PolyChrome: A Cross-Device Framework for Collaborative Web Visualization

Sriram Karthik Badam[1] **and Niklas Elmqvist**[2]
[1]Department of Computer Science and [2]College of Information Studies
University of Maryland, College Park, MD, USA
{sbadam, elm}@umd.edu

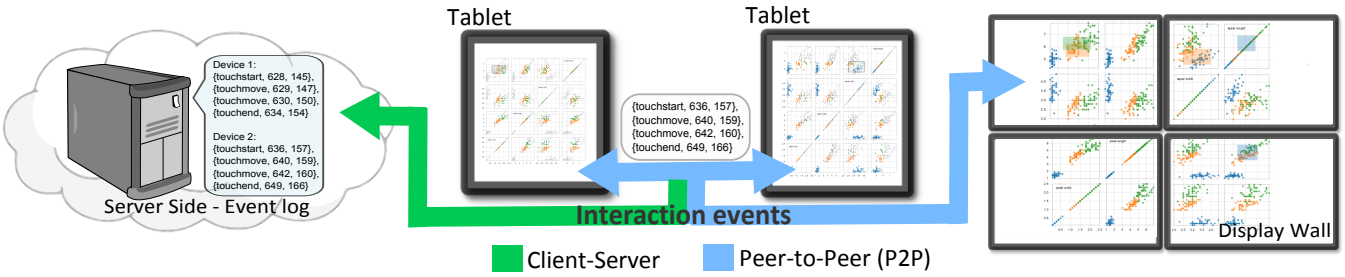

Figure 1. A web-based collaborative visualization of a scatterplot matrix of Anderson's *Iris* dataset with brush and link interaction enabled. Using PolyChrome, the brushes created on the tablets are represented on the display wall through operation distribution, and are also stored on a server.

ABSTRACT

We present PolyChrome, an application framework for creating web-based collaborative visualizations that can span multiple devices. The framework supports (1) co-browsing new web applications as well as legacy websites with no migration costs (i.e., a distributed web browser); (2) an API to develop new web applications that can synchronize the UI state on multiple devices to support synchronous and asynchronous collaboration; and (3) maintenance of state and input events on a server to handle common issues with distributed applications such as consistency management, conflict resolution, and undo operations. We describe PolyChrome's general design, architecture, and implementation followed by application examples showcasing collaborative web visualizations created using the framework. Finally, we present performance results that suggest that PolyChrome adds minimal overhead compared to single-device applications.

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces—*Interaction styles*; I.3.6 Computer Graphics: Methodology and Techniques—*Interaction techniques*

Author Keywords

Co-browsing; cross-device interaction; multi-display environments; distributed visualization;

ITS 2014, November 16–19, 2014, Dresden, Germany.

http://dx.doi.org/10.1145/2669485.2669518

INTRODUCTION

Given today's relentless advancements in display technology, mobile devices, and wireless connectivity, the concept of *collaborative visualization* on multiple devices such as large displays, tabletops, tablets, and smartphones is becoming increasingly prominent [23]. In parallel development, the web is quickly turning into a prime platform for managing and analyzing large amounts of data using visualization [7] and visual analytics [33]. Collaborative web browsing [9, 14] (co-browsing), where browsers on multiple different devices are connected, is one paradigm that allow us to combine both of these trends to enable *cross-device interaction* [42]. So far mostly restricted to applications within human-computer interaction, co-browsing synchronizes browsers by sharing both current browser state and interaction events, including navigation through scrolling, selection through mouse clicks, and input from keyboard and touch displays. Applying this idea to visualization and visual analytics would turn the web browser into an environment for cross-device visualization.

The concept of harnessing multiple networked devices for deep data analytics anywhere and anytime is called *ubiquitious analytics* (ubilytics) [12], and draws upon a significant body of existing work that supports synchronous and co-located collaboration using distributed user interfaces [30], distributed visualization [18], and post-WIMP interaction models [16]. Ubiquitous analytics is inspired from Mark Weiser's vision (ubiquitous computing [42]) of seamless interaction with everyday objects and activities for information processing. Utilizing the web as a medium for collaboration can easily be the most platform-independent way to make further progress towards these common visions of ubiquitous computing and analytics. However, to build ubilytics environments using multi-device ecosystems over the web, we need

a unified framework to replicate shared state and manage the shared visual space of multiple devices.

In this paper, we address this need by presenting POLYCHROME, a web application framework for exploring the design choices in sharing and synchronizing web applications in collaborative settings. Because the framework is built entirely in JavaScript, participating devices need no special software beyond a modern web browser. Interaction sharing and synchronization among devices using PolyChrome is performed through a secure peer-to-peer (P2P) network, while persistent data such as login details, display configuration preferences, shared state, and consistency management is managed by a dedicated server. PolyChrome provides three contributions to the visualization on interactive surfaces community:

- *Augmented legacy websites:* Augmenting standard web browsers to distribute legacy websites and visualizations across multiple devices and synchronize interaction between them. We target both dynamic websites for visualization created using toolkits such as D3 [7], as well as static websites such as Wikipedia.
- *Collaborative web visualizations:* Providing the interaction and display space distribution mechanisms to create new collaborative web visualizations that utilize multiple devices. This is widely useful in all types of collaboration, including the four combinations of synchronous vs. asynchronous and co-located vs. distributed collaboration [3].
- *Consistency and synchronization management:* Providing framework modules to store the user interaction (represented as operations). Combined with the initial state of a website, the interaction logs are useful for synchronizing devices within the collaborative environment, consistency management, and interaction replay [27].

The remainder of this paper is structured as follows: we first review existing research on web-based visualizations, collaborative visualizations, collaborative web browsing, and operation transformation algorithms for groupware. We then derive and motivate design considerations for our framework. We present the PolyChrome architecture and give several application examples of using PolyChrome for both new and legacy web-based visualization. We close the paper by discussing advantages and disadvantages of PolyChrome along with a preliminary evaluation.

BACKGROUND

In this section, we review existing research in the related areas in visualization, cross-device interaction, and operation transformation for groupware.

