A Virtual Mouse Interface for Supporting Multi-user Interactions

Matthew Peveler[1(✉)], Jeffery O. Kephart[2], Xiangyang Mou[1], Gordon Clement[1], and Hui Su[1,2]

[1] Rensselaer Polytechnic Institute, Troy, NJ 12180, USA
{pevelm,moux4,clemeg2}@rpi.edu
[2] IBM Thomas J Watson Research Center, Yorktown Heights, NY 10598, USA
{kephart,huisuibmres}@us.ibm.com

Abstract. Traditionally, two approaches have been used to build intelligent room applications. Mouse-based control schemes allow developers to leverage a wealth of existing user-interaction libraries that respond to clicks and other events. However, systems built in this manner cannot distinguish among multiple users. To realize the potential of intelligent rooms to support multi-user interactions, a second approach is often used, whereby applications are custom-built for this purpose, which is costly to create and maintain. We introduce a new framework that supports building multi-user intelligent room applications in a much more general and portable way, using a combination of existing web technologies that we have extended to better enable simultaneous interactions among multiple users, plus speech recognition and voice synthesis technologies that support multi-modal interactions.

Keywords: Large display interface · Multi-user interface · Cellphone interface · Interface framework

1 Introduction

When people engage with one another in meetings, they utilize a mixture of modalities to communicate and illustrate their points, often times simultaneously. To support natural and effective interaction [9,12], artificial assistants embedded into meeting spaces (or *intelligent rooms*) must be able to cope with multiple users across a number of modalities beyond just voice, including gesture and pointing. There exist a number of mechanisms to harness these additional modalities, such as leveraging skeleton data from a Kinect camera [18], pointing a phone at a display [2,10], or using a wand, remote, or similar pointing device [8].

Traditional approaches to capturing pointing and gestures for interacting with these spaces generally fall into one of two classes. One approach is to use a custom framework/display layer in which all content must be developed for that framework. This provides well-integrated content for the input modalities,

M. Kurosu (Ed.): HCII 2020, LNCS 12182, pp. 497–508, 2020.
https://doi.org/10.1007/978-3-030-49062-1_33

but at the cost that it is hard to reuse it outside of the specific framework it was developed for due to being written in a domain specific language. Additionally, some of these frameworks require building deep hooks into the operating system, preventing portability. A second approach is to leverage a web-based platform, and map input events (possibly with some additional processing such as is required with the Kinect) directly onto the mouse via automation frameworks, like RobotJS [15]. An advantage of this approach is that the content can be made available easily elsewhere such as via a web browser. Moreover, there exists an extensive set of libraries to support building web content and mouse-based interactions. Unfortunately, as this approach ultimately relies upon the mouse to trigger events, the systems end up being inherently single-user, thus leading to an interaction style in which a designated human is primarily responsible for mediating interactions with the system.

This paper describes our work on a framework that overcomes the limitations of prior approaches by allowing developers to build multi-user, multi-modal intelligent applications built on top of general web technologies, allowing us to take advantage of the wide-ranging work on websites and web based tools. To accomplish this, we present our *Virtual Mouse Interface* that in essence mimics a physical mouse for each user. Multiple users can simultaneously utilize the interface to achieve various well-understood mouse actions such as moving their mouse around a web page, clicking, and scrolling on the displayed web page within the system. The structure for the rest of the paper is as follows. In the next section, we bring up and discuss related work. Next, we present an overview of the architecture of our framework and its underlying technologies. Next, we describe in detail how the *Virtual Mouse Interface* functions and how it supports multi-user interaction. After, we describe some use-cases we have explored with our interface. Finally, we conclude the paper with a summary of our work and contributions, and some thoughts about promising lines of future work.

