HYDRASCOPE: ADAPTING EXISTING WEB APPLICATIONS FOR MULTI--DISPLAY WALLS

Viraj Kulkarni
Björn Hartmann

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-135
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-135.html

May 30, 2012

Acknowledgement

I would like to thank my advisor, Bjoern Hartmann, and my project partners, Hong Wu and Yun Jin for their support and encouragement. I also want to thank Michel Beaudouin-Lafon and Wendy Mackay for their continuous guidance.

HYDRASCOPE: ADAPTING EXISTING WEB APPLICATIONS FOR MULTI-DISPLAY WALLS

By

Viraj Kulkarni

May 2012

TABLE OF CONTENTS

ABSTRACT **3**

INTRODUCTION **3**

RELATED WORK **7**

MULTI-DISPLAY ARCHITECTURES 7
USER INTERACTION ON MULTI-DISPLAY WALLS 8
REVERSE ENGINEERING INTERFACES 9

DESIGN **9**

MOBILE REMOTE CONTROL 10
APPLICATION INSTANCE MANAGER 11
SYNCHRONIZATION SERVER 12
SYSTEM INPUT MANAGER 12

IMPLEMENTATION **13**

MOBILE REMOTE CONTROL 13
APPLICATION INSTANCE MANAGER 14
SYNCHRONIZATION SERVER 15
SYSTEM INPUT MANAGER 15

HYDRASCOPE APPLICATION DEVELOPMENT **15**

EXAMPLE APPLICATIONS **17**

PRESENTATION VIEWER 17
MULTI-DISPLAY STOCKS 18
TILED MAPS 18
AGGREGATED SEARCH 19
COLLABORATIVE TEXT EDITOR 19

EVALUATION **20**

USER INTERACTION WITH APPLICATIONS 20
APPLICATION DEVELOPMENT PROCESS 21

DISCUSSION AND FUTURE WORK **22**

CONCLUSION **23**

REFERENCES **24**

ABSTRACT

Although large wall-sized displays are becoming increasingly available, their rate of adoption in research and business environments has been limited due to (1) the high cost of developing applications that scale to cluster-driven displays and (2) lack of interaction techniques for multi-user input on such shared displays. In this report, we introduce Hydrascope, a framework for creating multi-view *meta-applications* for cluster-driven displays by adapting existing web applications without modifying their source code. Hydrascope meta-applications work by running multiple instances of an application in parallel and synchronizing their views. We demonstrate the capabilities of our framework with five example applications. We also report on informal evaluations of a developer writing a Hydrascope meta-application, and five pairs of users interacting with our example meta-applications.

INTRODUCTION

High-resolution multi-display walls are becoming increasingly available due to reducing hardware costs. Such display walls support resolutions that are an order of magnitude higher than standard personal computing workstations and offer opportunities in the fields of large data visualization and collaboration. A big hindrance for adoption of wall-sized displays, however, is the scarcity of applications that can take advantage of them. Such displays are usually assembled by tiling a number of smaller displays together in the form of a grid. Since there are

limitations on the number of displays that can be connected to a single computer, they often need computing clusters to drive them and are commonly used in multi-user environments. Most existing applications are developed for a single user using a personal display connected to a single computer and, hence, they often do not scale to such shared cluster-driven displays.

A number of UI frameworks exist for developing applications to take advantage of the high resolution and shared nature of these displays, such as jBricks [14] and Shared Substance [6]. However, rewriting applications from scratch is expensive and requires a significant amount of engineering expertise and effort.

Our project takes a complementary position and investigates how existing web applications can be adapted to run on multi-display walls without modifying their source code. Most web technologies follow design patterns such as MVC[1] that promote separation of the *data model* and *view* components of the system. We take advantage of the fact that an increasing number of web applications are now designed to support multiple views that work on the same data model. For example Google Docs[2] and Zoho Office[3], among others, enable multiple instances of an application to edit the same data simultaneously. These projects are based on a long lineage of research in collaborative editors [15]. Hydrascope is a framework that enables developers to instantiate multiple *views* of an application and control

[1] http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
[2] http://docs.google.com
[3] http://www.zoho.com

