2013-08-12

# Interactive Techniques Between Collaborative Handheld Devices and Wall Displays

Daniel Leon Schulte
*Brigham Young University - Provo*

# Interactive Techniques Between Collaborative Handheld Devices and Wall Displays

Daniel L. Schulte

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Dan R. Olsen, Chair
Bryan S. Morse
Robert P. Burton

Department of Computer Science

Brigham Young University

August 2013

ABSTRACT

Interactive Techniques Between Collaborative
Handheld Devices and Wall Displays

Daniel L. Schulte
Department of Computer Science, BYU
Master of Science

Handheld device users want to work collaboratively on large wall-sized displays with other handheld device users. However, no software frameworks exist to support this type of collaborative activity. This thesis introduces a collaborative application framework that allows users to collaborate with each other across handheld devices and large wall displays. The framework is comprised of a data storage system and a set of generic interactive techniques that can be utilized by applications. The data synchronization system allows data to be synchronized across multiple handheld devices and wall displays. The interactive techniques enable users to create data items and to form relationships between those data items. The framework is evaluated by creating two sample applications and by conducting a set of user study interactive tasks. The data recorded from these evaluations shows that the framework is easy to extend, and that with minimal training, the generic interactive techniques are easy to learn and effective.

Keywords: interactive techniques, collaboration, handheld devices, data storage

Table of Contents

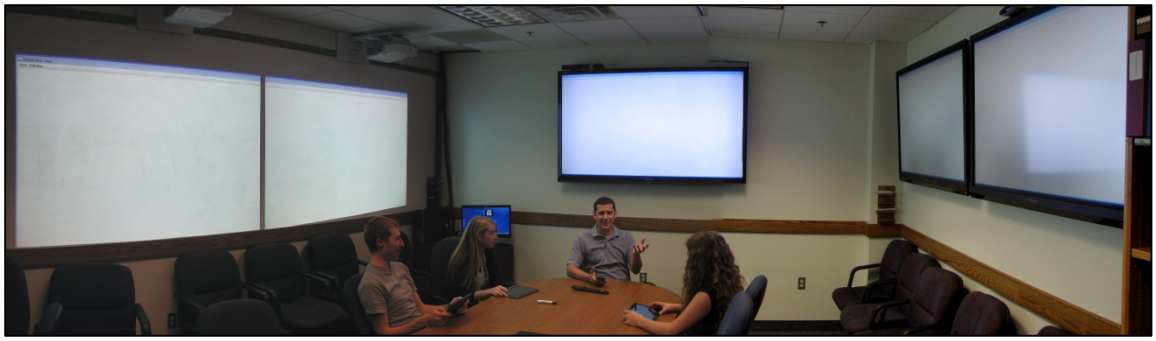

Figure 1: Users sit around a table during the beginning of a meeting.

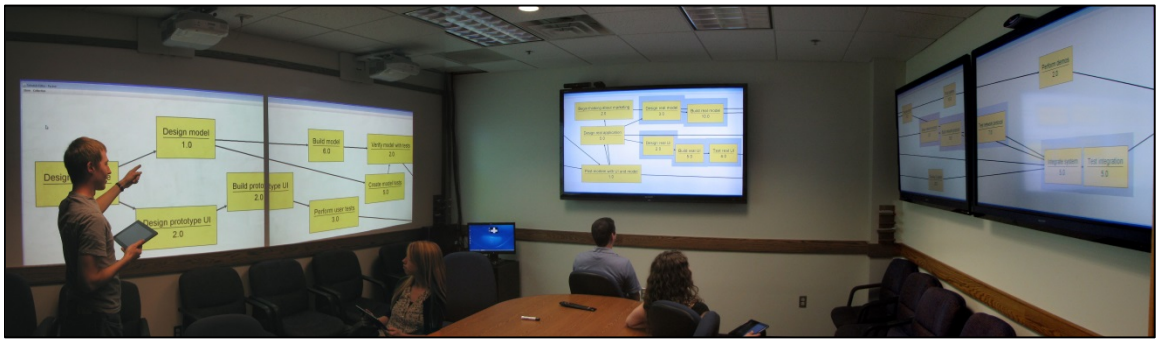


Figure 2: Users perform individual work during a portion of the meeting.

## Chapter 1. Introduction

Handheld device users want to work collaboratively on large wall-sized displays with other handheld device users. No software systems are currently in place to allow this type of work. By working together in the same area at the same time, people are able to bring different perspectives to the group. When people collaborate face-to-face, they address and confront issues as they arise, rather than waiting for someone to return an e-mail request or, even worse, wait for the next scheduled meeting. Although research is being done in remote collaborative work, our work focuses on scenarios where people are working in the same room. These types of co-

located meetings occur frequently in a variety of different scenarios, which means that our work can be broadly applied to both industrial and educational settings.

Large wall-sized displays are quickly becoming more and more feasible for businesses and educational settings. Not long ago, 40" monitors cost around $4000, but today a monitor of the same size with a much higher resolution can be purchased for around $400. A similar trend has begun to occur with displays that are 80" and higher. We expect the cost of these displays to follow the historic trend to the point that it will be reasonable to put several large wall displays in conference rooms and possibly even classrooms.

Handheld devices are critical in these real-time collaboration sessions. These sessions could be held in rooms that already contain several large wall-sized screens, such as many current business conference rooms. Using their handheld devices, users need to be able to control content of the large screens. The challenge is that no techniques currently exist for interacting with the contents of large wall-sized displays from a set of handheld devices.

With these small handheld devices, people are free to get up, move around, and form smaller sub-groups in whatever way feels appropriate for the current task. In Figure 1, we see four individuals sitting down at the beginning of a meeting. Once the group is finished discussing their goals for their collaborative work, they turn to face the nearest wall display and begin using their handheld devices and the wall displays to accomplish their project goals, as shown in Figure 2. As they work, some members of the group work as individuals, while others form sub-groups as they discuss data they see on the wall displays. When they are finished working on the task that the sub-group was interested in, the members of the sub-group return to their own individual tasks until a new sub-group needs to be formed for a different task. This regrouping of

people to fit the current task can happen several times over the course of a collaborative work session. As people form subgroups, their discussions focus around the data displayed on the large screen, rather than on the data on their handheld devices. Focusing on wall displays instead of handheld devices allows users to communicate more naturally in a group, as they do currently with non-digital mediums such a whiteboards. Creating a more natural working environment for groups enables users to communicate easier and complete their tasks faster.

The type of work described above cannot happen at the present time because no software frameworks exist to support it. This type of framework does not exist because useful techniques for interactions between a handheld devices and large wall-sized displays are not yet understood. For example, how do we allow users to interact on handheld devices and have the results of those interactions broadcast to other users' handheld devices as well as the shared wall displays? Establishing this framework and these interactive techniques will support collaboration between handheld device users and wall-sized displays that is currently unavailable.

To demonstrate what users of these collaborative applications need, we will consider two example scenarios. Our scenarios depict groups using a schedule creating application and a mood board creating application. To evaluate our framework, we created two sample applications that contain all of the features discussed in these example scenarios.

Schedule Example

To demonstrate the needs of these users, consider a meeting where a group of managers has been assigned to create a schedule for an upcoming project. This schedule will contain all of the milestones, or reasonably large tasks, that pertain to the project. Each milestone has a set of other milestones that must be finished before it can be started. Each manager in the meeting is

responsible for a different part of the project, such as a manager over the testing team or a manager over the development team. As the meeting begins, each member of the meeting turns on their handheld device, which could be either a tablet or a smartphone.

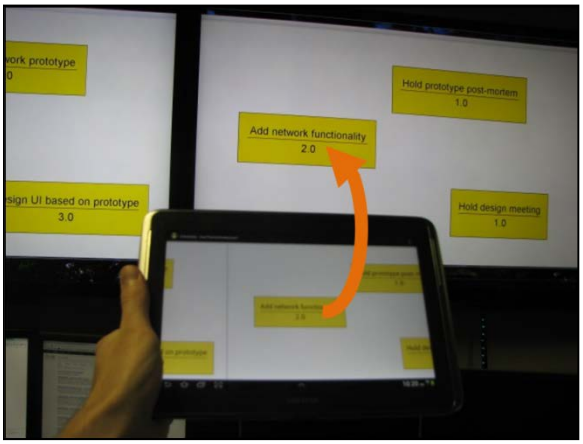


Figure 3: Users add milestones to the schedule.

The meeting begins with a brainstorming task to list out the milestones of the software project. Using their handheld devices, each member of the group enters items associated with their specialty. As the milestones are added using their handheld devices, they are instantly displayed on the large screens in the room, as shown in Figure 3.

Once the group is satisfied with the set of milestones that has been created, they move onto creating dependencies between the milestones. They begin by working individually on the sets of milestones that they each care about. Using their handheld devices, they establish links between milestones. Some of the work can be done while looking at a single large screen, but occasionally a task needs to be done across a long distance that cannot be easily performed on a user's handheld device. For example, while the project manager is looking at items related to the prototype stage of the schedule, he may think of a milestone that the schedule does not yet

include. He creates the milestone and it is placed around the prototype milestones. He needs to move this item from the prototype stage of the schedule to the completion stage of the schedule. However, these two milestones are located two different wall displays that are not located next to each other in the layout of the room, as show in Figure 4. Using our new interactive techniques, users can easily create links between two milestones regardless of where they are located and how far apart they are on the large screens.

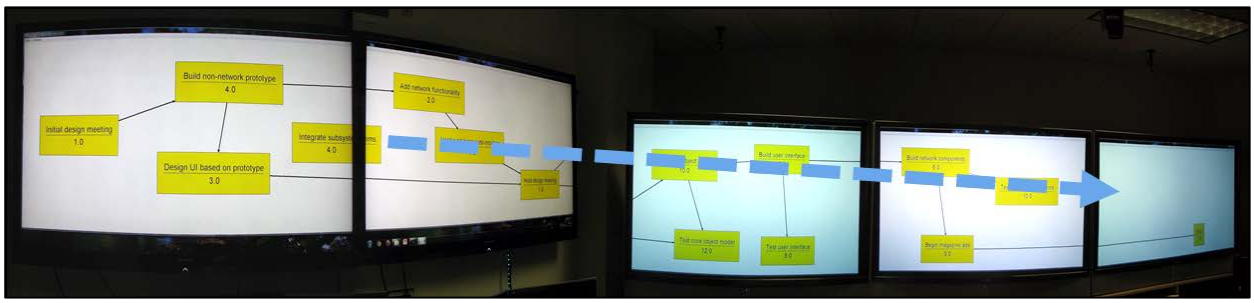


Figure 4: A milestone item that needs to be moved across several wall displays.

As the individual work continues, dependencies are created between milestones that multiple managers are interested in. When these managers notice changes occurring to the milestones that directly affect them, they discuss with each other the changes to the milestones and dependencies. During these discussions, the individuals focus on the wall displays, rather than their handheld devices.

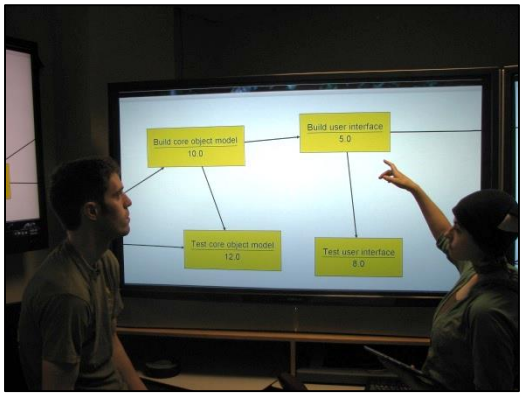


Figure 5: Users focus on the wall displays during conversations about the data.

In Figure 5, we see that focusing on the data shown on the wall displays allows the participants to use natural body language, such as pointing, to communicate their ideas to each other. Discussing data on the wall displays also allows the managers to easily see the context around the specific milestones they are talking about. Being able to see more data on the wall displays allows the conversation to be focused on the big picture of the schedule they are trying to create. Several of these conversations occur during the milestone organization phase of the collaborative meeting.

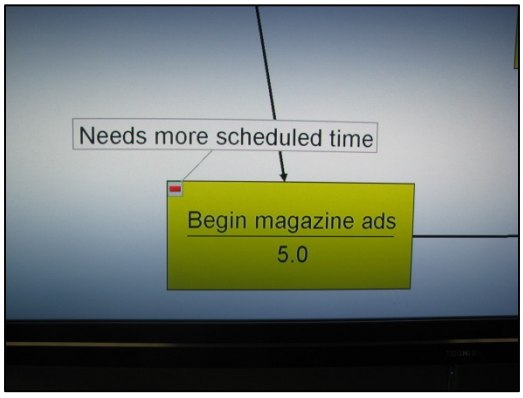


Figure 6: A note on a milestone that requests more time for the task.

The final task of the meeting is to go through and rate the work that has been done. Each member of the group walks in front of the large displays and uses their personal devices to add comments and notes to the milestones items. Some managers add notes that not enough time is scheduled for a milestone's completion, as shown in Figure 6.


Figure 7: A note on a milestone that reminds the developers to contact Dan when the milestone is completed.

Others add notes that they want to be contacted when a milestone is finished, as shown in Figure 7. When the group is done commenting on the work, they have a fully connected graph of milestones that shows the work that must be done to complete the project.

When the meeting finishes, each manager holds a meeting with their team. Using the completed schedule, the managers show their team members what aspect of the project they need to work on next.

Mood Board Example

The second sample application we built was a mood board application. A mood board is a collection of images that are combined to portray a mood or a feeling for a user interface design. For example, a good mood board for an anti-virus application would probably contain images of

locks, keys, and maybe even a security officer. Each of these pictures conveys a feeling of security and safety, and therefore a mood board that contains these pictures would convey the same feelings. These mood boards are created in group settings where each member of the group discusses what they do and do not like about how each mood applies to the current project. A small team of designers (Alice, Mallory, and Bob) has been assigned to create a new user interface. They begin their collaborative work meeting by creating individual mood boards.

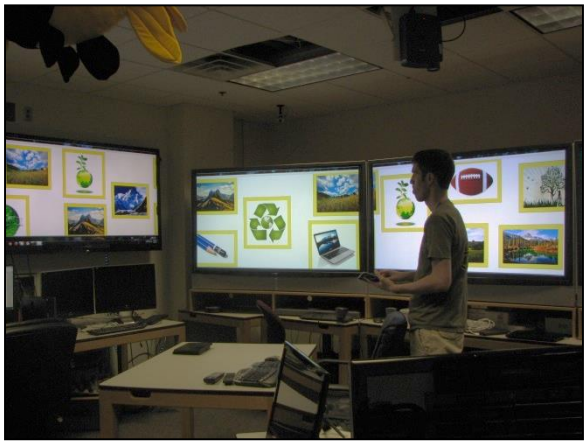

Figure 8: Displaying images on large screens.

When they arrive at the meeting room, they begin by displaying previously collected images on the large shared displays in the room, as shown in Figure 8. First, they group the images together based on the image moods. On the wall display nearest to her, Alice sees an image of a mountain range and an image of a meadow, and she decides that they are related. Using her smartphone, she drags the two images together and forms a mood group. The grouping actions are reflected on the large displays, as well as on the other designers' devices so that each member of the group can see the newly created mood. Alice decides that she will give the group a label later when it has more images.

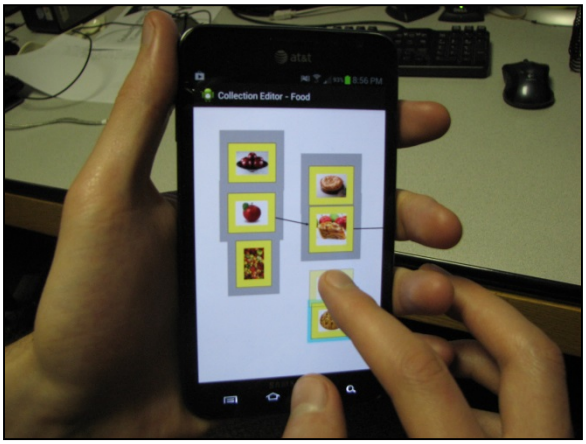


Figure 9: Editing groups on handheld device

While Alice is creating her group, Mallory thinks of the eco-friendly moods that are currently popular. She uses her tablet to group several images that have an eco-friendly mood, such as an image of the recycling symbol and a picture of the Earth. When Bob sees Alice and Mallory's new groups, he decides that they are similar, so he uses his handheld device to merge these two groups into one.

Figure 10: Bob and Alice discuss merging two mood board groups.

Both Alice and Mallory notice the merging of the groups and a discussion begins about whether those two groups should be united. After a brief period of interaction, as shown in Figure 10,

they decide that the merge was a mistake, so they undo the change and continue grouping images together. Even though the designers decided that Alice and Mallory's groups should be separate, Bob still wants to make a group that contains some of the images from both groups. Without removing the images from their current groupings, Bob creates references to the images he is interested in and forms a new mood group. With these references, as shown in Figure 11, he is able to create the group of image items that he originally wanted, but he doesn't have to interfere with the work of his fellow designers.


Figure 11: Alice's group on the left and Mallory's Eco-Friendly group on the right. In the middle, Bob has created references to items in both groups to create his own mood board.

The process of looking through the images and putting them into groups continues until all of the images are organized, as shown in Figure 11.

Figure 12: Viewing the finished image groups

They decide that they need more images to convey the types of moods they have created. After they have collected more images, they will meet again to continue organizing and to them critique each other's work. With this decision, the meeting concludes and each of the designers goes back to their individual workstations. Working as individuals at their own desks, they each can each continue to work on the shared mood board data. All of the changes that they make while at their own workstations are still distributed to each of the other designers so that they always are aware of the progress each other are making.

Although the scenarios we present use the sample applications we created, keep in mind that the interactions and techniques used within these applications are general enough that they can be used in several other applications. These other applications could be a brainstorming application that allows users to create data items from their handheld devices that appear on large wall displays. As mentioned earlier, once the ideas are on the wall displays, they can be viewed and commented on by the entire group. Another possibility is an art education application that would allow students to collect pieces of art on their own time, and then during class time they could share their collected pieces with their classmates. Once shared, the class could then find

relationships between different pieces. These relationships could be visually demonstrated using links between the pictures or by arranging the pictures into groups. Even spreadsheet-like applications could be based on our framework. Rather than being restricted to a tabular format, however, our framework can allow the relationships of spreadsheet cells to move into a more fluid and visual environment.

