with pens to write on paper and styluses for a more comfortable screen writing interaction. We also reminded them that annotation on the screen were possible even in the mouse configuration, since the mouse perfectly emulates a touch in the application.

What we noticed is that the majority of the users did not try to annotate on the screen in the mouse input configuration. One could say that it is harder to write using a mouse, and that is definitely true, but people did not even try, so they could not know. In the multi-touch input sessions, instead, users wrote almost exclusively on the screen, sometimes with styluses and sometimes with fingers, disregarding completely the paper solution. The amount of annotations written by each single user in the two different configuration was always roughly the same, so there is no evidence that one of the two solutions induces the user to write more or less. What emerged from the experiment is that when using a mouse input, people would not even consider writing with it on the screen, while it felt natural to them to annotate on it when using the touch input.

Another interesting finding here is that people are not necessarily attracted by the stylus when they want to write, some users used it, but many others naturally used their fingers to annotate, highlighting how our solution can provide even more solutions than a plain white-board.

6.8.5 Questionnaire Results

Let's start by discussing about the first two questions:

- Please rate the **ease of use** of freehand drawing with the whiteboard application on a scale of 1 to 10, with 1 being difficult and 10 being easy.
- Please rate the **ease of use** of writing with the whiteboard application on a scale of 1 to 10, with 1 being difficult and 10 being easy.

These two questions asked the user for a usability evaluation of two different use cases in SAGEBoard, only when dealing with the multi-touch input. At first glance, the results of these questions were very discordant, as you can see in Figure 19, and it was unclear how two users could have such different feelings of the same experience. However, after rewatching the video materials, we noticed that the users that gave lower scores were using their fingers to interact with the whiteboard, while the higher grades were given by people that used one of the available styluses. The styluses are not processed in any different way with respect to fingers, the reason for this difference is probably related with the hand position that the user keeps in the two different situations. When the user is drawing with his fingers, sometimes other fingers or the hand are detected as well, because the touch overlay we used in this setup is not very precise , and it detects touches even when objects are near the screen without touching it. For this reason, when drawing with fingers, the quality decreases, due to undesired lines drawn near the user finger position. When using a stylus, instead, the user holds it like a common pen and no other touches are detected near the stylus tip, resulting in a much better drawing quality. So we separated the grades given by the users, differentiating the ones given by people that used stylus from the others, and the result is shown in Figure 20. As you can see the grades coming from people that used a stylus, shown in blue, are much higher than the others.

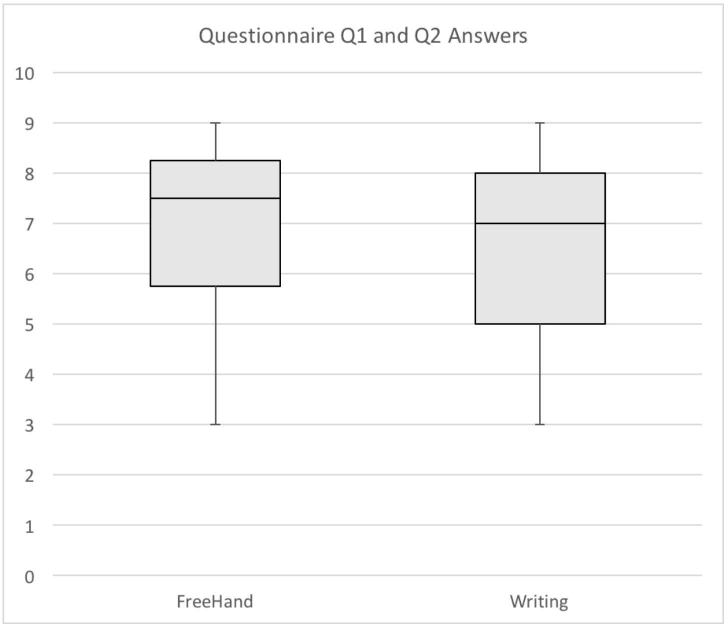


Figure 19: Results for questionnaire questions 1 and 2.

