HydraScope: Creating Multi-Surface Meta-Applications Through View Synchronization and Input Multiplexing

Björn Hartmann
Computer Science Division
University of California
Berkeley, USA
bjoern@eecs.berkeley.edu

Michel Beaudouin-Lafon
in|situ|
Université Paris-Sud
Institut Universitaire de France
mbl@lri.fr

Wendy E. Mackay
in|situ|
INRIA Saclay
France
mackay@lri.fr

ABSTRACT

As computing environments that combine multiple displays and input devices become more common, the need for applications that take advantage of these capabilities becomes more pressing. However, few applications are designed to support such multi-surface environments. We investigate how to adapt existing applications without access to their source code. We introduce *HydraScope*, a framework for transforming existing web applications into *meta-applications* that execute and synchronize multiple copies of applications in parallel, with a multi-user input layer for interacting with it. We describe the Hydra-Scope architecture, validated with five meta-applications.

Categories and Subject Descriptors

H5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Graphical User Interfaces*

Keywords

Application architectures, Multi-screen displays, Toolkits

1. INTRODUCTION

The decreasing cost of technology has led to the rise of multi-surface environments: In the home, newly interoperable interactive TVs and tablets share content and distribute control. In school and at work, smartphones, tablets and laptops link to interactive whiteboards, video projectors and tabletop displays. In labs, high-resolution wall-sized displays offer a hundred times more pixels than a desktop computer.

Unfortunately, existing applications rarely scale to these multi-surface environments for several reasons. First, few are designed to run in a distributed environment, where different machines control different displays but offer the illusion of a single interconnected display surface. Second, many applications are restricted to single-user input even though multi-surface environments encourage collaborative work. Finally, they are written for a specific screen resolution, making it difficult to resize to significantly larger – or smaller – displays. For example, when a web browser is displayed on a wall, the user must read excessively long lines of text and traverse a long distance to reach the back button.

Figure 1: HydraScope enables construction of multi-display meta-applications, controlled by mobile devices.

One solution is to rewrite applications using specially designed user interface frameworks, such as jBricks [12]. This offers maximum control but requires a major upfront investment to re-engineer common applications. An alternative is to adapt existing applications to run in multi-surface environments – ideally, without modifying the applications' source code. Users can thus leverage their existing expertise instead of learning new, dedicated applications.

We introduce the concept of a *meta-application*, which assembles and synchronizes existing applications to run across multiple interactive surfaces. A meta-application coordinates multiple application instances by sensing changes in application state due to user input and propagating appropriate changes across applications according to specific coordination rules. Prior work has identified several promising patterns of use that meta-applications should support [1]:

- Viewing a large section of a document, e.g., a map, at full resolution, with no pixel scaling;
- Viewing "small multiples" or coordinated views from one document or related documents, e.g., several stock quotes or slides from a presentation (see Fig. 1);
- Compositing related documents, e.g., a list of search results and a subset of the resulting documents; and
- Sharing content with remote users, e.g., extending a multi-surface environment to include several locations.

To enable these scenarios, meta-applications can either present multiple synchronized instances of a single application, or orchestrate the state of different applications.

