Kanto: A Multi-participant Screen-Sharing System for Etoys, Snap!, and GP

Yoshiki Ohshima
Human Advancement Research
Commumity/Y Combinator Research
Japan
yoshiki.ohshima@acm.org

Bert Freudenberg
Human Advancement Research
Commumity/Y Combinator Research
Germany
bert@freudenbergs.de

Dan Amelang
Viewpoints Research Institute
USA
daniel.amelang@gmail.com

Abstract

This paper demonstrates an implementation strategy for a general real-time remote collaboration framework called Kanto. Kanto is a web-based library that provides screen sharing, voice chat and bi-directional user interaction among participants over the Internet. The generality of Kanto's design makes it straightforward to add its facilities to the programming systems Squeak Etoys, Snap! and GP with little modification to those systems. Because Kanto is web-based, no additional software installation is required on the computers that use it.

Kanto takes advantage of the WebRTC framework, which supports peer-to-peer video, voice and other data transmission. One insight is that if an application uses a single HTML canvas to render all graphics, we can simply stream the contents of the canvas to other hosts to do screen sharing. Luckily, the above-mentioned blocks-based programming languages follow this single-canvas implementation strategy, which is influenced by Smalltalk.

Kanto embodies a particular set of choices within the vast design space of collaboration systems. For example, Kanto maintains its application state by designating one node as the state holder, and streaming just that node's display contents to the other nodes. This simplifies the implementation, but for a remote user introduces a delay between an action and its corresponding display update. In our experience the speed of response is acceptable even at intercontinental distances, but below we discuss alternative designs that would avoid this issue.

PX/17.2, October 22, 2017, Vancouver, BC, Canada

ACM ISBN 978-1-4503-5522-3/17/10…$15.00
https://doi.org/10.1145/3167106

The system can be tested by visiting https://tinlizzie.org:8080/snap/learner.html in one Google Chrome tab, then https://tinlizzie.org:8080/snap/teacher.html in another.

CCS Concepts • **Applied computing** → **Collaborative learning**; • **Information systems** → *Web conferencing*; • **Software and its engineering** → Visual languages;

Keywords Blocks-based languages, Collaboration

ACM Reference Format:
Yoshiki Ohshima, Bert Freudenberg, and Dan Amelang. 2017. Kanto: A Multi-participant Screen-Sharing System for Etoys, Snap!, and GP. In *Proceedings of 3rd ACM SIGPLAN International Workshop on Programming Experience (PX/17.2).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3167106

1 Introduction

As our devices become increasingly connected on the Internet, we anticipate that learning activities on computers will become more and more social, including remote collaboration between teachers and learners. This is already seen in the Scratch community, where children ask each other for suggestions and help in the text-based online forums.

When a learner is learning how to program in a visual environment, it is often difficult to provide guidance by text, or voice, because these are somewhat indirect. It is certainly more difficult than the situation where a tutor is sitting next to the learner, being able to point on the screen and explain things in person. A general video chat system such as Skype or Google Hangouts still falls short, as the tutor cannot point and actually demonstrate how to use the programming system.

From this observation we decided to create a web-based library that adds real-time collaboration features to blocks-based programming systems. We named the library Kanto, following the tradition of using the name of a large, flat region, such as Kansas [15] or Nebraska [4].

Kanto was originally created for the Etoys programming system running on top of SqueakJS [9][6]. However, its design does not depend on Etoys, so we were able to add collaboration features to two more programming systems, Snap! [3] and GP [10], with little modification to them.

Kanto adds screen sharing, voice conversation, and simultaneous user interaction to existing programming systems.

We use the WebRTC mechanism to enable peer-to-peer transmission of video, audio and user events (i.e., mouse pointer and keyboard interaction) among participating nodes.

For sharing graphics, Kanto takes advantage of the fact that these systems, Etoys on SqueakJS, Snap!, and the web version of GP, follow a tradition that is influenced by Smalltalk: the systems use a single HTML canvas element, and render all user interface details for themselves. Because WebRTC in modern browsers can stream the content out of a canvas, we can use the feature to stream the entire graphics of such an application. The audio content can be captured by the browser's `getUserMedia()` feature, and the user events are encoded into binary data.

There are some modifications needed to these programming environments to support multiple "hands", or objects that stand in for user interaction. See some more details in the implementation section.

The intended use case of Kanto is as follows: there is a "learner", who is trying to learn programming, or the interface of a programming system. When the learner feels in need of help, he or she can solicit a teacher to join the session as a remote collaborator and tutor.