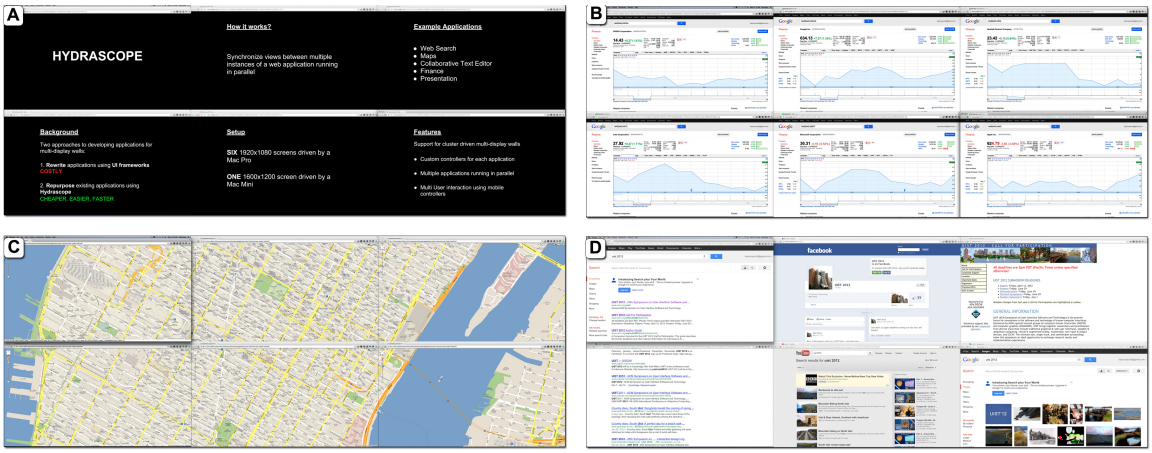

Figure 1: Four applications we built with Hydrascope: (A) A presentation viewer, (B) a stocks viewer, (C) A tiled map viewer with pan and zoom control, (D) A multi-screen search application where the left two screens show search results, and the right four screens automatically load pages from search results (here for the query ”UIST 2012”).

synchronization of these views across displays running on different computers. While such adaptations cannot provide the full benefits of rewriting applications, they have the advantages that they are much faster to produce, and can be used for applications for which source code is not available. Users continue to operate the existing software packages that they are familiar with and gain additional value from additional displays. The primary limitation here is that any approach that does not modify source code can only control or modify the functionality that is exposed in the existing view of the application. We restrict our investigation to ‘multiple view interfaces’ [17] that shows two or more views of a single concept or document, as we believe such interfaces are possible to produce using Hydrascope.

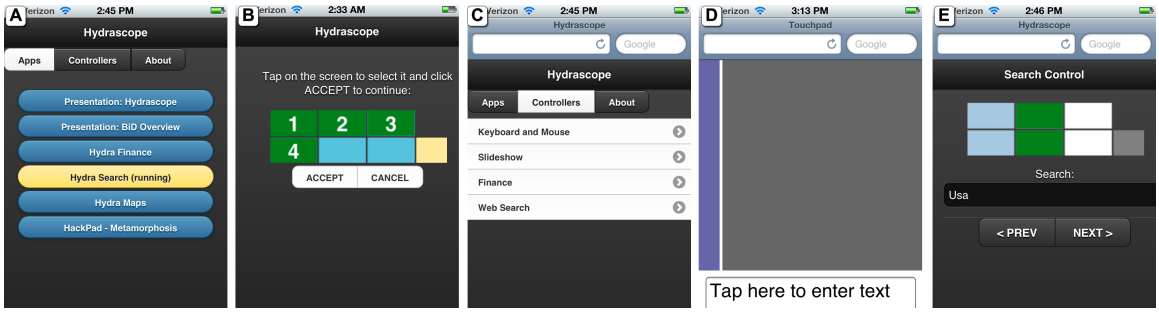

Figure 2: (A) Hydrascope application Launchpad. When users tap on an application that is not yet running, they can (B) assign available screens to that application. (C) When an application has launched, users can select an appropriate control interface: either (D) a touch pad and text entry controller; or (E) application-specific controllers. The search controller shown here enables users to "pin" search results to screens, and to enter new query terms without having to first navigate to the wall's on screen query box.