PerDis'13 June 4-5, 2013, Mountain View, CA, USA

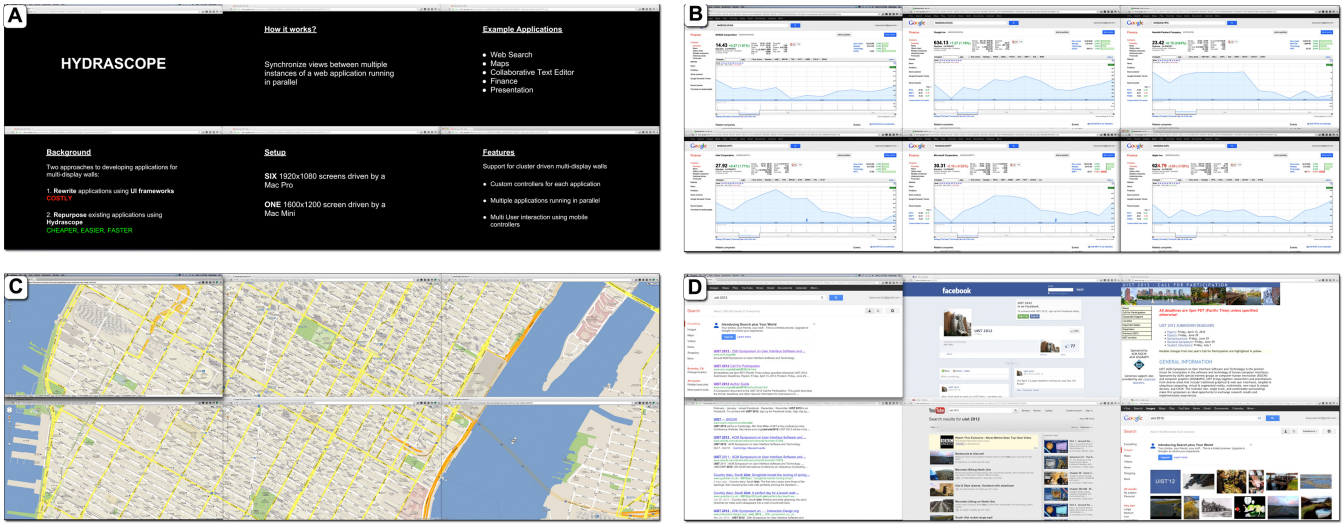

Figure 2: HydraScope applications: (A) Presentation viewer; (B) Stock viewer; (C) Tiled map viewer with pan and zoom. (D) Multi-screen search: left two screens show search results, right four screens load corresponding pages.

This paper presents *HydraScope*, our meta-application framework for web applications. HydraScope uses DOM inspection and event injection methods to coordinate web applications without having access to application source code. Any approach that does not modify source code can only control or modify some aspects of an application. To investigate what *can* be achieved, we focus on *multiple view interfaces* [14], which display several views of a single document, and on compositing related documents from different applications. By building several meta-applications (see Fig. 2), we demonstrate that our approach can be used to rapidly produce a useful range of applications.

2. HYDRASCOPE

HydraScope supports web-based meta-applications by executing multiple copies of one or more web applications in parallel and keeping the copies synchronized. HydraScope leverages the fact that many web applications already support data synchronization across multiple clients, and adds view synchronization logic to create a meta-application out of individual application instances.

Users manage and control meta-applications using the HydraScope *mobile controller* (Fig. 3A), delivered over the web to mobile devices. The *application manager* (Fig. 3B) lets users launch meta-applications and assign them to a subset of the available display surfaces (Fig. 3C). Several users can interact simultaneously with the meta-applications using HydraScope's extensible architecture: First, users can provide standard desktop input events (mouse pointing, scrolling, and text entry) to any application and display surface (Fig. 3D). Second, meta-applications can provide a dedicated interface to control specific features, such as navigating search results in a web search application (Fig. 3E).

HydraScope meta-applications must assemble a coherent user interface from multiple application instances, for multiple users. Achieving this conceptual coherence requires: (1) data synchronization, (2) view synchronization, (3) meta-application management and (4) handling multi-user input.

2.1 Data Synchronization

Application instances need a coherent, shared model of the data they present, e.g., the document being viewed or edited. In cases such as maps, on-line catalogs and search results, the underlying document is essentially static. The necessary information is often encoded in the document's URL, which can simply be shared.

For meta-applications that support document editing, real-time data synchronization among application instances is required. Fortunately, an increasing number of web applications, such as Google Docs[1], already support simultaneous editing. HydraScope takes advantage of their synchronization mechanisms to execute multiple copies of each.

HydraScope can also create meta-applications out of non-collaborative web applications by using web frameworks for real-time data sharing, such as ShareJS[2]. The developer first adds data synchronization to the application, then uses HydraScope to add view synchronization.

2.2 View Synchronization

View synchronization manages how the various instances of an application are presented: together, they must present a coherent view of the overall application. For example, view synchronization for a tiled map may entail calculating appropriate latitude and longitude offsets for each application instance to generate a single large map. Similarly, a large document may appear as a series of pages that are kept in sync as the document is scrolled.