Web-based and Collaborative Visualizations

Rohrer and Swing [35] more than a decade ago discussed the possiblity of utilizing the web as a visualization platform, citing its inherent device and platform independence as well as its pervasiveness, but it is only recently that the web became both the source of information as well as the underlying delivery mechanism for interactive visualization. Many Eyes [41] was among the first to allow a visualization of arbitrary datasets over the web. By doing so, they opened up a new opportunity for social analysis and enhanced collaboration in analytics by acting as a space to share visualizations, opinions, and conjectures.

In the past decade, toolkits designed for supporting web visualization such as Protovis [6], JavaScript InfoVis toolkit [5], and D3 [7] have come into prominent use. Protovis [6] aimed at making visualizations more accessible to the web and interaction designers by using a declarative specification of visualization as consisting of graphical primitivates called "marks", such as bars (for bar charts), lines (for line charts), and labels. In contrast, D3 [7], created by Bostock et al., supports web visualization through direct manipulation of the document object model (DOM) of a webpage without any intermediate representations. D3 uses the document as the scene graph and performs at least twice as fast as Protovis [6]. While these toolkits are oriented towards building visualizations over the web, they have not yet attempted to support features for collaboration between multiple users.

Collaboration in the analytics process has been shown to increase sensemaking performance: Mark et al. [28] showed significant benefits in using collaborative visualizations in both distributed and co-located settings, and Balakrishnan et al. [4] found better performance for analysts using shared visualization compared to single-user ones. Drawing on this body of work, Isenberg et al. [23] define the research area of collaborative visualization and derive its unique challenges. Collaborative visualizations in co-located settings often utilize large wall displays, floor displays, tabletops and tablets. Accordingly, several visualizations have been adapted to such settings [22, 25, 40]. Some approaches exist to facilitate the collaborative sensemaking itself; for example, Branch-Explore-Merge [29] supports the whole spectrum of coupled and decoupled collaboration when analyzing datasets.

Collaborative Browsing and Cross-Device Interaction

Collaborative web browsing [9, 14] involves multiple users viewing a single webpage collaboratively using their individual devices through synchronous or asynchronous coupling of their browser interfaces. Traditionally, this is accomplished either by replicating the content of one browser over the connected browsers in distributed collaboration or by allowing virtual avatars to operate on a single web browser in case of co-located collaboration. GroupWeb [15, 17] presents one of the earliest groupware adaptations of a web browser. It allows (1) document slaving for synchronized pages by making sure that the browsers display the same webpage, (2) relaxed "what you see is what I see" by keeping the visuals similar across devices, (3) view slaving for synchronous scrolling, gestures through telepointers, and (4) group annotations.

Another way to look at co-browsing is to see it as a medium for *cross-device interaction* [42]: interaction that spans two or more devices. Accordingly, over the past decade and half, collaborative browsing has evolved from simplistic methods to chain multiple displays into full multi-device environments. Websplitter [20] targets web-based presentations involving multiple displays and hand-held devices. Instead of mirroring webpages across devices, they provide the ability to orchestrate a composite presentation, including audio

and video, across multiple devices according to their capabilities. PlayByPlay [43] supports synchronous and asynchronous collaborative browsing and allows the user to toggle options for receiving actions of other users, sending their own actions, and resyncing with a previously synchronized user. Synchronite [39] works with more dynamic webpages by capturing the mousing events and replicates them among connected devices, thus creating low latency collaboration and reduced data traffic.

Hydrascope [21], a framework for adapting existing web applications (legacy) to multi-surface environments, faces issues with its approach for reverse-engineering these applications, and they counter this by providing some general design guidelines for web applications. Calderon et al. [8] mention the general requirements for going beyond common browser interactions and coupling mobile displays as lessons learned while developing the RED framework for multi-display applications. More recently, Panelrama [44] allows development of web-based collaborative applications through custom panels, but this work does not fully support legacy applications or dynamic web visualizations. Other research work in development of cross-device web interfaces includes XDStudio [31]. In contrast to these frameworks, PolyChrome utilizes existing HTML elements, supports legacy applications while providing a developer API for designing cross-device applications, couples any two devices with a web broswer, and synchronizes using interaction events.

Operation Transformation

Real-time collaborative applications often require management of consistency and conflicts inherently caused by simultaneous collaborative use. In the quest to allow concurrent use in distributed environments, *operation transformation* (OT) has become an established method compared to the alternatives such as turn-taking, locking, and serialization [10, 11].

The early approaches GROVE [10] and REDUCE [37, 38] both adopt fully distributed architectures and maintain a buffer of executed operations on each device. In particular, the REDUCE approach promises intention preservation—the final state on all devices being what the user intended—along with convergence and causality preservation. REDUCE introduces two kinds of operation transformations: inclusion transformations (IT) that transforms an operation O_a against operation O_b such that the impact of O_b is included, and exclusion transformation (ET) that transforms an operation O_a against operation O_b such that the impact of O_b is excluded.

The JUPITER approach [32] serves as an inspiration for recent OT algorithms used in Google Docs. Kumawat and Khunteta [26] review OT algorithms along with the challenges faced in the field and recent achievements. Current frameworks designed with built-in mechanisms for consistency management include Apache Wave [1] and ShareJS [2] that work with a centralized server, linear history buffer, and operation transformation for managing the global state. In PolyChrome, we draw upon this work to achieve consistency in collaborative visualizations by utilizing operation transformation algorithms through a framework that sees user interaction on the web as a series of native browser-level operations.

DESIGN: CROSS-DEVICE VISUALIZATION TOOLKITS

Multi-device ecosystems are becoming increasingly popular in office and academic settings [13, 34]. The presence of large screens, tabletops, and tablets not only increases the screen space but also the interaction space to utilize the various unique abilities of each device. Many existing platforms utilize the idea of connecting the devices in the ecosystem with a client-server or a peer-to-peer architecture [25, 29], to allow for ubiquitous analytics [12]. With the increasing use of the web as a platform for visualization and visual analytics, we believe that it is time to augment toolkits for web visualizations with the ability to allow collaboration at a device level. Here we describe the design choices involved in building ubiquitous visual analytics applications over the web and how these choices can be supported through a software framework for binding web browsers on separate devices together.