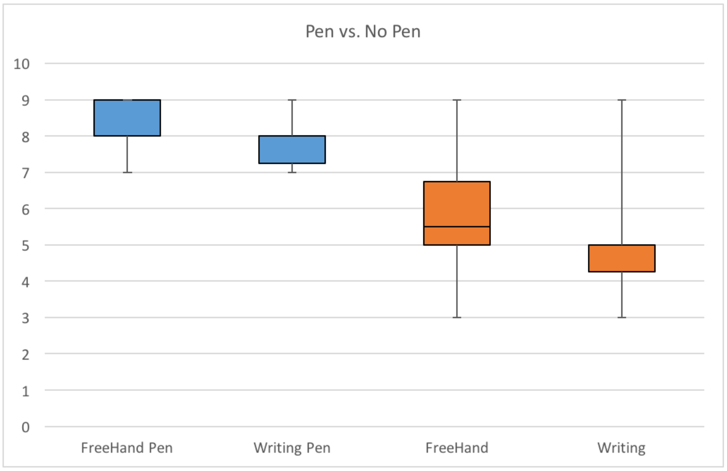


Figure 20: Results for questionnaire questions 1 and 2, differentiating users that used a stylus.

Question 3 was about the palette interface, but since the palette was not strictly needed to solve the assigned task, many users did not use it. So, only a minor fraction of the users observed was able to answer the question, and consequently we do not have enough answers to analyze the question from a statistically accurate point of view. However, the few evaluation we had were very positive, with an usability mean grade of 8.5.

Questions 4 through 7 were more open to suggestions for SAGEBoard:

- Please describe any difficulties you faced while using the Whiteboard application.
- How could the Whiteboard application be improved? Are there any features that you felt are missing?
- Do you think this application could substitute a real whiteboard? (Yes/No)
- Any further comments about the Whiteboard application used during this study

Many users found writing on SAGEBoard difficult because undesired lines were drawn even when they were not actually touching the screen, but this is a known hardware problem of the overlay we used in the experiment and we cannot do much about it. We tried SAGEBoard on more precise hardware and the problem is not present there. Also, the iPad extension we already described in Section 5 addresses this problem in an indirect way.

Some users commented that erasing should be more precise, this is again a known problem: the way we structured the drawing data described in Section 4.3.2 forces us to erase line sections instead of single pixels, because we store the lines with a low precision in order to reduce latency. Anyway we will try to better tune the stored lines precision in order to enhance the user experience.

The answers to question 6, asking if SAGEBoard could substitute a real whiteboard in the user's opinion, were encouraging: a surprisingly high percentage of testers answered positively. This is even better news when we consider that the hardware used in the experiment was not the best available, the users were probably attracted by the many functionalities of the application.

Most of the users that used a stylus during the experiment commented how the drawing experience was way better when using it. This is probably because people are not used to finger-based interaction in real life whiteboard, so the stylus turns out to be more comfortable to use. For this reason in the future we will encourage the users to use some sort of stylus to interact with SAGEBoard.

Finally, questions 8 and 9 were related to the task actually performed during the user study, with question 8 composed by two subquestions:

- How **satisfied** are you with the resulting path?
  - When using a simple **mouse input**?
  - When using the **multi-touch input**?
- Which type of input did you like more?

As shown in Figure 21 the results were clearly in favor of the multi-touch input, even though the mouse scored pretty high too.

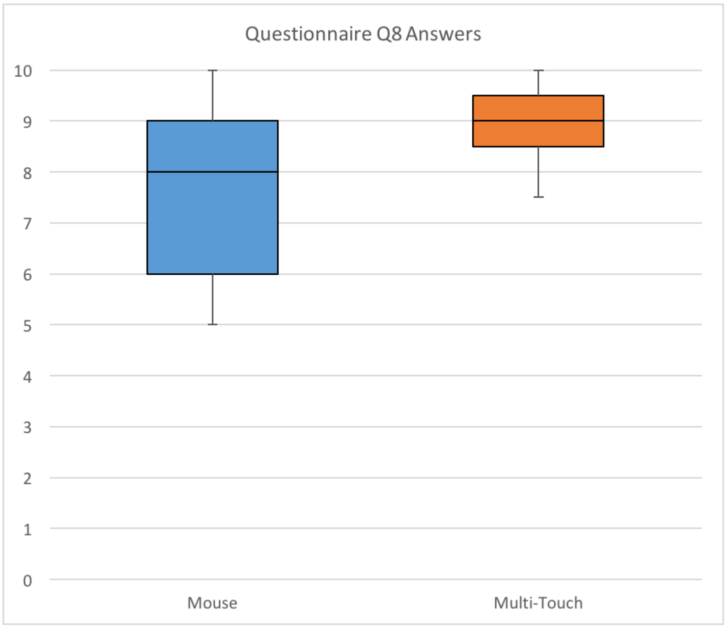


Figure 21: Results for questionnaire question 8.

