Extension of the WebRTC Data Channel Towards Remote Collaboration and Control

Nikos Pinikas, Spyros Panagiotakis, Despina Athanasaki, Athanasios Malamos
Department of Informatics Engineering
Technological Educational Institute of Crete, Heraklion, Greece
npinikas@hotmail.com, spanag@teicrete.gr, despinadev@gmail.com, amalamos@ie.teicrete.gr

***Abstract*— WebRTC is a project that allows browser-to-browser voice, video and data communication without the use of plugins, offering a more immediate communication without the need of a centralized system. It enables rich, high quality RTC applications to be developed for the browser, mobile platforms, and Internet of Things (IoT) devices, and allows them all to communicate via a common set of protocols. By utilizing these technologies, WebRTC-enabled IoT devices could enhance the "telerobotic" and "telepresence" experience of their users, allowing them to not only "interact" with these "things" but also "see" the interaction taking place and even interact collaboratively. In this paper we propose a communication protocol that allows calling and evaluating JavaScript functions and performing actions on remote peers. This protocol can be used in online collaboration platforms and by extension in IoT devices by using IoT JavaScript libraries such as Cylon.js. We also describe a language for exchanging collaboration information and metadata including whiteboard sketches and video annotations. We have developed an online collaboration platform that takes advantage of these technologies and protocols to offer peers the ability to collaborate in real time using whiteboards and text annotations on video streams which are generated from a variety of sources including web cams, screen captures and local video files.**

Keywords*— *WebRTC; Whiteboard; Online collaboration; Screen casting; Internet of Things

I. INTRODUCTION

Collaboration can be defined as the common effort of a group of people to create something. Tools that aid collaboration were around long before computers - whiteboards, flipcharts or even a piece of paper can be used to support collaboration [1]. Computers and the Web revolutionized the way people work together in groups. In the 80s the term "groupware" was coined by C. A. Ellis who defined it as a "computer-based system that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment" [2]. Popular groupware software packages included Lotus Notes and Microsoft Exchange.

Today online collaboration tools can be classified in two categories [3]:

- **Asynchronous collaboration tools.** These tools enable participants to collaborate at different times and different locations. These tools are useful for collaborating over time and providing resources and information that are accessible at any time. For example, by checking the revision history participants are able to see who has contributed, when they have contributed, and what they have contributed. Plus, the use of comments allow participants to agree, debate, or explain changes needed in the work.
- **Synchronous collaboration tools**. These tools enable participants to collaborate in real-time, whether in the same location or in different places. The key point of synchronous tools is that the technology lets the communicators work together at the same time.

The emphasis of this paper is on synchronous online collaboration since these kinds of tools are now made possible on the Web with the introduction of Web real-time communication technologies such as WebRTC. Synchronous collaboration can have many advantages over asynchronous collaboration methods that include immediate response and feedback, video/web conferencing allowing for body language and tone of voice, increased motivation and engagement with discussed concepts and increased social presence. Disadvantages of synchronous collaboration include the lack of reflection between collaborators, the requirement for large time commitment of the collaborators, the difficulty to achieve one to many communication and the fact that if the technology fails the collaboration session not possible [4].

Tools that aid synchronous collaboration include whiteboards, video and audio communication, text chat and screen sharing. Whiteboarding in particular is a teaching and collaboration practice in which participants use a whiteboard area to draw or write concepts, charts, maps, tables, diagrams, equations etc. Smith et al. in [5] conducted a literature review on interactive whiteboard and found among other things that they are particularly effective in education and virtual classrooms allowing teachers to use teaching time to discuss student-generated ideas rather than merely presenting information.

Prior to developing a system aimed at synchronous online collaboration and control, a protocol defining a uniform way in which all the messages that carry the required data that are

exchanged through the WebRTC data channel must be developed. This is necessary if the system is going to be easily expandable, interoperable and maintainable. What we propose is a "language" defining the actions and their parameters that take place in such a collaboration environment. The proposed protocol predefines a number of common collaborative functions such as text chat, whiteboard sketches and video annotation but also defines a way for evaluating JavaScript functions in remote peers. This gives developers the ability to adapt any existing JavaScript library for remote use. In the field of IoT devices for example, one such library is Cylon.js which is a JavaScript framework for robotics, physical computing, and the Internet of Things [6]. Using this library in combination with the peer-to-peer capabilities of WebRTC gives us the ability to create a collaboration environment in which users can not only collaborate and communicate with each other but also collaboratively use an IoT device in real time. IoT Devices that have this capability can also connect directly with other peers using the Janus general purpose WebRTC gateway which allows devices to setup a WebRTC media communication with the browser, exchanging JSON messages with it [7], as summarized in figure 1.

