Sage River Disaster Information (SageRDI): Demonstrating Application Data Sharing In SAGE2

Dylan Kobayashi¹, Matthew Ready², Alberto Gonzalez Martinez³, Nurit Kirshenbaum⁴,
Tyson Seto-Mook⁵, Jason Leigh⁶, Jason Haga⁷

^{1,3,4,5,6}University of Hawaii, Honolulu, USA, {dylank, agon, nuritk, tmook, leighj}@hawaii.edu

²Monash University, Melbourne, Australia, mgrea2@student.monash.edu

⁷AIST, Tsukuba, Japan, jh.haga@aist.go.jp

Figure 1: Users of SageRDI exploring Japan's water sensor data. Those on the left are viewing a specific sensors' details. Those on the right are pointing out map locations they are interested in.

ABSTRACT

The Scalable Amplified Group Environment (SAGE2) is an open-source, web-based middleware for driving tiled display walls. SAGE2 allows running multiple applications at once within its workspace. In large display walls, users tend to collaborate using multiple applications in the same space for simultaneous interaction and review. Unfortunately, many of these applications were created independently by different developers and were never intended to interoperate which greatly limits their potential reusability. SAGE2 developers face system limitations where applications are data segregated and cannot easily communicate with others. To counter this problem, we developed the SAGE2 data sharing components. We describe the Sage River Disaster Information (SageRDI) application and the SAGE2 architectural implementations

necessary for its operation. SageRDI enables river.go.jp, an existing website that provides water sensor data for Japan, to interact with other SAGE2 applications without modifying the website's server, hosted files, nor any of the default SAGE2 applications.

Author Keywords

Exploratory data analysis; work flows; tiled display walls; visualization.

INTRODUCTION

Tiled display walls (TDW) have evidence for significantly improving performance of visualization tasks [4, 11]. With the cost of large, high resolution displays (LHRD) decreasing and computing power increasing, TDW are becoming more common. The SAGE2 [24, 29] middleware was developed in response to the increasing demand to use large scale TDW to enable users to come to conclusions about their data with greater speed, accuracy, comprehensiveness, creativity, and confidence [21, 28].

SAGE2 takes multiple displays, potentially across different physical systems, and unifies them as a multiuser workspace.

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

ISS '18, November 25–28, 2018, Tokyo, Japan

© 2018 ACM. ISBN 978-1-4503-5694-7/18/11...\$15.00

DOI: <https://doi.org/10.1145/3279778.3279798>

Multiple applications may be open simultaneously, in full view without obscuring each other where users may freely reposition, resize, and interact with them. We believe enabling communication links between applications removes the need to tediously translate changes across them, thus accelerating workflow. The goal is to create links between applications such that if they have a commonality between them, when a user updates one, the rest automatically update which has the benefit of avoiding the need to update them one by one. For example, applications with time dependent data (e.g. videos), could be controlled all at once with a slide bar application acting like a remote.

Many applications are developed independently, and were never intended to interoperate. Designing them to interoperate is a challenge and is unlikely to happen unless there is native support for cross-application communication. To address this, we developed an application communication scheme for the SAGE2 environment. Using this scheme, developers can write applications capable of sending data to already existing ones without modifying their code and easily have bi-directional data links in future applications. New applications can be created using other applications as components based on the data they handle and functionality. This can simplify the way applications are added to a workflow.

To showcase this communication scheme, we developed a river disaster information application, SageRDI, that links with a chart application and the preexisting SAGE2 web browser and map applications. This paper describes our implementation for the SAGE2 communication scheme as well as SageRDI. The next section covers our motivation and interest in linked applications. Following is software background on linking application data. We then present the design choices for the data sharing schemes. After the description is an outline of SageRDI usage and capabilities (Figure 1). We end with discussion and conclusions drawn from SageRDI's implementation and initial feedback.

MOTIVATION AND BACKGROUND

When collaborating within TDWs, we have observed ourselves and others frequently filling the space with multiple applications containing complementary data. Placement and sizing of applications often relates to collaborators' interaction or interest in them. This would appear to enforce the observations made by Andrews et al. [3] where users encode additional meaning based on physical organization of data. The following describes two groups' usage with the TDW which had direct influence on our motivation to add data sharing capabilities to SAGE2.

One group regularly meets for energy planning. They view multiple PDFs at once, each representing one energy portfolio to evaluate. The layout and page count across the PDFs are standardized. Supplementary material includes images, maps, and a web application developed within their group. Although all portfolios are possible in theory, they may not be possible in practice due to land acquisition or zoning restrictions. For this reason, they value the ability to view all information at once, which enables easy comparison between the different portfolios. When reviewing the different portfolios, they frequently

open multiple copies of the SAGE2 native application (SNA) for Google Maps and their own web application. The maps are set to show different levels of zoom and map type (terrain and satellite). One map is zoomed out to maintain location awareness of the site, while the other two maps are zoomed in for detail. The web application pulls data from their servers to generate graphs and charts. Users spend effort to individually keep all PDFs showing the same page, maps centered on the same location, and the data their web application uses to show the related graphs.