Question 9, being more open to comments, helped us figure out why the users preferred the multi-touch input, also it is important to point out that, while question 8 was asking about the task results, question 9 asked the user about a preference between the two input devices, regardless of the results . Almost all the observed users answered multi-touch input to this question, apart from one user that was not satisfied with the multi-touch writing precision, and mentioned in his answer that he liked the mouse input more but “not by much”, and that it was “mainly for better note taking abilities”. The comments praising multi-touch input were various, and the most common sounded like “because we could collaborate ”, “the mouse felt restricting”, or “because it is easier to share the items in the environment”.

6.9 Discussion

From the user study results it emerges that the multi-touch input, even if it does not necessarily lead to better scores in this particular task, allows the user to collaborate way more than the mouse input. This conclusion is confirmed both by the results observation and by the users answers to the questionnaire, showing that not only the multi-touch configuration is better for collaboration, but also that the users themselves can understand it. The problem with the mouse configuration is mainly that the mouse can only be used by one user at a time, and this makes the users feel like they are not peers in the interaction. It must be said that all the people in the user study were right-handed, so even when they were sitting side by side, when one of the two users wanted to use the mouse he had to bring it away from the other's reach. We do not think that having a different mixture of right-handed and left-handed users would have changed something, because the mouse can only be used by one person at a time anyway, regardless of its position.

Most of the bad comments about the application in its multi-touch configuration were related to the precision, but we know we had a faulty hardware, and trying the application on more reliable systems showed us that those precision problems were only related to the hardware. We are sure that these hardware problems will be solved by advancements in technology, so we are not planning to modify the application in order to better deal with imprecise devices.

So, after all, we can say that the user study was a success, it showed us the power of SAGEBoard and, more in general, of whiteboard applications for large displays.

CHAPTER 7

CONCLUSION

We introduced in this work SAGEBoard, a networked application that enables large scale multi-touch displays to be used as enhanced electronic whiteboards. Our solution enables multiple users to collaboratively work together, and offers extended functionalities, such as freehand drawing, annotations on documents, selection, movement, scaling of text, and the possibility to save a session or load a previously saved one. One of this application strengths lies in its freedom of implementation: being integrated in a middleware such as SAGE2 allows it to scale to displays of every size.

We analyzed the state of the art in both the large displays and electronic whiteboard applications fields and we proposed an overview of the problems commonly encountered, trying to propose solutions to them, in order to improve our application. Then we looked at similar applications to the one we developed and we outlined what are the features that make our application stand out, such as mixing different kind of collaboration paradigms and seamless scaling to every possible resolution.

We then described how the application was designed and implemented, motivating all our design choices and describing in detail architecture and features of SAGEBoard. We looked at the application from a software engineering point of view, arguing that its design was suitable for many effortless extensions. We confirmed our statements by describing an extension of SAGEBoard: a tablet application that allows users to remotely interact with the system. The remote control extensions also added a new level of collaboration to the application, making it deployable even on not touch enabled large displays. Our work proposes an architecture to support whiteboard applications and analyzes many problems related to whiteboard applications, such as gesture management and delay mitigation. These technical details are the main contributions that this work brings to the field.

Finally, we tested the power of our application through a small scale user study. The experiment showed us that multi-touch controlled applications, with respect to the common mouse restricted ones, create a much more collaborative environment among multiple users. The single mouse revealed to be deleterious for collaboration, since it can be seen as a symbol of leadership. The user study also tested SAGEBoard usability, showing that the users were very impressed by its ease of use, and that the majority of them think that SAGEBoard could substitute a real whiteboard.

CITED LITERATURE

1. Tan, D. S., Robertson, G. G., and Czerwinski, M.: Exploring 3d navigation: combining speed-coupled flying with orbiting. In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 418–425. ACM, 2001.

2. Tan, D. S., Czerwinski, M. P., and Robertson, G. G.: Large displays enhance optical flow cues and narrow the gender gap in 3-d virtual navigation. Human Factors: The Journal of the Human Factors and Ergonomics Society, 48(2):318–333, 2006.

3. Andrews, C., Endert, A., and North, C.: Space to think: large high-resolution displays for sensemaking. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 55–64. ACM, 2010.