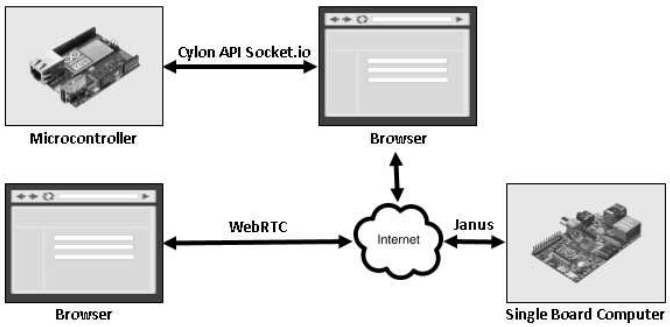


Fig. 1. Controlling an IoT device through the WebRTC data channel

The rest of the paper is organized as follows: In section II we discuss existing work on online collaboration, whiteboards and the combination of WebRTC and IoT. In section III we introduce to the inner details of WebRTC. Section IV details in our proposed protocol aimed at online collaboration and control. Finally section V concludes the paper.

II. Related Work

A number of synchronous online collaboration platforms have been proposed or implemented commercially. Jara C. in [8] proposed a web learning system which combines synchronous collaborative learning in 3D virtual laboratories. In their work, they intergraded their framework in the popular EJS physics program, allowing users to collaborate using the WebGL platform. Andrioti Z. in [9] combined WebRTC and the Evie-m platform [10] to create an online collaborative educational virtual environment for teaching mathematics. It is argued that online collaboration used in education leads to more positive learning outcomes (learning through participation in a group) and more engaged learners [11].

Whiteboarding is one of the most popular applications of synchronous online collaboration. Interactive whiteboards offer a considerable potential to enhance student learning and are excellent pedagogical tools when used appropriately [12]. For example Metz et al. in [13] designed a collaborative whiteboard which was then evaluated by assigning tasks to a group of users and collecting data from user interactions and chat communication. They showed that whiteboard can be an effective collaboration tool. Interestingly, they observed that the collective consciousness of the group of users is created through off-task interactions and so we can deduce that this capability to have "off-task interaction" is one of the reasons that video communication and text chat significantly improve collaboration efficiency and is one of the advantages of synchronous collaboration. Today many online whiteboards are commercially available on the web.

The possible combination of WebRTC with the IoT has not been explored in depth. One of the few examples where WebRC and IoT meet is in the work of Sandholm et al. in [14] who have developed a WebRTC gateway that can tunnel sensor and control data from medical devices. Another example is Neil Stratford's use of the data channel to control LED strips that are connected to a Raspberry Pi that runs Janus [15]. Finally Fan Yung in [16] proposes a system where audio captured by smartphones is send via the WebRTC audio channel to Wireless Sensor Networks in order to monitor environmental condition with a minimum cost.

III. WebRTC

WebRTC (Web Real Time Communication) is a technology that allows real-time peer-to-peer communication between browsers without the use of additional plugins. The mission of WebRTC is "to enable rich, high-quality RTC applications to be developed for the browser, mobile platforms, and IoT devices, and allow them all to communicate via a common set of protocols" [17]. WebRTC was open-sourced by Google in 2011 and after that an ongoing work started to standardize the protocols associated with it by IETF and its browser APIs by W3C. Interest and support for WebRTC has been since growing steadily. Today, the most advanced WebRTC implementation is offered by Mozilla Firefox and Google Chrome. These browsers are now supporting the majority of the features of WebRTC that are proposed by the corresponding W3C drafts [18]. Other platforms that support WebRTC to some extend include the Opera browser, the Android platform and Apple's iOS platform. Microsoft in its Edge browser supports another set protocols named ORTC which does not use the SDP for session descriptions but it is planned to be interoperable with WebRTC [19]. It is expected that by 2018 WebRTC will be supported by 4.7 billion mobile devices [20] and 1.5 billion PCs that run WebRTC enabled browsers bringing the total number to over 6.2 billion

WebRTC enabled devices. WebRTC implements three APIs: a. The MediaStream API which is responsible for capturing streams of media that includes video taken from the user's web camera, a stream from a canvas or video element or a screen capturing stream. b. The RTCPeerConnection API which is used to send these streams between browsers and c. The

RTCDataChannel API is used to exchange arbitrary data such as application and game data but also metadata between peers.