## Solution Requirements

The previous two example scenarios demonstrated several features that users in collaborative environments need. In order for our framework to be successful, we must meet the following needs of a collaborative application.

### Synchronization

Users of our collaborative environment work together on group projects. These group projects consist of several data items. As shown in our previous examples, during a collaborative work session, users create and modify a project's data items simultaneously. When two users are working on the same set of items within a given project, they need to see each other's data changes as soon as possible. In the mood board example, Bob merged two groups and his coworkers were immediately able to see the changes he had made. Since the system updated the wall displays and each user's handheld device immediately, the group quickly noticed the change and was able to discuss the change when it happened. Users of our collaborative environment need to see data changes as they happen so that collaboration can be seamless.

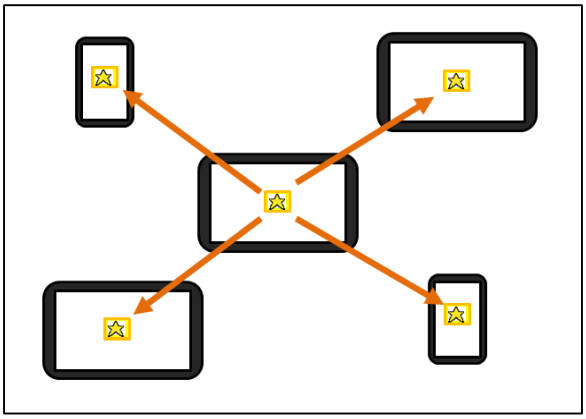

Figure 13: When an item is created or modified, the data on each user's device and on the wall displays should all be synchronized automatically.

## Data Access

Users must be able to access their project data regardless of whether they are in a collaborative work room or at their personal workstations. In order for collaborative work to help its creators, it must be persistent beyond the original time and place of its creation. In the schedule creating example, the managers worked together to create a schedule. When the meeting was over, however, each manager still needed access to the schedule in order to inform their own teams what the team needed to work on next. Similarly, in the mood board example, the three designers began the meeting by loading images into the application that they had previously collected. When the meeting was over, they continued to work on the mood board project individually in preparation for their next meeting. As shown in Figure 14, users must be able to access and modify their data regardless of whether they are in a collaborative work room or at the own personal workstation.

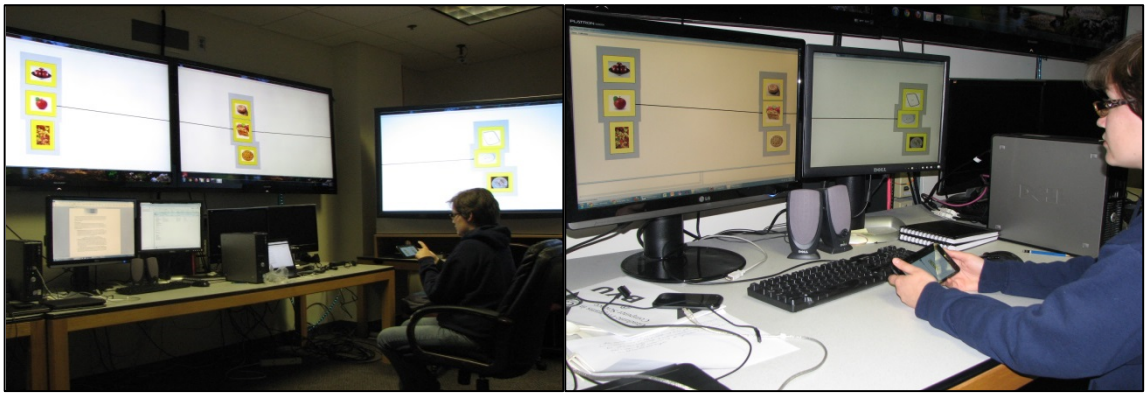

Figure 14: Data should be accessible in collaborative spaces as well as at personal workstations.

### Wall Interaction

Users must be able to interact with the entire span of the wall displays from their handheld device. Handheld devices provide users with a small personal space that they can use to interact with a large shared wall display space. Users need a way to interact with the entire wall display space, not just a small portion of it that can fit on their handheld device's screen. These types of handheld to wall display interactions must be possible, regardless of 1) the distance between the user and the wall displays, 2) how large the handheld device's screen is, and 3) how tall or short the individual user is. For example, a user who is sitting on the opposite side of the room from the shared wall displays should still be able to add and modify items from their sitting position. A user who is interacting with a project on a smartphone should be able to perform all of the interactions that a user with a tablet can perform. Finally, a user who cannot reach the top of a wall display with their hands should be able to use their handheld device to move items at the top of a wall display. Also, without needing to wait their turn at a shared display, several users can interact at the same time on data that is closely spaced together. This type of interaction can occur because can modify data from anywhere in the room.

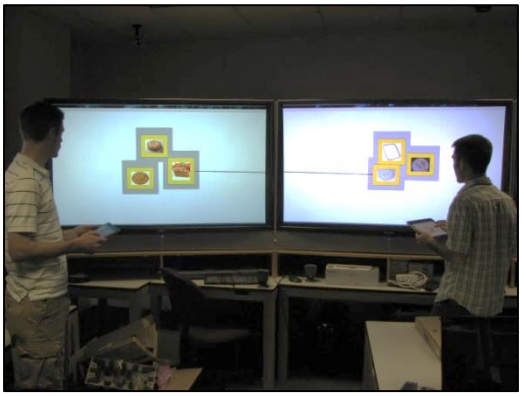

Figure 15: Two users modifying data items independently on two different wall displays.

## Several Simultaneous Users

In both the schedule and mood board examples, the collaborative work contained periods of individual work sessions. Whether it is forming mood board image groups or creating milestones for a schedule, this individual work is important because it creates data that the group can later work on together. These types of collaborative applications must allow users to create items in their own space where they will not interfere with others' work, as shown in Figure 15.

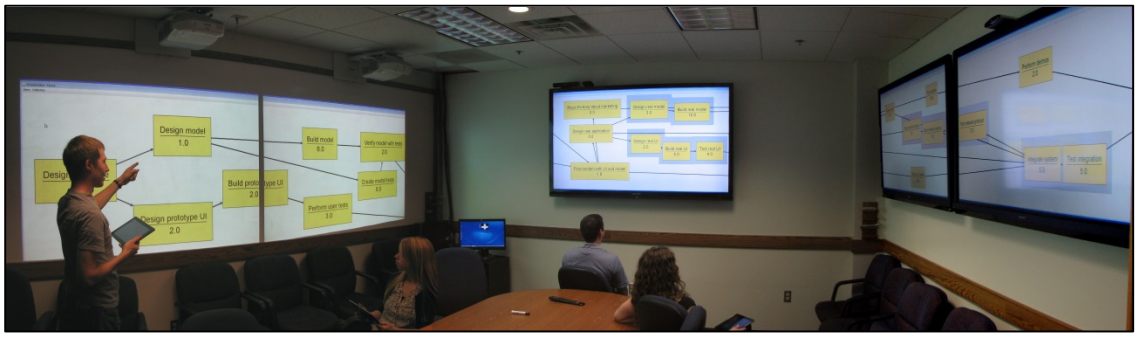


Figure 16: Multiple users on the same project at the same time.

In the collaborative environments that we have described, multiple users must be able to work together on the same project at the same time. Having one active user while several other uses

wait for a turn to interact with the system creates an interruption that inhibits creative and collaborative flow. A successful collaborative system must allow several users to be actively creating and modifying items at the same time as other users.

### Common Actions

As we brainstormed different types of applications that this type of system could create, we noticed a common set of actions that each of the applications needed to be able to perform. In the mood board application, users needed to

- create items,
- move items,
- group items,
- create links between items,
- create references of items,
- create copies of items, and
- add notes to items.

In the schedule application, we found that users needed to be able to perform these same set of actions on the items of the schedule. For our system to be successful, it needed to provide these common data actions in a way that other developers could easily extend and utilize in their own applications. Providing these common actions in a framework frees developers from creating their own instances of these familiar actions. Instead of developing these actions, developers can now focus on what the actions mean in the context of their own applications. For example, instead of making a system for creating links between art image items, a UML diagram

application developer can focus on using links to show classes that could be impacted from a change made to a specific class.

Chapter 2. Prior Collaborative Systems

The body of collaborative systems that relate to our research can be broken down into three sub-groups:

- Shared displays with multiple input devices
- Individual devices with no shared displays
- Shared displays with additional individual displays and devices

Shared Displays with Multiple Input Devices

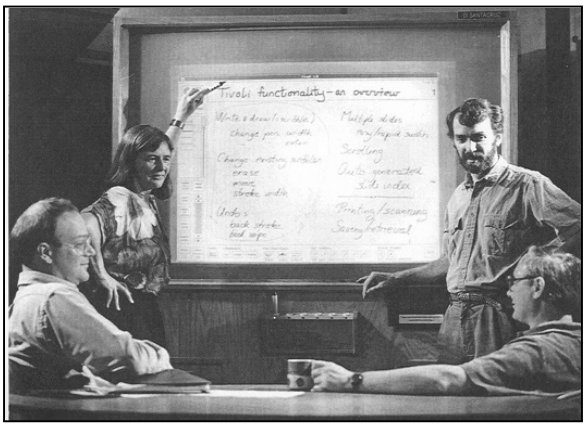


Figure 17: Tivoli uses pen inputs on a shared wall display.

Figure 18: Dynamo uses keyboards and mice to interaction on shared wall displays.

This first group of collaborative systems focuses on using one central computer and providing an input device for each user. As demonstrated in Figures 17 and 18, users typically interact with this system by using either a digital pen or finger [4,6,14], or with a keyboard and mouse [3,7,8,9,12]. Even though using keyboards and mice is more common, we have found that systems that allow pen or finger input generally just treat the touch input as a mouse cursor. This translation from touch events to cursor events means that the underlying systems remain similar, in spite of what type of input device users manipulate to interact with the shared data. Because the underlying systems are the same regardless of what type of input device the users are interacting with, we consider these systems to belong to the same group.

These systems easily meet our synchronization requirement because each of them only uses a single shared set of displays. When a user changes a data item, the data item change is immediately visible to the other users because only a single computer is used in the system.

None of these systems allowed users to access data after the combined work session was over. This missing feature means that none of these systems meet our data access requirement. Dynamo [7], TeamWorkStation [12], and Tivoli [14] all had ways of saving data to a personal data storage after a collaborative session was finished. However, these methods of saving data were not synchronized once the work session was over. If changes were made afterwards, the data would no longer synchronized, which means that each member of the group could potentially have their own modified version of the project. This leads to undesirable project file merges and potential conflict.

These systems meet our wall interaction requirement. When users used a keyboard and a mouse [3,7,8,9,12], they are easily able to move the cursor to anywhere on the shared display to edit

data in the current project. In the systems that used a user's finger or pen input [4,6,14], the displays were always small enough that a user could easily reach the entire workspace. It is important to note, however, that finger and pen-input systems do not scale to larger sized displays, particularly for users who are shorter, as shown in Figure 19. As shared touch-sensitive displays get bigger, there will be regions where users are unable to reach.

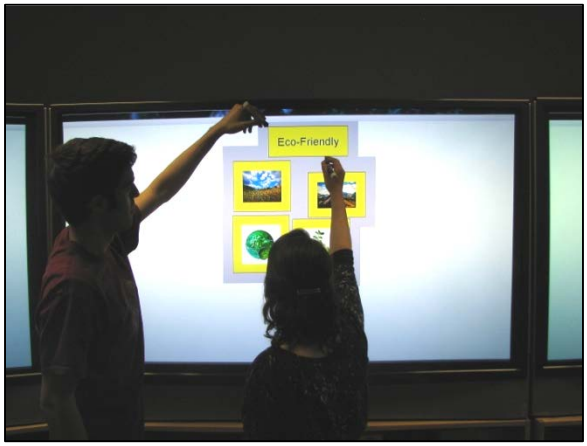


Figure 19: Reaching the top or bottom of a shared display with a finger/pen input can be difficult for users.

All of these systems also struggle to support multiple users. Each system has the software capabilities to support several simultaneous users, but their physical implementations generally do not scale to support more than a handful of users. For example, in Figure 20 we see Dynamo being used during its user study. The creators of Dynamo only provided two sets of keyboards and mice to their user study participants because of space constraints [21]. They did not provide more keyboards and mice because the user study room only had space for two sets of keyboards and mice. Each of the other systems in this group have a similar spatial problem when they attempt to support several simultaneous users. Although these systems do support multiple users, they do not scale to support more than a handful of users at one time.

Figure 20: Dynamo was limited to the number of users who could use a keyboard and mouse in front of the wall displays.

Figure 21: Firestorm was limited to the number of users who could fit around the tabletop display.

Finally, none of the systems in this group provided any type of common actions that future developers could use in their own applications. Although the systems allowed users to perform some of the required interactive techniques (Dynamo allowed users to add notes and Firestorm allowed users to group items together), they did not provide of all of the interactive techniques we have previously described. Also, these systems do not provide a clear framework that future developers can use to create their own applications. Rather, each of the systems in this group are standalone applications.

Individual Devices with No Shared Displays

The second group of related work provided users with their own individual devices, but provided no shared displays the users could collaborate with [1,6,20]. These systems are much less common than the previous group that used individual input devices with a set of shared displays.

Figure 22: Multiple users collaborating through wirelessly connected handheld devices.

Each of the systems in this group synchronized data between the users' devices. In Zurita's [20] and Alvarez's [1] work, users participated in educational games that synchronized the current game data and state between the user's devices. An example of collaborating users can be seen in Figure 22. In ClearBoard [6], users were positioned in front one of two tabletop digital drawing surfaces that shared data by using a camera and projector pair. As one user interacted with a tabletop display, a camera would capture that user's display and interactions. This video stream would then be sent to the second user's projector and would be overlaid with the second's users interactive display. In these three systems, the data that collaborative users interacted with was synchronized automatically without any user input necessary.

Although these systems do well at synchronizing user data at run-time, none of them mention any type of collaborative data access after a collaborative work session is completed. Once the

work session was complete, the data was ignored or thrown away. This way of handling the shared project data does not satisfy our users' need to have persistent data that they can access across multiple work sessions.

These systems do not have shared displays, so our wall interaction requirement does not apply to this group. It could be argued that these systems allow users to access all of the data that is being used in the current project through their handheld devices, but that data usually consisted only of what could fit on the display of the handheld devices. These systems do not scale well to larger projects that could potentially fill several wall displays. For this reason, we still consider this group of related work to not satisfy our need to be able to interact with the entire span of the available wall displays.

Figure 23: The ClearBoard system in use. This system only allows two users to collaborate together.

These systems do meet our simultaneous user requirement. As shown in Figure 23, ClearBoard [6] was specifically designed for two users to work together on a drawing, however, ClearBoard does limit the number of simultaneous users to two. This decision was made because they were interested in creating "gaze-aware" collaboration that enabled the users to see where each other were looking. In Zurita [20] and Alvarez's [1] work, more than two users could be supported, but

without shared displays their ability to communicate with each other would be limited to what could be visualized on small handheld devices. The system was designed to handle small educational games, but if the data were extended to larger scales, such as maps or databases, discussing the data would become problematic without large shared displays. For this reason, we feel that this work does not scale well with larger groups.

Finally, none of the systems in this group provide a common set of interactions that future developers can build with. In fact, none of the systems in this group provided any of the required interactions we described earlier. The types of applications we need users to interact with require individual data items that can be created, organized, and critiqued. However, the applications in this group do not organize their application data in terms of data items. For this reason, many of the data-item-centered interactions we previously discussed would not even be possible in the contexts of these applications.

Shared Displays with Individual Devices

The final group of collaborative systems contains systems that use shared displays but still provide each member of the group with their own device. Unlike the devices in the first group of collaborative systems, these devices provide individual displays for users to see the data in the current collaborative project. They also provide input tools that users can use to modify the group data. Of the related work groups that we have previously talked about, these systems are the most similar to the type of system that we have described in our example scenarios.

This group can be separated based on how the shared data is displayed on the individual user devices. The first version of Cognoter [5] and Argnoter [16] employed a WYSIWIS paradigm (What You See Is What I See). In this type of visual synchronization, each member of the group

sees the exact same view on their individual display. For example, when a user begins using their workstation in a WYSIWIS environment, the user's cursor as well as the user's data changes would be visible on the shared wall displays as well as the displays of each of the other members of the group. The second version of Cognoter [15] and i-LAND [17] used a WYSIWIMS paradigm (What You See Is What I May See). In this paradigm, each user could organize their display however they liked without interfering with other users' displays. As a user changed data, the modifications were broadcast to the other users in the group, but would only update the display if the user was viewing the modified data.

Figure 24: Shared display with multiple identical workstations.

All of systems in this group meet our data synchronization requirement. These systems synchronize the current project's data in the background while users are making modifications. As data modifications are received, they are displayed on each users' workstation quickly and without any user effort. This helps to maintain the feeling that each user is participating on the same project and are helping the group to accomplish something larger than the users could do alone.

None of these systems provided any mechanisms for data to be collaboratively accessed after a work session has been completed. In Cognoter [5] and Argnoter [16], the group's data was saved to a database so that it could be synchronized among the users' workstations, however, we were never told whether users could access that data later on. We believe that this type of persistent storage would be possible on these types of systems since they already have multiple independent computers accessing data remotely. However, these systems never mention whether the data can be accessed later, which does not meet our data access requirement.

Both the WYSIWIS and WYSIWIMS systems meet our wall interaction requirement, because users could access all of the data displayed on the shared displays, regardless of where they were in the room. This was possible because each of them were using a keyboard and a mouse for input, as was the case in Dynamo [7] and MMM [3].

These systems also meet our need to support several simultaneous users. Each user in these systems has their own workstation that they can use to modify the shared project data. The group also has their project data displayed on a set of large wall displays. This setup allows users to collaborate and discuss the project on the large wall displays, but they can use their personal workstation to create and modify data. This type of data synchronization between large shared displays and individual workstations allows users to be positioned in different places throughout a collaborative room, unlike in Dynamo where users had to be position in front of the shared display. Because there are less limitations on where users can be positioned in a room, these types of systems can support more simultaneous users than any of the other groups we have discussed previously.