4. Baudisch, P., Good, N., Bellotti, V., and Schraedley, P.: Keeping things in context: a comparative evaluation of focus plus context screens, overviews, and zooming. In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 259–266. ACM, 2002.

5. Czerwinski, M., Robertson, G., Meyers, B., Smith, G., Robbins, D., and Tan, D.: Large display research overview. In CHI'06 extended abstracts on Human factors in computing systems, pages 69–74. ACM, 2006.

6. Moreland, K.: Redirecting research in large-format displays for visualization. In Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on, pages 91–95. IEEE, 2012.

7. Ni, T., Schmidt, G. S., Staadt, O. G., Livingston, M. A., Ball, R., and May, R.: A survey of large high-resolution display technologies, techniques, and applications. In Virtual Reality Conference, 2006, pages 223–236. IEEE, 2006.

8. Febretti, A., Nishimoto, A., Thigpen, T., Talandis, J., Long, L., Pirtle, J., Peterka, T., Verlo, A., Brown, M., Plepys, D., et al.: Cave2: a hybrid reality environment for immersive simulation and information analysis. In IS&T/SPIE Electronic Imaging, pages 864903–864903. International Society for Optics and Photonics, 2013.

9. Jakobsen, M. and Hornbæk, K.: Proximity and physical navigation in collaborative work with a multi-touch wall-display. In CHI'12 Extended Abstracts on Human Factors in Computing Systems, pages 2519–2524. ACM, 2012.

10. Jakobsen, M. R. and Hornbæk, K.: Up close and personal: Collaborative work on a high-resolution multitouch wall display. ACM Transactions on Computer-Human Interaction (TOCHI), 21(2):11, 2014.

11. Kruger, R., Carpendale, S., Scott, S. D., and Greenberg, S.: Roles of orientation in tabletop collaboration: Comprehension, coordination and communication. Computer Supported Cooperative Work (CSCW), 13(5-6):501–537, 2004.

12. Scott, S. D., Carpendale, M. S. T., and Inkpen, K. M.: Territoriality in collaborative tabletop workspaces. In Proceedings of the 2004 ACM conference on Computer supported cooperative work, pages 294–303. ACM, 2004.

13. Elrod, S., Bruce, R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., Lee, D., McCall, K., Pedersen, E., Pier, K., et al.: Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 599–607. ACM, 1992.

14. Pedersen, E. R., McCall, K., Moran, T. P., and Halasz, F. G.: Tivoli: An electronic whiteboard for informal workgroup meetings. In Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems, pages 391–398. ACM, 1993.

15. Agostini, A. and Di Biase, E.: Large multitouch screens to enhance collaboration in the classroom of the 21st century: an italian experiment. IxD&A, 15:40–56, 2012.

16. Ashdown, M. and Robinson, P.: The writings on the wall: Large, remotely controlled displays. In Proceedings of the First European Conference on Computer-Supported Collaborative Learning (Euro-CSCL 2001), pages 83–88, 2001.

17. France, D. J., Levin, S., Hemphill, R., Chen, K., Rickard, D., Makowski, R., Jones, I., and Aronsky, D.: Emergency physicians behaviors and workload in the presence of an electronic whiteboard. International journal of medical informatics, 74(10):827–837, 2005.

CITED LITERATURE (continued)

18. Google docs support. https://support.google.com/docs/topic/21008?hl=en&ref_topic=2811805.

19. Microsoft office sharing support. https://support.office.com/en-us/article/Simultaneously-edit-a-document-with-other-authors-2a6059e7-9fe9-4e66-8ecd-f3d5372

20. Smart iq official page. http://education.smarttech.com/en/products/smart-kapp-iq.

21. Marrinan, T., Aurisano, J., Nishimoto, A., Bharadwaj, K., Mateevitsi, V., Renambot, L., Long, L., Johnson, A., and Leigh, J.: Sage2: A new approach for data intensive collaboration using scalable resolution shared displays. In Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on, pages 177–186. IEEE, 2014.

22. Swift socket library official page. https://github.com/swiftsocket/SwiftSocket.

VITA

NAME	Filippo Pellolio
EDUCATION	
	Bachelor's Degree in Computer Engineering
	Jul 2014, Politecnico di Milano, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2015 - TOEFL examination (107/120)
SCHOLARSHIPS	
Spring 2016	Research Assistantship (RA) position (20 hours/week) with full tuition waiver plus monthly stipend
Spring 2015	Teaching Assistantship (TA) position (32 hours/term)