Operation Distribution

At a software level, interaction events or operations, a common term in Operation Transformation, form the building blocks for any interaction with a visualization. For example, brush-and-link coordination is one of the common interaction techniques used in multi-view visualizations, and it is performed through one or more selection operations. In multi-device ecosystems for ubiquitous analytics, these operations need to be captured and shared with other devices to propagate the effects of an interaction on the display space. While a simple model for operation distribution would involve sending every operation to all the devices in the network, this may not turn out to be useful in some collaborative scenarios. We have identified various styles (design patterns) of operation distribution or replication (Figure 2).

P1 **Explicit Sharing:** Here the visualization user explicitly decides when the operation sharing should happen, which operations to share and whom to share them with. This is helpful in building private and public workspaces that can allow for branch and merge style collaboration [29]. While an operation that happens on the private workspace is not shared with others, the operation that happens on a public space is passed on to all the devices with user intervention.

P2 **Implicit Sharing:** Here the operations are automatically shared with the connected devices without the need for additional application logic. This creates a fully-aware environment where each device knows the interaction happening on others. This method requires additional application logic for consistency management since interaction can happen at the same time on multiple devices. Typical usage scenarios include distributed user collaboration scenarios that provide the visual feedback of a user interaction to all the other users.

The explicit and implicit sharing models form the two extremes of a hypothetical operation sharing scale (Figure 2). In a typical collaboration, some of the devices may use explicit sharing through user intervention, while others may prefer implicit sharing. Another sharing model on this hypothetical scale, is the **unilateral sharing** model where one device (leader) always shares its operations automatically, while the other devices (lagger) only listen to the operations shared by

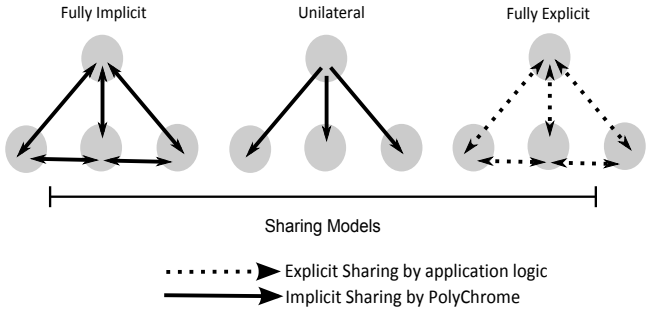

Figure 2. A hypothetical sharing scale ranging from fully implicit to fully explicit sharing. During implicit sharing, the operations are automatically shared with all connected devices. On the other end, explicit sharing allows the developer of the collaborative visualization to define explicit application logic for sharing interaction. Unilateral sharing model is a real-world example that can be applied during presentations.

the leader. This model is useful during presentations and many other guided collaboration scenarios.

To achieve the various design choices in operation distribution especially in concurrent use, transformation of these operations (OT) to the context also needs to be taken into account. Previous research in OT mostly deals with collaborative text editors and document editors where the operations are clearly defined (for example, insert and delete) and the applications allow undo/redo by default. In contrast, typical web visualizations directly deal with the document object model (DOM), thus providing different ways to realize operations. The definition of operation guides the operation transformation algorithms [26, 37, 38] for concurrent use of web applications. Some example operation definitions include,

P3 **Data-centric operation:** Here an operation can be defined on the data structures guiding the visualizations. Any interaction performed in this approach needs to be translated into a change in the data variables (operation) that is shared with other devices in the network.

P4 **Interaction-centric operation:** Here an operation can be defined as an interaction event that is handled by the web browser. This includes different types of browser-level events such as scroll, mouse events such as mouse click, move and mouse up, touch events such as touch start, touch move, and touch end, and key presses. These events/operations can be captured, shared, and then performed on the DOM of the webpage on each device.

As most OT algorithms dictate, collaborative visualizations also require a way to save operations on a server (i.e., an interaction log) to maintain the global state and resynchronize devices that are out-of-sync using context-specific algorithms.

Display Space Management

Multi-device environments often consist of devices of different resolutions, aspect ratios, and screen sizes. This causes a distribution of the unified display space, i.e., the rendering of the web visualization in the browser between multiple devices in the ecosystem. The distribution of the display space leads to different renderings on the devices that may cover

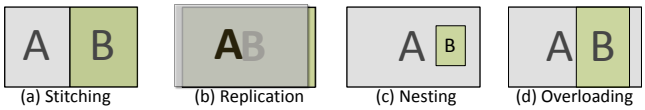

Figure 3. Various display space configurations for a two-device environment. (a) Splitting the display space between the two devices is useful for multi-screen displays, (b) replicating (mirroring) the display space among the two devices is useful during distributed collaboration, and (c, d) nesting and overloading are useful for small-screen mobile devices.

whole or part of the global display space, or the unified display space covering all the devices. Figure 3 shows several local display configurations for a two-display environment. Considering these aspects, one of the design considerations here is to support the following patterns for distributing the display space between multiple devices (inspired by Javed and Elmqvist [24]).

P5 **Replicating:** This strategy shows the exact same view on all devices. The global display space of the ecosystem is, therefore, a *superimposition* of the individual screens. This pattern typically occurs during webpage mirroring in co-browsing, and is widely useful in distributed collaboration where the users work from different locations.

Usage scenarios: Public presentations where the view of the presentor can be mirrored on the displays held by the audience, or distributed collaboration between multiple users to perform document analysis through tools such as Jigsaw [36] that require building synchronized views.

P6 **Stitching:** Here the entire display space is split between devices, with no two devices sharing any part of the display. The global display space is now formed by *juxtaposing*, or stitching, the individual displays.

Usage scenarios: Multi-display visual analytics.

P7 **Nesting:** Here one or more devices hold the entire display space, while others show bits and pieces of the display space relevant to the device or the user.

Usage scenarios: Co-located settings with mobile devices alongside large displays and desktop computers.

Javed et al. [24] also discuss *overloading*, where a view utilizes the space allocated for another view, and this is covered by nesting in the context of web visualizations. Figure 3 shows the design patterns in display space configuration.