HydraScope features an application-specific interface instance manager (IIM) that synchronizes different views using runtime interface inspection and event injection. When a user interacts with an application, the IIM senses the corresponding view changes and tells the other applications to update their views accordingly. Because these view synchronization rules are unique to each application and encapsulate the multiple display logic, they must be created by the meta-application developer. Rules can either be hardcoded

[1] `http://docs.google.com`
[2] `http://sharejs.org`

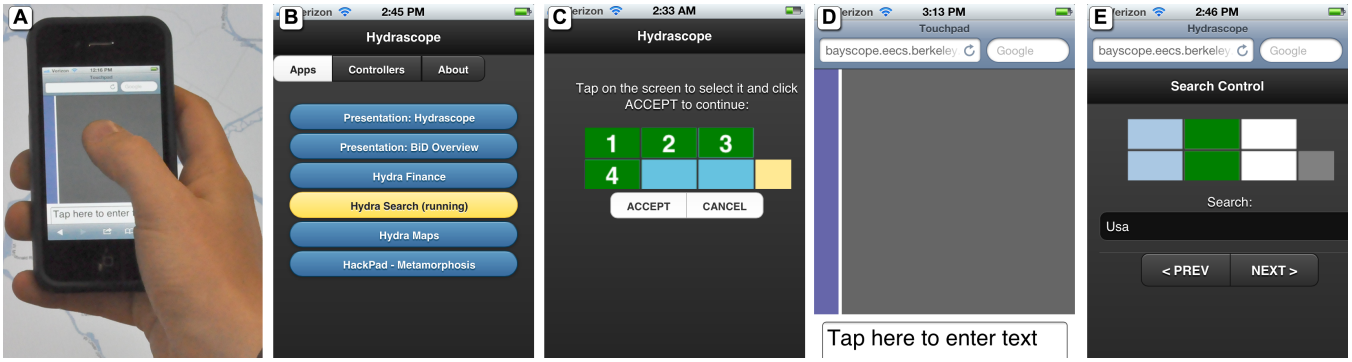

Figure 3: (A) Meta-applications can be controlled on mobile devices: (B) When users launch a new meta-application in the manager UI, they can (C) assign available screens to that application. Users can then select an appropriate control interface: either (D) a generic touch pad and text entry controller; or (E) application-specific controllers. The search controller shown here enables users to "pin" search results to screens and to enter new query terms.

in the meta-application or controlled by users at runtime via auxiliary user interfaces. For example, a meta-application may enable users to switch between a large tiled map or diverse views of the same map at runtime. Communication among interface instance managers on different machines is coordinated by a single, application-agnostic synchronization server.

2.3 Meta-Application Management and Input

Large-screen displays and multi-surface environments are especially well-suited for co-located collaboration. While traditional pointing devices can be scaled to multiple users working across multiple displays [9], users may also need to physically move in the room, e.g., to see detail on a wall display. Handheld input devices that support such movement complement desk-bound devices and offer additional displays for auxiliary interactions. Hydrascope's mobile input controllers unify two key functions:

(1) managing meta-applications (start, close, and manage display real estate), and (2) providing input to running meta-applications.

The HydraScope application manager lists the available meta-applications and lets the user choose on which displays they should run. It also lets users switch between a meta-application-specific mobile control interface and a generic interface to control cursor and text input.

Meta-application developers can define mobile control interfaces to deliver specific events to their applications. For example, a slideshow controller may provide buttons to navigate the slide deck, while a search controller may provide a search box on the mobile device so users need not point to a search box on the multi-screen display. To manage input conflicts, we use stateless commands such as next/previous slide, or temporary locks for commands that require continuous control, such as a slider to adjust the scale of a map.