2 Related Work

There is a rich tradition of work centered around so-called intelligent or smart rooms. These spaces combine a variety of sensors to allow a variety of inputs, such as voice and gesture. Bolt demonstrated using a combination of voice and gesture to issue commands to display simple shapes on a large screen [1]. Brooks demonstrated a distributed architecture that was bound to the underlying X display server and used to resolve multi-modal commands to the system [3]. Further work showed how these spaces could be utilized by multiple users across multiple modalities to play chess on a giant screen [4], using a custom built framework and application. Recent breakthroughs in the underlying technology have allowed for these systems to be used with less constraints on the inputs of voice and gesture [6] and to approach more complex domains, such as analyzing exoplanet data [8], while also relying on displaying and interacting with content shown in webpages to the user. However, in both of these works, their systems allow for multiple participants to speak, but the gestural and pointing input

was driven by a single person holding a "wand", which tied directly into the underlying mouse.

While pointing remains a much desired capacity of these types of systems, it's important to recognize alternative modalities that have been developed to enable multi-user systems. Examples of this include a digital table with a touch-screen [16] or allowing participants to have personal tablets that they can use to modify globally shown information [13], and through virtual reality and holography [11]. The table allows for users to use their hands as containers for objects for speech commands as well as handwriting content, though the authors note that it forces a high level of co-locality among the users which ends up negatively impacting their ability to cooperatively achieve tasks simultaneously. The personal devices help to allow users to type out fuller content to be shared with the other users without having to rely on the voice transcription service, which often carries some errors especially over transcribing longer sentences, however, it's potentially at the cost of a feeling of decreased connection both to the room at large and the other participants as each user heavily looks at their own screen. The virtual reality environment allows for the highest level of immersion within a space, but is perhaps the most costly to develop and create content for, which potentially renders in impractical for wide-spread deployment.

3 The Cognitive and Immersive System Architecture

Our system builds upon the Cognitive Immersive Room Architecture CIRA [5], which supports and augments group decision-making with cognitive artificial agents in so-called human scale environments. The architecture features a modular approach of separate components handling specific concerns, such as a transcript-worker for receiving input from microphones, a conversation-worker for translating the transcribed speech into dialogue, etc. We extended the architecture by augmenting the existing display-worker and adding in two new components, the spatial-context-system and Reagent, which are described below. As part of this work, and to demonstrate its effectiveness, we deployed it in two unique environments, where the first features a panoramic screen with a diameter of 12 m and height of 3.8 m that users stand inside of (shown in Fig. 2), and the second features a screen that is flat against a wall and measures 11 by 1 m that users stand in front of (shown in Fig. 5). In both environments, the users are equipped with lapel microphones that picks up what they say, transcribing it to text on the fly, which is then converted to an intent and entities utilizing IBM Watson Cloud services [7]. This intent is then fed into an orchestrator which matches it to an action within the domain, and sees if all entities necessary for the action are satisfied. If some are missing, the system will attempt to resolve it based on information that comes from the gesture system (e.g. what is the user actively pointed or just recently pointed at) as well as historical context of prior intents. If this resolution succeeds, the system carries out the command, else if it fails it asks the user for additional information. In carrying out the command, the orchestrator can call out to external web services to gather additional information, display content on the screen, or use speakers to output a synthesized

voice to the users. Figure 1 shows an overview of this architecture and how the pieces are connected. Communication between the core modules of the system utilize RabbitMQ and a publisher/subscriber model to allow modules to be easily swapped out or new modules put in with the only potential change is just the routing key the modules listen to or output on. The core modules of the architecture that enable our key contributions are highlighted below.