The presence of a data channel is one of the most important features of WebRTC allowing the development of all kind of P2P applications and collaborative solutions ranging from synchronized development [21] to telehealth services [22] and language learning [23] to "whiteboard collaboration" which is the subject of this paper. The architecture of WebRTC including the signaling server is shown in the following schematic:

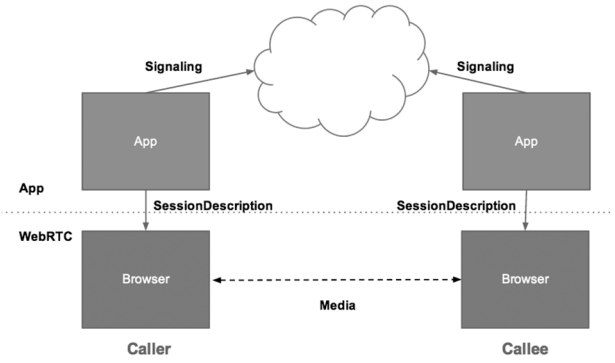


Fig. 2. WebRTC Architecture

Although WebRTC aspires to enable Peer-to-Peer communication between browsers without relaying data through a server, a use of a server is still required for two reasons: The first reason is the obvious one, a web server is needed to "serve" the actual JavaScript application that utilizes WebRTC. The second reason is less obvious. A server is required in order to initialize sessions between the clients that need to communicate. This process is known as "Signaling" and is responsible for the exchange of the initial (meta) data of session descriptions (using SDP) which contain details on the form and nature of the data which will be transmitted [24]. These information can include network data, such as IP addresses and ports, media metadata such as codecs and codec settings, bandwidth and media types, error messages or user and room information.

IV. A Protocol For Synchronous Collaboration & Control

The first step towards defining this "language" targeted at synchronous online collaboration is to develop some form of abstraction layer sitting on top of the native WebRTC RTCDataChannel interface. There are two basic reasons for this: First, we need functions to uniformly handle messages exchanged between peers and second, because the use of the internal WebRTC functions to exchange data through the data channel is often a complicated task requiring many lines of code and customizations. We propose wrapper functions encapsulation the data channel functionality into simple send and receive functions which simplify the development and maintenance of the system. Finally we propose a "language" for uniformly exchanging collaboration information and calling function on a remote peer.

The communication model described above is shown in figure 3. The foundation of the system is the native WebRTC data channel RTCDataChannel.send() function and onmessage property.

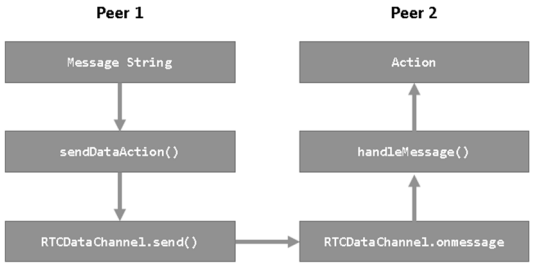


Fig. 3. Communication Model

For sending data we have developed a function called sendDataAction() for sending strings and a function called sendDataFile() for sending binary data. The option to compress data using the Lempel-Ziv-Welch algorithm is also supported through an optional argument. In the following listing the definition of sendDataAction is given:

```
void sendDataAction(string message,
                    bool compression)

Sends data through the WebRTC Data channel
using the native WebRTC RTCDataChannel.send()
method.

message         The string to be sent

[compression]   Optional. A Boolean
                representing whether the
                data should be compressed
                before sending
                true: Compresses data using
                the LZW algorithm
                false: No data compression
                (default)
```

Listing 1. Definition of the sendDataAction function

Note that while in our implementation the sendDataAction function broadcasts all messages to the other peer in the room, in a multiple-user environment this function could be adapted to send messages to specific users only (for example by adding an extra parameter defining a username)

The function handleMessage evaluates incoming messages into function calls. Messages are comprised of array elements separated by a double colon (::). The first element of the array is always the name of the function to be called while the other elements correspond to the parameters of that function. For example when the system receives the string "::MSGNM::PAR1::PAR2::" the callback function

handleMessage will look for a function named msgnm(p1, p2) and call it with “PAR1” and “PAR2” as its parameters as shown in the following schematic:

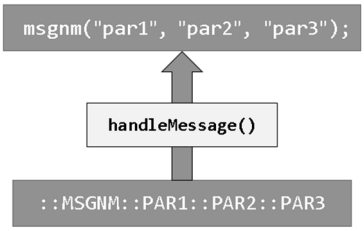


Fig. 4. Converting messages into function calls

Using the function handleMessage we can execute functions of other JavaScript libraries, such as Cylon.js in combination with Socket.io to control IoT devices as shown in fig. 1. A remote peer can send messages to the local peer who is in direct control of the device. These messages are in turn translated into function calls of the Cylon Socket.io API. For example an incoming message to blink LED x every y seconds could be in the form of “LED::1::1000”. The function led(led_no, ms) which would utilize the appropriate Cylon.js function call, would then be called by the system.

Finally the top layer of figure 2 consists of the exchanged standardized messages in the form of strings which represent the actions and function calls. This ensures that the system is well defined and can be easily expanded, but also interoperable so that any WebRTC applications that use this protocol can communicate with each other. The proposed protocol can be used for presenting metadata on video streams, which can include sketching information (Whiteboarding), or chat messaging but can be equally used for any data exchanged between peers including file data (binary), alerts etc.

As we explained, the way the system communicates actions between peers is done using a very simple language. Two colons (::, Unicode U+003A) are used to indicate that what follows is system data in the form of either strings or “stringified” JSON objects. Chat messages or any other data must be filtered and barred from containing this set of characters. Messages also contain an array of information, the elements of which are separated by a double colon (::). The first element of the array is always a 5 letter string defining the message type e.g.:

- URMSG : A chat message
- FILES : An incoming binary file
- SKTCH : Sketching data etc….

We theorize here that an online collaboration platform is comprised of these two elements: video streams and users. Therefore, messages come in two distinct forms: Messages that are intended for canvases and messages that are intended for users. For example a drawing corresponds to a canvas while a chat message corresponds to a user (since a user can have more than one canvas or video shared). We assume that in a peer-to - peer environment messages are broadcast to all peers (all users share the same streams). Of course it is possible to include the name of the recipient in the message in order to send data targeted at specific users.

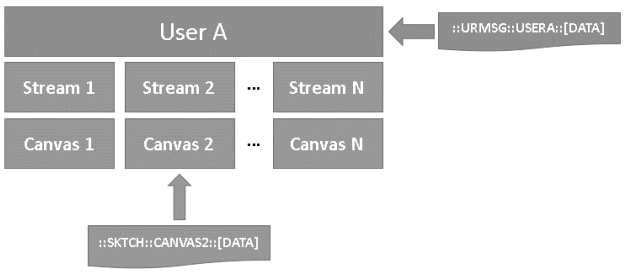


Fig. 5. Message types

Finally we need to add instructions on how the message should be handled once it is received. This is done as we explained using function handleMessage. Developers using our proposed protocol must set the RTCdatachannel.onmessage property to the provided handleMessage function:

```
datachannel.onmessage = handleMessage;
```

The user of the libray must also have the function sendDataAction available. Then the system is able to process incoming messages in the way we have explained.

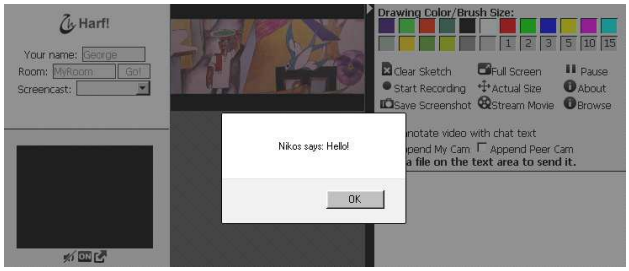
To summarize the expandability of the system, we give an example of how we could add a “poke” function that would broadcast a JavaScript alert to the other peer. First we need to handle the onclick event of an HTML element. The event would use the sendDataAction function with the prefix ::POKE::.

```
pokeLink.onclick=function(){
    var msg="::POKE::"+username+"::"+message;
    sendDataAction(msg);
};
```

We then proceed to write a function called “POKE” with as many parameters as those defined by the received string (in our example two parameters, one containing the sender’s username and one with the message to display):

```
function poke(username, message) {
    alert(username + ' says: ' + message);
}
```