Supporting Legacy Applications

While supporting development of new web-based collaborative visualizations, it is equally important to provide as much backward compatibility to the numerous existing web-based visualizations. For example, there are web visualizations built using D3, Protovis, and the JavaScript InfoVis Toolkit. These generally depend on the native browser events for the event handling, and SVG, CSS, and HTML5 for graphical rendering. To support these legacy applications, we propose to use a proxy module that augments a standard web visualization or web application with additional support for event sharing and display configuration between the involved devices.

Additional Design Considerations

The intended user of this design treatment for cross-device visualization toolkits is an application programmer (i.e., an "end-user" programmer), who is building ubiquitous web visualizations. Thus, we must consider the above design choices for operation management, display space configuration, conflict resolution, consistency management, and synchronization among devices. In addition to these, we must also support multiple device modalities for both input and output including tabletops, multi-screen displays, desktop computers, and mobile devices. Finally, a software framework for this should also accommodate future technologies and allow third parties to contribute new functionality.

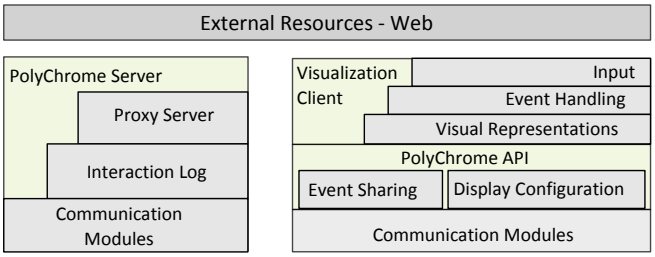

Figure 4. Architecture: The visualization client is connected to the PolyChrome API modules for event sharing and display configuration. The PolyChrome API automatically connects all the devices over a peer-to-peer network for event sharing. Every client is also connected to the PolyChrome server to store the interaction. This framework also provides proxy server modules for using legacy applications collaboratively.

THE POLYCHROME FRAMEWORK

PolyChrome is a generic software framework for building web-based cross-device visualizations. It is implemented using HTML, JavaScript, and CSS. Due to this strict reliance on standard web technology, it is entirely cross-platform and works on any device with a modern web browser. PolyChrome consists of both client and server-side modules. The server modules of PolyChrome have been built using Node.js, as the event-driven nature and non-blocking I/O of Node.js helps in efficiently managing interaction logs of multiple users over time. The client-side modules of PolyChrome contain the PolyChrome API that supports operation distribution and display space configuration. Figure 4 depicts the PolyChrome architecture, including the proxy server that converts legacy web applications into collaborative applications, operation distribution, input, and rendering (visual representation) layers. The framework interacts directly with the *document object model* (DOM) structure within the browser for display space configuration and also capturing browser events as operations (design pattern P4). PolyChrome framework uses PeerJS[1] for P2P communication (using WebRTC[2]) between the devices such as tabletops, tablets, and multi-screen displays, and communication via sockets from a client to server.

PolyChrome consists of four modules: (1) operation (event) sharing; (2) display space configuration; (3) conflict, concurrency, and synchronization management; and (4) a proxy server for serving legacy applications. The design philosophy of PolyChrome is to treat the view of a web visualization at any time instance as a state that changes with user interaction (as in Figure 5). In essence, browser-level DOM events are the operations in PolyChrome applications. By capturing the user operations at the most atomic level on the browser in the form of DOM events, we can reconstruct the state of the visualization from the original rendering. For this reason, the communication between clients of PolyChrome only happens in the form of event sharing, thus leaving the job of rendering to the clients (a philosophy inspired by Synchronite [39]). Browser-level events can be recreated using the event details such as event type, coordinates with respect to the page and client, and the target element in the DOM, but interaction over time with continuous movement of the mouse can lead to conflicts and consistency issues when concurrently performed on multiple devices. For example, in a two-device environment, if the mouse movement is attached to an event handler, sharing the mouse movement with other devices can lead to conflicts in terms of both events and order of the events, and this can propagate into a consistency issue if the handling sequence and context are different. In fact, the effects can be fatal if the application logic for handling such scenarios through OT logic is not encoded into the system.

The PolyChrome clients are connected over a peer-to-peer channel for event sharing, and they can also choose to connect to the server to store their interaction logs. This counters the major setback of using client-level web technologies, i.e., the lack of proper persistent storage of the interaction of a user when the webpage is closed. For example, when a client drops out of the network, the interaction made on the client is lost if it is not stored on some other host. Unless there is a common global state maintained by all devices, this leads to data loss and consistency issues. To counter this, PolyChrome maintains the state of all devices on a server in the form of an event log that contains the information and order of all events generated on all connected devices in a session over time.

The PolyChrome framework currently allows space configuration, i.e., the choice of rendering part or whole of the web application based on the design pattens (P5 to P7) in the Design section by assuming the global space to be the default state of a webpage rendered in a normal browser. The global space configuration is therefore device and webpage independent, and is encoded into each PolyChrome client to facilitate seemless conversion from the local to the global configurations. Every PolyChrome client transforms events to the global space configuration, and these events are interpreted accordingly on other clients by an inverse transformation.

The concurrency and synchronization modules of PolyChrome can be used for handling (1) newly joined clients that require the previous user interactions to work in collaboration or interaction replay, (2) multiple collaboration scenarios including co-located collaboration on a display wall involving multiple mobile devices, and unilateral collaboration during presentations involving a presentor and audience with their own personal devices, and (3) synchronization of clients that are out-of-sync with interaction on other devices. This

[1] `http://peerjs.com/`

[2] `http://www.w3.org/TR/webrtc/`

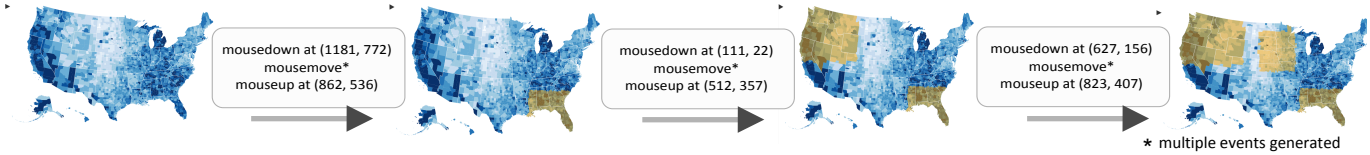

Figure 5. An example interaction with a Choropleth map of unemployment rates in U.S from 2008, created using D3. Upon user interaction (by drawing a rectangular shape on the map), the counties within a selected region are highlighted. PolyChrome provides API-level support for capturing and sharing the browser events that are triggered during this interaction with other connected devices, thus supporting collaborative visualization.

happens when the users work on their private space without sharing or receiving interaction events [29]. The PolyChrome server modules also provide the ability to load collaborative versions of legacy applications with the help of a proxy server that attaches PolyChrome client software to the web source of legacy websites. In the next few subsections we will elaborate on PolyChrome's modules in more detail.