Providing cursor and text input to a meta-application is challenging because it requires controlling the system cursors and keyboard foci of each computer running a display surface, allowing several users to interact simultaneously with the same or different meta-applications. HydraScope extends prior work on remote cursor control schemes to move input across different computers [9] by sharing a single system cursor among several simulated cursors, one per user, using optimistic locking: the system cursor is used only when absolutely necessary, essentially for clicks and drags, therefore minimizing (but not entirely avoiding) conflicts.

3. ARCHITECTURE & IMPLEMENTATION

The HydraScope architecture comprises four distinct components (Fig. 4): (1) application specific interface instance managers (IIM) that run on each display server[3] to monitor view changes and update views; (2) a system input manager (SIM) that runs on each display server to permit use of multiple cursors across multiple machines; (3) mobile meta-application managers to assign applications to displays and control them; (4) a master synchronization server that marshals messages across display servers and mobile controllers.

Developers write individual meta-applications by providing (1) a brief meta-application definition in the synchronization server; (2) interface instance manager code; and, optionally, (3) a mobile control user interface for their meta-application.

Our implementation uses the Google Chrome browser running on OS X, however the architectural principles are not tied to this browser or operating system. Each display server runs one or more Chrome windows. Interface instance managers (IIM) are written as browser extensions. System input managers (SIM) are native applications written in Objective-C to directly interact with the operating system's input APIs. The synchronization server is implemented in Javascript using the *node.js* event-driven I/O server[4]. The mobile devices, synchronization server and browser extensions communicate using *socket.io*[5].

Our test setup comprises two MacPro display servers, each with six screens, with a total display resolution of 11520px × 2160px.

3.1 Interface Instance Manager

On starting the meta-application, the IIM launches one or more browsers per screen, positions their windows to fill each screen, and initializes each view by loading appropriate application URLs, possibly with different parameters for each window. Because each display server can have multiple

[3]A display server is any computer in the environment that controls one or more display surfaces.

[4]`http://nodejs.org`

[5]`http://socket.io`, uses web sockets when available

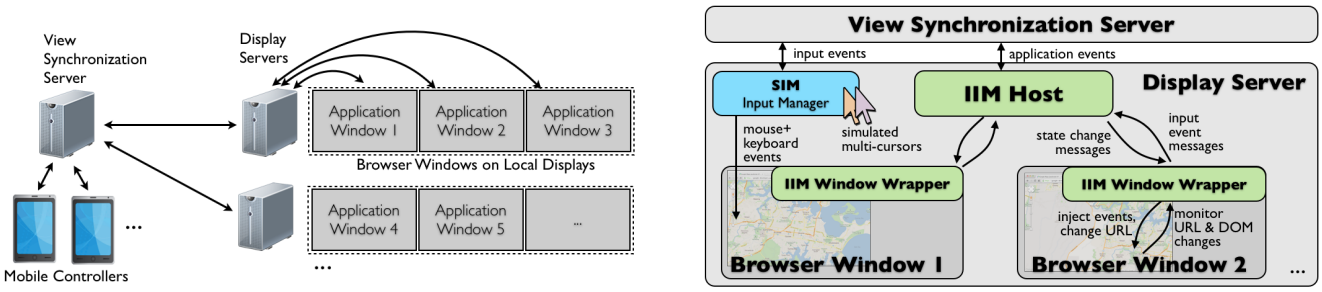

Figure 4: (Left) Global architecture of HydraScope; (Right) Local architecture on a single HydraScope display server.

screens running multiple windows, the IIM is split into a *host* component with one instance per display server, and *window wrappers* with one instance per browser window (Fig. 4 Right). Wrappers observe as users interact with the application. When actions change the application view, they pass a message to the IIM host, which then determines the proper steps for view synchronization. The IIM host then issues update commands to the other IIM wrappers on the same machine. In our implementation, the IIM host is implemented as a Chrome *background* page and the IIM wrappers are written as Chrome *content scripts*.

For multi-display environments that involve multiple display servers, HydraScope uses an additional synchronization server: IIM hosts forward the view update messages they receive from an IIM wrapper to the synchronization server, which in turn broadcasts it to all other display servers.