Human interaction with shared wall-sized displays is different from interaction with smaller personal displays. Personal displays are operated using single-user devices such as keyboard and mice that require a fixed supporting surface. On the other hand, wall-sized displays call for multi-user input as well as user mobility – users need to move about to observe different parts of the display from different distances. To support these needs, Hydrascope features a *mobile remote control* (Fig. 2) that allows users to manage applications and interact with them.

In this report, we present the motivation behind Hydrascope and introduce related concepts. We describe the design and implementation of the system and outline the process of developing applications using Hydrascope. We demonstrate the capabilities of the framework with five example applications: a slides viewer, a stock chart navigator, a tiled map, a search application, and a multi-user text editor (Fig. 1). We evaluate the system from the standpoint of both developers and users with

an informal study involving a developer using the framework to repurpose an application, and 10 different users interacting with our example applications in pairs. We conclude by discussing the challenges and limitations of our approach and our future vision for Hydrascope.

RELATED WORK

Hydrascope builds on prior work in three main research areas: multi-display architectures, multi-user interaction in collaborative environments and reverse engineering of existing applications.

Multi-display Architectures

Early work on developing visualization applications for clustered displays, such as Chromium [9], focused on using low-level graphics APIs such as DirectX and OpenGL. They worked well for high performance applications, but were difficult to develop and required a large engineering effort. The approach of running multiple instances of an application simultaneously on different screens and synchronizing their views has been studied. jBricks [14] is a Java toolkit that integrates a graphics rendering engine and an input configuration module into a coherent framework enabling rapid development of visualization applications. Shared Substance [6] provides a middleware that offers data sharing abstractions and has been use for developing multi-display applications. WeSpace [18] is a collaborative workspace that integrates a large data wall with a multi-user multi-touch table. An alternative to this distributed rendering approach is pixel streaming, as used in the SAGE

environment[4], where applications are run on a single machine and their outputs are streamed to display servers using protocols such as VNC [16]. The drawbacks of this approach include high bandwidth requirements and visual artifacts caused by scaling images. Hydrascope uses a version of the distributed rendering approach by running a copy of the application on each display server. Unlike previous work, it does not require access to the source code of the application.

User Interaction on Multi-display Walls

Small personal displays are operated using single-user devices like keyboard and mice that require a fixed supporting surface. Wall-sized displays, due to their larger size and shared nature, call for multi-user interaction techniques that allow users to move freely in front of the displays while interacting with them. Nancel et al. [13] study different families of location independent mid-air interaction techniques for pan-zoom navigation on wall-sized displays. For interacting with distant unreachable display areas, Khan et al. [11] proposed a new widget known as a "Frisbee", whereas Boring et al. [3] proposed using a live video of the display taken by a mobile phone. Malacria et al. [12] proposed clutch-free panning and integrated pan-zoom control on touch-sensitive surfaces by drawing circles. We did not incorporate these techniques in our mobile controllers, but we may do so in the future.

Supporting multi-user interaction is a challenging problem since the existing infrastructure consisting of hardware devices, operating systems and applications

[4] http://www.sagecommons.org

has been designed for single user input using mice or keyboards. PointRight [10] distributes multi-user mouse and keyboard inputs to multiple screens running across multiple computers. Since the existing infrastructure supports a single mouse cursor, these systems either multiplex the cursor, such as in Mighty Mouse [2] or support multiple cursors for specially designed applications, such as in Mischief [1]. Hydrascope uses cursor multiplexing for supporting multi-user input.

Reverse Engineering Interfaces

For programmatic interaction with applications, developers need to reverse engineer the interfaces to extract state information or execute interaction events. One approach includes extracting information from the code structure of the application, such as d.mix [8] that extracts DOM[5] information from webpages or Scotty [5] that programmatically queries the state of desktop applications on Mac OS X. Another approach is to apply computer vision techniques to screen pixels, such as Sikuli [19] and Prefab [4]. Hydrascope is closely related to DOM based reverse engineering such as d.mix.

DESIGN

Wall-sized screens usually require multiple display servers[6] to drive them. Hydrascope works by running one or more instances of the application on each display server, and synchronizing the views of these instances across all servers. The Hydrascope system comprises of four components (1) Mobile Remote Control,

[5] http://en.wikipedia.org/wiki/Document_Object_Model
[6] A display server is any computer in the environment that controls one or more display surfaces.