Another group has been accessing Japan's Ministry of Land, Infrastructure, and Transportation (MLIT) website called Kawa no Bousai Jouhou¹ (KBJ). The name roughly translates to, "river disaster prevention information" and was created as a centralized web location for the public to retrieve information and data from sensors across the country. KBJ is information dense and the group wanted to view the data on a TDW. The website had natural UI oddities further amplified when run on the TDW. Some sensor links redirect the page, while others updated an in-page placeholder. Navigation to locations required full top down knowledge of where you wanted to view (region, prefecture, and municipality). This prompted the group to open multiple copies of KBJ and the SNA Google Maps. One KBJ was the "primary" which they used to keep the view of the sensor map, while the other KBJ instances were for individual sensor details pages that would be moved around for easy comparison. But like the energy group, users had to individually update each sensor page. The SNA Google Map increased their navigation speed and positional awareness within KBJ. By zooming out and panning, those familiar with the area could quickly figure out the top down location rather than needing to rely on memory; different map types (e.g. satellite) provided contextual location. While the group liked being able to view KBJ on the wall, their difficulties with usage (both on TDW and normal access on their own laptop), gave us the opportunity to create SageRDI and the supporting infrastructure to SAGE2. SageRDI was developed to address some of the difficulties identified when using KBJ and to test our data linking infrastructure. These difficulties and how we addressed them are described later in the Implementation of SageRDI section.

Tiled Display Walls in Control Rooms

The advantage of large displays to share information for collaborative work in operations centers has long been an interest of the research community [14]. Despite the decreasing cost of a TDW, they are still considered uncommon in the workspace except for control rooms [23, 15]. The size and amount of information provided by TDW improves shared awareness and is critical when monitoring complex systems or responding to crisis. Their use has been noted and considered valuable at national science laboratories [1], network operation centers [36], traffic control center [27], transportation control centers [25], and hospitals [5].

¹<http://www.river.go.jp/>

Tiled Display Wall Middleware

There are many obstacles to tackle when developing applications for Cluster-Based Large High-Resolution Displays. Frameworks often involve more complex rendering, event handling, programming, and tend to run one application at a time [7]. Some middleware options available for driving display walls include XDMX, HiReD, and SAGE to name a few. XDMX is a Linux based software that enables the GNOME desktop manager and the X windows system to run across multiple systems. However, low frame rates and latency may make interaction difficult [16]. HiReD [31] builds upon VRJuggler [6] which intercepts OpenGL calls to provide geometry data to the different display nodes of a tiled display wall. VRJuggler was designed to run one application at a time; HiReD adds a general purpose window manager to support multiple applications. SAGE [22] focuses on enabling high-definition video and high-resolution graphics streaming over high-speed networks. Applications must use its interface library, where visuals are rendered off-screen and the final output is streamed to the display clients.

We are using SAGE2 [24, 29], the successor to SAGE. Key differences are: SAGE2 was written in JavaScript, can use JavaScript libraries available to webpages, and SAGE2 natively supports web browsing capabilities. For these reasons, we felt it was a natural fit for supporting infrastructure capable to enable data linked applications and web pages.

At its core, SAGE2's head node is a web server. Display clients provide a viewport into the workspace. Users interact with the workspace through user interface (UI) clients. Display and UI clients access the head node through a modern browser. The URL path requested dictates the web browser's role, that of a display or UI client. Due to this dynamic behavior, displays on different systems can be joined together to form the view of SAGE2's workspace. There is no restriction on how many display clients may request the same viewport, this enables view duplication at remote locations.

SAGE2 applications are written in JavaScript and run natively within the display client's browser, enabling full pixel usage. There are two types of applications: a) SAGE2 native applications which can take advantage of the SAGE2 API for features like synchronization support between display clients; and b) web portals that bring external web applications to SAGE2 by using the Webview application.

SAGE2 can run multiple applications simultaneously on its workspace where their visuals are contained within windows, like an operating system. Being a middle ware for TDWs, SAGE2 falls under the distributed application and event model. Each display client has their own instance of running applications. Users' input device events (e.g. mouse and keyboard) from the UI client are passed to the head node, where the events are redistributed to each display client for handling.

Webpage Linking

Brushing and linking is a well-known visualization interaction. Many visualization applications come with support for data linking between multiple views. Such systems include

Tableau [34], Spotfire ², Graphana ³, and Kibana ⁴. Multiple views provide easy comparison between different aspects of the data [35], levels of data [26], and aid in supporting investigation and enable the discovery of unseen relationships [33]. Brushing and linking are considered a basic interaction of information visualizations with multiple views due to their effectiveness for analysis and exploration of data [17, 19, 30, 37]. These are usually features a developer implements within an application. But we want to extend the benefits of multiple views to multiple applications running side-by-side. Being able to reuse applications also reduces the workload of an application developer.