View Synchronization involves several strategies. Some web applications implement a *stateless* design: all parameters needed to determine the view are passed from the browser to the web application server, usually through URL parameters. For such applications, IIM wrappers can simply monitor the URL for changes. Other web applications maintain internal state. For these, a meta-application developer must reverse-engineer relevant state through inspection. We have had success in identifying relevant DOM elements and watching them for changes. For example, in a slide presentation, sensing which slide an application is currently showing can be done by parsing a label that states *Slide x of y*.

IIM wrappers must also *actuate* the interface programmatically to change the view. For stateless applications, an appropriate URL can be constructed and loaded, but this causes a complete reload of the view at each change. Alternatively, a view can be controlled by injecting appropriate input events into the application. For example, the IIM wrapper of a slideshow viewer can synthesize clicks on the next / previous slide navigation elements. Finally, an application could also expose an API to explicitly support its integration into meta-applications.

3.2 System Input Manager

The System Input Manager (SIM) manages multiple cursors across multiple display servers and allows users to send mouse and text input to the meta-applications. It is composed of a native OS X application running on each display server (*OScontrol*), a web interface running on each mobile device, and glue code running in the synchronization server.

The mobile interface features a touchpad to control the cursor, a vertical bar to control scrolling, and an input field to send text input (Fig. 3D). Since the touchpad can be quite small relative to the available display surfaces, our cursor control features both a traditional gain and a less traditional cursor inertia: when lifting the finger at the end of a quick flicking gesture, the cursor continues with an amortized motion, therefore anticipating the subsequent movement. *OScontrol* can also be used with a regular mouse and keyboard for a result similar to, e.g., PointRight [9].

OScontrol uses optimistic locking to acquire the system cursor for button and drag events: If the system cursor is not locked, the request succeeds and the events are relayed to the system cursor; If the system cursor is locked, which happens only if another user is performing a drag operation on the same display server, the operation fails silently. In practice, conflicts are rare because web applications rarely use dragging, especially since scrolling is handled separately. Applications that use dragging extensively should probably use an application-specific control interface.

3.3 Mobile Meta-Application Manager

The synchronization server maintains a global configuration of all displays in a multi-display wall or multi-surface environment, which it delivers to the mobile meta-application manager (Fig. 3B). When the user chooses a set of displays on which to run an application (Fig. 3C), the configuration is broadcast to all IIM hosts, which then load the application's URL if their screens are affected by the launch.

4. SAMPLE APPLICATIONS

We developed five meta-applications to demonstrate the HydraScope design space (see Fig. 2 and the accompanying video). These projects show that meta-applications are more like scripts than system programs and can be implemented with only a few hundred lines of JavaScript (Fig. 5).

4.1 Slide Presenter (Fig. 2A)

Users can give or review presentations on the multi-screen display, with one slide per monitor. They can navigate through the slide deck by pressing two buttons on a mobile controller or clicking on navigation controls inside each application instance. Advancing beyond the last currently visible slide shifts all the slides to the left.

The Slide Presenter builds on Google Documents' Presentation Viewer. Each monitor runs one Chrome window with one presentation page. Since each slide has a unique id, the user can start the deck at a specified slide by passing the slide's id as a URL parameter, which is used to initialize the views. For efficient updating, we inject mouse events on the "previous" and "next" buttons inside the player whenever the user presses the corresponding mobile controller buttons.

Application	IIM Code Size	Mobile UI Size
Slide Viewer	189 lines	35 lines
Stocks	158	N/A
Tiled Map	223	N/A
Wall Search	231	196
Document Editor	144	N/A

Figure 5: Code size for five meta-applications.

4.2 Stock Viewer (Fig. 2B)

Users can compare the financial performance of several stocks over time, with one stock chart per monitor. With the system cursor, the user can pan the graph or use its slider widget to change the time interval shown in one window; all other windows update to show the same time interval.

The Stock Viewer builds on Google Finance. Unlike the Slide Viewer, the stock chart is an embedded Flash object, with no easily accessible way to inject events. Instead, we reload the page with new URL parameters, which precludes interactive data "scrubbing". We take advantage of an embedded link designed to let users share their current view, checking the URL to monitor if the graph was scrolled.

