FH JOANNEUM (University of Applied Sciences)

WebRTC

Development of a browser based real-time peer-to-peer remote support application

Bachelor Thesis

submitted in conformity with the requirements
for the degree of
Bachelor of Science in Engineering (BSc)

Bachelor's degree program **Internettechnik**
FH JOANNEUM (University of Applied Sciences), Kapfenberg

supervisor: Johannes Feiner

submitted by: Michael Stifter
personal identifier: 1310418054

01 / 2016

info

(nicht alles gelesen)

Obligatory signed declaration:

I hereby declare that the present bachelor's thesis was composed by myself and that the work contained herein is my own. I also confirm that I have only used the specified resources. All formulations and concepts taken verbatim or in substance from printed or unprinted material or from the Internet have been cited according to the rules of good scientific practice and indicated by footnotes or other exact references to the original source.

The present thesis has not been submitted to another university for the award of an academic degree in this form. This thesis has been submitted in printed and electronic form. I hereby confirm that the content of the digital version is the same as in the printed version.

I understand that the provision of incorrect information may have legal consequences.

Michael Stifter

Graz, 30.01.2016

pdf feedback

2016-01-27

Table of contents

Problem
Hypothesis
(Supposed Solut.)
Approach = Method

State of the art

= criteria ?

• Overall "maturity"
• personal opinion

= outlook ??

Abstract

Content

including results

1 Introduction

Assistive technology has been in use in factories for a few years now. Also known as remote support applications, they enable on-site personnel to repair malfunctions under support of experts, while they are connected via audio and video streams. For companies, this brings the substantial advantage that disruptions can be repaired significantly quicker, without the necessity of an expert having to be physically present.

Problem Need ✓

Proprietary video chat applications like Skype come with a few disadvantages, though: The data flows over a third party server. Companies dealing with sensitive data might not want that, as they can never be sure that their data does not fall into the wrong hands. Furthermore, the data always streaming over a server is automatically coming with higher network latency for the data transfer.

pro / con ✓

To eradicate these problems, Web Real Time Communication (WebRTC) could be used instead of a proprietary video chat application. With WebRTC, a server is only needed in order for users to find each other and set up a connection. After that, the data is going directly from user to user, or peer-to-peer, thus reducing network latency significantly. Furthermore, data encryption is mandatory for all components of WebRTC.

suggestion idea ✓

expected results ✓

criteria ✓
alternatives ✓

This thesis is structured as follows: In the first part, the key features of remote support applications described, as well as an overview of possible technologies to implement such applications. In addition, similar research regarding this field of study will be compared. In the second part, WebRTC will be presented as a potential implementation method and its advantages and disadvantages discussed. Third, the essential insights regarding the development of a prototype application will be highlighted. After this, the results of the evaluation of the prototype application will be addressed. Finally, the findings and experiences of the prototype development will be summarised and possible ways of extending the current work will be outlined.

Structure
≈ method ✓

prototype ✓
eval ✓

Idea:
"story"
Use Case Scenario
for examples
for the whole document

Breite Stride
auch mal
andere Quelle
zwischen durch

block

2 Analysis of remote support applications

In factories, troubleshooting malfunctioning machines can be an arduous task. With complex devices, it is often necessary for an expert to conduct the repair or maintenance because local factory personnel is not trained to do it themselves. In the past, this required physical presence of the expert on site of the factory. However, in most cases it is sufficient for the well-trained person to assist a technician without special knowledge in this field through the use of video and audio streams, thus reducing costs for travel and the time until work completion significantly (cf. Chen et al. 2013, p. 1).

This chapter will take a deeper look into the necessary requirements of remote support applications, their essential features as well as the technology and hardware that are required to implement them. Finally, a conclusion will be drawn how a remote support application in a factory setting can be implemented to exploit its full potential.

2.1 Requirements

Huang et al. (2013, p. 1f) define several requirements that remote support applications need to fulfil in order to be beneficial: First, it is necessary for the cooperating parties to communicate with each other by speech. Second, the helping expert should be able to have the same visual field as the worker to be able to provide a maximum level of assistance. Furthermore, „the helper should be able to point to the objects in the workspace and use [...] hand gestures to guide the worker" (Huang et al. 2013, p. 2), which the worker should in turn be able to see. Ideally, both hands of the worker are free to enable the performance of physical tasks under a minimum of confinement in the working environment and, additionally, increase the safety conditions for the worker. Last, it is important for the worker to have the ability to move around the factory (cf. Huang et al. 2013, p. 2).

2.2 Essential features

2.2.1 Audio and video stream

Huang et al. (2013, p. 2) argue that a setting where the participating parties are able to hear and to speak to each other is substantially more effective than when they have to communicate over text messages. They proposed the most efficient workspace setup to be one where the helper has a „panoramic view of the worker's workspace“ (Huang et al. 2013, p. 2). This is essential in keeping the overall awareness of the working environment. The worker, on the other hand, does not need to see the helper or his enviroment on screen, but instead his own video feed enhanced with overlay indicators suggesting possible solutions to a task. Additionally, they should able to communicate over a wireless network over microphones and speakers (cf. Huang et al. 2013, p. 2).