None of these projects provide a set of common interactions that future applications can use to modify data items. These groups come the closest to our needs because many of them do interact with data on item-level. By this we mean that they allow data items to be created, moved, given notes, grouped, and linked together. This is more than any of the previous systems allowed. However, these actions are not exposed in a framework that other developers can build on to create new applications for different contexts.

### The Pebbles Project

The Pebbles project [10,11] fits into this group but deserves special mention because it uses handheld devices to collaborate on a shared screen. Using specialized software that is installed on both a PalmPilot device and a shared computer, multiple users can write notes on top of PowerPoint slides. Using another program through this system allows multiple users to work in a collaborative painting program. All of the users provide input through their individual PalmPilot devices, and those inputs are then transmitted to the shared workstation, as shown in Figure 25.

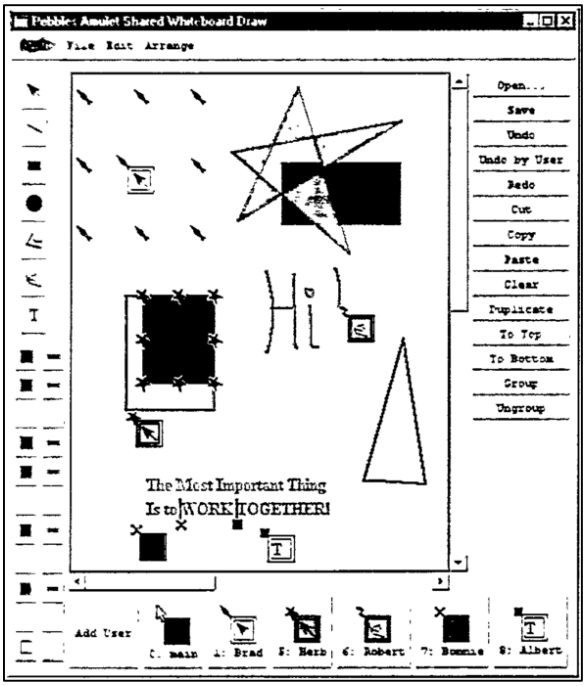


Figure 25: View of the shared display used in Pebbles' collaborative painting application.

Although this work closely resembles ours, the main difference is the amount of responsibility given to the handheld device. In our desired system, a user's personal device displays the current project as well as allows the user to create and modify data. In the Pebbles Project, a user's handheld device was only used as an input device. Users could annotate presentation slides or collaborate on a shared drawing. However, since the Pebbles Project did not store data in data item format, they could not provide the common framework of interactions that our system needs to provide.

From this set of previous work, nothing has been explored pertaining to using small, personal devices in the context of collaborative work on large shared screens that meets all of our requirements. Particularly, we have not found any work done on the interactive techniques that

are required for these handheld device users to interact collaboratively with each other on these large displays.

## Chapter 3. Framework Architecture

With no solution existing to meet our needs, we created a system that fulfills each of our requirements for success. We chose to create a framework instead of creating a specific application. We created a framework so that future researchers could easily continue where we left off. If we had created an application, it would be more difficult for future work to explore different avenues because they would have to recreate much of what we had already done. Our framework consists of several generic components, such as our common interactions, that can be used together to create several different types of applications that were not previously possible. This framework is the application "glue" that connects handheld devices to shared wall displays. Our framework architecture consists of three components: a cloud data store, shared wall displays, and individual handheld devices.

### Cloud Data Store

The first component in our architecture is the cloud data store. The cloud data store is responsible for storing all of the data that will be shared across multiple devices in our framework. Each data item consists of a set of descriptive tags and some type of data content. These tags could be system-defined (such as which user created it and when it was created), or they could be more application specific (such as the latitude and longitude of a point of interest in a map application). The data content could be any series of bytes, such as the contents of an image, the text of a paragraph, or even an xml document describing a milestone item. Each of an item's tags and an item's content needs to be stored in the cloud data store so that many simultaneous users can access it at the same time.

Each item has a unique id associated with it that applications can use to request item tags or content from the data store. When an item's tags or content are changed, a new version of the item is saved in the data store that can be referenced based on the timestamp of when the change occurred. Once an item version is created in the data store, it cannot be changed. The only way to change an item is to create a new version. Implementing item data changes in this way greatly simplified synchronization issues. When an item's tags are requested from the data store, the tags of the most recent version of the item will be returned. When two requests that modify the same item tag arrive in the data store at the same time, the requests are placed in a queue and processed in order. In this way, the next time the item tag is requested, the value that is returned is the value stored in the most recently processed data change.

The cloud data store is also responsible for sending change notifications to devices that have expressed an interest in a particular set of items. For example, if two users are working on a set of moodboards, the devices that the designers are using should only receive change notifications for the items that are contained in their mood board project. If a set of accountants are working on a collaborative spreadsheet application at the same time, the accountants' devices should not receive the change notifications for the mood board project. Rather, they should be receiving change notifications for the spreadsheet project that they are currently working on. The cloud data store needs to send notifications when items are modified, and these notifications should only be sent to devices that have expressed an interest in those specific modified items.

To implement the cloud data store, we created a web server based on the Apache Tomcat web server technology. By building our data store on top of the Apache web server, we were able to

focus on managing data items and sending out notifications, instead of creating networking protocols and handling incoming requests in an efficient manner.

Incoming requests to our cloud data store, such as creating a new item or retrieving the tags for a specific item, were transmitted using HTTP requests. Embedded in these requests was all of the necessary data that the cloud data store needed to be able to process the request. For example, if a user were to request the content for a particular image data item, the user's device would send a item content request to the cloud data store with the item's id as a parameter in the requested URL. If a user were sending item modifications to the data store, such as setting or delete item tag values, those modifications would be encoded in XML format in the body of a POST request. This request would then be read by the cloud data store where the encoded changes would be applied.

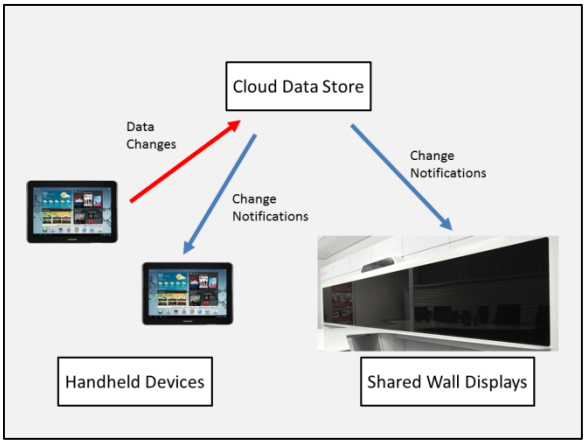


Figure 26: Solution architecture depicting item change notifications to interested devices.

As data change requests are sent to the cloud data store, devices that are interested in the changing items need to be notified of the item changes, as shown in Figure 26. To receive change notifications, devices first must indicate to the cloud data store which items they are

interested in. They do this by providing a tag value filter that describes the items the device is interested in. For example, consider a map application that has hundreds of user defined points of interest spread across the globe. However, a particular user has navigated his device to point only to a small portion of the United States. Instead of receiving change notifications for all of the points of interest across the globe, the device's application can add a listener to the cloud data store using the longitude and latitude coordinates that describe where the user's screen is currently looking. Now the device will only receive change notifications when an item is changed that contains a longitude and latitude coordinate that is currently visible to the user's device. If the user changes the area that they are viewing on their handheld device, the device will remove the previous listening filter and will add a new listening filter when the user settles on a new part of the map.

Since the cloud data store is accessible through a web server interface, we can also perform all of our data requests through the network from any location. This accessibility allows us to access our data in collaborative work space rooms as well as at our personal workstations. This also means that when a meeting has concluded, users can continue receiving data change notifications for items that they are still interested in. This type of remote access functionality was not provided by any of the previous work that we discussed earlier.

Handheld Devices

Users create and modify project data by using handheld devices. Common types of handheld devices that may be used with our system are depicted in Figure 27. The portion of our framework that runs on these devices allows users to provide input into the current project. These inputs can consist of creating new data items in the cloud data store, modifying existing data

items, or creating relationships between data items. Gestures and/or systems to create these data changes are provided in our framework in a generic way so that future developers can easily add these interactions to their own applications.

Figure 27: A collection of handheld devices users may use with our system.

Applications that extend our framework display their data in a two-dimensional virtual work space. Users can use pinch-and-zoom and two-finger-scroll gestures to zoom and pan around the project that they are currently working on. These basic navigation gestures are provided in the handheld device section of our framework. By navigating around the project on her handheld device, a user can successfully access and modify any part of the shared wall displays, regardless of where she is currently located in the room. This is a similar benefit that we saw earlier with a keyboard and a mouse.

The handheld device portion of our framework also provides future developers with the tools necessary to retrieve data item tags and contents from the cloud data store. For example, when an application connects to a schedule project, the handheld device needs to know which data items belong to the specified project. First, the handheld device sends a request to the cloud data store

to discover which items are contained by the current project. The cloud data store then returns a list of data item identifiers that the current project contains. The handheld device is then responsible for requesting the item tags and content for each of these items from the cloud data store. The handheld device is responsible for drawing the incoming data items so that the user can see the contents of the project. Ideally these data items will be drawn as they are received, so as to give the user a sense of progress as the project is loaded.

### Shared Wall Displays

The final component of our framework is the shared wall displays. These shared displays are treated architecturally very similar to how the framework treats handheld devices. For example, when an application that extends our framework runs on a set of wall displays, the wall displays must have a project provided that it can connect to. Once connected to a project, the wall displays then have to request the data items that belong to the project from the cloud data store. As these data items are received from the data store, the wall displays then have to display these data items so that users can collaborate using the wall displays in conjunction with their handheld devices. The wall displays also must register a data item filter with the cloud data store so that they can receive change notifications when items are changed within the current project. When these change notifications are received, the wall displays need to update themselves to show the most recent versions of the data items.

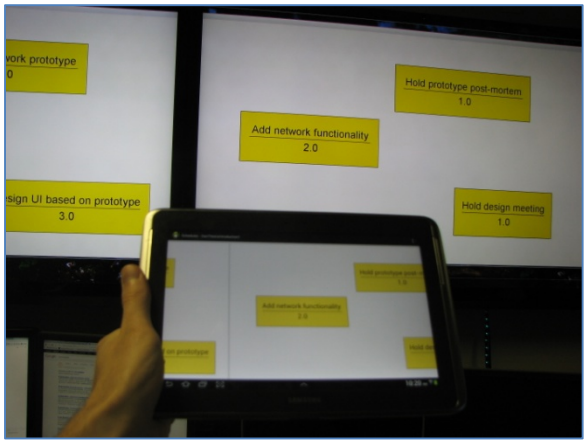


Figure 28: Both handheld devices and wall displays draw data items identically using a shared portion of our framework.

Because there is so much overlap in our framework between the applications that run handheld device applications and the applications that run on the shared wall displays, we combined much of our code into one platform-independent section. As a consequence, future developers can easily implement their application display and data item modification logic in one place, but be able to deploy this logic to both handheld devices and wall displays, as can be seen in Figure 28.

One area where our wall displays differ from our handheld devices is that the wall displays provide no way for a user to modify items. In our current framework, the wall displays are used as a view-only system. Future researchers could change this by adding novel ways to interact with large wall displays that don't involve handheld devices. However, in this work the shared wall displays allow users to gain a quick understanding of where items are in the current project.

Primary Challenges

As we created the framework that enabled collaborative applications to work on handheld devices and shared wall displays, we recognized that there would be several challenges that we

would need to focus on. These particular issues were problems that needed to be solved in order for our solution to even be possible.

The first of these challenges was efficiently synchronizing data among all of the devices in a collaborative work space. When users create data for a project, the data gets sent to the cloud data store. Whenever a device needs to retrieve data about the created item, it must go to the data store to get the information. Forcing the data to be in one location makes it much easier for the system to keep all of the data synchronized but it also can become a huge delay in the system. For example, consider the designers working on a mood board project. Each time a designer navigated to a different position in the project, the application would need to redraw the images at that new location. To redraw the images, the application needs to retrieve their pixel data. If the pixel data for the images is not retrieved and loaded quickly into the application, then users will experience an extremely slow application that will not allow them to make acceptable progress. In order for the collaboration between our users to be fast enough that they can get work done, our system must be able to synchronize and distribute data about the project's items quickly.

The second challenge is allowing handheld devices and wall displays to work together on a single project. As mentioned in the mood board example from earlier, sometimes users begin a collaborative work session by adding several data items to the shared wall displays. Once the items are on the wall displays, each user will walk in front of the wall displays and look for images that are interesting to her. If a user finds a data item that she is interested in, she needs a way to quickly navigate her handheld device to that data item that she found visually on the wall. Once she has navigated her handheld device to the interesting item, she now needs to move the

new item to a specific location on a different wall display where she will create a new mood board. How does she know where to drag the item on her handheld device so that it appears in the wall display location where she will create her mood board? Similarly, several users may be discussing a certain set of data around a specific wall display when one of the participants notices an item on another wall display that pertains to the current conversation. How can that user know where to move the additional item on their handheld device so as to make it appear on the wall display area that the conversation is currently focused around?

The final challenge to our system is that by creating a framework, we were building a system that other applications can build on top of. This means that we needed to provide several interactive techniques that were not only simple to learn and use, but also were generic enough that they could be used across several different applications. The needs of a mood board creating application are slightly different than the needs of a schedule creating application. For example, the mood board application needs to be able to group items and create references of items so that a single item can appear in several different mood boards. In the schedule application users do not need to create multiple copies of a single milestone, but they do need to create links between milestones. In this challenge, we recognized that although there are differences between these two applications, there were a set of core interactions that were common that we could build into our framework. This final challenge was to create those core interactions in such a way that developers could quickly extend them to meet the needs of their specific applications.

## Chapter 4. Efficient Data Synchronization

In creating our framework, we recognized that applications that used our system would be dealing with several data items at one time across several displays. These data on these displays must be synchronized so as to not confuse the user with two different unsynchronized versions of the same data set. For example, if a user is trying to group two images to create a new mood board, it can be distracting if the user sees the group form on his handheld device, but has to wait for the group to be displayed on the wall display in front of him. This problem only gets worse if two users are working in the same area and both attempt to interact with the same item. If the data item's position and attributes are synchronized efficiently and quickly, then the two users will see quickly that they are both using the same data item. This will spur a conversation between the two users, which could result in the users creating a reference of the item so that they can both interact with the same image in two different mood boards.

There are two different situations where data needs to be synchronized in our system. The first situation is when a user loads a project on her handheld device. In this type of scenario, a user walks into a collaborative room where other users are already interacting with data that is displayed on shared wall displays and their handheld devices. The new user starts up her handheld device, and then directs the application to connect to the same project as the other group members in the room. At this point, her handheld device will need to request all of the data associated with the given project's items, which consists of item tags and item content. She will be unable to participate with the group until her handheld device has retrieved the project's data.

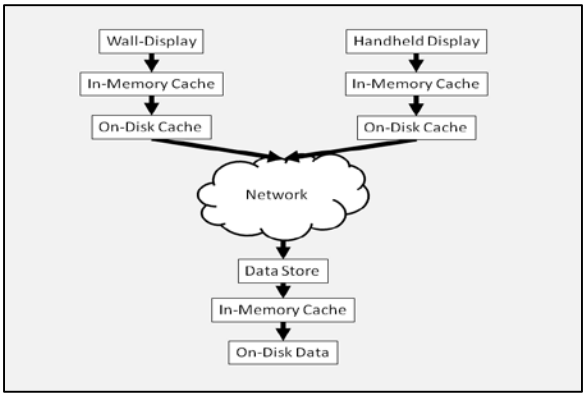


Figure 29: Several layers of caching to improve data access performance.

In order to make loading a project as quick as possible, we introduced several layers of caching into our data retrieval process. As can be seen in Figure 29, we introduce two layers of caching on handheld devices and shared wall displays. We also introduce a single layer of caching on our cloud data store. Each layer of caching is designed to reduce the amount of work that must be performed to retrieve a piece of data. For example, when the cloud data store saves an item's tags in it's in-memory cache, the next time the item's tags are requested the data store will not have to parse the tags from its disk. Instead, it can access the tags directly from memory. Similarly, an image item's pixel content is stored on a shared wall-display's on-disk cache, the wall display will no longer have to request the image's content from the data store. Instead, it will have that image in its permanent storage whenever it loads the image.

The wall-display and handheld device on-disk caches really create the biggest improvement in project load times after the project has previously been loaded. Once this occurs, all of the items' tags and content will be in the device's permanent storage. This means that the next time that the project is loaded, the only requests that the devices will need to make to the data store will be for new changes that have occurred to the project's data items since the last time the device

connected to the project. This decrease in network requests greatly improves our project load times. There is the chance that a user will access a vast amount of data items to the point that they need to replace older data items with new ones. This situation is unlikely since our initial on-disk cache size is several gigabytes and a single item occupies between one kilobyte (text items) and a few megabytes (image items). However, if this did happen and the user then needed the replaced data item, the device would just request the data item from the store again.

The second situation in which data needs to be synchronized across several devices is when data changes occur. For example, if a user moves an item to a different location in the project, each of the wall displays and handheld devices need to instantly be notified of the item movement so that they can update their displays. To facilitate these updates, we put in place a system of notifications that allows the cloud data store to notify handheld devices and wall displays when data items change. When an application on a handheld device changes an item based on a user interaction, that data change is sent to the cloud data store. When the data store receives the change, it stores the change in its permanent storage, updates its cache, and then checks which registered devices need to be notified of the change.

As mentioned in our Framework Architecture section, applications can register a tag filter that describes a set of items that the application is interested in. For example, perhaps the device is running a schedule application and the application only needs to update itself when an item that is currently visible on its display changes. When other devices send in data changes for items that the registered device cannot see, these changes will be stored in the cloud data store, but the registered device will not clog up the network receiving data changes that it does not need to see. The device can request them later when it navigates to a different portion of the project and then

can see these changed items. However, if items are changed that the registered device can see, then the cloud data store will create an item change notification and will send it to the registered device. This change notification contains the item's id, the previous timestamp that the item was modified at, and the current timestamp for the item's most recent modification. When the user's handheld device receives this change notification, it first compares the modification's timestamp against the most recent timestamp that the device has for the changed item. It is possible that a network delay may have caused one version of an item to reach a device after a second version has arrived. In this case, the handheld device should reject the older version of the item, even though it received the older version's change notification after it received the newer version's notification. After the device has verified that the changed timestamp is more recent than the item's current timestamp, the device will then request the data changes from the cloud data store.