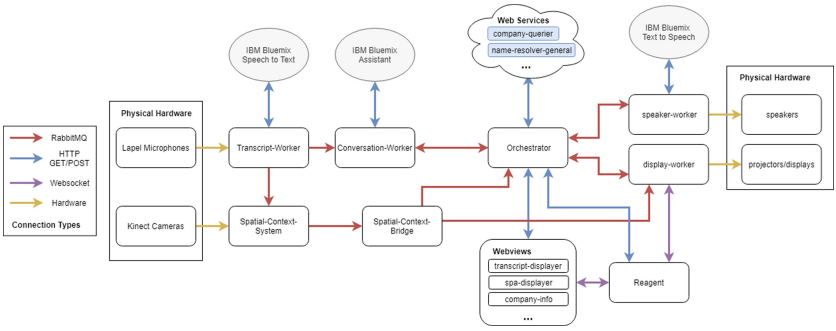

Fig. 1. Architecture of the cognitive and immersive system

3.1 Spatial-Context-System

Within our system, to handle capturing gestural information, users can utilize Kinect cameras [18] or HTC Vive controllers [17], utilizing a common interface API[1]. For both, the underlying implementation follows a similar development path. First, a unified 3D coordinate system is defined for a model of the space, giving all fixed physical objects, such as the displays, a unique location in this model. Next, the sensors, including the Kinect and HTC Vive controller, are calibrated against this coordinate system so that the data generated from the sensors is translated into the coordinate system of the space. Using this, we can create a pointing approximation for the Kinect by using a 3D location of the joints of the human arm and for the HTC Vive by using the posture of the controller to estimate a pointing ray. This approximation is then used to calculate the spatial interaction against the fixed objects, giving us a corresponding [x, y] pixel against the fixed object which acts as the final pointing result. To support multiple users, we dynamically maintain IDs for users in the space. For the Kinect, a unique ID is automatically assigned to a user when they enter the field of view of the camera. For the HTC Vive, an ID is assigned when the controller is turned on and connects to the space. This "spatial ID" is then tied to the unique ID of the lapel microphone for a user to allow for fusion and reference of speech data with gestural information. For either system, it then passes the

[1] It is important to note that through this interface, additional types of input can be supported beyond the two presented here.

pointing information, as well as gestures for the Kinect and button presses for the HTC Vive to the *spatial-context-bridge* which then acts as a normalization layer on the inputs to standard mouse interactions. The HTC Vive requires little normalization as it already has buttons that correspond to how a mouse functions that we can leverage (left and right buttons, scroll wheel, etc.). The Kinect camera on the other hand is transformed from the gestural information of hand actions to mouse actions. For example, opening and closing the right hand corresponds to clicking and then releasing the left mouse button while closing and moving the left hand corresponds to using the scroll wheel to move about a page in the four cardinal directions. Additionally, it provides a smoothing operation on the quickness of hand state changes such as to prevent a rapid hand close, open, close chain within a few milliseconds rising from a brief misclassification of hand state. This is to prevent accidental clicks or ends of clicks that a user might wish to avoid, at the cost of actions taking a few extra milliseconds.

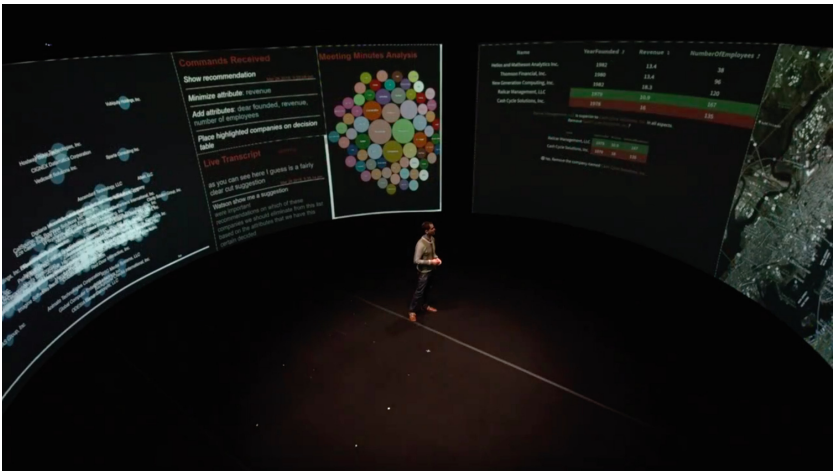

Fig. 2. Aerial shot of half of the 360-degree panoramic screen

3.2 Display Worker

Displaying output on the screen is managed by the Electron framework, which uses a modified chromium engine[2] to render content from websites contained within "webviews". The display-worker provides the user with a grid with a set number of rows and columns in which the user can open as many websites as they would like, with each taking up one or more cells. An example of this is