Event Sharing Modules

The event sharing modules provide the basic network-level communication support to share and synchronize the user operations on different devices. PolyChrome achieves this by capturing browser events, communicating these events in a serialized form through a shared peer-to-peer channel created using PeerJS, and then triggering DOM events encapsulated in a PolyChrome Event class on each client. PolyChrome allows two types of event capture mechanisms:

1. **Explicit capture:** An application developer using PolyChrome API can choose to handle event sharing explicitly. By verifying whether an event is generated PolyChrome (encapsulated in a PolyChrome event class) or otherwise, the users can recycle native DOM events, which involves sharing the event with all devices in the P2P network (design pattern P1). The recycled DOM event is then triggered by PolyChrome. The event is at the same time shared with the server by the PolyChrome client (on which the event was created). The events are stored on the server asynchronously on an interaction/operation log in the filesystem with details such as the client identifier, timestamp, event type, spatial co-ordinates and target DOM element details.
2. **Implicit capture:** PolyChrome also allows the users to choose an implicit capture style in which all the events generated on a client are automatically captured at the document level irrespective of their targets. The events captured are analyzed to find their actual targets and are automatically shared with other clients without explicit application logic (design pattern P2). The events are then handled by the original client (on which they were created) through the application logic created for handling the respective event.

These methods for event capture allow for different styles of collaboration as discussed in the Design section. These capture methods can also be applied to settings such as collaborative search, team analytics [25], and general brainstorming processes involving divergence and convergence [19], where the users enjoy the private space on their tablets and merge their ideas with the common displays after a significant breakthrough. PolyChrome can also be used to emulate a unilateral sharing model, as defined previously in the Design section, during presentations (for example) where the presentor performs the interaction which is shared with the audience, with no communication in the opposite direction.

Display Space Configuration Modules

Webpage rendering on a standard browser is defined through the HTML for the static elements, JavaScript for the dynamic structure, and CSS for the style elements for managing alignment, colors, transforms, and effects such as shadows. For PolyChrome, the rendering of a webpage or a dynamic visualization depends on the device type on which it is rendered. For example, rendering on a multi-screen display requires splitting and stitching the webpage across the multiple screens. PolyChrome allows individual devices to manage the local configuration while keeping them informed of the global space. The global space configuration is the DOM rendering of the webpage on a standard web browser. Events generated on each device are scaled to this global space, and then sent to other devices by the PolyChrome API, where they are transformed to the corresponding local configuration. For example, in a multiple monitor setup, the webpage rendering is split and stitched from multiple parts, and PolyChrome maintains the local transforms of each monitor to convert the spatial coordinates of an event into the local space using an affine 2D transform that can take the local translation, scaling, and rotation of each display into account. Events generated on a device can accordingly be converted to the global space using the inverse transform.

This method applies to any screen configuration through CSS-level element transformations that are graphics accelerated on modern browsers, such as Google Chrome. An alternative mechanism would require device-dependent display states that are prone to complex operation distribution logic.

Persistence, Consistency, and Synchronization

Persistence in PolyChrome is achieved by clients sharing their events with a server, which stores the timestamped events in an interaction log. Since the Node.js server is asynchronous in nature, the clients can continue their workflow once the events are sent to the server. This event log can act as an history buffer that is typically used in some modern OT algorithms [26, 32] for consistency management. Furthermore, the PolyChrome framework creates an opportunity to fully allow collaborative use of web visualizations by embedding a concurrency management model. This event log on the server can also be used to update newly joined devices with past interaction events, and also opens up the design space for event caching and chunking on the server to distinguish the semantic importance of various events. For example, in a

visual exploration scenario, while all the events are recorded, only a few might be responsible for the detection of a trend or an anomaly during the sensemaking process. Tagging and caching these events can help during (for e.g.) report generation when these events can be replayed to recreate the process by which the corresponding trend or the anomaly was found.

Proxy Server

To support legacy applications currently available on the web, PolyChrome also contains a proxy server built using Node.js that automatically injects the PolyChrome client modules into legacy websites and web applications. PolyChrome uses an invisible HTML *div* element spanning the entire webpage to capture an event and then identify the target behind this invisible element. The events captured are automatically synchronized, thus allowing collaborative use of legacy applications. This automatic sharing approach for legacy applications can face issues in the presence of custom application-specific events (such as D3's brush events). Although these modules are currently experimental, they allow for running a wide range of sample visualizations created with D3[3] in a multi-device environment.

IMPLEMENTATION

We have implemented PolyChrome as a heterogeneous JavaScript framework that uses PeerJS for peer-to-peer event sharing and a Node.js server application with modules for proxy server, interaction log management, and server-based consistency and synchronization. The basic PolyChrome client consists of approximately 1,500 lines of JavaScript code and works with the most popular D3 visualization toolkit for both computers and mobile devices.

To avoid dependency problems of asynchronous updates, PolyChrome events maintain the ID of the target element in the DOM. Based on the target ID, the corresponding element in the DOM becomes the target on the connected devices. Since one cannot be sure whether every element in the DOM has an ID, the deterministic structure of a DOM tree is utilized to assign an ID to each element during PolyChrome initialization. Newly created DOM elements are also assigned with an ID by the PolyChrome API when needed, thus accounting for the dynamic nature of modern websites. PolyChrome also provides modules to generate pseudo-random values to achieve similar DOM trees even for webpages with random alignment. This mechanism can also be used to distribute identical random numbers across the lightly-connected components in the network. For example, since a force-directed layout starts with random spatial positions, distributing them on multiple devices can lead to inconsistencies.