4.3 Tiled Map (Fig. 2C)

Users control a tiled map that spans all monitors, using the mobile controller to pan and zoom. Users can view a single large map or "small multiples" with different data, such as roads, satellite images and topographic views.

The Tiled Map builds on Google Maps. Like the Stock Viewer, we take advantage of an embedded link that serializes the view state into URL parameters. For smooth scrolling, we use Google's JavaScript Maps API to get and set map parameters and a separate HTML page that stores map boundary and zoom information in the DOM.

4.4 Wall Search (Fig. 2D)

Users can build up a set of Google search results: The two left screens synchronize scrolling to create the illusion of a single, tall window; the four right screens each show a top search result. The user can type new terms into the query box on the mobile controller or *pin* search result pages onto its display by tapping the corresponding screen icon (see Fig.3E). The *previous* and *next* buttons switch between results, displaying them on screens not yet pinned.

Wall Search builds on Google Search. The interface instance manager maintains two types of screens: result listings and individual results pages. The IIM wrapper parses Google's results listing and extracts links to external sites. When a user searches for a new term, the IIM wrappers for both results listing screens construct the appropriate search URL, reload the page, and adjust the scroll position. A similar synchronization occurs when users arrive at the end of one results page and advance to the next.

4.5 Document Editor

Multiple users can edit a single document simultaneously, by manipulating different text carets in different windows (albeit only one text caret per window) using multiple cursors from the System Input Manager. The Document Editor builds on HackPad[6], a real-time collaborative editor that already performs the necessary data synchronization. We synchronize scroll positions so that scrolling one document window updates the other windows.

[6] `http://hackpad.com/`

5. LIMITATIONS AND GUIDELINES

We encountered a few anticipated limitations when developing meta-applications with HydraScope. Generally, reverse engineering state can be both *limited* and *brittle*. It may not be possible to detect internal application state changes that do not result in user interface changes. Web applications can also change their interface layout frequently. Such changes can break both detection and injection code written by meta-application authors. Event injection code can also fail because the web application interfaces may have different latencies based on network traffic. Finally, applications may not fully support concurrent interactions.

To support re-use as meta-applications, web applications should follow several design guidelines:

1. give access to the view state for use by a remote controller: a) encode the application's state in the URL to facilitate initial loading or latecomer synchronization; and b) store important state in documented DOM elements for monitoring by the meta-application.

2. separate the interface from the rest of the application using well-known patterns such as MVC. The Stock Viewer shows how opaque containers such as Flash make it impossible to finely monitor interaction and synchronize views if such separation is absent.

3. allow meta-applications to replicate application widgets for direct use on a remote controller. For example, Google search's query box, the Stock Viewer's slider and the Slide Presenter's navigation should be easy to migrate, instead of requiring reverse-engineering and reimplementation.

6. RELATED WORK

HydraScope builds on three main research areas: multi-display environments, multi-user input, and reverse engineering of existing applications.

6.1 Multi-Display Environments

Creating interactive applications that span multiple displays and multiple computers has been studied primarily in the context of large tiled displays and immersive environments. Early work focused on rendering performance and used OpenGL, e.g. Chromium [8]. More recent work simply runs a copy of the application on each display server, using different viewports to compose part of the overall image. jBricks [12] runs the Java-based ZVTM toolkit on a visualization cluster, while Shiffman's Most Pixels Ever[7] extends Processing programs to multiple screens. Shared Substance [5] provides sharing mechanisms to create distributed applications on wall-sized displays. In Google Earth's[8] LiquidGalaxy mode, a master instance controls several slaves to generate high-resolution interactive maps on tiled displays.

Another common approach, e.g., the SAGE environment[9], runs the application on a single machine and moves pixels to the display servers using protocols such as VNC [13]. The drawback is that scaling pixels creates blocky images, losing the benefits of high-resolution wall-size displays.

[7] `http://github.com/shiffman/Most-Pixels-Ever/`
[8] `http://earth.google.com`
[9] `http://www.sagecommons.org`

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.