One limitation of the way that we synchronized data between devices is that we are unable to show items in the process of being changed. For example, when a user begins moving an item on her handheld device, she can see the item moving as her finger drags it across the screen. However, her handheld device will not send the data change to the cloud data store until she releases her finger from the item. Once she releases her finger, the device will send the change to the cloud data store and then all of the other devices in the project will update the position of the moved item. In an attempt to solve this situation, we considered sending data change notifications whenever the item was moved at all. However, we were concerned about flooding the network with data changes and reducing the overall performance of the system. We were also concerned about flooding the cloud data store with several additional item versions as an item is dragged around the screen. In the framework architecture section we mentioned how we store

every version of a data item in the cloud data store. As "in-process" changes are broadcast to other devices through the cloud data store, these changes would also need to create new versions of the moved data item in the store. This might result in a hundred or more new versions of an item being created just because a user dragged the item across their handheld device. We decided that flooding the network and the data store with the item changes was not worth having real-time item changes broadcast across every device in the project.

Chapter 5. Using Handhelds Devices and Wall Displays Together

Using wall displays together with handheld devices allows users to have more meaningful collaborative discussions. The wall displays also allow users to quickly understand the larger context of the data set that they are currently working in. These benefits, however, do not happen without careful communication between handheld devices and wall displays. As discussed earlier, if a user finds a data item on the wall that they are interested in working on, how can she quickly navigate her handheld device to that item of interest? Similarly, if a user is working on a specific data item and wants extra context information around that data item, how can he know where his data item is on the wall while still looking at his handheld device? We will discuss each of these three problems one at a time.

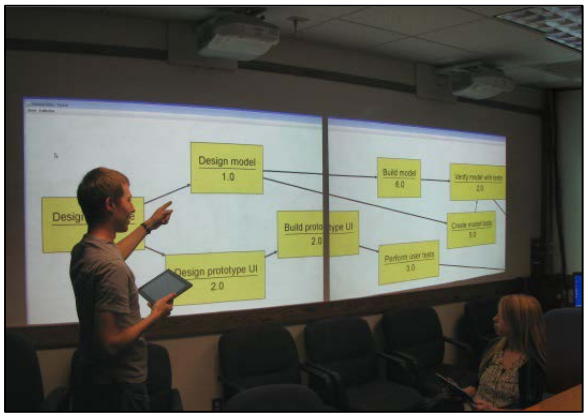


Figure 30: Users need to be able to navigate their handheld devices to items that they find on shared-wall displays.

The first problem in this section is navigating a handheld device to a specific position on the set of wall displays, as shown in Figure 30. For example, in the mood board example that we discussed earlier, the designers first looked at the wall displays to see the different items that they could use to create mood boards. As they found items on the wall that they were interested in, they navigated their handheld devices to those items and then grouped them together to form

mood boards. The designers knew where the items were in reference to the wall displays, such as finding a particular image on the left-most wall display. To navigate their handheld devices to the desired items, the designers need a way to reference where their handheld devices are in relation to what the wall displays are showing. To allow users to perform these types of navigations, we created a navigation interactive technique called screen jumping. When screen jumping is initiated on a handheld device, users are presented with a series of rectangles that resemble the alignment of the wall displays. First, the user finds which rectangle on their screen corresponds to the wall display where their item of interest is located. Then, they click in that rectangle in the approximate region where the item is in relation to the wall display. The handheld device is then navigated directly to the position where the item is located. In the mood board example, if a user found an image on a wall display that his handheld device was not currently navigated to, rather than performing several pan-and-zoom techniques, he could just perform a screen jump operation. This way, the user could navigate to his desired destination with a single interaction. Using this navigation technique, users can quickly navigate a large project to specific items that they need to work with.

Figure 31: An example of a screen jump view. At this point the user can click on any of the white rectangles that represent wall displays and the users handheld will be jumped to that display.

In order to accomplish the screen jumping navigation, the wall displays needed to share information about their size and alignment to each of the handhelds in the collaborating group. We used our cloud data store to distribute the wall displays alignment and size. When a new project is created in the cloud data store, the data store creates a new data item to store data about the created project. This data item is used to store information about the project, such as the project's name and a tag filter than can be used to determine which data items belong to the project. We chose to use this project data item as a way to broadcast system related changes to all of the users that are connected to the project. For example, when a set of wall displays connect to a project, the sizes and positions of the wall displays are saved as tags on the project data item. When the wall displays are disconnected from the project, they remove their sizes and dimensions from the project's tags. With wall display information being stored on the project's data item, handheld devices can now register with the data store to be notified when changes occur to the project data item that they are connected to. When the wall displays modify a project

data item's tags with their dimensions, each connected handheld device will be notified of the new set of wall displays that are available to the current project.

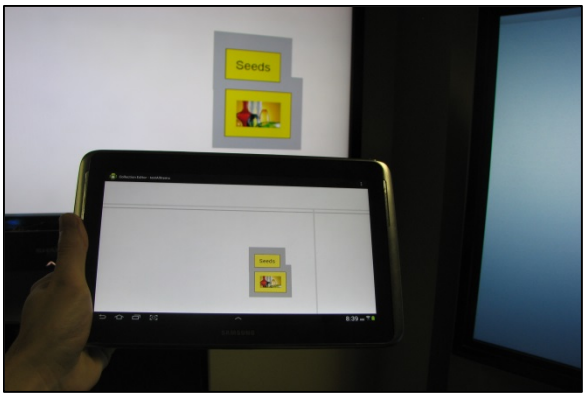


Figure 32: Wall display borders being displayed on a handheld device.

We noticed that when users were discussing information on a wall display, they had a tendency to say things such as, “Move that image over to the right side of this wall display.” Although the instructions given are very clear (move a data item to a specific location), performing this action on a handheld device can be difficult without any wall display information on the user’s handheld device. Typically, the user would move the data item to a position on her handheld device where she thought the wall display was. She would then look up at the wall displays to see how close she got the item to the right-side of the wall display. If she wasn’t close enough, then she would move the item again, and check the wall display again. This process would continue until she placed the item in the desired position on the wall display. This trial-and-error attempt to place a data item relative to a wall display's border is unacceptable.

When wall displays put their dimensions into a project data item's tags, these dimensions can then be used to show users where wall displays are relative to where the user's handheld device is currently navigated. By using this information about the shared wall displays, handheld devices

can display an outline of the displays on the users' handheld devices in relation to the data items of the current project. As users navigate around the project, they are able to see light-gray border lines where borders of wall displays lie. In Figure 32, we can see the gray lines on the handheld device that represent the borders of the wall displays the user is collaborating on. Adding these gray border lines allows users to move items to specific locations relative to the borders of wall displays with no trial and error necessary. With this added visual information, users can quickly use both the wall displays and handheld devices to work together to arrange data items.

We evaluated our success on this challenge by conducting a user study that required users to move items relative to the connected wall displays. We discuss the user study tasks as well as our results later in our evaluation section.

## Chapter 6. Generic Interactions

From the related systems we read about [4,5,15,19], we found that many of the applications that utilized our framework would have a similar work-flow. Users generally begin a new project by creating data items. Once the data items were created, users then would create relationships between the items. These relationships could be grouping the items together or creating links between items. After items were properly organized, the users then would place notes and comments on the items. Recognizing that these were common interactions that most, if not all, of the applications that built on our framework would use, we decided it would be valuable to application developers if our framework provided these interactions.

By providing these interactions to developers, our hope was that we would improve the expressive leverage [13] of our framework. Building on top of our framework would enable developers to not need to worry about how to create an item or a link between two items. Instead, these developers could focus on what the new item or link means in the context of their application. For example, perhaps they want to create a non-tabular spreadsheet application. To allow one cell's value to flow into another cell's formula, the developer could simply have his application wait for the creation of a link. Once the link was created, the application could update its data and cell-references. The developer would not need to know how the link was created in the cloud data store. They can simply focus on reacting to these types of interactions that our framework provides them. Limiting the focus of the developer also means that the amount of source code that the developer needs to write will be smaller and will take less time to write.

In discussing the interactions that our framework provides, we will follow the work-flow that users employ to create a project. We will begin with creating new items.

Item Creation

When a new project is created, users begin by creating new items. From the findings of prior work [4,19], we found that users generally begin a new project by creating several new items instead of just one or two. This led us to believe that the interaction for creating new items would need to allow for users to create several items in succession with as little interruption in the creation flow as possible. We also found that although there was a large surge of item creation at the beginning, items were occasionally created throughout the remainder of the collaborative experience. At first the fact that users created items throughout the project's lifetime seemed as though it would have no impact on the interactive technique that a user would perform to create a new item. However, with further consideration we realized that this did impact the interactive technique. In the mood board example from earlier, the designers began their meeting by adding several images to the project. This fits in line with our current understanding of the beginning of a collaborative meeting. Let's imagine that during this first meeting a user finds a new image that he wants to add to the mood board he is currently working on. If his mood board fills the entire display of his handheld device, what interactive technique should he perform to create a new item?

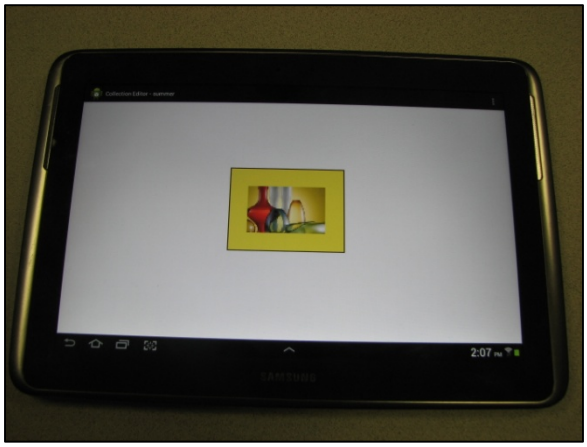

Figure 33: A newly created image item that is position in the center of the device's screen.

Originally we felt that a long-press interactive technique would be useful for creating new items. A long-press interactive technique is when a user presses and holds her finger against the screen for a certain period of time. When she releases her finger, the application receives a long-press event. By long-pressing on any point in the screen users could indicate that they wanted to create a new item, and they could also tell the system where the new item should be placed. However, this technique can become confusing to the user if the handheld display is full of existing items. The long-press technique is often used to modify currently existing items, and if a user cannot find any open space in which to long-press, they may be forced to lose their current focus by navigating to an open area outside of the current mood-board to create the new item. We felt this was too much of a distraction to the user. To get around this problem, we simplified where items would be placed by deciding that newly created items would always be placed in the middle of the user's screen. We also moved the option to create a new item to the application's menu. Using the application's menu to create new items allows users to create new items regardless of how congested with data items their handheld display is.


Figure 34: Different types of items that can be created with our framework.

As shown in Figure 34, our framework initially supports text, image, and ink data items. When a user clicks on the "Add Items" menu option, they are brought to a new screen that allows them to create new items. Users first choose the type of item that they want to create from a drop-down box at the top of the screen. They then provide input for the item's content.

Text, image, and ink item creation views are all provided within our framework, but users are also able to create their own item types and item creation views. In our schedule creator example from before, the developers of the scheduler creator application created a milestone item data type and a milestone item creator view. These views were then added to the framework's collection of item creator views so that when a user clicked on the "Add Item" menu item, they were brought to the milestone creator view.

## Item Organization

After users have added several data items to a project, the users then move on to organizing the data items that belong to the project. In our framework we allow users to organize data either by creating item groups or by creating links between two items.

### Item Groups

In our framework, groups are created based on how close items are to each other. We felt that during the organization phase, users would want to quickly add several items to a group. As they added items to various groups, we they would also want to begin exploring different possibilities of groupings. This meant that if two designers were working on a set of mood boards, they may want to create a group consisting of a handful of items just to see how well the items go together. With just as little effort, they may want to remove the items from the newly created group and try a different organization of the image items. With this in mind, we knew that creating groups could not be a multi-step process. This interaction needed to be a quick technique that users could perform in an exploratory way and then, just as quickly, cancel the interaction if needed. For this reason, we decided that groups should be formed based on proximity to each other, rather than by explicitly selecting an option to group several items together.

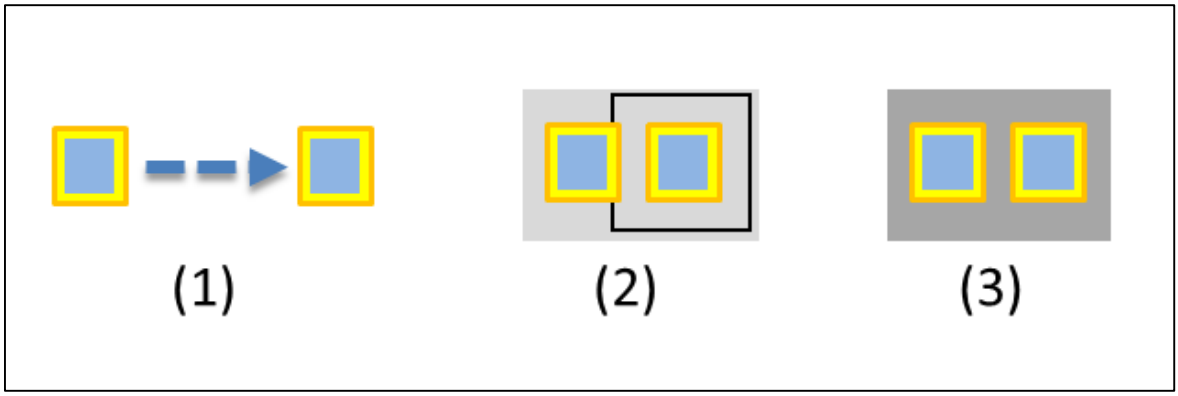


Figure 35: Grouping items. (1) An item is moved close to another item. (2) A light group border surrounds both items indicating that when the item is released, a group will be formed. (3) The finished group when the user releases the item.

To group two items together, a user begins by dragging one item towards the second, as shown in image 1 of Figure 35. When the items get within the minimum grouping distance from each other (represented as the black square in image 2), each item has a temporary grouping assignment drawn around it. This temporary background is visual feedback to the user that the two items will be grouped together if the dragged item is released. When the user releases the dragged item within the minimum grouping distance, the temporary grouping background changes to a darker shade of gray to indicate that they items have been grouped together. To remove an item from a group, the item is simply dragged away from the remainder of the group. Similar to creating a new group, an item's group assignment background changes back to a lighter shade when a user drags the item away from the group. This indicates that the item will be removed from the group when the user releases her finger. When user releases their finger, the group background disappears to indicate that the group assignment has been removed.

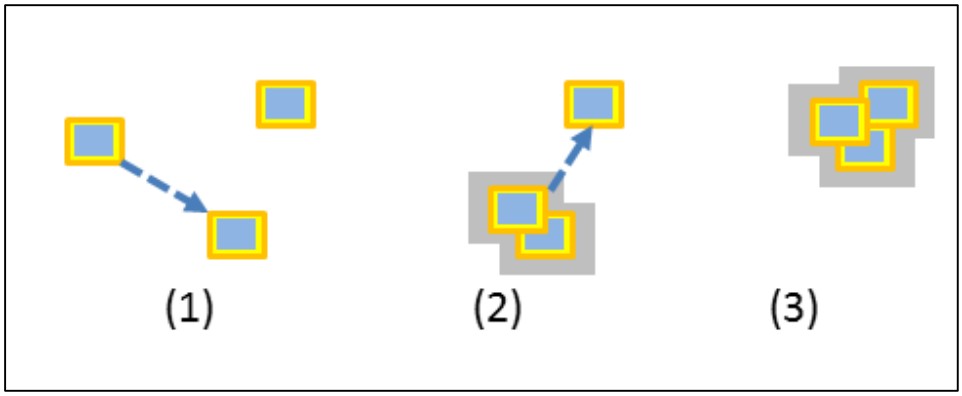


Figure 36: A user berry-picking drags an item onto another item (1), to form a group (2). Then, the user drags the group onto another item to absorb the third item into the group (3).

A common situation that occurs when users begin grouping items is that users began "berry-picking" items together [18]. For example, a user would drag two items together to form a group. Users can move an entire group by dragging the gray group border. Using this interaction, users would move the entire group to a new item and release the group on top of the new item. The new item would join the group. Then the users would grab the group again and drag it to the next item that they wanted to include in the group. This method proved to be very effective in collecting and moving several items together.

Although we wanted grouping to be a fluid interaction that could be easily done so that users could explore different types of groupings, we recognized that creating links between items would be a different type of interaction. In the schedule creation example from earlier, it would be hard to imagine the managers creating several links between milestones in an exploratory manner. We decided that creating links between items was different than creating item groups. Links are generally created with more fore-thought put into the linking decision. However, we also wanted the interaction to follow the structure of a link: it should start with the source of the

link and end at the destination of the link. With these ideas in mind, we chose to use introduce a long-press-drag interaction for link creation.

Item Links

To create a link through the long-press-drag interaction, the user begins by pressing down on the desired start item. With our system, when the user presses down on an item, the item receives a light-blue highlight to provide feedback that an item was successfully selected. At this point, the user holds their finger on the item until a predetermined amount of time has elapsed. When the time has passed, some devices vibrate to indicate that a long-press has been initiated, but some devices do not. To accommodate for devices that do not produce this haptic feedback, when the long-press timeout has occurred, we also change the color of our item highlight to a dark green. This allows users to visually see when the long-press has taken place, as shown in Figure 37. At this point, a user can begin to drag the item. To create a link, the start item is dragged to the location of the destination item. When the user's finger is inside the destination item, a green highlight will surround the end item to visually show that a link can be formed between the start and end items. In Figure 38 we see that when a user releases their finger, they are presented with a context menu from which they can create a new link between the two items. Once the option to create a new link is selected, the user will see a link form between the two items.