The PolyChrome framework contains modules for a control panel that is automatically attached to each client. This panel is opened through a toggle button and provides additional controls for event sharing and display configuration to the user, including shortcut methods for some display configurations. It also shows a list of connected devices along with the events generated by each device. Users can also use the control panel to toggle event sharing for specific event types.

[3] `https://github.com/mbostock/d3/wiki/Gallery`

APPLICATION EXAMPLES

We describe two collaborative visual applications for interactive surfaces built using PolyChrome as well as one example of a legacy visualization. The PolyChrome source, along with several examples, are publicly available on GitHub.[4]

Scatterplot Matrix Exploration

Scatterplot matrices (SPLOMs) allow exploration of the relationship between groups of every two variables in a multivariate dataset, and expands in size for every new dimension added to the dataset. For example, Anderson's *Iris* dataset consists of four variables: sepal width, sepal length, petal width, and petal lengths of three species of flowers. The SPLOM in Figure 6 contains sixteen scatterplots. With increasing dataset size, screen space becomes an increasingly valuable commodity for visual exploration. To facilitate collaborative analysis of SPLOMs, we have created a web application using PolyChrome and D3 that visualizes a given multivariate dataset. Brush-and-link interaction is enabled on the visualization to allow the user to see patterns of data points selected on one scatterplot (using brushing) on other scatterplots in the tabular view (linking) through highlighting. Using the PolyChrome API, a device-specific interaction handling is embedded within the code by identifying mobile devices. The brushes created on the mobile devices are shared with a display wall and are highlighed as rectangles on the display wall. The event sharing in this example only happens from mobile devices to the display wall (unilateral sharing model), thus creating an interaction overview on the shared space.

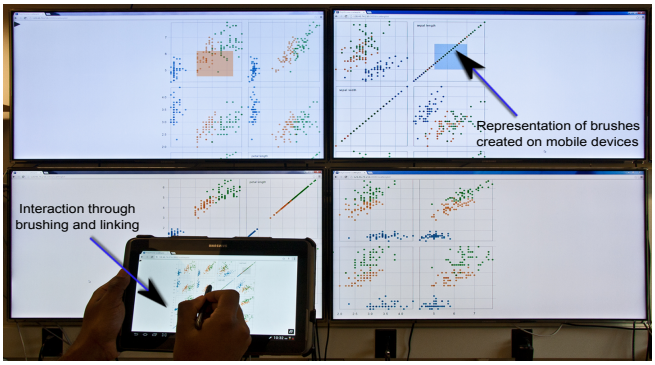

Figure 6. A collaborative web for exploring a scatterplot matrix using brush-and-link interaction on multiple devices.

The application allows users to work on their tablets using brush and link interaction to explore the dataset, while visualizing the brushes on an overview visualization on the display wall. The visualizations and the interaction are created using D3, and the entire application is written using HTML, JavaScriptS, and CSS with less than 30 lines of code to initiate PolyChrome, configure the display, and share events. This example is currently available with the PolyChrome source.

Geographical Map Exploration

The second example uses Google Maps for map exploration using pan and zooming on a web-based multi-display environment (Figure 7). Instead of using the native mouse

[4] `https://github.com/karthikbadam/PolyChrome`

movement events to share the pan interactions, we detect the change in the center of the Google Map (supported by Google Maps API) and share this visual event as a PolyChrome custom application-specific event. These type of events are more meaningful as they represent a visual change, and therefore, they can be semantically tagged for consistency management and synchronization.

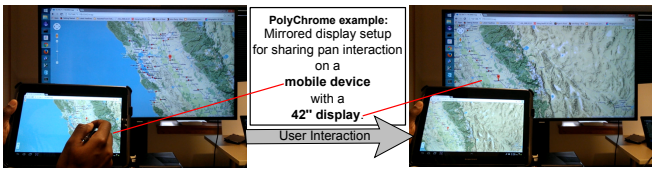

Figure 7. Cross-device Google Maps example where PolyChrome custom event modules are triggered when the map is panned on a device.

This example was created using HTML, JavaScript, and CSS using the Google Maps JavaScript API for the map view, and D3 to show some targets on the map to guide exploration. This example uses less than 20 lines of code using the PolyChrome client API modules, which includes creating an event handler, capturing, and sharing navigation events.

Legacy Application

PolyChrome supports legacy visualizations built using visualization toolkits such as D3. Figure 8 shows an example of a web-based ScatterDice tool applied to box office data.[5] PolyChrome fetches the legacy application using the proxy server and attaches the PolyChrome client modules to it, thus allowing for collaborative exploration of the movie data. The interaction events that happen on one device are shared implicitly with others by the PolyChrome client. This includes the click events, mouse events for drawing shapes, and selection events supported by the legacy application.

PolyChrome also works with the examples provided as part of the D3 toolkit, as well as other toolkits that enable exploration of SVG visualizations, such as VisDock.[6] However, running some of these legacy applications depends on the security configuration, event handling mechanisms in the source webpage, and the capabilities of the proxy server.

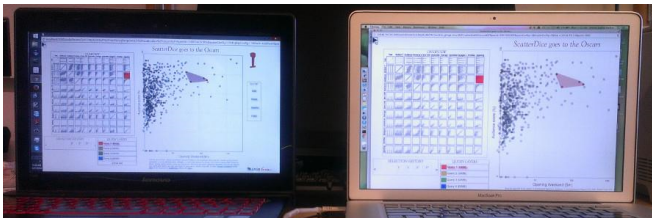

Figure 8. ScatterDice project attached with PolyChrome client modules to synchronize interaction among the devices.