Frameworks like Angular ⁵, Meteor ⁶, and Vue.js ⁷ can produce linked web applications. But developers of such applications have access to the full stack, and usually tailor it to the server it resides on. When working across different servers, applications often require credentials for read and write access. OAuth [12] enables websites to request credentials from a content provider with limited functionality. Web applications may use a representational state transfer (REST) [9] for dynamic data update, but the web application initiates all actions. Web applications may query a server at any time, but the connections are not persistent and the server is unable to push data. A web application using REST detects resource changes by constantly requesting and checking for differences.

Other solutions developed with the intention to work with web applications include Shared Substance [10], HydraScope [13], and Webstrates [20]. Shared Substance uses a data oriented approach for programming and a scene graph that is synchronized between displays as the base for data sharing. Capable of running multiple applications at once, each application may choose to broadcast data or functions that can be accessed by others. Applications must be written to work with Shared Substance and know ahead of time how to request and handle the resources. When the source application closes, the resource will no longer be available. HydraScope applications run one at a time and implement coordinated views for the web applications it shows. The authors say usage requires reverse-engineering relevant state change detection combined with script injection to utilize the web applications. Web applications with built-in synchronization (like Google Docs) or those containing the state in the URL make HydraScope development easier. Webstrates enables synchronized web applications by keeping the document object model (DOM) consistent across all clients. Synchronized pages load a Webstrate client that communicates to the Webstrate server any changes detected in the DOM, those changes are distributed to each client connected to that Webstrate page.

IMPLEMENTATION OF SAGE2 COMPONENTS

There were two main components added to SAGE2 to enable linking applications. First was the parent-child paradigm and

²<https://spotfire.tibco.com/>

³<https://grafana.com/>

⁴<https://www.elastic.co/products/kibana>

⁵<https://angular.io/>

⁶<https://www.meteor.com/>

⁷<https://vuejs.org/>

Figure 2: An overview of the data sharing components. (a) Applications can launch others to form parent-child relationships. Those in this relation can directly call functions on one another. (b) Any application may freely set, request, or subscribe to variables in the server. (c) Application Webview may inject code into the pages it is displaying to alter functionality, visuals, or retrieve data from within.

second was a data relay component added to the server. Developers may envision an ideal setup or layout when linking applications, including creating additional views during usage. The parent-child component ensures applications relying on others will have the means to launch them, then communicate. The data relay component is based on a publish-subscribe system which resides in the server. This has the benefit of one centralized location for data and ensures data survival after an application closes. Interfacing with the data relay is available to any application. Figure 2 shows an overview of the SAGE2 data sharing components⁸.

Parent-Child Relation

Normally only users can launch a SAGE2 application. But if an application is dependent on others, there must be an assurance that the necessary application(s) are available. Providing applications with the ability to launch others increases the flexibility of linking, and reduces requirements of the user.

We added functionality to the SAGE2 application class allowing any application to send the server a launch-application command. The application issuing the launch event becomes the parent and the child is the newly launched application (Figure 2.a). For a parent to communicate it must know the child's application ID for unique identification in case multiple copies are open. For example, multiple image viewers may be open, each showing the same or different images. The server's launch-application event handler was modified to take an optional parent ID. If provided, the server broadcasts a notification to all display clients to inform the parent and child of each other's ID.

SAGE2 display clients are browsers that run SAGE2 applications natively. One can gain direct access to an application object through its unique ID. JavaScript does not natively support private properties. However, by using an object reference,

parents or children may read, alter, or activate each other's variables or functions. This method offers proactive parent applications the highest level of control possible over a child application. This also allows communication from new applications to old applications, but for a child to proactively communicate back its code must be updated to account for this.

Data Relay System

To provide a generalized way of passing data between applications, a data relay component based on publish-subscribe systems [8] was added to the SAGE2 server (Figure 2.b). Helper functions were added to the SAGE2 application class, any application may interact with the data relay in the following ways: `setValue`, `subscribeToValue`, `getValue`, `removeValue`, `subscribeToNewValueCreation`, and `getAllTrackedDescriptions`. The helper functions also prevent duplication of actions when SAGE2 has multiple display clients.

For the `setValue`, `subscribeToValue`, `getValue`, and `removeValue`, a name for the desired variable must be specified. When setting or subscribing, the variables do not need to exist. Setting a non-existing variable will create it; when subscribing to a non-existing variable the data relay will automatically subscribe them when the named variable is created. Preloading the data relay with variables is not necessary, developers are free to use any valid variable name they desire. The helper function for setting a value accepts an optional description object and flag to prefix the variable name with the application ID to ensure unique values from running copies while still using the same code base.