6.2 Multi-User Input

Distributing input across multiple computers and managing input from multiple users is a long-standing problem in collaborative applications. MMM [2] is an early single-display groupware system that supports multiple mice, while PointRight [9] distributes mouse and keyboard input to multiple displays across multiple computers. FourBySix [6] supports multiple cursors on a tabletop driven by a single machine by drawing simulated cursors and multiplexing access to the system cursor. HydraScope uses a similar approach, except that it works with multiple computers.

While HydraScope manages a simple 2D geometry of the display surfaces, PointRight [9] manages a 2D-manifold and the Perspective Cursor [11] can manage arbitrary 3D geometry. We may consider these approaches in the future.

6.3 Reverse-Engineering Interfaces

Our work is based on reverse-engineering web applications to extract view state information. This is closely related to prior work that uses similar DOM-based reverse engineering, such as d.mix [7], or the early Multibrowser [10], which moved web content across multiple displays. To facilitate the work of meta-application developers, Hydrascope could leverage existing tools that simplify or automate access to web applications, such as ChickenFoot [3].

Other approaches use pixels instead of the DOM structure. For example, Sikuli [15] monitors application state by applying computer vision to screen pixels. Such approaches could be used to create meta-applications without access to the application source code nor the introspection facilities of web applications, enhancing the scope of HydraScope.

Another way to modify existing applications without access to their source code is code injection. Scotty [4] combines this approach with reflection in Objective-C to programmatically query the state of desktop applications. Scotty has been used to remote control applications and to teleport their display to a remote computer, and could be used by HydraScope to create meta-applications on Mac OS X.

7. CONCLUSION

Today's applications do not readily scale to multi-surface environments in which collections of display surfaces are run by different computers. Rather than rewriting applications from scratch, we introduce the concept of a *meta-application*: a collection of application instances with coordinated contents and views that are accessible to multiple users via mobile interfaces. We present HydraScope, a first step towards building and deploying meta-applications from existing web applications and demonstrate its effectiveness through five meta-applications.

Realizing the full potential of meta-applications requires important future work. First, we need to expand beyond multiple view interfaces and improve the tools used by developers to create meta-applications. Second, we need to better understand how meta-applications can meet user needs by deploying them in real settings, including remote collaboration scenarios. Finally, we need to investigate meta-applications that include native as well as web applications.

8. ACKNOWLEDGMENTS

We thank Viraj Kulkarni, Yun Jin, and Hong Wu for their contributions. This work was supported by NSF award OISE-1157574.

9. REFERENCES

[1] M. Beaudouin-Lafon, S. Huot, M. Nancel, W. Mackay, E. Pietriga, R. Primet, J. Wagner, O. Chapuis, C. Pillias, J. R. Eagan, T. Gjerlufsen, and C. Klokmose. Multisurface interaction in the WILD room. *IEEE Computer*, 45:48–56, 2012.

[2] E. A. Bier and S. Freeman. MMM: a user interface architecture for shared editors on a single screen. In *Proc. User Interface Software and Technology*, UIST '91, 79–86. ACM, 1991.

[3] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller. Automation and customization of rendered web pages. In *Proc. User Interface Software and Technology*, UIST '05, 163–172. ACM, 2005.

[4] 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.

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

[6] B. Hartmann, M. R. Morris, H. Benko, and A. D. Wilson. Augmenting interactive tables with mice & keyboards. In *Proc. User Interface Software and Technology*, UIST '09, 149–152. ACM, 2009.

[7] 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.

[8] 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.

[9] 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.

[10] B. Johanson, S. Ponnekanti, C. Sengupta, and A. Fox. Multibrowsing: Moving web content across multiple displays. In *Proc. Ubiquitous Computing*, UbiComp '01, 346–353. Springer-Verlag, 2001.

[11] M. A. Nacenta, S. Sallam, B. Champoux, S. Subramanian, and C. Gutwin. Perspective cursor: perspective-based interaction for multi-display environments. In *Proc. Human Factors in Computing Systems*, CHI '06, 289–298. ACM, 2006.

[12] 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.

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

[14] 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.

[15] 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.