EVALUATION

We performed a preliminary analysis to see the impact of the number of clients using PolyChrome on the time delay between the creation of an event and the execution of the event

[5] `http://tiny.cc/scatterdice`

[6] `https://github.com/VisDockHub/NewVisDock`

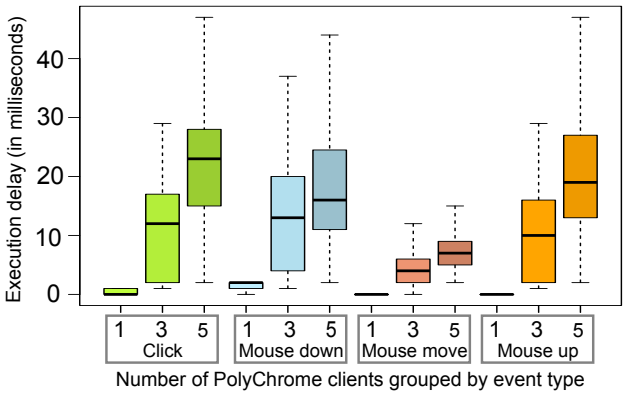

Figure 9. A preliminary performance evaluation of PolyChrome: Three different setups were tested for delays caused by the PolyChrome client in executing various events.

on all connected devices (execution delay) during the implicit sharing model (used in legacy applications). We used a basic drawing application as a surrogate for the legacy visualizations with options to add strokes and also perform mouse clicks. Ten strokes spanning a diagonal of the displays were drawn ($>$ 100 *mousemove* events per stroke) and ten mouse clicks were performed on each client for one, three, and five PolyChrome client setups. These interactions were handled through *click*, *mousedown*, *mousemove*, and *mouseup* events. When the execution delays were analyzed, we found that for the three setups (one, three, and five clients), execution delays increased as the number of clients increased, but they were still in the order of milliseconds (Figure 9). Furthermore, we also found out that the operation of identifying the target element is the root cause of the these delays as it requires a traversal of the DOM tree. The delay was minimal for *mousemove* events, because when this event follows a *mousedown* the targets of both are the same and therefore, PolyChrome skips the target identification step in this case. This analysis helped us identify the potential aspects for further analysis, and also the median increase in delay ($<$ 10 ms from three to five clients) with more clients in the environment.

This suggests that we should strive to achieve better target identification with lower delays. Furthermore, it guides our future evaluation efforts for PolyChrome in terms of the overall performance by profiling event distribution process, developer adoption, and even the mechanics of multi-device visualization and display space configuration strategies.

IMPLICATIONS

The definition of operation as discussed in the design patterns P3 and P4 opens a new space in the field of collaborative visualization for Operation Transformation. This leads to the concurrent use of collaborative web visualization without the need for locking and turn-taking methods to restrict concurrency. The operations in PolyChrome are browser events, and this design choice may lack support for "undo" in some cases. On the other hand, using a data-centric operation that essentially works with the data variables guiding the visualization lacks the ability to support legacy applications and custom data structures. Further evaluation of this is needed.

The choice of hybrid communication modules with peer-to-peer (P2P) and client/server for operation distribution and interaction log respectively leads to fault tolerant yet real-time system. The hybrid system has the ability to surpass the liabilities of both communication types. P2P-based distributed systems face consistency issues in regards to display synchronization and event sharing, while client-server systems face scalability, robustness, security, and trust issues. By combining both and using client/server only for operation/interaction log (for conflict management) the PolyChrome framework gains the advantages of both communication methods.

In the end, no single toolkit works for all situations, and PolyChrome is just the first offering of many potential toolkits for cross-device visualization. We look forward to seeing other toolkits that take different design approaches than us.

CONCLUSION AND FUTURE WORK

We have presented the PolyChrome framework for building web-based collaborative visualizations for multi-surface environments. Beyond the basic functionality to utilize implicit, explicit sharing models, and display space configuration through transformation to the global shared space, PolyChrome comes with a proxy server that supports collaborative use of legacy websites. Due to the presence of a server, operations on various devices can be stored, utilized, and analyzed for many purposes. These interaction logs can be used to serve newly joined devices with previous interaction, replay events during asynchronous collaboration, and maintain synchronization among devices. We validated PolyChrome using several applications built using the framework as well as with some informal performance testing.

In the future, we plan on extending the framework to provide ways to support standard collaboration styles out of the box, embed operation transformation strategies into the framework, allow interpretion through tagging event logs on the devices, adapt the framework to additional hardware and software platforms, and perform an in-depth evaluation.

ACKNOWLEDGMENTS

This work was partially supported by U.S. National Science Foundation award IIS-1253863. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the funding agency.

REFERENCES

1. Apache Wave. `http://incubator.apache.org/wave/`, accessed June 2014.
2. ShareJS. `http://sharejs.org/`, accessed June 2014.
3. Baecker, R. M. *Readings in Groupware and Computer-Supported Cooperative Work*. Morgan Kaufmann Publishers, San Francisco, 1993.
4. Balakrishnan, A. D., Fussell, S. R., and Kiesler, S. Do visualizations improve synchronous remote collaboration? In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2008), 1227–1236.
5. Belmonte, N. G. The JavaScript InfoVis Toolkit. `http://philogb.github.io/jit/`, accessed March 2014.
6. Bostock, M., and Heer, J. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1121–1128.
7. Bostock, M., Ogievetsky, V., and Heer, J. D^3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2301–2309.
8. Calderon, R., Blackstock, M., Lea, R., Fels, S., de Oliveira Bueno, A., and Anacleto, J. Red: a framework for prototyping multi-display applications using web technologies. In *Proceedings of the ACM International Symposium on Pervasive Displays* (2014).
9. Domingue, J., Dzbor, M., and Motta, E. Collaborative semantic web browsing with magpie. In *The Semantic Web: Research and Applications*. 2004, 388–401.
10. Ellis, C. A., and Gibbs, S. J. Concurrency control in groupware systems. In *Proceedings of the ACM SIGMOD Record* (1989), 399–407.
11. Ellis, C. A., Gibbs, S. J., and Rein, G. Groupware: some issues and experiences. *Communications of the ACM 34*, 1 (1991), 39–58.
12. Elmqvist, N., and Irani, P. Ubiquitous analytics: Interacting with big data anywhere, anytime. *IEEE Computer 46*, 4 (2013), 86–89.
13. Endert, A., Bradel, L., Zeitz, J., Andrews, C., and North, C. Designing large high-resolution display workspaces. In *Proceedings of the ACM Conference on Advanced Visual Interfaces* (2012), 58–65.
14. Esenther, A. W. Instant co-browsing: Lightweight real-time collaborative web browsing. In *Proceedings of the World Wide Web Conference* (2002), 107–114.
15. Greenberg, S. Collaborative interfaces for the Web. In *Human Factors and Web Development* (1997), 241–254.
16. Greenberg, S., Marquardt, N., Ballendat, T., Diaz-Marino, R., and Wang, M. Proxemic interactions: the new ubicomp? *Interactions 18*, 1 (2011), 42–50.
17. Greenberg, S., and Roseman, M. GroupWeb: A WWW browser as real time groupware. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems*, ACM (1996), 271–272.
18. Grimstead, I. J., Walker, D. W., and Avis, N. J. Collaborative visualization: A review and taxonomy. In *Proceedings of the Symposium on Distributed Simulation and Real-Time Applications* (2005), 61–69.
19. Hailpern, J., Hinterbichler, E., Leppert, C., Cook, D., and Bailey, B. P. TEAM STORM: demonstrating an interaction model for working with multiple ideas during creative group work. In *Proceedings of the ACM Conference on Creativity & Cognition* (2007), 193–202.