Applications that `subscribeToValue` and `getValue` will also need to specify the name of a receiver function. `getValue` is a one-time action which the data relay will send back to the application's specified receiver function. Anytime a variable value is set, all subscribers will have the value sent

⁸SAGE2 is an open source software located at <http://sage2.sagecommons.org/>

to their specified receiver function. Variables remain on the server until an application issues `removeValue`.

Applications may `subscribeToNewValueCreation`. Whenever a variable is created in the data relay, new value subscribers will receive the variable name and its description object. To see variables that were created prior, `getAllTrackedDescriptions` will provide a list of all variable names and their description objects. This is when the description objects plays a major role in helping to identify desired/usable data.

A helper function, `broadcastDestination`, is available to broadcast the type of data that an application can receive. This will first create a unique variable using ID prefix, then send a subscription event, adding the application as a subscriber to the variable it just created. Like any other variable, all applications are still free to subscribe and set the value of the variable. But the purpose has changed from distributing data from an application to providing a means for an application to broadcast a location where it is able to receive data.

IMPLEMENTATION OF SAGERDI

To validate our approach, we developed a river disaster information application, SageRDI, which links multiple applications to provide more information context when investigating data. SageRDI leverages KBJ, the website described above. KBJ provides access to over 17,000 live sensor measurements like rainfall, river height levels, dam water levels, snowfall, shoreline, and water quality. The information comes from XRAIN, an observation network managed by MLIT⁹.

We wanted to address difficulties experienced when interacting with the website: First is non-intuitive browsing of the sensor map. KBJ provides a map with sensor icons specific to a municipality. Getting to a particular municipality requires knowing the region and prefecture it is located in. Second, scanning through the land for sensors is not possible as each map is associated to a different URL. Third, getting to sensor data is not always obvious. To view sensor data, one has to hover over the sensor icon, which then reveals a URL. However, if the cursor leaves the icon or the URL, the page hides the URL. Clicking the URL would inconsistently open the data in a new browser tab. Not all icons did this, a few would adjust an area in the sensor map page causing immediate confusion even for those who had already been using KBJ. Finally, KBJ is not optimized for large-scale, high-resolution display environments.

SageRDI is built upon the existing SAGE2 native application, Webview, which enables web browsing within SAGE2. Webview uses Electron¹⁰, which isolates the pages in separate processes and allows code injection prior to webpage load and throughout runtime (Figure 2.c). Code injection into KBJ allows event detection, new interaction, and facilitates communication back to SageRDI. The advantage of this approach is SageRDI can leverage the existing KBJ website, removing the need to rebuild a UI or establish KBJ database connections. Notably, database connections were not possible as MLIT

⁹<http://www.mlit.go.jp/en/index.html>

¹⁰<https://electron.atom.io/>

Figure 3: SageRDI launches and forms parent-child relations with 8 Webviews. A script injection provides KBJ with additional interaction. Now, when the map is clicked, the eight closest sensors will have their data displayed in the surrounding Webviews.

does not provide a public API for access. Prior knowledge of how KBJ works was used to design the injection code. Modifications were applied to the KBJ without altering its server or original files.

The webpage is sandboxed and hides console prints from the SAGE2 display client. But, an event detectable by SageRDI is produced. Getting the webpage to communicate data back to SAGE2 is possible through two means: a) console printing stringified JSON objects events which are detected and handled by SageRDI, or b) injecting JavaScript code which connects the webpage back to the data relay. SageRDI uses method a) and acts as an intermediary for data sharing between KBJ and other SAGE2 applications.

During SageRDI's load process, it launches and forms parent-child relationships with eight child Webviews (Figure 3). Through parent-child relation, IDs are exchanged. SageRDI uses the IDs to establish data sharing and form a 3x3 layout starting position, placing itself in the center. Webviews may be individually resized and repositioned as normal. When SageRDI is repositioned, it uses the data link to move each of the child Webviews an equal amount to maintain the formation. This effect may be disabled if users find they wish to break layout formation.

After loading is complete SageRDI accesses KBJ with the starting view of Shonan Town and performs the code injection. The injection improves interaction with the webpage by adding a `mousedown` event listener to the map canvas. A calculation is made to determine the eight closest sensors of the click to populate the child Webviews with those sensors' data. Visual cues, including circling the sensors and drawing reference lines are part of the code injection (Figure 3). The script injection also enables data sharing from KBJ to SageRDI, which then passes URLs through the data link to the child Webviews.

URLs determine which municipality KBJ retrieves information for. Only that municipality's map is shown. KBJ does not support panning or zooming the map. Limited in detail, determining exact location of the sensor or local environment

Figure 4: Panning the Google Map application will update the sensor map shown in KBJ. After the KBJ map loads, sensor locations are passed to the Google Map, where they will be plotted.

features are not possible. However, each sensor from the website has GPS data associated with it. Therefore, SageRDI can launch a child SAGE2 native Google Map application to provide more details and interactivity. Its ability to switch between road, terrain, and satellite views provides more insight to the environment. For example, how developed the surrounding area is or the presence of nearby bodies of water.