The most important mode of collaboration we would like to support is that the teacher should be able to show how to perform a certain task by pointing the location, talking what to do, and sometimes actually demonstrating. For this reason, typical desktop sharing applications, such as Skype and Google Hangouts don't fulfill our need.

The network topology used in implementing the system is asymmetric: the host session is the one running in the learner's browser tab, while teacher's browser shows an HTML `video` element that is receiving the streamed video from the learner. User interaction, such as mouse pointer and keyboard events on the teacher's tab have to be transmitted to the learner to take effects, and then the resulting screen has to be transmitted back to the teacher, so there is a noticeable lag. However, with sessions across the Atlantic, we managed to maintain a real time interactive session. from our experience, we conclude that the system is usable.

2 A Typical Use Case

Figure 1 depicts a typical session. In the figure, time flows towards the right. The top row shows the learner's screen and the bottom row the teacher's.

At step 1, the learner visits the programming system page (the address is shown in the browser's address bar as `learner.html`). At step 2, the page is loaded but the server adds a short random number to the URL (in this example, `#7071`) as a session identifier.

The learner attempts a programming task but encountered some difficulty (at step 3). In the current implementation, the learner would then ask a teacher, with an out-of-band communication, to join the session.

At step 4, the teacher requests the teacher-side web page, adding the session ID (in this example, `teacher.html#7071`); the web page then loads (step 5). Unlike the learner's page, the teacher's page has an HTML video element to receive the video stream from the learner's page. At the same time, the server mediates in establishing a WebRTC connection between the learner and the teacher using the Interactive Connectivity Establishment (ICE [13]) protocol (step 6). The ICE information is exchanged over WebSocket connections between clients and the server.

Once the connection is established, the contents of the HTML canvas element in the learner's page are streamed to the teacher (step 7), and the session carries on without requiring further intervention from the server. Step 8 shows the flow of a user event from the teacher to the learner, and step 9 shows further video update to the teacher from the learner.

Kanto allows more than one teacher to join the same session. In that case, each participant sends audio data to all others, while user events are still only sent to the learner's computer, and video is only sent from the learner's.

3 Implementation

The Kanto library is written in about 1,000 lines of JavaScript code. The general structure for discovering peers and establishing connection follows the standard tutorial and sample implementation found on Google's WebRTC tutorial site [11]. In other words, we did not have to devise all lines from scratch but merely adapted an existing code base for our purpose.

There is a server that handles the peer discovery. The server keeps track of connected sessions and exchanging the ICE protocol information over WebSocket among the learner and the teachers sessions. It is also based on the sample implementation, and written in about 300 lines of JavaScript.

There are some modifications needed to these programming environments to support multiple "hands" Etoys already has multiple-hand support in its Nebraska [4] screen-sharing subsystem. Because of this, we only needed to adapt its encoding and decoding of user events. We modified Snap! to support multiple hands. It involved the core part of Morphic JS as well as some parts of Snap! that embodied assumptions about the number of hands, such as the highlight in block editing. GP so far has not been changed to support multiple hands, yet Kanto allows the position of remote users' cursors to be visualized.

The code is available at:
https://github.com/yoshikiohshima/WebRTC-Events.

4 Related Work

Supporting collaboration through computers is a holy grail of computing. We can trace the idea back to Engelbart's Big