[2] The modification here is that all generated JavaScript events have the "isTrusted" flag set to true, which is usually only set to true for user generated actions. This allows us to interact with inputs, selects, etc. on a page that do not have an explicitly created "EventListener".

shown in Fig. 3, which shows a 4×4 grid with 5 web pages of different sizes open. It is possible that open webviews may overlap each other, or be kept completely separated, depending on the needs of the application. When each webview opens, the display-worker preloads a small JavaScript file on-top of the opened webpage which helps deliver a payload from the Reagent system, described below. From the spatial context system, it receives the absolute [x,y] coordinate, which it shows to the user as an icon on the screen with their user id in the center. This icon then follows where the user is pointing on the screen, and gives a visual indication of the particular action they are attempting to make (such as clicking on the screen). To help translate what webview a user is interacting with, the display-worker provides an API to understand the dimensions of open webviews, as well as providing a mechanism to translate an absolute [x,y] coordinate on the overall screen into a relative [x,y] coordinate within a given webview. To accomplish this, the display-worker maintains a sorted list of webviews based on their "height" in the DOM tree where elements lower down the tree are on top and overlap elements higher up in the tree, and traverses through that list until it finds a webview in which the [x,y] coordinate lies. To speed this process up on subsequent look-ups, we exploit the principle that users repeat actions within the same webview usually, so we cache the webview the previous look-up was in and check it first on subsequent calls, unless a new webview is opened in the intermediary time. If the new action falls outside the webview, we instantiate the above search again to find the new webview.

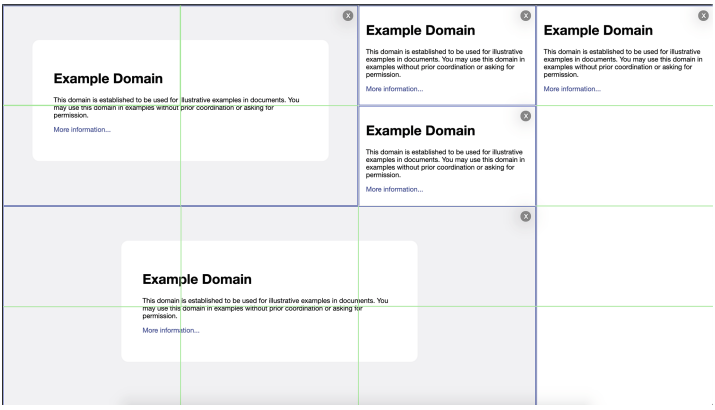

Fig. 3. View of display with 4×4 grid with 5 open pages. Green lines represent the grid and blue lines the borders of the web pages. (Color figure online)

3.3 Reagent

The Reagent system [14], once its bootstrap is pre-loaded by the display-worker, injects further JavaScript code from the Reagent server, which then in-turn constructs an open websocket on the webpage to the Reagent system. Additionally,

while important to the operation of the voice system, but orthogonal to the Virtual Mouse, the injected JavaScript inserts a transparent layer on top of the open page. This layer then analyzes the site, captures the salient semantic information of the page, build listeners for user interactions with the mouse on that content, and attach a so-called MutationObserver to detect any changes in the content of the page, causing the above process to repeat. Additionally, it sets up a websocket on the open page that is used for bidirectional communication between the webpage and the central Reagent server. This server then utilizes a REST API to allow other modules, such as the orchestrator, to get information about what elements are at a particular [x,y] coordinate, get a record of prior interactions, or to run arbitrary JavaScript within a page, such as triggering custom interaction events and which returns the affected elements.