Navigating to a municipality map in KBJ requires knowledge of the region and prefecture containing it. Our implementation streamlines this process by linking the geo coordinates of the Google Map child, back to SageRDI. If interaction with the child map causes the center coordinates to change, the new location will be sent to SageRDI through the data relay. When this happens SageRDI performs a distance check between that point and the municipalities. If another town than the one currently shown by KBJ is closer, SageRDI will redirect KBJ to view that municipality's webpage. The distance check is made possible by the group SageRDI was developed for; they had scraped KBJ for a month and provided us with the information collected.

When KBJ changes views to another municipality, SageRDI uses the parent-child data link to send a list of that municipality's sensor GPS coordinates to the Google Map child for plotting (Figure 4). Sensor selection in KBJ will also send geo locations of the selected sensors for plotting in the child map (Figure 5). As mentioned before, the KBJ map does not have much detail, plotting sensor coordinates on the Google Map

allows users to gain more information about sensor location by zooming down to street level view and changing the map view type.

The eight selected sensors may have their historic data recalled from the data scrape and viewed in a chart. If no sensor data is available, this is displayed as part of the title. Charts can be generated for individual sensors or all eight closest sensors. Navigating to a new town will not close the charts. The charts are generated using the plotly.js library¹¹. Each of the charts can be panned, zoomed, or have their view bounds specified by click and drag. Views are synchronized by subscribing and pushing values to the data relay (Figure 6). The final feature added is the ability to flag sensors which creates and editable virtual notes with sensor information. While working with KBJ and linking applications, we found that there was a hand full of sensors with incorrect geo locations. Flagging the sensor turned out to be useful for the group SageRDI was developed for, as they found further errors in the sensor data while using SageRDI¹².

Voice Recognition

By default, SageRDI's starting view is the nine windows seen in Figure 3, SageRDI plus eight surrounding Webviews. During run time, it can open applications that are either link with

¹¹<https://plot.ly/plotly-js-scientific-d3-charting-library/>

¹²SageRDI is an open source software available at https://bitbucket.org/sage2/sage2_apps/src/master/javascript/SageRDI/

Figure 5: With the Google Map open, selected sensors will have their location plotted

itself or each other. (Figure 1 shows some of the optional applications, e.g. Google Maps and charts showing sensor history). The SageRDI's context menu, which is accessible from the client UI, allows launching these applications. This way, the view of KBJ is preserved without being obscured by additional UI elements. Interaction with applications on the wall requires the user to start a SAGE2 pointer. Doing so will hide their computer cursor and send events to SAGE2 in order to manipulate the user's wall cursor. Based on feedback during development, we found users became disoriented when switching between interaction with an application on the wall and accessing the context menu on the client UI.

We addressed the problem of disorientation by adding a voice-to-command (VTC) component to SAGE2. This was inspired by one user thinking aloud that it would be great if he could ask the application instead of going through the menu. Originally, we intended to implement voice commands to improve interaction with SageRDI, but saw an opportunity for the VTC component to be generalized and automatically work with any application's context menu. Research shows a key benefit of voice interaction with display walls is increased productivity by bypassing user interface menu hierarchies [18], aiding situational awareness, and increased collaboration [15]. For these reasons, we decided to make voice commands activate context menu entries which allows generic support in any application.

SAGE2 developers recommend Chrome for the UI client, allowing us to leverage the built-in Chrome component complying with Web Speech API [32]. With no additional installation, we can convert voice to text with high accuracy [2]. Voice recognition is initiated with a long press over an application, either in the UI or with the SAGE2 pointer. Once initiated an icon will appear in the UI indicating to users their voice will be interpreted as a command. After roughly one second of silence, the transcript will be sent to the server for command interpretation. VTC uses the text of a context menu entry as the keywords to determine which command to activate. This

Figure 6: View changes like panning and zooming in one chart will update the view in all linked charts. (a) A user modifies the view of the bottom chart. (b) All charts have their view updated to match the change.

feature allows full interaction with SageRDI without leaving SAGE2 pointer mode.

DISCUSSION AND FUTURE WORK

Although no formal user study was performed, the components added to SAGE2 enabling application data sharing were successfully implemented and tested using the SageRDI application. SageRDI can launch children applications, use the parent-child relation for fine grain control (like positioning), and establish data links with other applications through the data relay. Future work includes a formal user study to evaluate the difficulty of developing applications using the data sharing components and how SageRDI provided benefits beyond using KBJ normally (if any). Informally, we observed the group for which SageRDI was developed and got some feedback. We noticed users spent more time looking for sensors using the Google Map linked with SageRDI. The ability to swap map view types (road, satellite, terrain), generated

interest regarding the sensor environment. Some became interested in finding rural areas or locations with sensors while others attempted to guess where the sensor was on the satellite view.