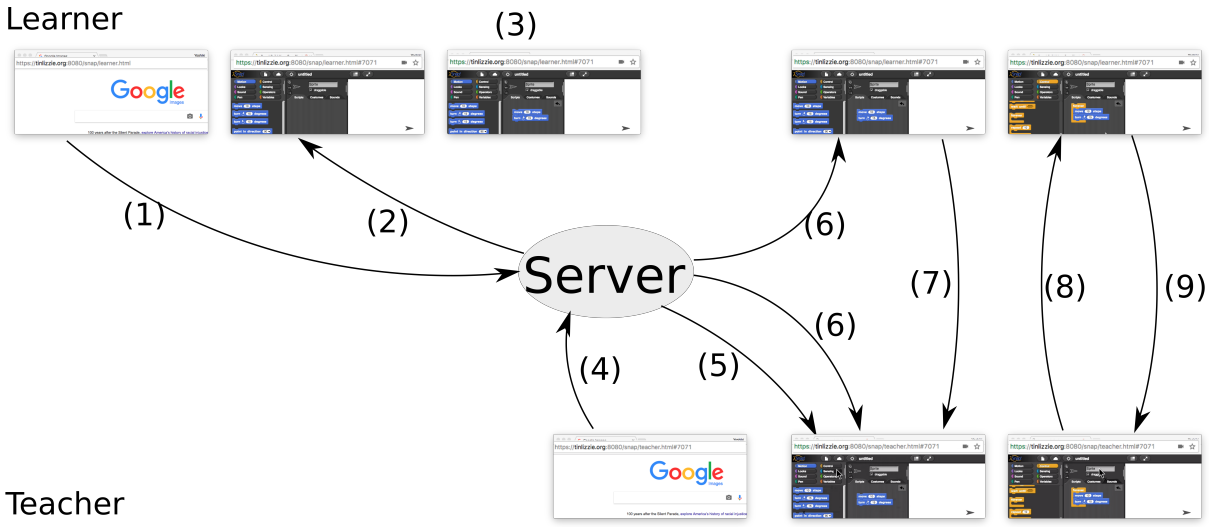

Figure 1. An illustration of typical user case. The time goes to right, and the top row shows the learner's situation while the bottom row shows a teacher's.

Demo on NLS [2]. NLS included on-screen video stream of one's collaborator, and multiple mouse pointers controlled by the respective users. In the "Big Demo", the users (Doug Engelbart and Bill Paxton) were collaborating in real time to navigate the structured and hyper-linked files.

The Self system supported multi-user collaboration [15]. The subsystem for Self was called Kansas, where each participant can view a part of a large virtual 2D space, using facilities for panning and zooming. When the views of multiple users overlap, those users can collaboratively work on objects in the overlapping area. When users wish to work on objects in private, they simply make their views not overlap with others'. While the idea of multiple users working in the same space is now supported in Google Docs and has become a common practice for editing documents, collaborative programming in this form has not been fully realized, even 20 years after the Kansas research.

The Squeak Etoys system draws upon Kansas with its own collaboration subsystem called Nebraska [4]. Nebraska, while named as an obvious homage to the Kansas system, does not support the pan and zoom model. The interaction design of Nebraska is based on the idea of a teacher being invited to a learner's session, with a mechanism called "badges" that is used as the contact list of helpers. Kanto's idea of inviting teachers draws upon this.

NetsBlox [1] is a collaboration environment for Snap!. NetsBlox adds RPC and broadcast-based network features to Snap!, as well as sending block editing commands to the remote nodes. Unlike Kanto, which just streams video, NetsBlox has a more sophisticated collaboration model, and is thus more efficient and robust. But it required deeper changes to the existing Snap! implementation. In addition, as of writing, NetsBlox does not support multiple hands.

VNC [12] is an OS-level screen-sharing system. It is general in the sense that any application (or the entire OS display) can be shared. But it has some disadvantages. One is that only some operating systems have VNC built in; for others, the users must install the VNC software for themselves. Another is that customizations such as specifying a different cursor shape for a remote hand are not possible. Lastly, audio communication (for voice chat) is not supported by VNC.

Croquet [14] offers a collaborative environment that uses a sophisticated network model based on the concept of "replicated computation". In Croquet, all participating nodes hold identical state. Each node sends user events to a small server called the router, and the router sends properly sequenced user events back to all participants. The system is designed so that such this event distribution causes identical computation on the participating nodes, thus maintaining consistency.

Lively Web is a system to support live development on the Web [8][7]. While the sytems has deep root in Smalltalk, the graphics model uses HTML DOM objects as morphs, and our approach cannot be applied. On the other hand, it has its own session to sessionn communication framework (called Lively to Lively), and there were some attempts to create a collaborative environment for Lively.

For document editing, Google Docs uses the concept of "Differential Synchronization" [5]. This too is a sophisticated and robust mechanism, that ensures eventual convergence of remote sites even when occasional errors occur. But for a live programming environment, care must be taken so that execution does not diverge on different nodes.

Skype, Google Hangouts, and other similar conferencing technologies offer real-time collaboration with screen sharing and voice chat. They are robust, but they do not support transmission of user interaction, such as mouse pointer and

keyboard events. Trying to tell someone over Skype which button to click, for example, can be a frustrating conversation. If Skype were to offer a remote hand feature, it would be very useful.

5 Discussion

We think that the strength of Kanto is the balance between its simple implementation and the benefit it provides to multiple applications. Kanto requires minimum changes to an application to get a simple collaboration started, as long as the application's graphics are rendered on a single HTML canvas. Only transmitting higher-level events, as done in Croquet, NetsBlox and Google Docs, gives better performance, but would require deeper changes.

One strength of NetsBlox and Google Docs is that general support for undo is available thanks to the mechanisms for coordinating user events. Kanto only relies on what is already implemented in the application, so providing undo on a per-user basis would require further changes to the application. Also, when the application does not use a single canvas but multiple DOM elements for its UI, Kanto cannot support video transmission.