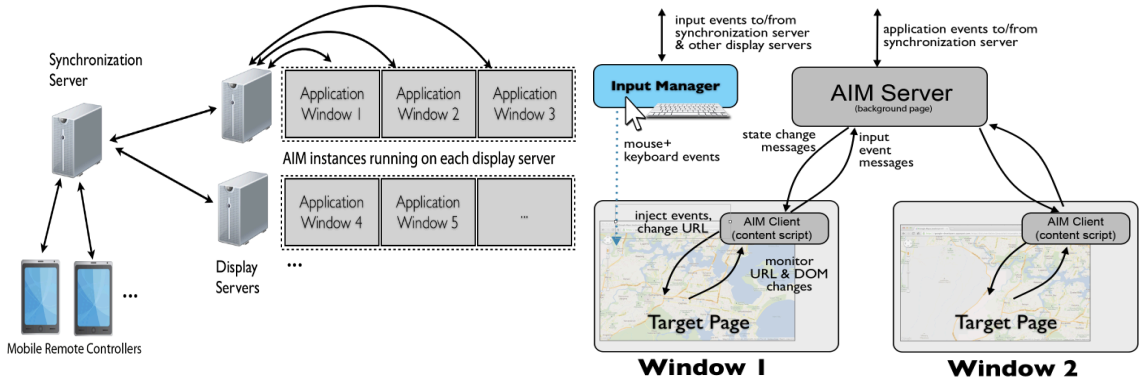

Figure 3: (left) Global architecture of Hydrascope (right) Local architecture on a single display server

(2) Application Instance Managers, (3) Synchronization Server and (4) System Input Manager (Fig. 3). The *Mobile Remote Control* is an application that enables users to launch and manage applications on the multi-display wall and interact with them. The *Application Instance Manager* (AIM) is a process that runs on each display server and is responsible for launching multiple application instances and synchronizing them. The *Synchronization Server* is a central server that facilitates communication between the *Mobile Remote Control* and the AIMs. The *System Input Manager* (SIM) runs on each display server and executes system-level mouse and keyboard events on that server.

MOBILE REMOTE CONTROL

The mobile remote control consists of an *Application Launchpad* for launching and killing applications, the *Screenscape* screen manager for assigning different parts of the wall to different applications, and a set of *controllers* for interacting with the multi-display wall. On running the mobile remote control, the *Launchpad* presents the user a list of installed applications. On selecting an application to launch,

Screenscape displays a screen map that allows the user to assign screens to the application. The user then launches the application. This process is illustrated in Fig. 2. The application configurations and screen topologies are stored on the synchronization server.

The mobile remote control supports a keyboard and mouse controller that can be used to redirect text and mouse events to the multi-display wall. This controller provides an interface for low-level input such as mouse click, drag, scroll and pinch-zoom. For applications that can benefit from more sophisticated forms of interaction, Hydrascope also supports custom controllers. These controllers provide high-level interfaces for application specific tasks that otherwise require the user to perform a series of low-level mouse interactions to accomplish. This can be seen in our example search application where the user pins search results to screens using the custom controller (Fig. 2E). Custom controllers are especially important in shared environments where multiple users may simultaneously perform different tasks.

APPLICATION INSTANCE MANAGER

Every display server runs one instance of the AIM for each application. The AIM is responsible for (1) launching multiple application instances on the display server, (2) listening for view changes in each instance and synchronizing changes across all instances and (3) executing input events and application specific interactions performed using the mobile controllers (Fig 3, right). There are two components to an AIM: (1) a system level *AIM Server*, one per display server and (2) multiple *AIM*

Clients, one per application instance. The *AIM Server* launches multiple application instances and handles their synchronization. There is an *AIM Client* for every application instance that the AIM Server launches, and it is responsible for monitoring view changes and executing interaction events for that instance. The AIM clients execute application-level interaction events, while the *system input manager* executes system-level interactions such as mouse and keyboard events. Synchronizing application instances running on different display servers needs communication between AIMs running on these servers and is handled by the *synchronization server*.

SYNCHRONIZATION SERVER

Instances of an AIM running on different display servers are synchronized by the synchronization server. The synchronization server also facilitates message passing between the mobile remote control and the AIMs. This server resides on a networked machine that has an IP address accessible by all display servers and mobile devices.

SYSTEM INPUT MANAGER

The *system input manager* is a separate module running on each display server that executes system level events such as text entry and mouse click, drag, scroll and zoom.