The system will call the function poke upon arrival of the ::POKE:: message and display an alert box as shown in the following screenshot:



We also have predefined and implemented a number of messages that are useful in a synchronous online collaboration environment. Some of them are shown in the following table:

Prefix	Data
::URMSG:: UNAME::DATA	A chat message from a user with username “UNAME”
::SKTCH:: TARGET::WIDTH::DATA	JSON Sketch data including text annotation for the stream named “TARGET”
::FILES:: DATA	Data for incoming files
::PAUSE:: TARGET::TIME	Pauses a stream at a specified time

Table 1. Sample messages

We have developed a sample application that uses the protocol described in this paper. The users have the ability to send video streams to each other. These streams can be sourced from a webcam, an application or window, a monitor, or a local video file. Both users can add text and sketch annotations on any video stream (fig. 7). Capturing the screen or a part of it and then converting it into a stream is done using the “Screen Capture” API which is an extension to the Media Capture API (getUserMedia) and defines a way for the user’s display or parts of it to be used as the source of a media stream [25]. The ability to capture local media files and then send them to other peers as a WebRTC media stream is made possible using the “Stream Capture from Media Elements API” [26]. Users also have the ability to record streams and the annotations and sketches on them, for storage on their local computer as WebM files using the “HTML5 Media Recording API”which provides a simple mechanism by which developers can record media streams from input devices and instantly [27] (fig. 6).

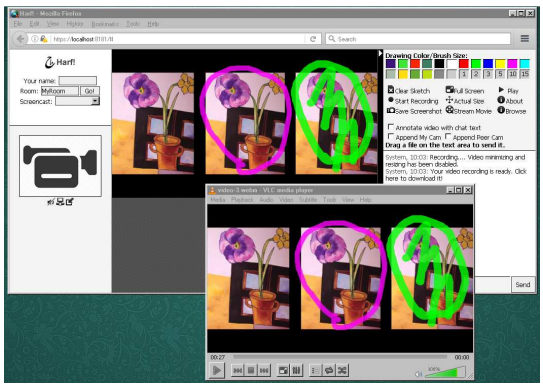


Fig. 6. Users have the ability to share streams generated from local video files, annotate these streams and record them as WebM files

Sketching data are in the form of JSON objects which are comprised of mouse actions and mouse coordinates and text annotation information. Note that the format of the JSON object containing the sketching data and text annotations is not standardized, new features can be added to it as long as the sktch(target, width, data) function (see table 1) is adapted accordingly. A possible SKTCH message is shown in listing 2.

```
::SKTCH
::pbt6HN5gVideoSketch
::640
[{
textcaption: "Hello World",
text_x:10,
text_y:10,
text_size:20
}]
```

Listing 2. A sample message containin information about a text annotation targeted on a stream named ‘pbt6HN5gVideoSketch’

For simplicity the above example specifies that that function SKTCH should be called for the canvas element named “pbt6HN5gVideoSketch”. The next line specifies that on the senders screen this element has a width of 640 pixels (aspect ratio is always equal to 1). Finally the “stringified” JSON object containing information about a text caption (in this case a label with the message “Hello World, size 20 at position (10,10) of the canvas) is attached to the message.

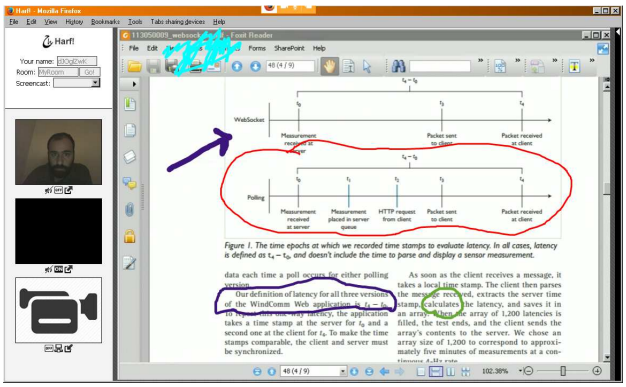


Fig. 7. Users sketching on an open PDF document

It becomes obvious that this protocol can be extended to IoT devices by using existing IoT JavaScript libraries (figure 1). For example the SocketIO API plugin of the Cylon.js library can be used to remotely interact with an Arduino microcontroller in real-time. The user of such a library can use our proposed protocol to call functions of this library through the WebRTC data channel as seen in figure 8. In this example a Cylon.js Socket.io connection is established between one of the peers’ computers. The developer has defined the message:

```
::USELED::user:action::par1::par2
```