Initial feedback seems to indicate SageRDI provides an easier means of traversing municipalities. We want to note two major benefits. The first is that navigation with the Google Map allowed those with no knowledge of Japan's geography to navigate with confidence. The second is that KBJ is written in Japanese, those who could not read Japanese became more willing to look for and were able to find sensors at locations using the Google Map.

The injected interaction to display sensor data was easier to work with than the original interface. Having the eight child Webviews provide dedicated sensor data views was much easier to review and organize. The freedom to reposition and resize combined with the general resolution affordance provided by the TDW gave the impression of being easier to investigate sensor data within SageRDI than on their own laptop directly accessing KBJ. But it should be noted that most kept the layout as it was presented to them, rarely moving or resizing them.

SageRDI helped to detect several incorrect details about the sensor data provided by KBJ. One major error being that all sensor locations had their latitude and longitude values swapped. One of the initial SageRDI tests did not show any sensors because with the values swapped, their locations were invalid. After correction, there were still sensors with clearly incorrect locations (one was in Africa). Although locations can be fixed easily, it is representative of the larger problem of data provenance, integrity, and usability for these types of scenarios.

The sharing components allow SageRDI to communicate and link with other applications. This opens a powerful new paradigm for SAGE2 developers where applications can make use of other applications as though they were widgets. Moreover, this system creates an important foundation for the existence of a data broker in SAGE2. Applications can now submit data and broadcast data destinations. One possibility with the data broker is to handle automated movement of data. For example, if an application were to broadcast GPS data and another application could accept it, the data broker could suggest creating a link or automate the passing of data between them. This concept was briefly tested with the Image Viewer application, which was modified to check for geotags and if present create a new variable in the data relay after loading an image. The Google Maps application was also modified to detect variable creation. If a new variable was an image geo location, it would then be plotted on the map. Another data broker extension could be automated data wrangling. Where related data would be automatically reshaped to match format requirements.

Voice commands were originally implemented to help reduce user disorientation caused when users were unsure where to look: at their computer or the tiled display wall. For example, SAGE2 applications are written by different people and the

interaction controls may be in different places. Some put controls in the application window as part of the visuals, while others place controls in the client UI application's context menu with the intention to reduce application view clutter. For SageRDI, voice commands allow usage without leaving SAGE2 pointer mode.

It turns out voice commands not only improved users' opinions regarding interaction, they were also more willing to interact with the system. We noticed that users spent more time looking for sensors, specifically through using the Google Map linked with SageRDI. Once users discovered they could issue location search through voice commands, they spent more time trying to find obscure areas or locations. When the map failed to display the correct location, perhaps because of pronunciation, users were unexpectedly forgiving. After using voice controls, users became more inquisitive about what could be performed with voice, and if not available would it be possible to add that feature. Future work includes expanding the voice capabilities of SAGE2 beyond context menu entries and using better techniques to improve interpretation and handling.

CONCLUSION

We added components to the SAGE2 system which enables applications to communicate and establish meaningful data links between each other. These links enable custom-made knowledge chains to improve data visualizations and understanding. We then validated the approach by building an application, SageRDI. It leverages an existing website Kawa no Bousai Jouhou and extends functionality by adding new interactions and data sharing capabilities with SAGE2 applications. Notably, the original Kawa no Bousai Jouhou webpages were not designed for this, but can benefit from this new functionality.

Initial feedback and observations show that the links were meaningful and did improve the experience and engagement. The added components enhance the SAGE2 middleware by enabling application parent-child relations and between application communication. Component implementation was generalized to enable future application usage, not just limited to SageRDI. These components create a foundation from which further investigation can be made into SAGE2 performing automated duties like data conversion and automated data passing. We discussed a range of future work that these components can enable and the options they provide to developers.

ACKNOWLEDGEMENTS

The co-operative effort in this project was enabled by CEN-TRA (Collaborations to Enable Transnational Cyberinfrastructure Applications). SAGE2 is funded by the National Science Foundation award 1441963 and the Department of Energy & Hawaii State Energy Office's HAVEN (Hawaii Advanced Visualization Energy Nexus) project. This work is also supported by the ICT International Team Grant of the National Institute of Advanced Industrial Science and Technology (AIST), Japan.