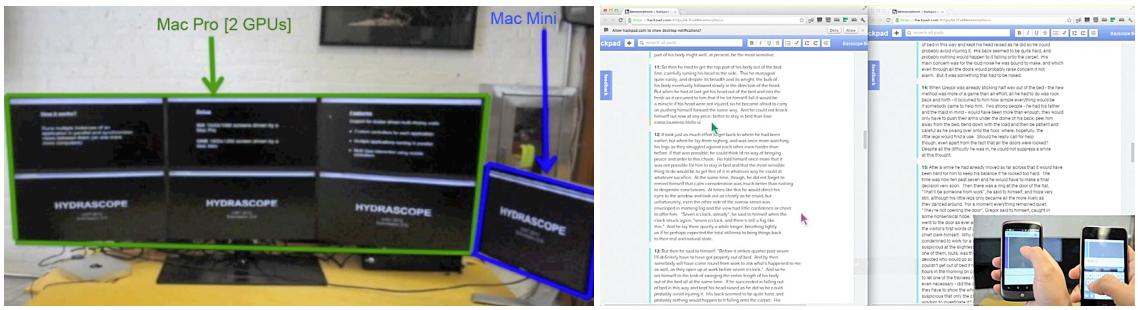

Figure 4: (left) In our lab, a Mac Pro drives 6 monitors with a total resolution of 5760x2160 and a Mac Mini drives a single display of 1600x1200 (right) Our collaborative document editor enables multiple users to edit a single document simultaneously by providing input from their mobile devices.

IMPLEMENTATION

In our lab, we use a setup with two display servers. A Mac Pro with 2 GPUs drives a 3x2 tiled display of 6 monitors with a total resolution of 5760x2160 (1980x1080 per monitor). A Mac Mini drives a single monitor with a resolution of 1600x1200. We use Android and iOS devices to run the mobile controllers.

MOBILE REMOTE CONTROL

The *mobile remote control* is packaged as a web application written in HTML5 and jQuery Mobile[7]. It is optimized to run on mobile devices with the form factor of a phone, but can be used from any device that supports a web browser. The *remote control* features the *Screenscape* screen manager that is responsible for managing screen ownership for applications and resolving screen assignment conflicts that occur frequently between users in multi-user environments.

[7] http://jquerymobile.com/ is an HTML5 web development framework for mobile devices

The *remote control* also supports custom controllers for application specific tasks. These are supplied by the application developer as standalone HTML files. At present, communication between the controllers and the AIMs must be managed entirely by the developer. In future, we plan to standardize this process by encapsulating it in the form of an API.

APPLICATION INSTANCE MANAGER

We developed the AIMs as Google Chrome Extensions[8]. The extension *background page* performs the task of the AIM Server launching multiple application instances in the form of chrome windows with each window running one instance of the application. It also redirects custom input events from the mobile controllers to these windows. We used *Chrome content scripts* to perform the tasks of *AIM Clients.* The *content scripts* listen for events and *view* changes in webpages in these windows and communicate these changes to the *background page,* which then synchronizes all other windows with these changes. Monitoring the webpage for *view* changes is performed by extracting *view* information by reverse engineering the DOM of the webpage. The background page also communicates these changes to other background pages running on different display servers through the *synchronization server*.

[8] http://support.google.com/chrome/bin/answer.py?hl=en&answer=154007

SYNCHRONIZATION SERVER

We implemented the communication server using *NodeJS*[9] and *socket.io*[10]. There is no application specific code inside the server. A server configuration file is used to specify configuration data such as topology of the multi-display wall, display servers and the list of applications and mobile controllers.

SYSTEM INPUT MANAGER

The *system input manager* is a native OS X application that runs on each display server. It receives system level interaction events such as keyboard text entry or mouse events from the *keyboard and mouse mobile controller* or any of the *application instance managers*. On receiving these events, the *system input manager* executes them on the display server.

HYDRASCOPE APPLICATION DEVELOPMENT

Hydrascope works by running multiple instances of the application in parallel and synchronizing views between them. For this approach to work, application instances need a shared coherent data model (e.g., the document that is being edited). We leverage the fact that an increasing number of web applications now come with built-in data synchronization to support multiple simultaneous collaborators. As a result, the Hydrascope framework relies on the applications' built-in data synchronization instead of providing its own mechanisms.