2.2.2 Overlay indicators

It is possible that some technical details are difficult to explain by speech only, even when the helper has a complete understanding of the solution to the problem. Chen et al. (2013, p. 5) describe this situation as an „uneven […] knowledge distribution“ between the two involved parties. A solution to this problem could be the opportunity to draw simple sketches on the screen, which are transferred to the other person’s screen and subsequently rendered there on top the video feed. These overlay indicators can significantly improve the mutual understanding between the helper and the worker about the problem at hand and possible remedies (cf. Chen et al. 2013, p. 5).

2.2.3 Gestures

Chen et al. (2013, p. 5) propose another solution to improve the understanding in the assistance process: The display of hand gestures in addition to the previously described overlay indicators. This can be particularly useful when there is a large number of different components on the screen or the worker has a limited amount of knowledge about the malfunctioning system. In that case, hand gestures from the helper can be a valuable addition to verbal comments and overlay indicators. Possible gestures could include pointing to specific objects, holding and pressing, wiping as well as turning or screwing objects in certain directions.

While this feature is indisputably helpful, its implementation, however, cannot be considered trivial. It requires capturing the helper’s hand on top of a black background with a separate camera. Afterwards, the image is processed with a grey-scale function and, subsequently, converted into a mask image, where pixels below a certain threshold value are set to black, and those above are set to white. Finally, all black pixels are becoming transparent, and the resulting image can be shown on top of the video feed of the worker (cf. Chen et al. 2013, p. 5).

Idee: Matrix/ Table
criteria = requirements

2.2.4 Pause video feed

An additional essential feature is the possibility to pause the video feed. A constantly running video stream would aggravate the assistance of the helper immensely, because the helper would have to draw the indicators on a moving, most likely unsteady image. This is unacceptable in terms of usefulness for both the helper and the worker. Consequently, a remote support application must feature the possibility to freeze the video feed in order to provide a maximum level of assistance (cf. Chen et al. 2013, p. 4).

2.3 Connection architecture

See Figure X

Fig X

2.3.1 Client-server

In a client-server network, both parties exchange data over a network connection. Clients can request files from the server, for instance via protocols like HTTP, or any other protocol the server can process. The server’s task is to fulfil the request of the client. In other words, the server *responds* to it. In a standard HTTP environment, this principle is called request-response. (cf. Sinha 1992, p. 78f).

While HTTP is a stateless protocol, it is possible to maintain a full-duplex connection between client and server since the introduction of WebSockets. The standard HTTP connection is upgraded to a WebSocket connection with an additional initial handshake, which enables WebSockets to use the same port as used for the HTTP connection. A significant advantage of this approach is the low latency for network transfer (cf. Davies, Zeiss & Gabner 2012, p. 3).

For the implementation of a remote support application using this architecture, this implies that if two clients want to exchange data like audio and video streams, the server must implement the logic to facilitate this functionality. The main tasks are setting up a connection between two users and transfer the received data from one client to the other. It has to be noted, that in the scenario at hand there is no actual connection between the users but rather separate client-server request and response connections handled by the server. To the clients, however, it appears like they are connected together directly, while in reality the data is transferred over the server. Consequently, the latency for the network transfer is higher, even when WebSockets are used for maintaining connections between the server and the clients.

2.3.2 Peer-to-peer

Comparatively, with peer-to-peer connections, clients are linked together directly, over the shortest available network path. This setup results in lower latency for the network transfer compared to a client-server architecture. It has to be noted, however, that in order for setting up a peer-to-peer connection, a management server is still necessary for the clients to find each other and start the connection. After the connection setup, the data is exchanged directly, without passing the server. This paradigm benefits remote support applications because they are inherently data-intense due to the constant amount of data transfer of audio and video streaming.

One peer-to-peer technology that has emerged over the past few years is Web Real-Time Communication (WebRTC). It enables web browsers and mobile applications to share peer-to-peer connections without the installation of additional software or plugins. This offers a substantial advantage to web developers, who are now able to implement real-time communication applications like video chats for web browsers with the use of a JavaScript API, without any special knowledge about telecommunication technology.

2.4 Vision enhancing technologies

As indicated in Chapter 2.2.2. above, overlay indicators can significantly improve the usefulness of a remote support application. To implement this functionality, a vision technology like Augmented Reality or mixed reality could be used. The characteristics of these two technologies will be examined in the following.