REFERENCES

1. Gheni Abia, Grant Wallace, David Schissel, Sean M. Flanagan, Qiang Peng, and Justin R. Burruss. 2005.

- Shared display wall based collaboration environment in the control room of the DIII-D national fusion facility. *WACE*. (2005).
2. Julius Adorf. 2013. Web Speech API. *KTH Royal Institute of Technology*. (2013).
 3. Christopher Andrews, Alex Endert, and Chris North. 2010. Space to Think: Large High-resolution Displays for Sensemaking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 55–64. DOI: <http://dx.doi.org/10.1145/1753326.1753336>
 4. Robert Ball and Chris North. 2005. Effects of Tiled High-resolution Display on Basic Visualization and Navigation Tasks. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems (CHI EA '05)*. ACM, New York, NY, USA, 1196–1199. DOI: <http://dx.doi.org/10.1145/1056808.1056875>
 5. Jakob E Bardram, Thomas R Hansen, and Mads Soegaard. 2006. AwareMedia: a shared interactive display supporting social, temporal, and spatial awareness in surgery. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. ACM, 109–118. DOI: <http://dx.doi.org/10.1145/1180875.1180892>
 6. Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. 2001. VR Juggler: A virtual platform for virtual reality application development. In *Virtual Reality, 2001. Proceedings. IEEE*. IEEE, 89–96. DOI: <http://dx.doi.org/10.1109/VR.2001.913774>
 7. Haeyong Chung, Christopher Andrews, and Chris North. 2014. A Survey of Software Frameworks for Cluster-Based Large High-Resolution Displays. *IEEE Transactions on Visualization and Computer Graphics* 20, 8 (Aug 2014), 1158–1177. DOI: <http://dx.doi.org/10.1109/TVCG.2013.272>
 8. Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The Many Faces of Publish/Subscribe. *ACM Comput. Surv.* 35, 2 (June 2003), 114–131. DOI: <http://dx.doi.org/10.1145/857076.857078>
 9. Roy T Fielding and Richard N Taylor. 2000. *Architectural styles and the design of network-based software architectures*. Vol. 7. University of California, Irvine Doctoral dissertation.
 10. Tony Gjerlufsen, Clemens Nylandsted Klokmoose, James Eagan, Clément Pillias, and Michel Beaudouin-Lafon. 2011. Shared Substance: Developing Flexible Multi-surface Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 3383–3392. DOI: <http://dx.doi.org/10.1145/1978942.1979446>
 11. Tera Marie Green, William Ribarsky, and Brian Fisher. 2009. Building and Applying a Human Cognition Model for Visual Analytics. *Information Visualization* 8, 1 (2009), 1–13. DOI: <http://dx.doi.org/10.1057/ivs.2008.28>
 12. Dick Hardt. 2012. The OAuth 2.0 Authorization Framework. IETF RFC 6749. (2012). DOI: <http://dx.doi.org/10.17487/rfc6749>
 13. Björn Hartmann, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2013. HydraScope: Creating Multi-surface Meta-applications Through View Synchronization and Input Multiplexing. In *Proceedings of the 2Nd ACM International Symposium on Pervasive Displays (PerDis '13)*. ACM, New York, NY, USA, 43–48. DOI: <http://dx.doi.org/10.1145/2491568.2491578>
 14. Christian Heath and Paul Luff. 1992. Collaboration and controlCrisis management and multimedia technology in London Underground Line Control Rooms. *Computer Supported Cooperative Work (CSCW)* 1, 1-2 (1992), 69–94. DOI: <http://dx.doi.org/10.1007/BF00752451>
 15. Tomi Heimonen, Jaakko Hakulinen, Sumita Sharma, Markku Turunen, Lauri Lehtikunnas, and Hannu Paunonen. 2016. Multimodal interaction in process control rooms: are we there yet?. In *Proceedings of the 5th ACM International Symposium on Pervasive Displays*. ACM, 20–32. DOI: <http://dx.doi.org/10.1145/2914920.2915024>
 16. J Hodrien, J Wood, and R Ruddle. 2007. The design and implementation of a 50 million pixel Powerwall display. *VizNet report*. See <http://www.viznet.ac.uk> 1 (2007).
 17. Danny Holten and Jarke J. Van Wijk. 2008. Visual Comparison of Hierarchically Organized Data. *Computer Graphics Forum* 27, 3 (2008), 759–766. <https://doi.org/10.1111/j.1467-8659.2008.01205.x>
 18. Kenneth Huber. 2005. Does speech technology have a place in the control room?. In *Power Engineering Society General Meeting, 2005. IEEE*. IEEE, 2702–2703. DOI: <http://dx.doi.org/10.1109/PES.2005.1489558>
 19. Petra Isenberg and Danyel Fisher. 2009. Collaborative Brushing and Linking for Co-located Visual Analytics of Document Collections. *Computer Graphics Forum* 28, 3 (2009), 1031–1038. DOI: <http://dx.doi.org/10.1111/j.1467-8659.2009.01444.x>
 20. Clemens N. Klokmoose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 280–290. DOI: <http://dx.doi.org/10.1145/2807442.2807446>
 21. Jason Leigh, Andrew Johnson, Luc Renambot, Tom Peterka, Byungil Jeong, Daniel J. Sandin, Jonas Talandis, Ratko Jagodic, Sungwon Nam, Hyejung Hur, and Yiwen Sun. 2013. Scalable Resolution Display Walls. *Proc. IEEE* 101, 1 (Jan 2013), 115–129. DOI: <http://dx.doi.org/10.1109/JPROC.2012.2191609>