20. Han, R., Perret, V., and Naghshineh, M. WebSplitter: a unified XML framework for multi-device collaborative web browsing. In *Proc. ACM Conference on Computer Supported Cooperative Work* (2000), 221–230.
21. Hartmann, B., Beaudouin-Lafon, M., and Mackay, W. E. Hydrascope: creating multi-surface meta-applications through view synchronization and input multiplexing. In *Proceedings of the ACM International Symposium on Pervasive Displays* (2013), 43–48.
22. Isenberg, P., and Carpendale, S. Interactive tree comparison for co-located collaborative information visualization. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1232–1239.
23. Isenberg, P., Elmqvist, N., Scholtz, J., Cernea, D., Ma, K.-L., and Hagen, H. Collaborative visualization: definition, challenges, and research agenda. *Information Visualization 10*, 4 (2011), 310–326.
24. Javed, W., and Elmqvist, N. Exploring the design space of composite visualization. In *Proceedings of the IEEE Pacific Symposium on Visualization* (2012), 1–8.
25. Kim, K., Javed, W., Williams, C., Elmqvist, N., and Irani, P. Hugin: A framework for awareness and coordination in mixed-presence collaborative information visualization. In *Proceedings of the ACM Conference on Interactive Tabletops and Surfaces* (2010), 231–240.
26. Kumawat, S., and Khunteta, A. A survey on operational transformation algorithms: Challenges, issues and achievements. *International Journal of Computer Applications 3*, 12 (2010), 3038.
27. Manohar, N. R., and Prakash, A. The session capture and replay paradigm for asynchronous collaboration. In *Proceedings of the European Conference on Computer-Supported Cooperative Work* (1995), 149–164.
28. Mark, G., Kobsa, A., and Gonzalez, V. Do four eyes see better than two? collaborative versus individual discovery in data visualization systems. In *Proceedings of the International Conference on Information Visualisation* (2002), 249–255.
29. McGrath, W., Bowman, B., McCallum, D., Hincapie-Ramos, J.-D., Elmqvist, N., and Irani, P. Branch-explore-merge: Facilitating real-time revision control in collaborative visual exploration. In *Proceedings of the ACM Conference on Interactive Tabletops and Surfaces* (2012), 235–244.
30. Modahl, M., Bagrak, I., Wolenetz, M., Hutto, P., and Ramachandran, U. Mediabroker: An architecture for pervasive computing. In *Proceedings of the IEEE Conference on Pervasive Computing and Communications* (2004), 253–262.
31. Nebeling, M., Mintsi, T., Husmann, M., and Norrie, M. Interactive development of cross-device user interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2014).
32. Nichols, D. A., Curtis, P., Dixon, M., and Lamping, J. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Proceedings of the ACM Symposium on User Interface and Software Technology* (1995), 111–120.
33. Payne, J., Solomon, J., Sankar, R., and McGrew, B. Grand challenge award: Interactive visual analytics palantir: The future of analysis. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology* (2008), 201–202.
34. Raskar, R., Welch, G., Cutts, M., Lake, A., Stesin, L., and Fuchs, H. The office of the future: A unified approach to image-based modeling and spatially immersive displays. *Computer Graphics 32* (1998), 179–188.
35. Rohrer, R. M., and Swing, E. Web-based information visualization. *IEEE Computer Graphics & Applications 17*, 4 (1997), 52–59.
36. Stasko, J., Görg, C., and Liu, Z. Jigsaw: Supporting investigative analysis through interactive visualization. *Information visualization 7*, 2 (2008), 118–132.
37. Sun, C., and Ellis, C. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the ACM conference on Computer supported cooperative work* (1998), 59–68.
38. Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction 5*, 1 (1998), 63–108.
39. Thum, C., and Schwind, M. Synchronite – a service for real-time lightweight collaboration. In *Proceedings of the International Conference on P2P, Parallel, Grid, Cloud and Internet Computing* (2010), 215–221.
40. Tobiasz, M., Isenberg, P., and Carpendale, S. Lark: Coordinating co-located collaboration with information visualization. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1065–1072.
41. Viégas, F. B., Wattenberg, M., Van Ham, F., Kriss, J., and McKeon, M. ManyEyes: A site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1121–1128.
42. Weiser, M. The computer for the 21st Century. *Scientific American 265*, 3 (1991), 94–104.
43. Wiltse, H., and Nichols, J. PlayByPlay: collaborative web browsing for desktop and mobile devices. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2009), 1781–1790.
44. Yang, J., and Wigdor, D. Panelrama: enabling easy specification of cross-device web applications. In *Proceedings of the ACM conference on Human factors in computing systems* (2014), 2783–2792.