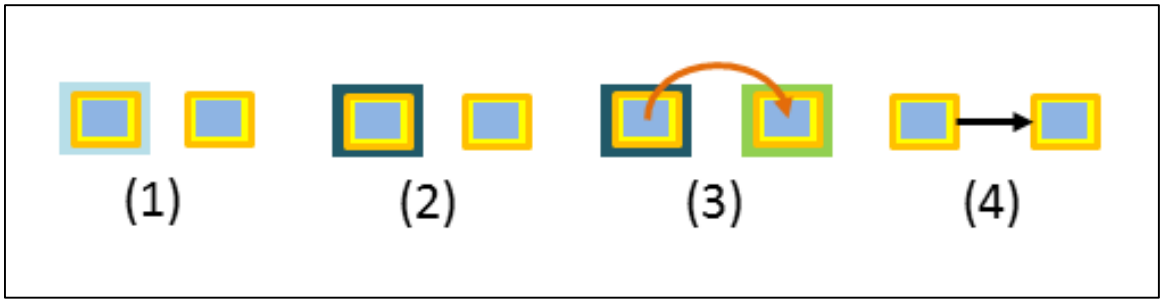


Figure 37: Creating a link. (1) A user's finger presses an item. (2) After a specified waiting period, the item highlight turns green indicating that a long-press has started. (3) The user drags the item to another item until the other item receives a green background highlight. (4) The user releases their finger and selects "Create Link". A link is created.

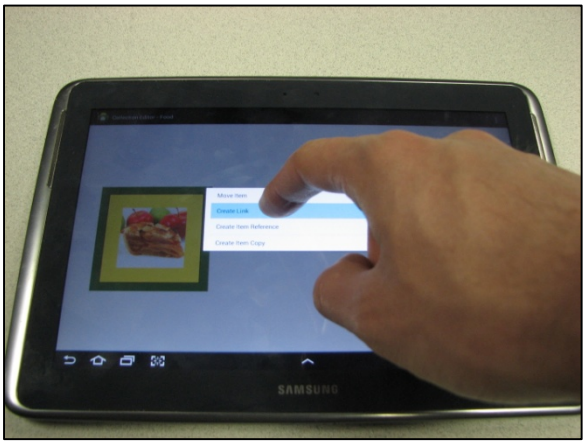

Figure 38: A user choosing the creating a link from a long-press-drag context menu.

We chose to display a context menu when the user released their finger from a long-press-drag operation because we felt that the interaction could be used for more than just creating links. For example, in the mood board creation example, one of the designers wants to use a specific image in a mood board that he is putting together. However, the image already belongs to a mood board that a different designer is creating. Rather than take the image from his co-worker, he simply creates a reference of the image and includes the reference in his mood board. When we

considered this type of situation, we decided that the long-press-drag interaction would be a desirable way to create a reference as well as create a link. When the context menu is displayed after a user releases their finger, they are not only presented with an option to create a link between two items, they also can create a reference of the dragged item. We view the long-press-drag operation as an alternate dragging operation. This means that any operations that involve either two items or a single item and a user-specified location (such as creating a reference at a new location) can be performed using the long-press-drag operation. For this reason, we also allow users to create item copies from the same interactive technique. The results of each of these three operations can be seen in Figure 39. From this figure we can see that when an item reference or an item copy is created, it also maintains the source item's attributes, such as links and notes.

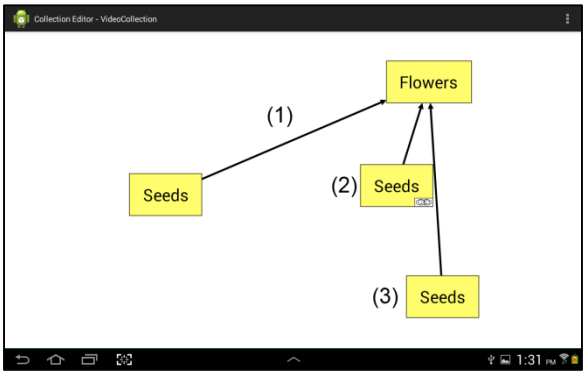


Figure 39: (1) A link created between two items. (2) A item reference to the original "Seeds" text item. (3) An item copy of the original "Seeds" text item.

### Item Critiques

During and after the process of creating and organizing items, users can also critique the work that they have performed. This critiquing could be a vote that the members of the collaborative

group make for which mood board they like the best. It could be that each member of the group places notes regarding how reasonable they feel the time estimates for each milestone item is. Some users may place notes indicating that the current time estimate for a milestone item is far too low, while others may simply place a note on the milestone item that requests that the user notify the note's creator when the milestone is completed. These critiques could also include just general comments about a particular user thinks that this milestone needs special attention because she has seen these types of tasks last longer than originally estimated. In any of these situations, there is a common need to attach additional information to any item that exists in the project. In our framework, we use note items to accomplish this need for critiques and comments.

Figure 40: A user selecting to add a note to an item.

In our framework, adding a note is similar to modifying an item, so we decided to allow users to create notes by long-pressing on an item. When the user releases their finger from an item after a long-press, a context menu appears above the item. From this menu, they can select to add a new

note. After a note has been created for a given item, the note appears as a small icon in the top left corner of the data item.

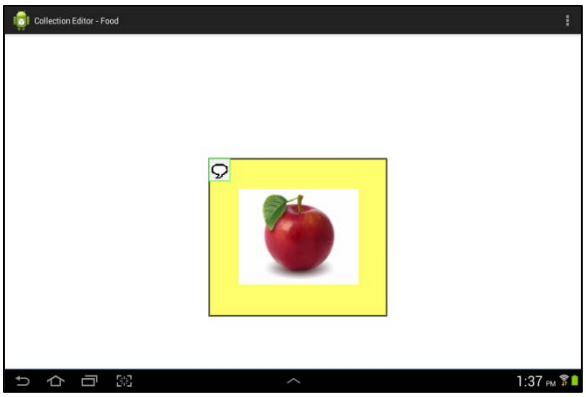


Figure 41: An item with a closed note attached to it.

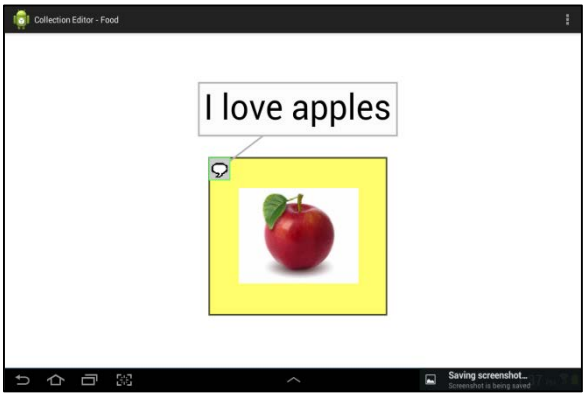


Figure 42: An item with an open note attached to it.

Although not required, notes can be given additional data information, such as textual messages or an image. As shown in Figure 42, this note content can be opened by clicking on the note's icon. If the item is moved in the context of the project, the note's content moves with the data item it belongs to. By clicking on the note's icon again, the user can close the note's content. If we consider how dense these projects potentially could become (as shown in , then we can recognize that it is important that users be able to hide the contents of an item's notes. If users

could not hide the contents of a note, then after several notes had been added to a project the display would be saturated with note contents. This would also discourage users from adding notes to items that belong to groups, because the contents of the note would hide the other items in the group. In an application like a mood board creator, this is especially problematic because the items are organized based on their proximity to each other. With these cases in mind, we see that it is necessary that notes with content can have their content open for viewing, or closed so that other work with the data items can be performed.

Long Distance Interactions

In Figure 43 and Figure 44 we see two data items on the furthest edges of a set of wall displays. When a user zooms out far enough to see the two items on their handheld, the user's finger occupies almost half of one of the wall display outlines as it shown on the user's device. In this type of scenario, the interactive techniques we have described previously break down. The human finger does not have the accuracy necessary to be able to select and move data items when the handheld device is zoomed-out at such a high level. There simply is no way that an interaction, such as creating a link between these two items, could take place at this zooming level.

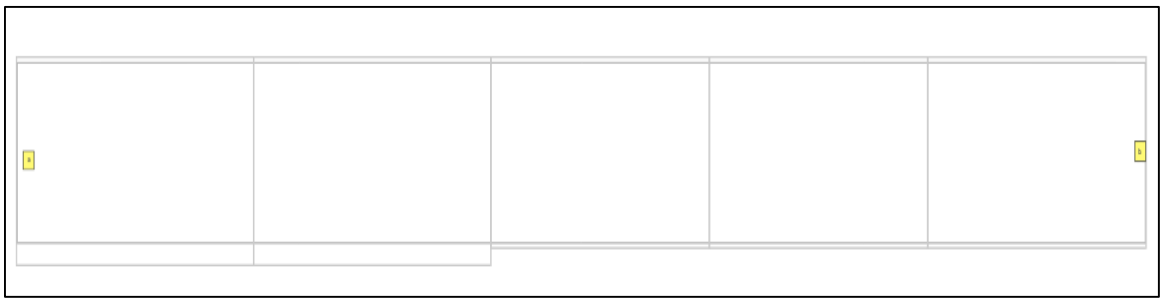

Figure 43: A screenshot from a handheld device looking at two distant items.

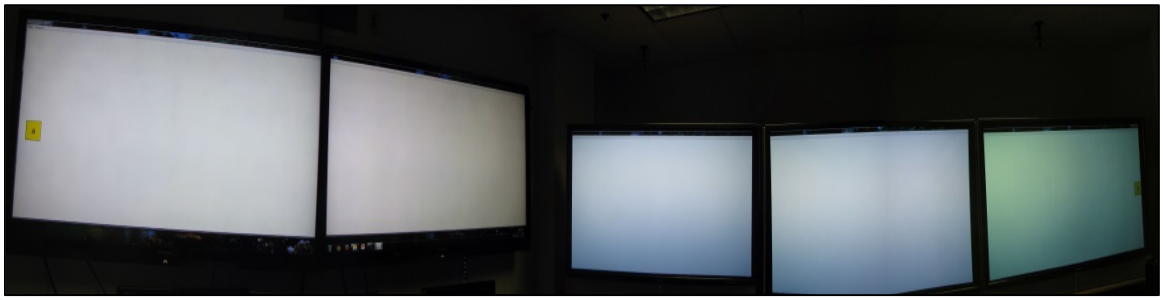

Figure 44: The two distant items from Figure 43 as they appear on the wall displays.

To solve this problem, we created the wormhole interactive technique. This technique allows users to split a handheld device's display into two independent views. These views can be zoomed, panned, and even screen jumped completely independently of each other. This means that with the wormhole interactive technique, we can convert the top image of Figure 44 into Figure 45. Now that the user has zoomed-in to view the two items more closely, her finger can now easily fit within the bounds of each of the items. At this point, the user can now drag items across the center boundary of the two views and have the item transported to the view on the other side of the wall displays. With this long-distance teleportation in place, she can now move items long distances, group items over long distances, or create links between two items over long distances. Anything that a user could do to an item or a set of items over a short distance can now be extended to long distances using the wormhole interaction.

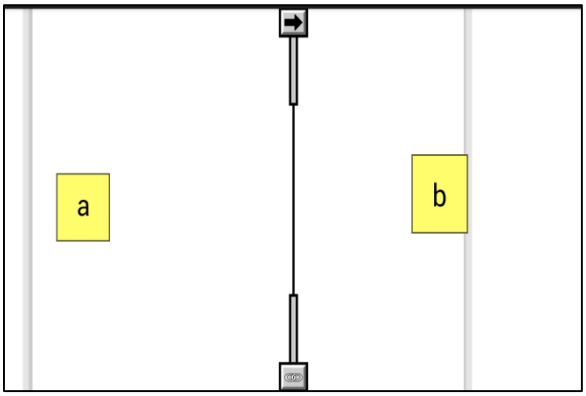


Figure 45: A wormhole view of the same project displayed in Figure 44.

Moving items long-distances through the wormhole can be extremely useful in several scenarios. For example, imagine that during the schedule creation example, after the initial brainstorming phase, the group decides that the marketing team put up all of their items in the wrong place. Originally they had put all of their milestones on one of the far-right wall displays, but they decided later that they really needed to start working on their marketing assignments for this project much sooner in the schedule. Without the wormhole interaction, the marketing manager would have to drag each of her milestone items one by one across the entire project space until they were in the new proper position. This could potentially require a lot of timely manual navigation and dragging. However, with the wormhole interaction she can split her screen and navigate the left-half to the location where she wanted to move her items to. Then, with only a few finger drags she could have moved her entire set of items to a new location. When she is finished, she closes the right side of her wormhole display and resumes interacting with her items in a single view mode.

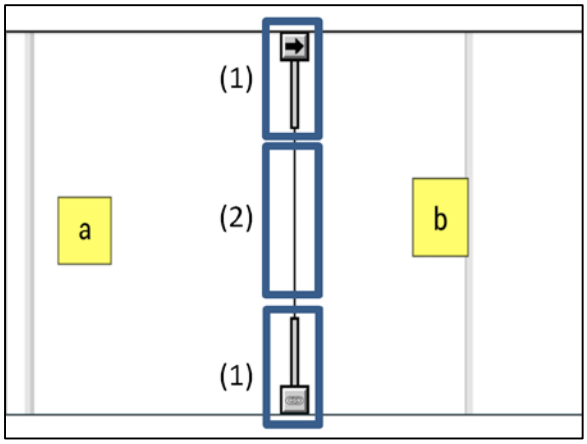


Figure 46: The three drag-action regions of a wormhole. (1) Customizable drag-action regions. (2) Standard drag-action region.

Figure 46 also exposes another functionality that is built into our wormhole feature. The wormhole has three drag-action regions built into the boundary between the two sub-views. The middle region is simply a boundary between the sub-views, as has been mentioned previously. If the user drags an item through this region, the user's current action does not change. For example, if a user long-press-drags and item through the middle region and then releases their finger on the other side of the wormhole boundary, then a context menu will appear, just as if they had long-press-dragged an item without crossing the wormhole boundary.

However, if the user drags a single item through the top or the bottom region, then the drag takes on the action that is represented in the image on the far edges of the boundary. For example, if the user drags item ‘a’ through the top region, it will take on a linking action, as shown in Figure 47 and Figure 48.

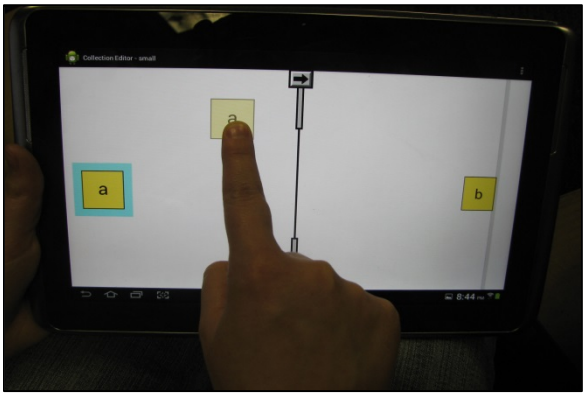


Figure 47: Dragging an item normally before crossing the action region.

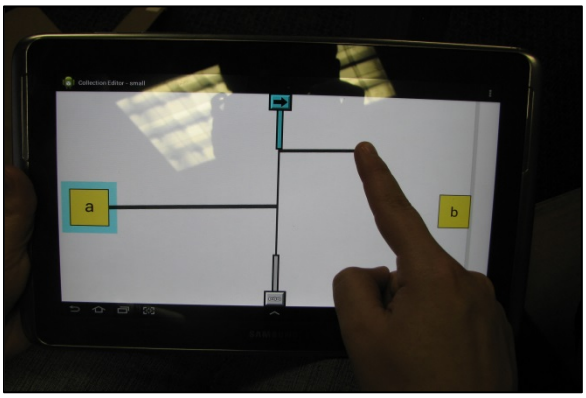


Figure 48: After crossing the create link region boundary. If the user's finger is released over the 'b' item, a link will be formed between the two items.

When the user crosses the action boundary at the top of the wormhole boundary, the action boundary turns a light-blue color to indicate that the action has been initiated. This crossing behavior is similar to previous work that was done in CrossY [2]. In Figure 48, the action that was crossed through was a "create link" action. To show this visually, the display no longer shows a transparent version of the item being moved, but rather now shows a temporary link between the source item and the user's finger. Just as we saw with the long-press-drag interaction, if a user drags his finger over a second item when the "create link" action is

activated, the second item will receive a light green highlight to indicate that if the user releases his finger, a link will be created. The same process applies to the bottom region. By using the action boundaries instead of the long-press-drag interaction, users can create more links in less time because they do not have to wait for the long-press timeout. In the long run, this can actually save users time and effort.

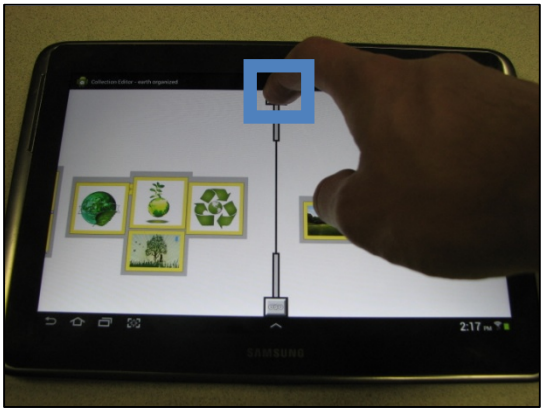

Figure 49: A user clicks on the action button for the top action region, which is marked with a blue square. The current action assignment for the region is “create link”.

Figure 50: A user chooses to change the top action region to be the "Create Item Copy" action.

A final feature of the wormhole interaction is that the actions that the boundary regions perform can be customized. In our current version of the framework, only Create Link, Create Item Reference, and Create Item Copy can be used if an item is dragged through one of the action regions. If a user clicked on the Create Link image at the top of their wormhole boundary, then a context menu would appear with the three previously stated options, as shown in Figure 49 and Figure 50. If the user selected the Create Item Copy option, then the Create Link image would change to a Create Item Copy image, as seen in Figure 51.