22. Jason Leigh, Luc Renambot, Andrew Johnson, Ratko Jagodic, Hyejung Hur, Erik Hofer, and David Lee. 2008. Scalable adaptive graphics middleware for visualization streaming and collaboration in ultra resolution display environments. In *Proc. of Workshop on Ultrascale Visualization*. 47–54. DOI: <http://dx.doi.org/10.1109/ULTRAVIS.2008.5154063>
23. Lars Lischke, Sven Mayer, Andreas Preikschat, Markus Schweizer, Ba Vu, Paweł W Woźniak, and Niels Henze. 2018. Understanding Large Display Environments: Contextual Inquiry in a Control Room. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, LBW134. DOI: <http://dx.doi.org/10.1145/3170427.3188621>
24. Thomas Marrinan, Jillian Aurisano, Arthur Nishimoto, Krishna Bharadwaj, Victor Mateevitsi, Luc Renambot, Lance Long, Andrew Johnson, and Jason Leigh. 2014. SAGE2: A new approach for data intensive collaboration using Scalable Resolution Shared Displays. In *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*. 177–186. DOI: <http://dx.doi.org/10.4108/icst.collaboratecom.2014.257337>
25. Torin Monahan. 2007. “War rooms” of the street: Surveillance practices in transportation control centers. *The Communication Review* 10, 4 (2007), 367–389. DOI: <http://dx.doi.org/10.1080/10714420701715456>
26. Chris North and Ben Shneiderman. 1997. *A taxonomy of multiple window coordination*. Technical Report. CS-TR-3854.
27. Arnaud Prouzeau, Anastasia Bezerianos, and Olivier Chapuis. 2016. Towards road traffic management with forecasting on wall displays. In *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces*. ACM, 119–128. DOI: <http://dx.doi.org/10.1145/2992154.2992158>
28. Khairi Reda, Catherine Offord, Andrew E. Johnson, and Jason Leigh. 2014. Expanding the Porthole: Leveraging Large, High-resolution Displays in Exploratory Visual Analysis. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems (CHI EA '14)*. ACM, New York, NY, USA, 2047–2052. DOI: <http://dx.doi.org/10.1145/2559206.2581202>
29. Luc Renambot, Thomas Marrinan, Jillian Aurisano, Arthur Nishimoto, Victor Mateevitsi, Krishna Bharadwaj, Lance Long, Andy Johnson, Maxine Brown, and Jason Leigh. 2016. SAGE2: A collaboration portal for scalable resolution displays. *Future Generation Computer Systems* 54 (2016), 296–305. DOI: <http://dx.doi.org/10.1016/j.future.2015.05.014>
30. Jonathan C. Roberts. 2007. State of the Art: Coordinated & Multiple Views in Exploratory Visualization. *Coordinated and Multiple Views in Exploratory Visualization, 2007. CMV '07. Fifth International Conference on* (2007), 61–71. DOI: <http://dx.doi.org/10.1109/CMV.2007.20>
31. Chris Rooney and Roy A Ruddle. 2015. HiReD: a high-resolution multi-window visualisation environment for cluster-driven displays. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 2–11. DOI: <http://dx.doi.org/10.1145/2774225.2774850>
32. Glen Shires and Hans Wennborg. 2012. W3C Web Speech API Specification. (2012). Retrieved October 1, 2017 from <https://dvcs.w3.org/hg/speech-api/raw-file/9a0075d25326/speechapi.html>.
33. Ben Shneiderman. 2010. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India. DOI: <http://dx.doi.org/10.1145/25065.950626>
34. Chris Stolte, Diane Tang, and Pat Hanrahan. 2008. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Databases. *Commun. ACM* 51, 11 (Nov. 2008), 75–84. DOI: <http://dx.doi.org/10.1145/1400214.1400234>
35. Edward R Tufte, Susan R McKay, Wolfgang Christian, and James R Matey. 1998. Visual explanations: images and quantities, evidence and narrative. (1998). DOI: <http://dx.doi.org/10.1063/1.168637>
36. Bin Wei, Claudio Silva, Eleftherios Koutsofios, Shankar Krishnan, and Stephen North. 2000. Visualization research with large displays. *IEEE Computer Graphics and Applications* 4 (2000), 50–54. DOI: <http://dx.doi.org/10.1109/38.851750>
37. Ji Soo Yi, Youn ah Kang, and John Stasko. 2007. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (Nov 2007), 1224–1231. DOI: <http://dx.doi.org/10.1109/TVCG.2007.70515>