In the future, we hope that operating systems will provide native support for collaboration. Once that happens, groups of users will be able to customize their collaborative work; groups of programmers will be able to customize their programming experience. What is holding us back today may be a lack of will among OS developers to consider collaboration as a fundamental need.

6 Conclusion

This paper describes a WebRTC-based collaboration framework called Kanto for augmenting existing web-based applications with real-time collaboration features. Because it is built on web technology, it does not require additional software installation. We have shown that the framework is general enough for us to add collaboration features to three programming environments, with little modification to these environments required.

We are aware of more sophisticated implementation strategies that can deliver greater efficiency but would require deeper modification of the applications. We believe that Kanto represents a useful point in the design space of collaborative environments.

Acknowledgments

The authors would like to thank the colleagues at YCR and Viewpoints Research Institute: especially Aran Lunzer for valuable suggestions on the structure of the paper, and John Maloney and Jens Mönig for making GP and Snap! and giving us encouragement.

References

[1] Brian Broll, Akos Lédeczi, Peter Volgyesi, Janos Sallai, Miklos Maroti, Alexia Carrillo, Stephanie L. Weeden-Wright, Chris Vanags, Joshua D. Swartz, and Melvin Lu. 2017. A Visual Programming Environment for Learning Distributed Programming. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 81–86. https://doi.org/10.1145/3017680.3017741

[2] Douglas C. Engelbart and William K. English. 1968. A Research Center for Augmenting Human Intellect. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I (AFIPS '68 (Fall, part I))*. ACM, New York, NY, USA, 395–410. https://doi.org/10.1145/1476589.1476645

[3] Jens Mönig et al. 2017. Snap! (2017). Retrieved Aug 1, 2017 from http://snap.berkeley.edu

[4] Lex Spoon et al. 1999. Nebraska. (1999). http://wiki.squeak.org/squeak/1356.

[5] Neil Fraser. 2009. Differential Synchronization. In *Proceedings of the 9th ACM Symposium on Document Engineering (DocEng '09)*. ACM, New York, NY, USA, 13–20. https://doi.org/10.1145/1600193.1600198

[6] Bert Freudenberg, Dan H.H. Ingalls, Tim Felgentreff, Tobias Pape, and Robert Hirschfeld. 2014. SqueakJS: A Modern and Practical Smalltalk That Runs in Any Browser. *SIGPLAN Not.* 50, 2 (Oct. 2014), 57–66. https://doi.org/10.1145/2775052.2661100

[7] Daniel Ingalls, Tim Felgentreff, Robert Hirschfeld, Robert Krahn, Jens Lincke, Marko Röder, Antero Taivalsaari, and Tommi Mikkonen. 2016. A World of Active Objects for Work and Play: The First Ten Years of Lively. In *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2016)*. ACM, New York, NY, USA, 238–249. https://doi.org/10.1145/2986012.2986029

[8] Dan Ingalls, Krzysztof Palacz, Stephen Uhler, Antero Taivalsaari, and Tommi Mikkonen. [n. d.]. The Lively Kernel – A Self-Supporting System on a Web Page. In *Proceedings of the Workshop on Self-Sfustaining Systems (LNCS 5146)*. Springer, 31–50.

[9] Alan Kay, Kim Rose, Dan Ingalls, Ted Kaehler, John Maloney, and Scott Wallace. 1997. Etoys & SimStories. (February 1997). ImagiLearning Internal Document.

[10] John Maloney, Jens Mönig, and Yoshiki Ohshima. 2017. GP: A general-purpose blocks programming language. (2017). Retrieved Aug 1, 2017 from http://gpblocks.org

[11] The WebRTC Project. [n. d.]. Real time communication with WebRTC. ([n. d.]). Retrieved Aug 1, 2017 from https://codelabs.developers.google.com/codelabs/webrtc-web/

[12] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. 1998. Virtual Network Computing. *IEEE Internet Computing* 2, 1 (Jan. 1998), 33–38. https://doi.org/10.1109/4236.656066

[13] J. Rosenberg. 2010. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols.* RFC 5245. RFC Editor. http://www.rfc-editor.org/rfc/rfc5245.txt http://www.rfc-editor.org/rfc/rfc5245.txt.

[14] David A. Smith, Alan Kay, Andreas Raab, and David P. Reed. 2003. Croquet - A Collaboration System Architecture. *Creating, Connecting and Collaborating through Computing, International Conference on* (2003), 2–9.

[15] Randall B. Smith, Mario Wolczko, and David Ungar. 1997. From Kansas to Oz: Collaborative Debugging when a Shared World Breaks. *Commun. ACM* 40, 4 (April 1997), 72–78. https://doi.org/10.1145/248448.248461