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 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.

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.

CSC CONCEPTS
• Applied computing → Collaborative learning; • Information systems → Web conferencing; • Software and its engineering → Visual languages;

KEYWORDS
Blocks-based languages, Collaboration

ACM Reference Format:
https://doi.org/xx.xx

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 child 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 or Nebraska.

Kanto was originally created for the Etoys programming system running on top of SqueakJS [7][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 [8], 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 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. 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 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 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. A user event on the teacher’s tab has to be transmitted to the learner to take effect, and then the resulting screen has to be transmitted back to the teacher, so there is a noticeable lag. However, from our experience with sessions across the Atlantic, we conclude that it is still 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) 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 [9].

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.

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 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 [12]. 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 views not overlap with others’. While the idea of multiple users working in the same space is now supported in Google Docs and views not overlap with others’. While the idea of multiple users working in the same space is now supported in Google Docs and views not overlap with others’. While the idea of multiple users working in the same space is now supported in Google Docs and views not overlap with others’.

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 [10] 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 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 [11] 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.

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 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

The strength of Kanto is its simplicity and general nature. Kanto requires minimal 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