[9] http://nodejs.org/
[10] http://socket.io/

In order to produce a new Hydrascope application, developers need to build or provide access to (1) an existing web application that supports data synchronization (2) an Application Instance Manager (AIM) and (3) optional mobile controllers. Although we developed Google Chrome Extensions as our AIMs in our example applications, developers can write their own AIMs using any development platform. They can provide their own mobile controllers, use our built-in generic controllers or choose to use the conventional keyboard and mouse devices. They must add their application to the server configuration file to make it visible to the *Application Launchpad* in the mobile remote control.

From our observation, most applications that can benefit from multi-display walls serve one of the following purposes:

1. View different parts of a document: View different segments of the same document. Examples include our presentation and maps applications. The tiled maps example displays a different part of the map on every display.
2. View different views (operations) on a document: Display different views or results of different operations on the same document. An example would be a photo editing application that displays the same photograph with different operations performed on it on different screens.
3. Time-lapse view of a document: Show snapshots of the same document at different points in time such as different revisions of a piece of code or a series of medical scans taken at different times.

4. Compare multiple documents: Compare different documents such as stock quotes of different companies.

Often applications fall in multiple categories listed above.

EXAMPLE APPLICATIONS

To demonstrate the capabilities of Hydrascope, we developed five example applications: presentation viewer, stocks viewer, maps viewer, search results aggregator and collaborative text editor (Fig. 1).

PRESENTATION VIEWER

We developed a presentation viewer for multi-display walls by repurposing the Google Docs Presentations web application. We display one slide per monitor. The user can navigate slides with the help of the custom controller or by using the mouse pointer to click on the previous and next buttons on the webpage.

The AIM's content script adds event handlers to the previous and next buttons on the Google Docs webpage to monitor slide change events, and notifies the background page when the event occurs. The background page, on receiving this event, notifies all other content scripts attached to instances of the Google Docs application and also the communication server. The communication server passes this event to the background pages running on other display servers. In this fashion, all running instances of the application are notified of view changes and synchronized across multiple display servers. Events generated using the mobile

controller are transmitted to the communication server, which then relays them to all background pages for the presentation application.

MULTI-DISPLAY STOCKS

Our stocks viewer application is based on Google Finance web application. We run multiple instances of the Google Finance application in different windows, one per monitor. Each instance shows a stocks chart of a different company during the same timeframe. Users can use the scrollbar on the webpage to change the timeframe, or they can use the mobile controller that features previous and next buttons similar to the presentation controller.

The Google Finance webpage uses URL parameters to set the timeframe for which it displays the stocks chart. The AIM uses content scripts to listen for view changes by tracking an embedded link in the page that exists sharing purposes. On detecting changes in this link, the AIM reloads all windows with an updated URL that reflects this change. These page refreshes are slow and may take up to a couple of seconds. Since the stocks charts are written in flash, they do not offer an accessible interface to send parameters.

TILED MAPS

We tile multiple instances of Google Maps across the multi-display wall to make a single giant map. Users can perform interactions such as panning and zooming using the *keyboard and mouse mobile controller*. We use Google Maps API to update views for different instances.

AGGREGATED SEARCH

The search application allows users to take advantage of multiple displays to perform web search. The two leftmost screens in our setup show the search result list of Google Search, while the remaining screens are used to display the webpages from the results. The two leftmost screens showing the results support synchronized scrolling to give the appearance of a single scrolling list, so that the results scroll across the physical monitors.

Users navigate results using the custom search controller. A new set of results can be loaded by pressing next or previous buttons. Users can enter a new search query using a textbox. The controller also displays a screen map of all the screens assigned to the search application. Users can tap on the screen icon to 'pin' or 'unpin' a result to a screen. Pinned results are not replaced on pressing previous and next buttons or on entering new search queries.

COLLABORATIVE TEXT EDITOR

Our text editor application lets multiple users simultaneously edit a text document (Fig. 4-right) Different pages are shown on different monitors and they support synchronized scrolling similar to the search application. Multiple users can simultaneously enter text using the keyboard and mouse controller.