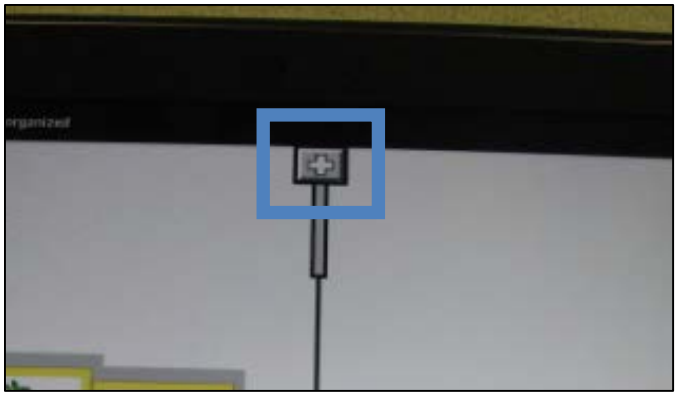

Figure 51: After the user changes the action assignment from “create link” to “create item copy”.

Now, if the user drags an item through the Create Item Copy action region, then a copy of the item they were dragging will be created wherever the user releases his finger. If the user wanted to move an item and accidentally slipped his finger through one of the action regions, then the action can be cancelled by dragging the user’s finger back through the wormhole. Once the user’s finger is back on its starting side, then the action is released and the original interaction is restored, whether it is a normal drag or a long-press drag.

The wormhole interaction is a customizable component of our framework as well. Developers who use our framework to build their own handheld-to-wall applications can create their own list

of actions that they want the wormhole regions to allow. Our framework provides a base action class that developers can extend with their own action functionality. They then have to register their action object with an action factory class. After this, the framework can then take advantage of their new actions. Perhaps an application wanted to be able to create links of different colors? One way to accomplish this would be to create a new action for each major color that users wanted to create. Then by adding the different colored link actions to the wormhole regions, users could choose at run-time which colors they use most often. This sort of customizability allows the wormhole to not only enhance the user's abilities, but also allows developers to enhance the amount of interactions their users can perform over long-distances.

We chose to evaluate our interactive techniques by verifying that our framework allowed developers to create new applications in a small and reasonable amount of code and time. We also performed a set of user studies to verify that our interactive techniques were learnable and that they improved users' ability to accomplish tasks.

## Chapter 7. Evaluation

We performed three different evaluations of our framework. This first evaluation tested our data item caching system. The second evaluation tested how difficult it was to use our framework to build applications. The third evaluation tested our generic interactions.

### Efficient Data Synchronization

The first evaluation of our system was to verify that our data caching mechanism improved the performance of our system. In this evaluation, we measured the amount of time that it took a handheld device to load a project. When a project is loaded, the system first must make a query to the cloud data store to discover which data items belong in the loaded project. Once the handheld device receives from the data store which items belong to the loaded project, it then has to retrieve the attributes of each of the data items within the project. Finally, once the data attributes are loaded, then the data contents of each item must be loaded. This is the most data intensive process within our framework, so we felt it would be the best place to measure how our caching system improved our performance.

We measured our project load-time under two different situations. The first situation was to measure the entire project loading time without caching. In this situation we removed our framework's caching mechanism on both the handheld device and cloud data store. This meant that every time the handheld device needed to access data, it had to make a request to the cloud data store. The cloud data store would then have to access the data from disk. We expected these load times to be slow.

Within each cache testing situation, we provided three different projects that we would load. The first project, named "Low", contained only five ink items. The next project, "Medium",

contained 15 ink items. The last project, “High”, contained 25 ink items. These ink items were simple handwritten numbers that counted up from 1. We loaded each of these projects three times without caching and recording the amount of time it took for each project to load. Our results from these set of experiments are shown in Figure 52.

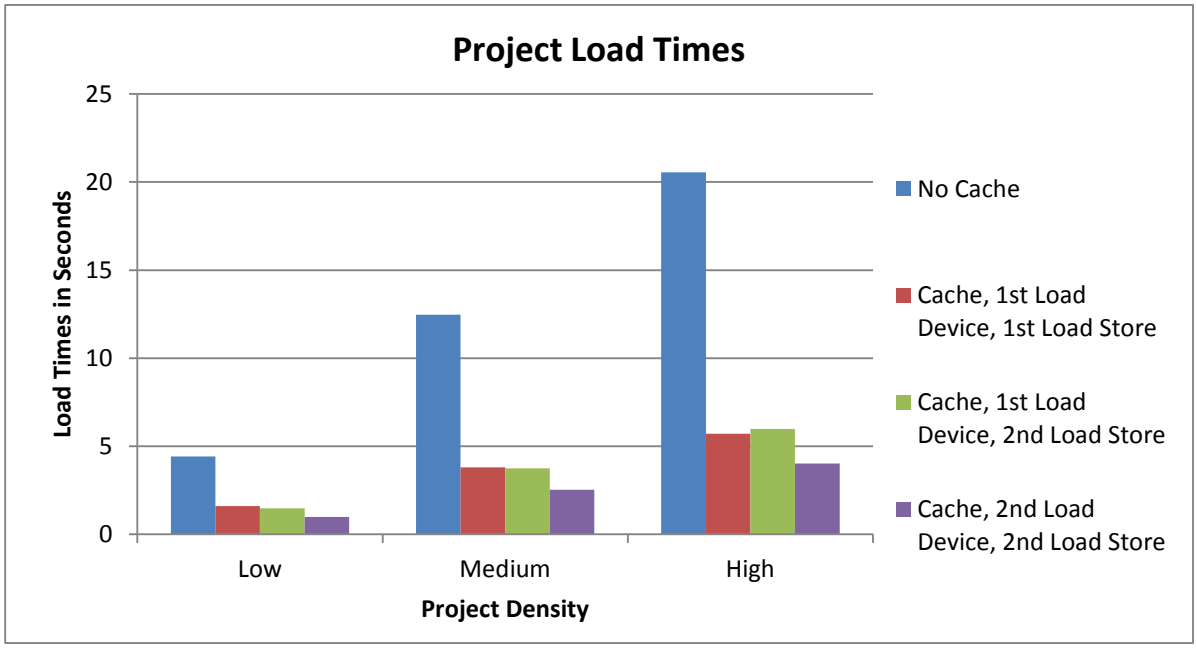


Figure 52: The load times for the three projects with varying amounts of caching, including no caching.

From Figure 52 we can see that, as we would expect, the amount of time it takes to load items is directly proportional to the number of items that need to be loaded. Loading a project without a cache system in place takes several seconds, even for the project with only five ink items. Load times of this duration are unacceptable for a collaborative application, and these delays were what motivated us to create an item caching system at the beginning of our framework’s development.

In our second cache testing situation, we introduced our caching system into our framework. We then loaded each of the three before-mentioned projects again. However, we recognized that when the caching system is engaged in our framework, there are actually four different scenarios that must be considered when loading a project from the cloud data store.

- **No Cache** - Neither the handheld device or the cloud data store had any caching enabled.
- **Cache, 1st Load Device, 1st Load Store** - Both the handheld device and data store have caching enabled. This is the first time that either the handheld device or the data store have ever loaded the project.
- **Cache, 1st Load Device, 2nd Load Store** - Both the handheld device and data store have caching enabled. The data store has loaded the project before, so the project's data will be cached in the store's memory. The handheld device has not loaded the project before.
- **Cache, 2nd Load Device, 2nd Load Store** - Both the handheld device and the data store have caching enabled and both the device and store have loaded the project previously.

The first comparison that is apparent from Figure 52 is the difference that introducing caching has on the project load times of our system. It may seem strange that simply enabling the cache system would have such an impressive speed enhancement to the system, especially when a project is loaded for the first time. When a project is loaded on a handheld device, the handheld device first queries the store for which items the project currently contains. In order to response to this query, the cloud data store must check the attributes of each item in the store. As the data store checks the attributes of the items, it stores the loaded attributes into its cache. This means that just by checking for which items belong to a project, the data store populates its cache. Then,

when the handheld device begins requesting data attributes on different items in the project, the cloud data store already contains the items' attributes in its cache. This produces the large 3x speed-up that we see in Figure 52.

Next we experimented with a scenario where the cloud data store had previously loaded the project, but the handheld device had not. We did not expect the load times in this scenario to be very different from the load times in our previous scenario. As we expected, the difference in the amount of time needed for 1st project loading and the 2nd project loading for the cloud data store are negligible.

The final scenario (Cache, 2nd Load Device, 2nd Load Store) we tested our caching system under was what we would consider to be more normal usage. In this scenario, both the handheld device and the data store had previously loaded the project in question. This scenario simulated re-opening a project that a user had recently worked on. We expected these load times to be the fastest of all. We found that providing the handheld's on-disk cache with the project's data gave us one-third speedup across each of the projects. Although the load times for this scenario were the fastest, they were not as fast as we had expected them to be. However, considering the fact that if we compare our slowest load times (no caching) against our fastest load times (with caching and previously loaded project), we find that adding caching to our system provided a 5x speedup to our project load times. This speedup successfully met our requirement that we needed to make our data synchronization efficient.

Extending the Framework

The second way we evaluated our system was by determining how easy it was to extend our framework to create a new application. To test how easy our framework was to extend, we

created the sample applications that we have discussed throughout this thesis: a mood board creator and a schedule creator.

The mood board creator fit very easily into the type of application our framework expects to be used for. In our mood board creator application, users create mood boards by grouping together images. To show images on the displays for users to interact with, users can create image items using the interactions we previously mentioned. Once the user's project contains a set of image items, they can then group the items into mood boards as they see fit.

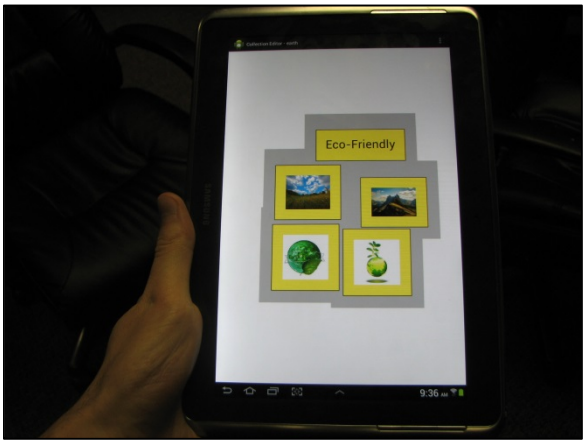


Figure 53: A user creating an Eco-Friendly mood board within the mood board creator application.

Users may also want to attach a label to mood board groups to describe what type of mood they intended the group to represent. To accomplish this task, users can create a text item with the label's text. This text item can then be added to the mood board group. The text item label will now move with the group and effectively will indicate what mood the group is attempting to convey.

In our mood board example from earlier, the three designers merged two mood board groups together. After they had performed the merge, however, they decided that the resulting group

was not the direction they wanted to go in after all. To correct this action, they performed an undo operation on the project. Our framework already provides undo and redo functionality, so that feature requirement was provided by our framework.

The last part from our mood board example is that one of the designers needed to be able to create references to some of the items in the project. This functionality is already built into the framework and has been discussed previously. Our mood board creator application used this built in functionality.

The schedule creator sample application was very similar to the mood board application in that most of its functionality already existed in our framework. Users can already create links between items in our framework. Our framework also provides interactions whereby users can add notes to items. However, the schedule creator application was different from the mood board creator application in that the scheduler creator used milestone items instead of images. These milestone items are not provided in our original framework, so the framework needed to be extended to include this type of item and to exclude the other types of pre-existing items.

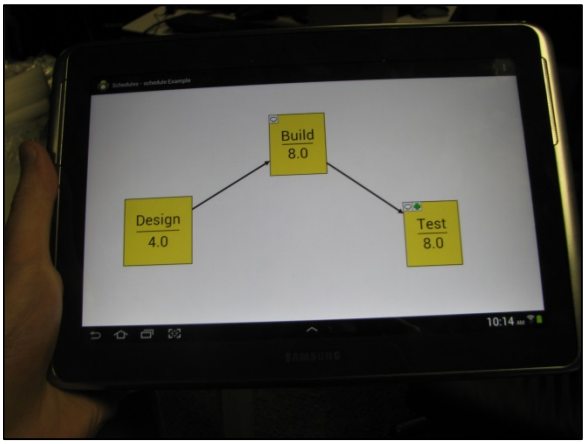


Figure 54: An example of using the schedule creator application. Milestone items with their names and estimated days to completion are shown.

To add the milestone item type, we extended the base data item type. We then provided methods on the new milestone type that allowed it to display the milestone's name as well as the estimated number of days it would take to complete the milestone. The final step in creating a new data type was to provide methods for writing the milestone item's data to the cloud data store. Once these two operations were in place, we now had a new data item type to use in our schedule creator application.

With a limited amount of work, we successfully were able to extend our framework to create two sample applications. In Table 1 we can see some of the work quantified into numbers that can be compared against each other.

| | Hours to Build | Source Files in Project | Lines of Code |
|---|---|---|---|
| Mood Board Creator | 15 | 5 | 209 |
| Schedule Creator | 8 | 7 | 733 |

Table 1: A comparison of the effort required to build the two sample applications.

From Table 1 we can see that within a day or two, a developer could have a collaborative application working on a handheld device and on a wall display. We chose to build the mood

board creator application first, which meant that while we were building it, we needed to make changes to the framework to support extensions that we didn't realize we would need. For example, our framework allows users to create ink, image, and text items. However, our mood board creator needed to limit users to creating only image items. Future applications would most likely need the feature to not allow users to create items of various types, so we needed to build in a registration system where a developer could indicate which types of items a user could create. This explains why the mood board creator application took almost twice as much time to build as the scheduler creator application did. For this reason, we consider the schedule creator to be much more similar to what developers could expect if they wanted to extend our framework for their own applications.

Interactive Technique User Studies

After verifying that with a small amount of effort our framework can be used to create various applications, we then conducted a set of user studies to determine whether our interactive techniques were learnable and useful. Our 18 user study participants (7 female, 11 male) were asked to perform a set of tasks to measure the effectiveness of our various interactive techniques. The study consisted of 16 training tasks. Each training task began with a short training video in which users were shown an interactive technique that our system uses. Then, they were given a task that used the technique they were trained on in the video. Once the task was complete, they were then given their next task.

When they were finished with the training tasks, they were then asked to complete an additional 20 tasks. During these additional tasks, we recorded data about the different interactions that the

participants used. Specifically, we recorded data that would allow us to answer three main questions about the system we had created.

- Does the Screen Jump interaction make users more efficient than the Pan and Zoom interaction?
- Does using the Wormhole interaction make users more efficient than they are without it?
- Do users look at the wall displays while they are completing tasks?

We rotated the order that our tasks were presented to our participants in order. This task rotation allowed our participants to learn task goals on different instances of the task. For example, tasks 1 through 4 had the same goal, but the original starting positions of the items were different. By rotating these similar tasks such that the second participant would start at task 2, our participants could learn the task's goal on different tasks, which would distribute the initial learning curve among each of the similar tasks. After the second participant completed task 4, she would then do task 1 before advancing to next group of similar tasks.

Navigation Comparison

We created two complementary sets of tasks in our study to determine whether the Screen Jump interaction allowed our users to navigate around a project in less time. In the first set of tasks, users were asked to use the pan-and-zoom technique to move various distances across the wall displays, as shown in Figure 56.

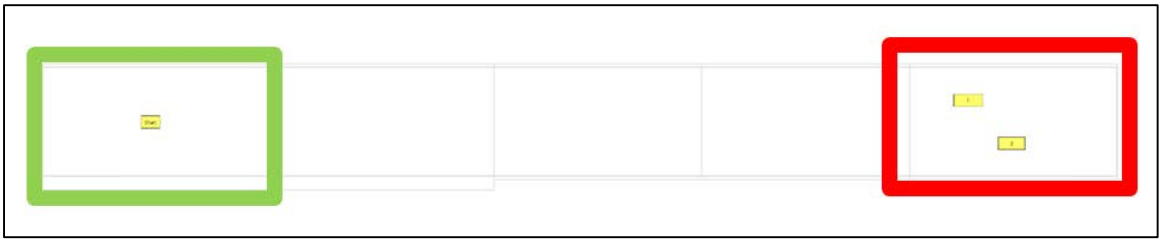

Figure 55: Users began the task with their device at the green square. They then had to navigate to the red square and group the two items in the red square together.

Once the task's destination was reached, users would group two items together to indicate that they had reached the desired location. We disabled the screen jump interaction for these tasks to ensure that users used the pan-and-zoom techniques for navigation. In the second tasks, users were forced to use the screen jump technique to navigate various distances across the wall displays. Similar to the first set of tasks, we disable pan-and-zoom navigation for this set of tasks. Although the distances were the same as in the pan-and-zoom set of tasks, we presented them in a different order to the user. This eliminated the possibility that screen jump may have been faster than pan-and-zoom simply because the users remember the order of the task navigations. Our results are shown in Figure 56.

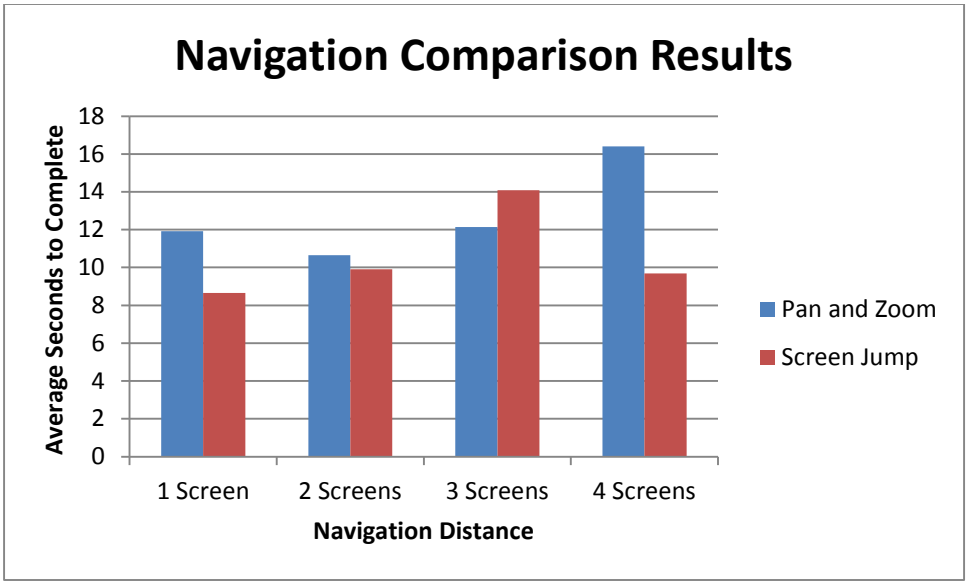


Figure 56: User study results for users navigating using the pan-and-zoom technique versus the screen jump technique.

| Navigation Distance | 1 Screen | 2 Screens | 3 Screens | 4 Screens |
|---|---|---|---|---|
| Paired t-test result | 0.168 | 0.343 | 0.356 | 0.048 |

Table 2: Paired t-test results from the original navigation comparison results.