4 Virtual Mouse Interface

Leveraging the spatial-context and Reagent systems, we enable a *Virtual Mouse Interface* that allows users to interact with content as well as to guide multimodal interactions. This interface gives each user the equivalent of their own personal mouse. The interface itself, under the hood is not one dedicated component, but rather conglomeration of functionality across modules described above. To start, the spatial-context-system provides us with an absolute [x,y] coordinate for a given device on the screen from a user, which has a unique ID attached to it. This is sent into the display-worker, which then displays an icon to the user on the screen that represents where they are pointing at that time, as well as the mouse action they are doing. This icon updates at a constant rate for the user, and scales to many concurrent users, where there is (due to RabbitMQ) a delay of about 4–8 ms, which is largely imperceptible to users. In addition to the display-worker, the spatial-context system sends the pointing and action data to the orchestrator. The orchestrator communicates with the display-worker to translate the absolute [x,y] coordinate into a relative [x,y] coordinate within a specific webview. From here, it communicates with Reagent in a number of ways. For each payload that it sends along, and the subsequent action JavaScript event that Reagent generates against a given WebView, the unique user ID is passed, which Reagent binds to the generated events that are dispatched to the page. First, it is important to denote that the orchestrator sets a limiter on the number of actions that can flow through the system, which is roughly 75 ms per action type, which allows adequate throughput for the system for a number of users such that they do not notice lag while also not sending too much information to the page and potentially causing a slow-down. Below, we describe the two types of actions with which we concern ourselves with, mouse and scroll.

4.1 Mouse Actions

Mouse actions represent the principle way in which people interact with the page. This includes the use of the left and right mouse buttons, though we mainly focus

on the usage of the left button here. The mouseitself can be thought of as being in three potential states, being held down for any period of time (MouseDown), being released after being held down for any period of time (MouseUp) and a rapid push down and release of the button (Click). Additionally, there is the act of just moving the mouse itself (MouseMove). To start with these actions, the orchestrator first sends a MouseMove event to Reagent, which then gets dispatched against the webview. From this, Reagent returns the element that the mouse is currently over. The orchestrator stores this element and if it different from the last stored element, issues to Reagent a leave event (MouseOut) on the old element and an enter (MouseEnter) on the new element. This chain of events allows for triggering of hover type events on a site for the given elements affected as you move your cursor across a page. Finally, it takes the mouse action (MouseUp or MouseDown and possibly Click), and sends that to Reagent to issue against the page. In all of three of these cases, Reagent sends details about the element that was clicked on, such that the orchestrator can drive subsequent interactions on it, such as if clicking on a form input, can ask the user what value do they want to input, which is picked up via voice input.

4.2 Scroll Actions

For scroll actions, a more involved sequence is followed to determine what type of scroll is meant by the user. Webpages may implement scroll to mean just moving around the content that has overflowed from the available displayed space (such as scrolling down on a news article), which is referred to as a ScrollEvent. Alternatively, they may use scroll to control zooming in and out of the content, or panning (common in graphs or maps), where these are WheelEvents. However, for both, we require the difference between the current mouse position and the previous mouse position to perform the action, which is stored within the orchestrator. To determine the appropriate action (especially on a page that includes both overflowed content and a graph), the orchestrator first sends a WheelEvent to Reagent. Reagent returns the event that was acted upon, as well as if the MutationObserver it attached to the page detected any changes to that page. If there are no changes, than the system determines that the user's intention was not to zoom or pan, but to simply scroll the page for overflowed content, at which point a ScrollEvent is issued to Reagent, and the page content is shifted.

5 Use Cases

We now describe some use-cases that we have explored with our interface, and describe implementation details. These use-cases cover a couple of different design paradigms that we envision content creators might follow. We first start with utilizing existing web sites in which it is expected only one particular user at a time is going. For this, it is important to remember that the display-worker allows us to open many different webviews at the same time across the available space. For example, in Fig. 2, there is 6 different open webviews in sight.

Each webview is self-contained and interactions within one does not bleed into any other. From this, the virtual mouse interface allows users to interact in different webviews simultaneously. An example of the ways a user might do this is shown in Fig. 4, where the user on the left is scrolling an article in the center, while the user on the right is scrolling the window next to it. Other examples of these sorts of pages include opening something off Google Maps, interacting with a graph, etc. where only one person can scroll and generally only one location or node selected at a time.