The message states that user USER requests to perform an action on a specific LED on the Arduino board. The message would be translated in the following function call:

```
function USELED("user", "action", "par1",
"par2")
```

Which would then in turn call the Socket.io emit function:

```
device.emit("action", "par1", "par2");
```

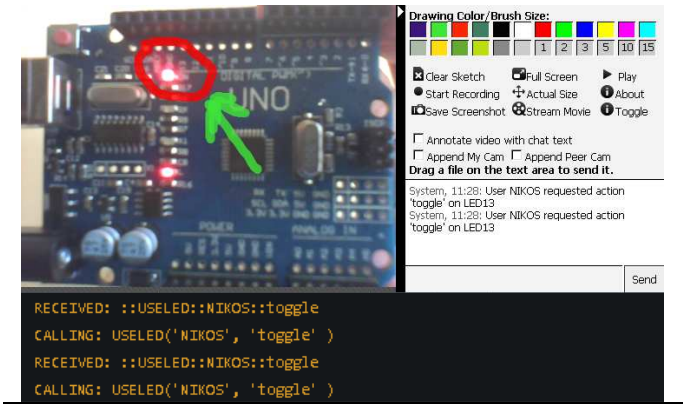


Fig. 8. A remote user observing an Arduino board through the webcam can issue commands to it through the WebRTC data channel (In this case a toggle of the status of the onboard led 13) and even sketch or record the video stream.

In the above example a Toggle button has been added so that when clicked it sends the "USELED::USERNAME:TOGGLE" string through the WebRTC data channel. Upon receiving that string, the peer whose computer has an established connection with the Arduino device will toggle the state of LED13 on the board. Applications of this technique could include the collaborative control and observation of more advanced devices such as motors, servomechanisms, analog sensors etc.

V. Conclusions

WebRTC is a relatively new technology that allows browser-to-browser communication. In this paper we presented a possible application of WebRTC technology in the fields of synchronous online collaboration and IoT control. The WebRTC data channel API is designed to mimic WebSocket exactly, and supports strings as well as some JavaScript binary types as Blob, ArrayBuffer and ArrayBufferView. The existence of a data channel on WebRTC makes possible a number of applications to be developed ranging from online gaming to file sharing.

We propose a uniform way of exchanging data through the WebRTC data channel that can be used to transfer metadata for online collaboration platforms and for evaluating JavaScript functions in remote systems and by extend we suggest an application of this protocol in combination with existing IoT JavaScript libraries such as Cylon.js or Janus to remotely control IoT devices.

We have also developed an application as a prototype intended to demonstrate the capabilities of WebRTC and the proposed protocol, and its potential use for online collaboration, whiteboarding and media streaming. The application takes advantage of modern HTML5 APIs such as the Screen Capture, Media Recording and Stream Capture from Media elements to offer users the ability to share video streams from a variety of sources and then use the WebRTC data channel to exchange collaboration metadata that include sketches, video annotations and IoT device actions.

References

[1] J. F. N. Jr, R. O. Briggs and N. C. Romano, Collaboration Systems: Concept, Value, and Use, New York: Routledge, 2014.

[2] C. A. Ellis, S. J. Gibbs and G. Rein, "Groupware: some issues and experiences," *Communications of the ACM,* vol. 34, no. 1, pp. 39-59, 1991.

[3] T. Walhert, "Synchronous or Asynchronous Tools," Green Hills Area Education Agency, [Online]. Available: https://sites.google.com/a/ghaea.org/aiw-iowacore-techintegration/synchronous-vs-asynchronous. [Accessed 16 4 2016].

[4] B. Kask and S. Wood, "Synchronous and Asynchronous Communication:Tools for Collaboration," University of British Columbia, [Online]. Available: http://etec.ctlt.ubc.ca/510wiki/Synchronous_and_Asynchronous_Communication:Tools_for_Collaboration. [Accessed 16 4 2016].

[5] H. J. Smith, S. Higgins, K. Wall and J. Miller, "Interactive whiteboards: boon or bandwagon? A critical review of the literature," *Journal of Computer Assisted Learning,* vol. 21, no. 2, pp. 91-101, 2005.

[6] "Cylon.js," [Online]. Available: https://cylonjs.com/. [Accessed 3 9 2016].