The first thing we noticed about these results was that there were several extreme outliers in our data. For example, if we look at the comparison of the techniques across three screens, we see that the majority of the results for screen jump are around ten seconds, but there is one example that is almost two minutes long. In our user studies, occasionally participants would do something that did not accurately reflect the time it took to navigate the task. One user forgot how to perform a screen jump and spent a minute trying to remember before she asked for help. Another user spent some time trying to remember how to link to items together, only to remember later that he only needed to group them together. These outlier points generally occurred on the first task in these sets that the user was presented with. On the second, third, and fourth tasks of the set, the user understood what was required of them. Once they understood the

task, their results were much faster and much more consistent. With these data exceptions in mind, the updated results with these inaccurate data points removed can be seen in Figure 57.

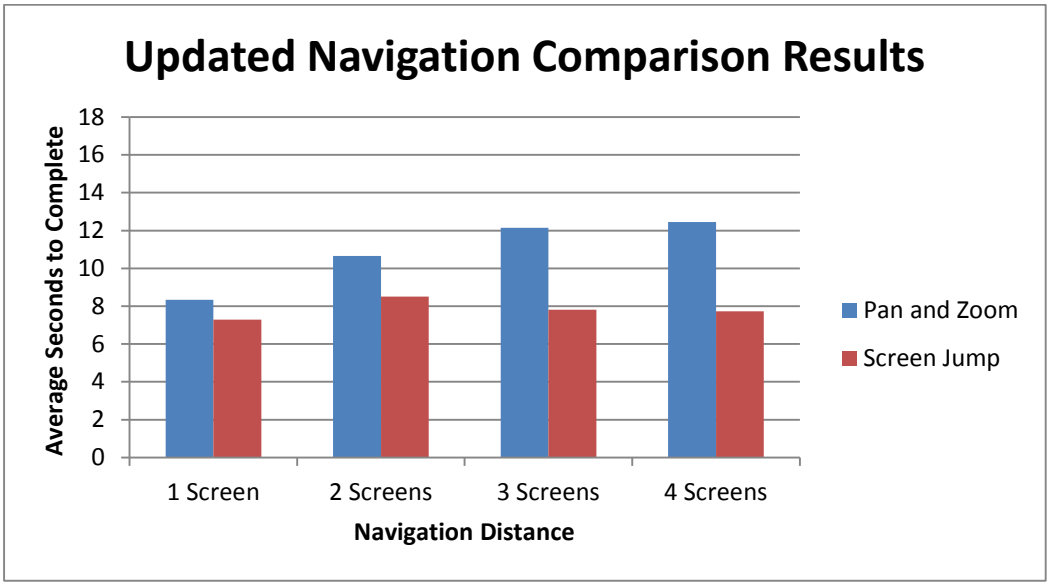


Figure 57: User study results for users navigating using the pan-and-zoom technique versus the screen jump technique when outliers were removed.

| Navigation Distance | 1 Screen | 2 Screens | 3 Screens | 4 Screens |
|---|---|---|---|---|
| Paired t-test result | 0.250 | 0.113 | 0.001 | 0.002 |

Table 3: Paired t-test results from the updated navigation comparison results.

With these updated results, we can now see that the screen jump interaction is faster than the pan-and-zoom interaction. Over short distances the improvement in navigation is not very significant. However, when the two interactions are compared over long distances, we can see a very significant improvement in user interaction time. We consider the screen-jump interactive technique userful because it decreases the amount of time it takes a user to navigate a long distance.

Another aspect of screen jumping that is better than pan-and-zoom is its scalability. From Figure 57 we can see that as the distance a user had to navigate increased, the amount of time that it took for the user to perform the navigation also increased. This trend is not true of the screen jump navigation. The amount of time that a user took to navigate using the screen jump interaction remained almost constant, regardless of the distance the user needed to navigate. This indicates that the screen jump interaction would scale much better than the pan-and-zoom interaction as additional screens were added to an environment.

Although there is already a significant difference between navigation times using pan-and-zoom and screen jumping, we feel that the current difference is only the beginning. In our user study, users initiated the screen jump interaction by long-pressing with two fingers for an entire second. With additional work put into optimizing the screen jump interaction, we feel that screen jumping could widen the time gap between itself and the pan-and-zoom interaction, even a smaller navigation distances. For example, one improvement would be to draw the project's items on the screen-jump view as they are drawn on the wall displays. This would help users to identify which wall display they want to jump to when they are looking at their handheld device. When users were forced to use the screen-jumping interaction during our user studies, they would often look between the wall displays and their handheld device several times during a screen-jump. This is because they would find an item they wanted to modify. Then they would open the screen-jump view. Then they would forget which screen the item was on, so they would look up at the wall displays to find it again. This multi-glance interaction was common enough that we feel drawing the items on the screen-jump view would greatly help the interaction to be faster.

Another way we could improve screen-jumping would be to place the screen-jump view on the project view (normal item view). As shown in Figure 58, the screen-jump view could be placed at the top of the screen in the title bar region. This would allow the user to jump to any part of the project by just tapping the position in the title bar where they wanted to navigate to. This would be much faster than our current two-finger-long-press interaction, and would provide for much better navigation accuracy as well.

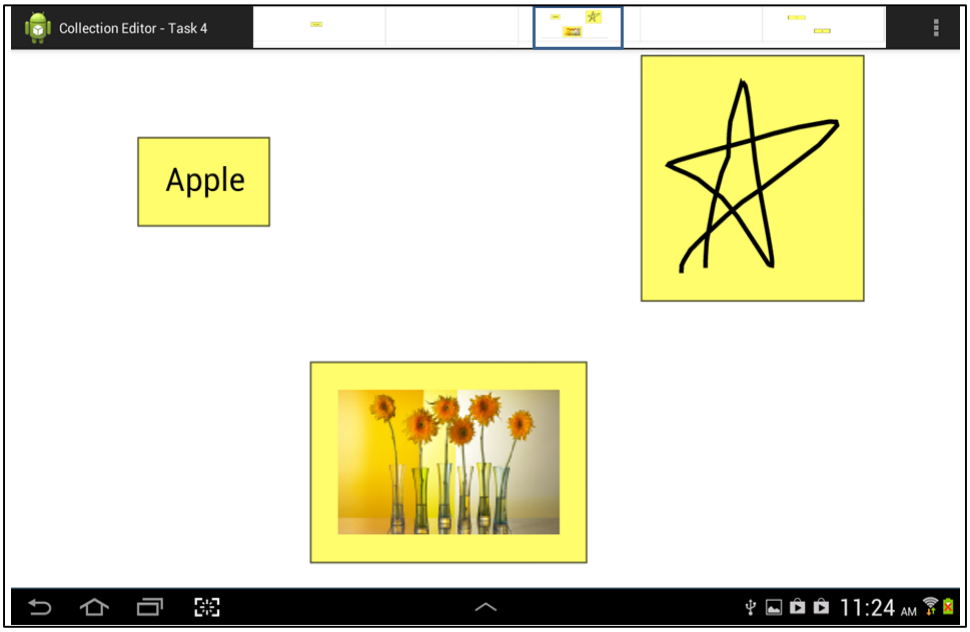


Figure 58: Possible layout of placing the screen-jump view in the title bar of the normal item view.

Wormhole Interaction Comparison

After testing our navigation techniques, we next tested whether the wormhole interaction allowed our users to move items over long-distances faster or not. For this test, we presented the participants with two sets of tasks. Each set consisted of four tasks. At the beginning of the tasks, users were presented with a set of text items on the farthest left wall display and a set of text items on the farthest right wall display, as can be seen in Figure 59. Each of the text items on

wall display contained either a 0 or a 1. In total, there were four 0's and four 1's. For this task, users had to group the text items based on their content (0's with other 0's and 1's with other 1's). Then, then were required to group the items based on their content, and then move the 0 text items to the farthest left wall display and the 1 text items to the farthest right wall display, as shown in Figure 59.

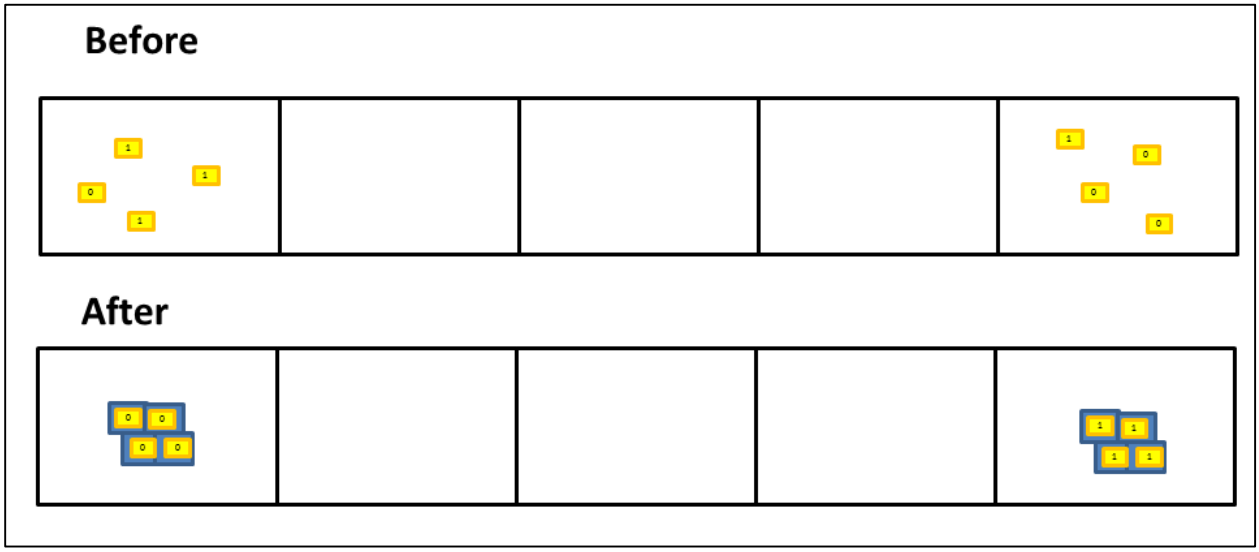


Figure 59: An example of a task used to determine whether using the wormhole interaction was faster than not using it.

Each task within a task set was slightly different than the others. Our four variations on this task were as follows:

- **2 and 2:** Each end display had two 1 text items and two 0 text items.
- **3 and 1:** The left-end display had three 1 text items and one 0 text item, and the right-end display had three 0 text items and one 1 text item.
- **4 and 0:** The right end display had all of the 0 text items and the left-end display had all of the 1 text items, but the items were not grouped.

- **Grouped:** The item arrangement was the same as "4 and 0", except that the items were already grouped together.

During the first set of tasks, users were required to complete the task without the wormhole interaction. We disable the wormhole interaction to enforce this constraint. During the second set of tasks, the wormhole interaction was enabled from the beginning of the task and users were unable to disable the wormhole interaction. The results of these tasks are show in Figure 60.

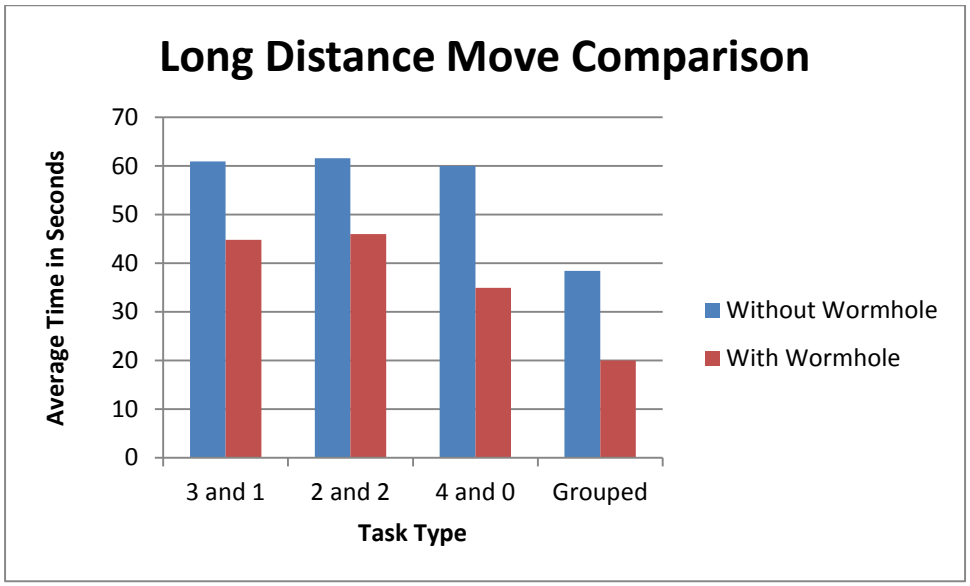


Figure 60: Comparing time to complete tasks using the wormhole interaction against not using the wormhole interaction.

| Navigation Distance | 3 and 1 | 2 and 2 | 4 and 0 | Grouped |
|---|---|---|---|---|
| Paired t-test result | 0.140 | 0.102 | 0.0001 | 0.012 |

Table 4: Paired t-test results from the long distance move comparison data.

From these results we can see that users were faster with the wormhole interaction on all of the tasks. Because our users were faster with the wormhole than they were without it, we consider it to be a userful interactive technique. It would be interesting to perform additional studies to

determine if there are certain types of tasks that the wormhole is more beneficial with. For example, in our study we performed a task where users had to group items and position those groups on certain wall displays. It would be interesting to see if we had a similar speed improvement if users were completing tasks that involved linking items together.

Looking at the Wall

The final data that we recovered from the user study was how often participants looked at the wall displays instead of the handheld device they were holding. Measuring how often users looked at the wall displays gave us an idea of how often users use the wall display to learn about the big-picture of their current project. During every task in our user studies, we used a camera to record the user interacting with the handheld device. We then watched the film afterwards and counted the number of times in each task that the user looked up at the wall displays. Our results are shown in Figure 61.

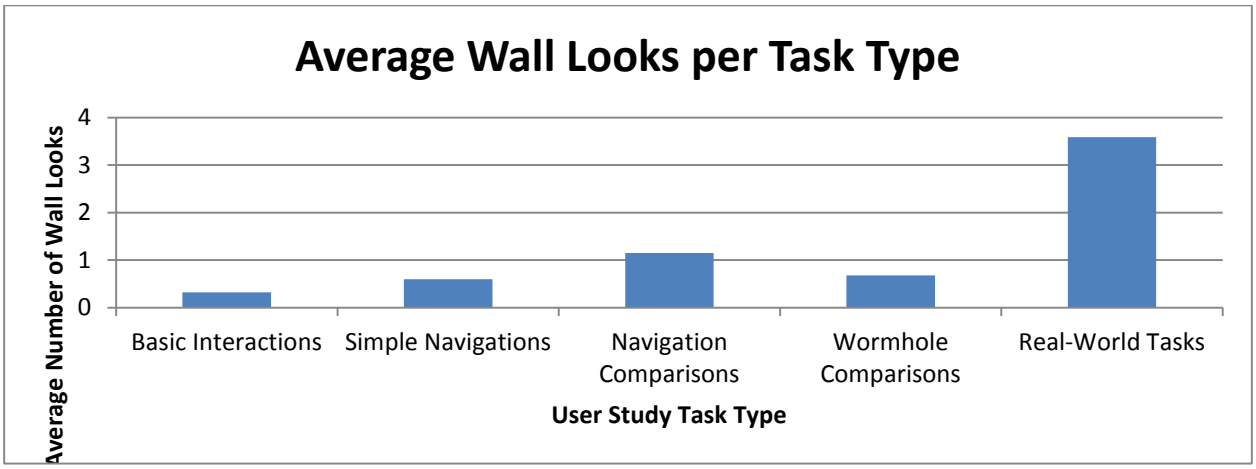


Figure 61: The average number of times that a user looked at the wall displays for each user study task.

The first ten tasks of our user study taught users basic interactions about using our system, so it is very understandable that most users did not look up at the wall displays during these tasks. Our

observations showed that the users that did look up at the wall displays during these tasks did so simply because they saw something move on the wall display. This movement was created when the wall display switched the task data that it was currently showing. Users could complete the first ten tasks using the starting handheld display position. With these things in mind, it makes complete sense why users did not look at the wall displays during these tasks.

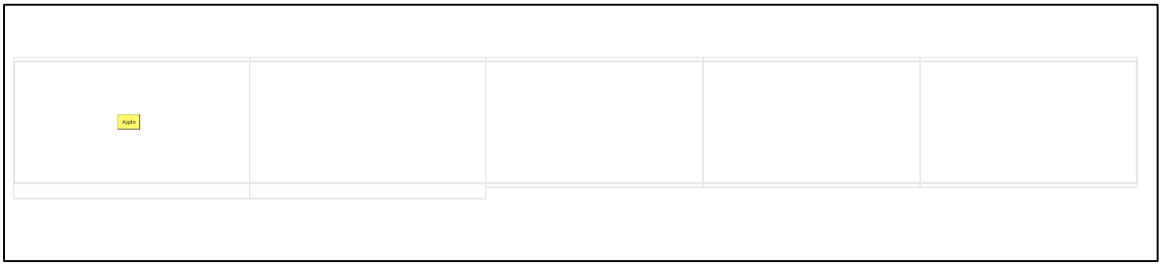

Figure 62: The item layout for the 11th user study task. In this task, users had to move the item to the right-most wall display.

The 11th task was the first task where users had to navigate the entire span of the wall displays, as shown in Figure 62. In this task, users had to move an item from the left-most wall display to the right-most wall display. At this point, several participants looked up for the first time of their user study session. However, as demonstrated in Figure 61, all they needed was one look. Once they understood the scope of the task, they quickly went back to their handheld device and were able to complete the task.