Figure 5: An information visualization researcher adapted a visualization of Tour de France race data to create this Hydrascope application

EVALUATION

USER INTERACTION WITH APPLICATIONS

We conducted an informal user study involving 10 participants aged between 20 and 30 years to gather feedback about the applications and controllers. All of them were familiar with computers and had previously used some of the applications on single displays. The participants were grouped in pairs and were asked to complete predefined tasks.

Mobile Controllers: Users liked the mobility of the mobile controllers as it allowed them to move around while interacting with applications. They found using custom controllers convenient because it often took them a single tap to accomplish what otherwise required a combination of cursor movement, pointing and clicking. For other applications, however, users preferred the conventional mouse since it was faster and more accurate than our keyboard and mouse controller.

Using a multi-display wall vs. a single display: Users preferred multiple displays for certain applications because it allowed them to view and compare a number of

documents simultaneously. Another benefit was a reduction in the number of window-switches performed for completing the given task. The disadvantages included distance and excess information.

Our implementation of the keyboard and mouse controller executed mouse events by capturing the system cursor. Since multiple users shared the system cursor, this gave rise to conflicts and unexpected results. Due to this model, only one user could enter text at a time. This limitation prevented us from exploring the full collaborative potential of Hydrascope applications.

APPLICATION DEVELOPMENT PROCESS

We recruited a graduate student with prior experience developing web applications for our study. He repurposed a visualization tool for Tour de France that shows stage timing data for each cyclist. He adapted this application to show data for six different cyclists on six displays. On changing the stages selected for a cyclist in one window, the other windows were automatically updated to reflect the new stages for the other cyclists. This task was similar to the stocks example application, where the required information (selected stages and cyclist in this case) was passed to the application through URL parameters.

The participant spent some time browsing the code of our example applications. His familiarity with developing web applications was beneficial and he completed the task in 2 hours 20 minutes. He did not develop any custom controllers. The participant encountered some difficulties due to peculiarities of the Chrome

development environment. For example, during debugging he sometimes modified the source code but forgot to reload the extensions. Since Chrome does not automatically reload extensions, he wasted time debugging stale copies of the code.

From these observations, we realized that we could encapsulate some of the message passing code and expose APIs to accomplish those tasks. In our present version, it is the developer's responsibility to ensure that his application does not process messages that were not meant for it. We could relieve the developer of this responsibility by performing intelligent message routing in our *communication server*. Lastly, we could provide a generic Chrome extension template that has the boilerplate code necessary for common tasks. The developer can then derive his AIM from this template rather than having to build his own.

DISCUSSION AND FUTURE WORK

Hydrascope is an attempt to ease the adoption barrier for multi-display walls by reducing the engineering investment required in developing applications for them. We identified areas of possible improvements from our own experience with using the framework and the subsequent user study. A more robust communication protocol and intelligent message routing in the server will make future development of more sophisticated applications possible. Providing extensible APIs for reusing the functionality of the *Screenscape* manager will enable developers to develop sophisticated custom controllers with capabilities of manipulating the multi-display layouts. Presently, users must explicitly assign screens to applications when

launching them. In the future, we plan to augment the *Screenscape* manager with functions for intelligent screen management and window positioning on the display wall similar to those discussed in [7] to make it robust enough to support dynamic reconfiguration. This will be especially helpful since users do not plan ahead and anticipate which applications they will need to launch on what screens. From the standpoint of developers, encapsulating the message passing code in the form of a library and providing templates for the boilerplate code will reduce time spent in navigating peculiarities of the Chrome extensions environment.

There are two challenges in improving user interaction with Hydrascope applications. First, we need to find ways of providing multi-user input to single-user applications designed to be driven by a single mouse pointer. Second, we need to engineer ways to redesign the mouse controller so that it offers speed and accuracy comparable to conventional mice and trackpads.

Lastly, we want to extend Hydrascope to repurpose native applications along with web applications using pixel scraping or code injection techniques.

CONCLUSION

As large multi-display walls become increasingly available, there is a growing need for a framework to rapidly develop applications that scale to such shared cluster-driven displays. Instead of developing applications from scratch, we presented the Hydrascope framework that allows developers to repurpose existing web

applications for multi-display walls in a matter of hours. We discussed the design and implementation of Hydrascope and demonstrated its capabilities with five example applications. We evaluated our system from the standpoint of both developers and users. We concluded by discussing the impact of Hydrascope, challenges and limitations of our approach and areas of future work.