[7] A. Amirante, T. Castaldi, L. Miniero and S. Romano, "Janus: A General Purpose WebRTC Gateway," in *Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications*, Chicago, IL, 2014.

[8] C. A. Jara, F. A. Candelas, F. Torres, C. Salzmann, D. Gillet, F. Esquembre and S. Dormido, "Synchronous collaboration between auto-generated WebGL applications and 3D virtual laboratories created with Easy Java Simulations," in *9th IFAC Symposium Advances in Control Education*, Nizhny Novgorod, 2013.

[9] Z.-E. Andrioti, Web3D Gaming Over HTML5 and Web-Based Communication, 2015.

[10] K. Kapetanakis, H. Andrioti, H. Vonorta, M. Zotos, N. Tsigkos and I. Pachoulakis, "Collaboration framework in the EViE-m platform," in *Proceedings of the 24th EAEEIE Annual Conference*, 2013.

[11] M. Hammond, "Online collaboration and cooperation: The recurring importance of evidence, rationale and viability," *Education and Information Technologies,* pp. 1-20, 2016.

[12] R. Zevenbergen and S. Lerman, "Learning Environments Using Interactive Whiteboards: New Learning Spaces or Reproduction of Old Technologies?," *Mathematics Education Research Journal,* vol. 20, no. 1, pp. 108-126, 2008.

[13] S. M.-V. Metz, P. Marin and E. Vayre, "The shared online whiteboard: An assistance tool to synchronous collaborative design," *European Review of Applied Psychology,* vol. 65, no. 5, pp. 253-269, 2014.

[14] T. Sandholm, B. Magnusson and B. A., "An On-Demand WebRTC and IoT Device Tunneling Service for Hospitals," in *2014 International Conference on Future Internet of Things and Cloud (FiCloud)*, 2014.

[15] N. Stratford, "WebRTC Lights," Binary Fen , 17 4 2015. [Online]. Available: http://www.slideshare.net/jaquayle/webrtc-lights-by-neil-stratford-of-binary-fen.

[16] F. Yang, "Enhancing the Internet of Things with Reconfigurable Hardware and Software," in *Middleware Doctoral Symposium*, Vancouver, BC, 2015.

[17] "WebRTC," [Online]. Available: http://www.webrtc.org/home. [Accessed 4 7 2015].

[18] "Is WebRTC ready yet?," [Online]. Available: http://iswebrtcreadyyet.com/. [Accessed 5 5 2016].

[19] J. Wager, "What Developers Should Know About ORTC Versus WebRTC," ProgrammableWeb, 12 10 2015. [Online]. Available: http:/

/www.programmableweb.com/news/what-developers-should-know-about-ortc-versus-webrtc/analysis/2015/10/12/.

[20] ABI Research, "4.7 Billion Mobile WebRTC Devices by 2018 Despite Lack of Open Support from Apple and Microsoft," 25 9 2013. [Online]. Available: https://www.abiresearch.com/press/47-billion-mobile-webrtc-devices-by-2018-despite-l/.

[21] K. Jain, A. Himmatramka, A. Bhandary, A. D'silva and D. Barge, "Synchronized Development Using WebRTC Real-Time Collaboration in WebRTC," *International Journal of Engineering Science,* vol. 6, no. 4, 2016.

[22] L. V. Ma, J. Kim, S. Park, J. Kim and J. Jang, "An efficient Session_Weight load balancing and scheduling methodology for high-quality telehealth care service based on WebRTC," *The Journal of Supercomputing,* pp. 1-18, 2016.

[23] I. V. Osipov, A. A. Volinsky and A. Y. Prasikova, "E-Learning Collaborative System for Practicing Foreign Languages with Native Speakers," *International Journal of Advanced Computer Science and Applications,* vol. 7, no. 3, 2016.

[24] S. Dutton, "WebRTC in the real world: STUN, TURN and signaling," [Online]. Available: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/. [Accessed 20 2 2016].

[25] W3C, "Screen Capture," 14 6 2016. [Online]. Available: https://www.w3.org/TR/screen-capture/.

[26] W3C, "Media Capture and Streams," 19 5 2016. [Online]. Available: https://www.w3.org/TR/mediacapture-streams/.

[27] "MediaRecorder API," Mozilla Developer Network, [Online]. Available: https://developer.mozilla.org/en/docs/Web/API/MediaRecorder_API. [Accessed 9 9 2016].