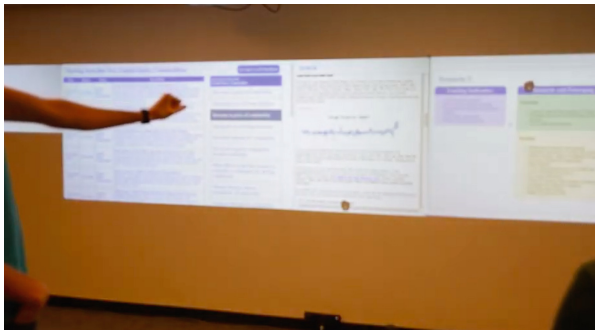

Fig. 4. Two users interacting with multiple open webviews

The other type of interaction we concern ourselves with is having one open webview in which multiple users interact simultaneously. For this, the content creator implements on their site content that has their own event listeners and that can be interacted with discretely, such as elements that are to be clicked on[3]. This can be further extended, if the content developer knows they are designing for our system, by taking advantage of the provided user id on a given event to tie subsequent actions to a single user. For example, in the context of a sticky note application, it may be desired that a MouseDown on a note selects the note, the mouse is moved, and then a MouseUp releases the note to that new location. Multiple users can be supported for that by storing the user id to the note on MouseDown, and ignoring any subsequent actions made against the note except for a MouseUp by that same user. An example of this is shown in Fig. 5. Through this extended paradigm, we easily allow existing sites with discrete mouse actions to function and take advantage of the provided interface.

[3] https://codepen.io/masterodin/pen/jOOPddy gives an example of this sort of content.

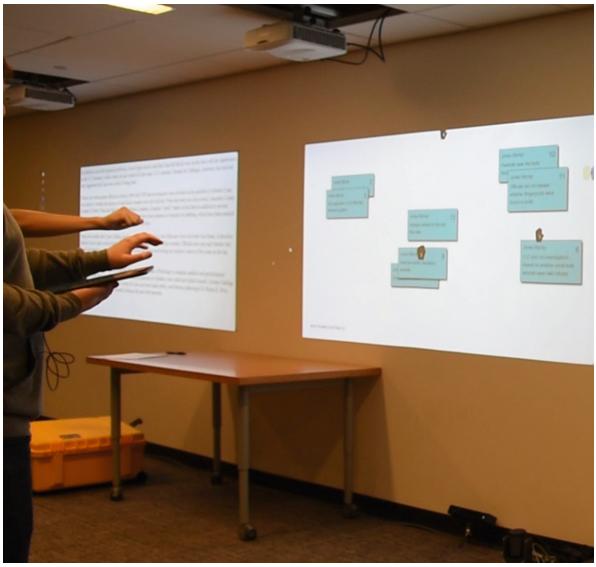

Fig. 5. Two users moving sticky notes around a screen

6 Conclusions and Future Work

In this paper, we present a *Virtual Mouse Interface*, powered through a cognitive and immersive system, to handle multi-user, multi-modal interactions. Through this interface, we can leverage existing web content and handle interactions to it in a way that mimics a physical mouse to those pages. To accomplish this, our interface leverages the Regent system to issue simulated mouse events through JavaScript, while attaching a unique ID per user to the event. From this, pages can be interacted with similarly to a regular mouse, however we can have many virtual mouses interacting at once, be it on the same webview or different open webviews. Additionally, pages can utilize the unique user ID to tailor interactions around our system. This helps remove a major hurdle of requiring a single driver of the system to handle the gestural interactions or development of full custom applications that existed within prior work. Future work would aim to principally lower the latency between events to increase fidelity. Additionally, when using the Kinect, creation of an interface to allow the user to map the inputs of the hands to the virtual mouse in a personalized way as well as for allowing additional actions such as right clicking. Finally, while we focus here on Kinect and HTC Vive as the principal input mechanisms, we are investigating the usage of cellphones, which would operate on a similar input scheme as the HTC Vive, but hopefully be more intuitive to users, as well as lowering entry costs to our system.