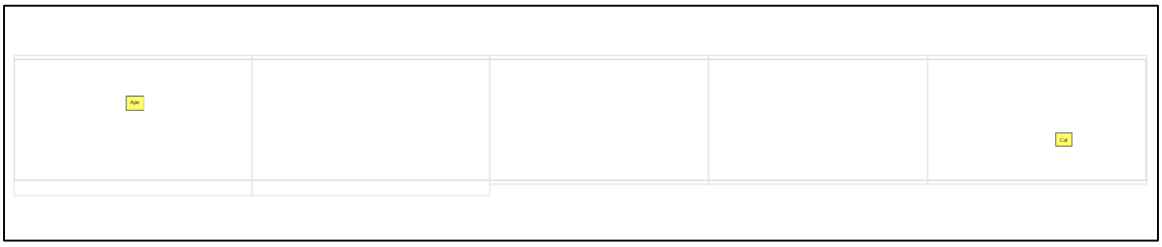

Figure 63: The item layout for the 14th user study task. In this task, users had to create a link between the two items.

Tasks 12 through 16 also required users to perform an interaction across the span of the wall displays. However, we can see that once users were familiar with how to interact across the span of the wall displays, they no longer needed to look up to perform these types of tasks. This shows us that users do not necessarily need to look at the wall displays to perform a long-distance interaction.

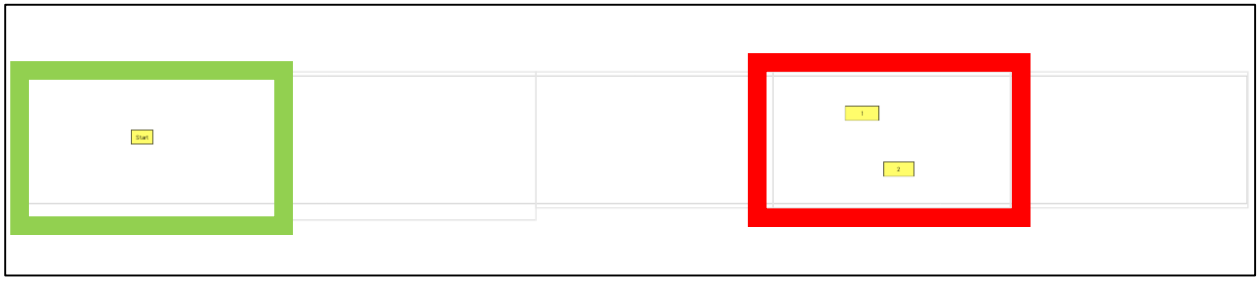

Figure 64: In task 17c, users had to navigate their handheld devices from the green region to the red region. Once they were at the red region, they had to group the two items within the red region.

Tasks 17a through 18d were designed to be a comparison between the pan-and-zoom (17) and screen jump (18) navigation techniques. In these tasks, the user's handheld device was positioned at either the left-most or right-most wall display. They were then asked to navigate to a pair of text items and group them together to show that they had successfully navigated to the target location. With the pan-and-zoom technique, some users would just zoom-out on their handheld device instead of looking at the wall displays. With task 18, users could not zoom-out since we disabled pan-and-zoom. We disabled pan-and-zoom to force them to use screen jump so that we could do an accurate comparison. This meant that the only way users could find the target text items that they needed to group was to look at the wall displays. This accounts for why the average number of looks at the wall displays for task 17 was lower than they were for task 18.

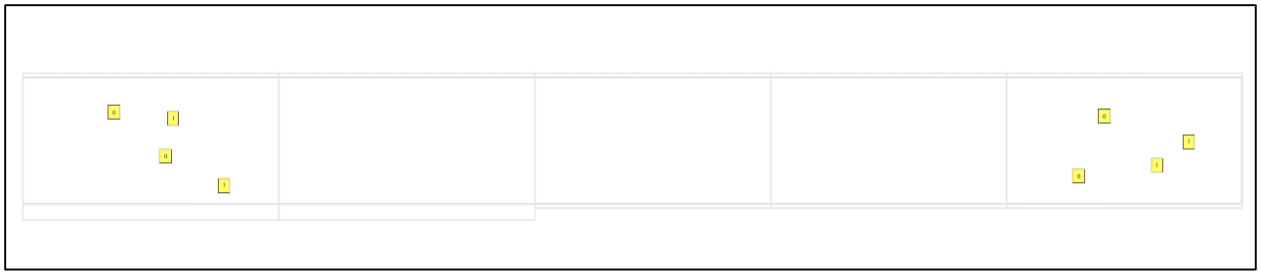

Figure 65: In Task 20c, users had to group text items based on their text contents. All of the text items with a 0 had to be grouped and placed in the left-most wall display. All of the text items with a 1 had to be gropued and placed in the right-most wall display.

Tasks 19a through 20d involved comparing the effectiveness of the wormhole interaction. To do this, we place a set of text items on the far-left wall display and a different set of text items on the far-right wall display. Once the participants learned that the items would always be on these two wall displays, they never needed to look up at the wall displays.

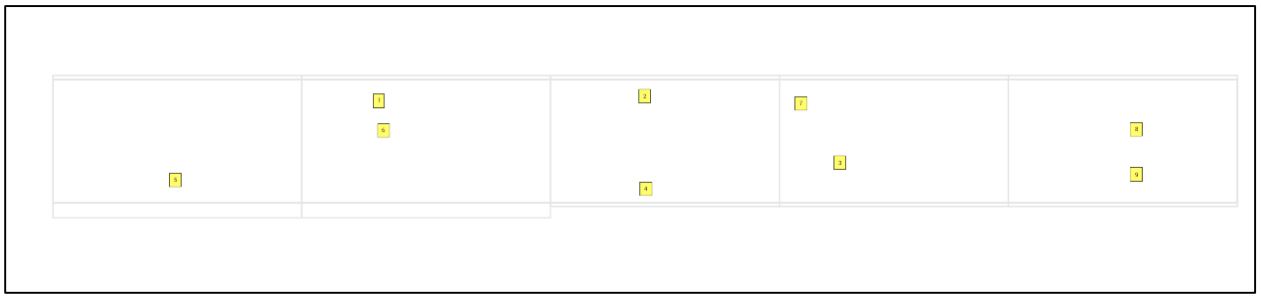

Figure 66: In Task 22a, users had to create a link between items in ascending order (1 linked to 2, 2 linked to 3, 3 linked to 4, etc.).

Tasks 21a through 22d were designed to simulate real-world usage more than the other tasks were. In the four tasks for task 21, users were given several text items that needed to be grouped by their text contents. Once the items were grouped, they then needed to be placed on a specific corresponding wall display. We expected users to look at the wall displays more during this set

of tasks than they actually did. By looking at the data, we found that users would typically not navigate long distances on task 21. Instead, they would zoom-out on their handheld device so that they could see two or three wall displays on their handheld. They would then group and move the text items that they could see on these displays. After they had finished working on the items they could see on their handheld device, they would then navigate to another nearby position and work on the items in the new location. There were very little long-distance navigations where users would need to jump several wall display lengths. Users did look up on average about two times per task in the Task 21 set. This usually happened at the end of the task when almost all of the items were properly grouped and positioned. At this point, the participants would have one or two groups that did not have all of their needed items, as shown in Figure 67, Figure 68, and Figure 69. In order to find these last items, users would then look up at the wall displays. When they located the items on the wall displays, participants were more likely to use screen-jump to navigate to the item more often than they were at any other time during the task.

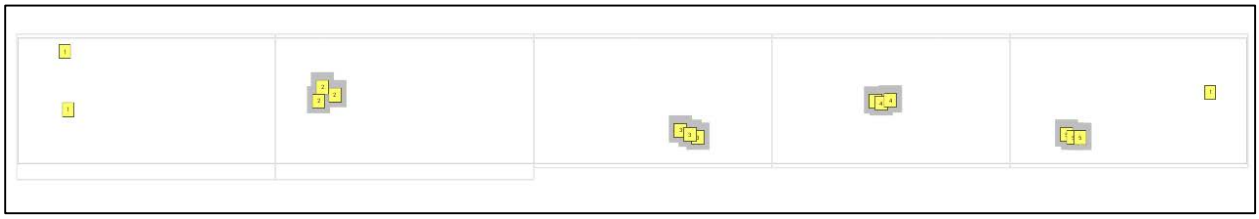

Figure 67: Items arranged on a wall display when a user has almost completed the task.

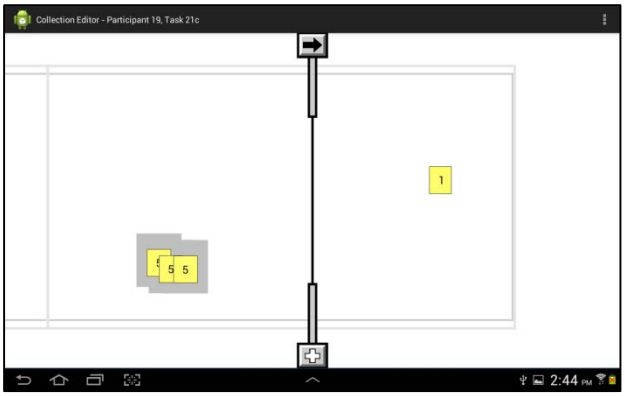


Figure 68: The user's view of the project before screen-jumping the left side of the wormhole to the other "1" text items.

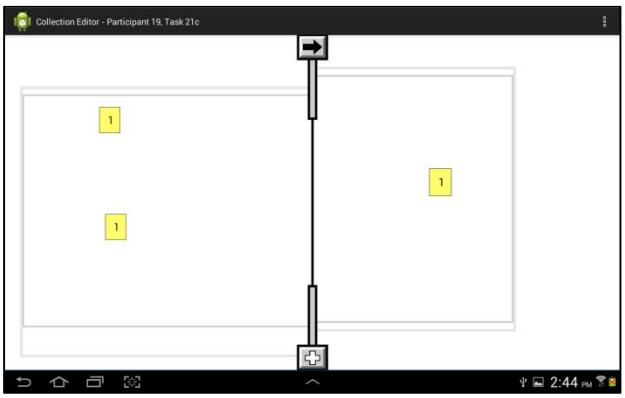


Figure 69: The result of the screen jump. Now the items can easily be grouped.

Tasks 22a through 22d contained nine text items that were scattered across the wall displays. Each text item contained a different number between 1 and 9. Users were asked to create links between the text items so that the items created a counting chain (1 links to 2, 2 links to 3, 3 links to 4, etc.). As can be seen from Figure 61, users looked at the wall display during these tasks more than any other. We feel that users looked at the wall display during these tasks for the same reason that users looked at the wall display on the last items of tasks 21a through 21d. So long as users could interact with the items that they could see on their handheld device's display, they did

not look at the wall displays. However, as soon as users needed to work with an item that they knew existed, but they didn't know where it was located, they would look at the wall displays.

All of our findings pertaining to why users look at the wall displays can be summarized in two points. Users look at while displays while editing a collection of items to:

- Gain an initial overview of the data items available in the current project,
- Find an item that the user knows exists somewhere in the project, but at an unknown location.

In our user studies, users would often look at the wall displays when a task was loaded, but never look at it again. We feel that this behavior directly relates to real working scenarios as well. It is easy to imagine a user loading up a project and looking at each wall display to try to remember where all of their items are and what their relationships are to each other. Once the user has built a mental model of the data they are working with, they will then look down at their handheld device and begin working with the items in the project. This point also relates to users navigating to a new set of items within the same project. If we consider the schedule creator example from earlier, if the development manager decides that she needs to talk to the testing manager, she will probably look at the wall display that contains the testing manager's milestones before she begins talking to the testing manager. This way, she can quickly gain an overview of what the testing manager has been working on before they begin talking about the relationships that need to be created between each other's items.

The second point summarizes the situation where a user has a memory of seeing an item in the current project but can't remember where the item is located. Instead of trying to find the item on

his handheld device, the user will more likely look up at the wall displays. This is a faster search than using the handheld device, because no navigation techniques need to be performed to find the desired item. In our user study tasks, users were aware that certain items existed and they performed this type of visual search in order to find specific items. This could happen in a real-world scenario, but it is more likely that users will use this type of visual search to determine if a specific item already exists. In the mood board sample application, a designer may look through the wall displays to see if an image of a hot-air balloon already exists. If she finds the picture, then she can use a screen-jump navigation technique to navigate to the item and to manipulate it. If she does not find a satisfactory image, then she knows that she needs to add one.

# Chapter 8. Conclusion

Handheld device users want to work collaboratively on large wall-sized displays with other handheld device users. This type of collaborative work is not possible at this time, however, because there are no applications that provide a collaborative environment for multiple users across handheld devices and wall displays. This type of work would greatly help these users by allowing them to work on data using their handheld devices with the larger data context that the wall displays can provide. Users would also be able to communicate in the context of the data on the wall displays, rather than the small display sizes of handheld devices.

To solve this problem, we provided a framework that enables collaborative work applications that incorporate handheld devices and large wall displays together. Our framework consists of a cloud data store that manages data items. These data items are shared between handheld devices and large wall displays.

Our framework also provides mechanisms that allow wall displays to share information about their size and positioning with other collaborating devices. This information is then used on handheld devices to allow users to jump to various parts of the current project based on what they find visually on the wall displays. This information also allows users to position data items relative to wall display landmarks, such as screen edges and corners.

Finally, our framework provides a set of generic interactive techniques that application developers can use to add items to a project, organize and create relationships between items, and to add comments and critiques to items.

As we look towards the future of handheld device and wall display collaborative applications, we can envision some areas that may be interesting to explore. The work we performed on helping users navigate handheld devices to items they located on the wall displays was informative, but we feel that this interaction could be greatly improved. Specifically, we feel that using depth cameras, such as Microsoft's Kinect [22], could enable users to be tracked in a three-dimensional space around the wall displays. It is possible then, that as users change the wall display that they are closest to, the system could navigate their handheld device automatically.

Another possible future work would be to use depth camera systems to provide data modifications without the handheld device. For example, if a user wanted to group two items that were spread out across a project, instead of using their handheld device, the user could simply reach out their hand and "grab" one of the items. By dragging her hand through the air and "releasing" the item next to the other data item, the user would be able to move items long distances without needing to use their handheld device at all.

Bibliography


1. Alvarez, C., Alarcon, R., and Nussbaum, M. Implementing collaborative learning activities in the classroom supported by one-to-one mobile computing: A design-based process. *Journal of Systems and Software 84*, 11 (2011), 1–27.

2. Apitz, G. and Guimbretière, F. CrossY: A crossing-based drawing application. *UIST 04 Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, ACM (2004), 3–12.

3. Bier, E.A. and Freeman, S. MMM: A user interface architecture for shared editors on a single screen. *Work 91*, (1991), 79–86.

4. Clayphan, A., Collins, A., Ackad, C., Kummerfeld, B., and Kay, J. Firestorm: A brainstorming application for collaborative group work at tabletops. *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces ITS 11*, ACM Press (2011), 162.

5. Foster, G. and Stefik, M. Cognoter: Theory and practice of a colab-orative tool. *Proceedings of the 1986 ACM Conference on Computer Supported Cooperative Work*, (1986), 7–15.

6. Ishii, H., Kobayashi, M., and Grudin, J. Integration of Inter-Personal Space and Shared Workspace: ClearBoard Design and Experiments. *Proceedings of the 1992 ACM Conference on Computer Supported Cooperative Work - CSCW '92*, ACM Press (1992), 33–42.

7. Izadi, S., Brignull, H., Rodden, T., Rogers, Y., and Underwood, M. Dynamo: A public interactive surface supporting the cooperative sharing and exchange of media. *UIST 03 Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology 5*, 2 (2003), 159–168.

8. Johanson, B., Fox, A., and Winograd, T. *The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms*. IEEE Educational Activities Department, 2002.

9. Johanson, B., Hutchins, G., and Winograd, T. PointRight: A System for Pointer/Keyboard Redirection among Multiple Displays and Machines. (2000).

10. Myers, B.A., Stiel, H., and Gargiulo, R. Collaboration using multiple PDAs connected to a PC. *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work CSCW 98 98*, (1998), 285–294.

11. Myers, B.A. Using handhelds and PCs together. *Communications of the ACM 44*, 11 (2001), 34–41.

12. Ohkubo, M. and Ishii, H. Design and implementation of a shared workspace by integrating individual workspaces. *Proceedings of the ACM SIGOIS and IEEE CS TCOA Conference on Office Information Systems*, ACM, New York, NY, USA (1990), 142–146.

13. Olsen, D.R. Evaluating user interface systems research. *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology - UIST '07*, ACM Press (2007), 251.

14. Pedersen, E.R., McCall, K., Moran, T.P., and Halasz, F.G. Tivoli: An electronic whiteboard for informal workgroup meetings. *Writing*, ACM (1993), 391–398.

15. Rodgers, Y. Dynamo. http://www.informatics.sussex.ac.uk. Retrieved June 29, 2013, from http://www.informatics.sussex.ac.uk/research/groups/interact/previousSite/projects/dynamo.htm

16. Stefik, M., Bobrow, D.G., Foster, G., Lanning, S., and Tatar, D. WYSIWIS Revised: Early experiences with multiuser interfaces. *ACM Transactions on Information Systems 5*, 2 (1987), 147–167.

17. Stefik, M., Foster, G., Bobrow, D.G., Kahn, K., Lanning, S., and Suchman, L. Beyond the Chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM 30*, 1 (1987), 32–47.

18. Streitz, N.A., Geibler, J., Holmer, T., et al. i-LAND: An Interactive Landscape for Creativity and Innovation. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: The CHI is the Limit*, ACM (1999), 120–127.

19. Watanabe, N., Washida, M., and Igarashi, T. Bubble clusters: An interface for manipulating spatial aggregation of graphical objects. *Human Factors*, ACM (2007), 173–182.

20. Whiteside, J. and Wixon, D. Contextualism as a world view for the reformation of meetings. *Proceedings of the 1988 ACM Conference on Computer Supported Cooperative Work CSCW 88*, (1988), 369–376.

21. XBox 360 + Kinect. *Xbox.com.* Retrieved May 15, 2013, from http://www.xbox.com/en-US/kinect.

22. Zurita, G. and Nussbaum, M. Computer supported collaborative learning using wirelessly interconnected handheld computers. *Computers & Education 42*, 3 (2004), 289–314.