REFERENCES

1. S. Amershi, M. R. Morris, N. Moraveji, R. Balakrishnan, K. Toyama. Multiple mouse text entry for single display groupware. In Proc. Computer Supported Coperative Work, CSCW ’10, 169–178. ACM, 2010.

2. K. S. Booth, B. D. Fisher, C. J. R. Lin, and R. Argue. The ”mighty mouse” multi screen collaboration tool. In Proc. User Interface Software and Technology, UIST ’02, 209–212. ACM, 2002.

3. S. Boring, S. Gehring, A. Wiethoff, M. Blöckner, J. Schöning, A. Butz. Mobile Interaction Through Video. In Proc. Human Factors in Computing Systems, CHI '10, 2287 – 2296. ACM, 2010

4. M. Dixon and J. Fogarty. Prefab: implementing advanced behaviors using pixel based reverse engineering of interface structure. In Proc. Human Factors in Computing Systems, CHI ’10, 1525–1534. ACM, 2010.

5. J. R. Eagan, M. Beaudouin-Lafon, and W. E. Mackay. Cracking the cocoa nut: user interface programming at runtime. In Proc. User Interface Software and Technology, UIST ’11, 225–234. ACM, 2011.

6. T. Gjerlufsen, C. N. Klokmose, J. Eagan, C. Pillias, M. Beaudouin-Lafon. Shared Substance: developing flexible multi-surface applications. In Proc. Human Factors in Computing Systems, CHI ’11, 3383–3392. ACM, 2011.

7. F. Guimbretikre, M. Stone, T. Winograd. Fluid Interaction with High-resolution Wall-size Displays. In Proc. User Interface Software and Technology, UIST '01, ACM, 2001

8. B. Hartmann, L. Wu, K. Collins, and S. R. Klemmer. Programming by a sample: rapidly creating web applications with d.mix. In Proc. User Interface Software and Technology, UIST ’07, 241–250. ACM, 2007.

9. G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream processing framework for interactive rendering on clusters. In Proc. Computer Graphics and Interactive Techniques, SIGGRAPH ’02, 693–702. ACM, 2002.

10. B. Johanson, G. Hutchins, T. Winograd, and M. Stone. PointRight: experience with flexible input redirection in interactive workspaces. In Proc. User Interface Software and Technology, UIST ’02, 227–234. ACM, 2002

11. A. Khan, G. Fitzmaurice, D. Almeida, N. Burtnyk, G. Kurtenbach A remote control interface for large displays. In Proc. User Interface Software and Technology, UIST '04, 127-136, ACM 2004

12. S. Malacria, E.Lecolinet, Y. Guiard Clutch-free panning and integrated pan-zoom control on touch-sensitive surfaces: the cyclostar approach, In Proc. Human Factors in Computing Systems, CHI '10, ACM, 2010

13. M. Nancel, J. Wagner, E. Pietriga, O. Chapuis, W. Mackay. Mid-air Pan-and-Zoom on Wall-sized Displays. In Proc. Human Factors in Computing Systems, CHI '11, 177 – 186. ACM, 2011

14. E. Pietriga, S. Huot, M. Nancel, and R. Primet. Rapid development of user interfaces on cluster-driven wall displays with jBricks. In Proc. Engineering Interactive Computing Systems, EICS ’11, 185–190. ACM, 2011.

15. A. Prakash. Group editors. In M. Beaudouin-Lafon, editor, Computer Supported Cooperative Work, Trends in software, 103–133. Wiley, 1999.

16. T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper. Virtual network computing. Internet Computing, IEEE, 2(1):33 –38, jan/feb 1998.

17. M. Q. Wang Baldonado, A. Woodruff, and A. Kuchinsky. Guidelines for using multiple views in information visualization. In Proc. Advanced Visual Interfaces, AVI ’00, 110–119. ACM, 2000.

18. Daniel Wigdor, Hao Jiang, Clifton Forlines, Michelle Borkin, Chia Shen WeSpace: The Design, Development, and Deployment of a Walk-Up and Share Multi-Surface Collaboration System, In Proc. Human Factors in Computing Systems, CHI '09, ACM, 2009

19. T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: using GUI screenshots for search and automation. In Proc. User Interface Software and Technology, UIST ’09, 183–192. ACM, 2009.