References

1. Bolt, R.: "Put-that-there": voice and gesture at the graphics interface. In: Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1980, Seattle, Washington, USA, 1980, pp. 262–270. ACM, July 1980
2. Boring, S., Jurmu, M., Butz, A.: Scroll, tilt or move it: using mobile phones to continuously control pointers on large public displays. In: Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group on Design: Open 24/7 - OZCHI 2009, Melbourne, Australia, p. 161. ACM Press (2009). https://doi.org/10.1145/1738826.1738853, http://portal.acm.org/citation.cfm?doid=1738826.1738853
3. Brooks, R.: The intelligent room project. In: Proceedings Second International Conference on Cognitive Technology Humanizing the Information Age, Aizu-Wakamatsu City, Japan, pp. 271–278. IEEE Computur Society (1997)
4. Carbini, S., Delphin-Poulat, L., Perron, L., Viallet, J.: From a Wizard of Oz experiment to a real time speech and gesture multimodal interface. Sig. Process. **86**(12), 3559–3577 (2006)
5. Divekar, R.R., et al.: CIRA: an architecture for building configurable immersive smart-rooms. In: Arai, K., Kapoor, S., Bhatia, R. (eds.) IntelliSys 2018. AISC, vol. 869, pp. 76–95. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-01057-7_7
6. Farrell, R.G., et al.: Symbiotic cognitive computing. AI Magazine **37**(3), 81 (2016)
7. IBM: IBM Watson Cloud, May 2019. https://www.ibm.com/cloud/ai
8. Kephart, J.O., Dibia, V.C., Ellis, J., Srivastava, B., Talamadupula, K., Dholakia, M.: An embodied cognitive assistant for visualizing and analyzing exoplanet data. IEEE Internet Comput. **23**(2), 31–39 (2019)
9. Krum, D., Omoteso, O., Ribarsky, W., Starner, T., Hodges, L.: Speech and gesture multimodal control of a whole earth 3D visualization environment. In: Proceedings of the Symposium on Data Visualization 2002, Barcelona, Spain, pp. 195–200. Eurographics Association (2002)
10. Langner, R., Kister, U., Dachselt, R.: Multiple coordinated views at large displays for multiple users: empirical findings on user behavior, movements, and distances. IEEE Trans. Visual. Comput. Graphics **25**(1), 608–618 (2019). https://doi.org/10.1109/TVCG.2018.2865235. https://ieeexplore.ieee.org/document/8440846/
11. Noor, A.K., Aras, R.: Potential of multimodal and multiuser interaction with virtual holography. Adv. Eng. Softw. **81**, 1–6 (2015)
12. Oviatt, S., Cohen, P.: Perceptual user interfaces: multimodal interfaces that process what comes naturally. Commun. ACM **43**(3), 45–53 (2000)
13. Peveler, M., et al.: Translating the pen and paper brainstorming process into a cognitive and immersive system. In: Kurosu, M. (ed.) HCII 2019. LNCS, vol. 11567, pp. 366–376. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22643-5_28
14. Peveler, M., Kephart, J.O., Su, H.: Reagent: converting ordinary webpages into interactive software agents. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Macao, China, pp. 6560–6562, August 2019. https://doi.org/10.24963/ijcai.2019/956, https://www.ijcai.org/proceedings/2019/956
15. Stallings, J.: RobotJS, October 2019. http://robotjs.io/

16. Tse, E., Greenberg, S., Shen, C., Forlines, C., Kodama, R.: Exploring true multi-user multimodal interaction over a digital table. In: Proceedings of the 7th ACM conference on Designing interactive systems - DIS 2008, Cape Town, South Africa. pp. 109–118. ACM Press (2008)
17. Zhang, Y.: Combining absolute and relative pointing for fast and accurate distant interaction. arXiv:1710.01778 [cs] October 2017
18. Zhao, R., Wang, K., Divekar, R., Rouhani, R., Su, H., Ji, Q.: An immersive system with multi-modal human-computer interaction. In: 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018), Xi'an, pp. 517–524. IEEE, May 2018