Definition ✓

2.4.1 Augmented Reality

Augmented Reality bridges the gap between the real world and Virtual Reality, a world that is entirely generated by computers. It does so by enhancing the senses of human beings, most commonly the visual sense, though less frequently also the sense of hearing or smell (cf. Bonsor n.d.). In other words, „Augmented Reality adds graphics, sounds, haptic feedback and smell to the natural world as it exists“ (Bonsor n.d.). Popular applications like Wikitude, for instance, recognize tourist attractions through user's camera feeds and label them with information about it on the screen (cf. Wikitude 2016). Most commonly, there is a tracking engine that examines still frames of the video feed and matches them against specific patterns. This technique originates from the field of computer vision.

Optional: hier würde Fußzeile mit URL genügen

Nevertheless, Augmented Reality applications suffers from several disadvantages. First, they commonly need some form of visual markers in order to be recognized by the tracking engine. The markers need to be placed on the object that should be augmented, which results in administrative effort. Some Augmented Reality frameworks, like Vuforia for instance, offer object recognition simply by scanning its outline (cf. Vuforia 2016). However, this technology could still prove to be complicated to use due to the necessity of a high resolution camera that can handle insufficient or mixed lighting conditions, which could be expected in factories.

2.4.2 Mixed reality

Mixed reality offers a flexible combination of Augmented and Virtual Reality. In mixed reality, users can see computer-generated objects in addition to the real world. Admittedly, the line between Augmented Reality and mixed reality can be described as vague at best and might have only been created for marketing purposes (cf. Johnson 2015). The main difference between them is that in mixed reality the computer-generated objects are not triggered by analysis of image features, but rather by a different, external source, for instance by a human being drawing overlay indicators on the screen.

2.5 Hardware

There are several different types of hardware that can be utilized in the context of remote support applications. As stated above, an essential characteristic is the ability to move around freely. As a result, only mobile devices qualify to be used by the worker. It must be noted that for the helper, on the other hand, desktop computers are entirely eligible devices, as there is no imminent need for moving around while assisting the worker. Broadly speaking, there are two main groups of mobile devices, which will be described in more detail in the following.

2.5.1 Handheld devices

The term handheld devices refers to all electronic telecommunication devices that can be carried around and, as the name implies, held in a hand. Handheld devices are in most cases equipped with a display screen that is able to process touch inputs. Most commonly, handheld devices are smartphones and tablets. For remote support applications, both types can be used. Tablets have a slight advantage of larger screen sizes, while smartphones, in comparison, are less disruptive as they demand less space when carried around. All handheld devices, however, prevent

URL....

workers from using both hands, thus limiting them in carrying out their work without distractions.

2.5.2 Wearable devices

Idea: Images, examples

In brief, a wearable device can be described as electronic technology which can be worn on the body, without the need of holding it (cf. Tehrani & Andrew 2014). Over the last years, there was a vast amount of newly presented wearable devices, ranging from smart glasses like the Google Glass to smart watches, like Apple's iWatch. While watches do not meet the requirements for being a part of a remote support applications due to the lack of a camera and a significantly limited screen size, smart glasses, on the other hand, can be employed to be used in such a setting. They have a limited screen size compared to tablets but come with the substantial advantage that the worker can perform tasks using both hands.

Huang et al. (2013) propose an alternative to the previously described devices, as they are rather targeting the consumer market than specific industries and environments. Their solution consists of a common safety helmet that is equipped with a camera capturing the visual field of the worker, a near-eye display as well as a microphone and headphones for speech communication with the helper. They decided to use a near-eye display because it enables users to keep a clear view of the surrounding environment, which is particularly important in safety-critical settings like production factories. (cf. Huang et al. 2013).

optional: personal opinions pro/con useable in your context

2.6 Software

2.6.1 Desktop applications

A desktop application is software that runs standalone on a PC or laptop. It needs to be installed on a single device before it can be executed. If one or more parts of the software need to be changed, a new version or update of the software has to be installed. Typical desktop applications include text processors and video or music players (cf. Smith n.d.). Popular programming languages for the development of desktop applications are Java and Microsoft .NET. While applications written in Java can be used on any desktop device that has Java installed, those written in Microsoft .NET can only be executed with their full potential on devices that run on the Microsoft Windows operating system.

2.6.2 Mobile applications

Mobile applications are computer programs that run on mobile, mostly handheld devices like smartphones and tablets. Similar to desktop applications, they run natively within the device's operating system. Native mobile application presents developers with the problem that applications have to be separately implemented for the different operating systems, for instance in Java for Android or in Swift for iOS.

There are several frameworks, like Apache Cordova (2016) for instance, which try to solve this problem by offering developers the option to generate applications for smartphones from a single code base to a variety of operating systems. Typically, these cross-platform applications are written in a similar way as web-based applications, using HTML, JavaScript and CSS, which is then displayed in a special web-view layer within an operating system specific native application. While this offers developers more flexibility regarding the portability of their application and a significantly reduced time required for the implementation, cross-platform applications typically cannot exploit the operating system's full potential, due to their one-size-fits-all approach (cf. Ciman, Gaggi & Gonzo 2014).

2.6.3 Web-based applications

In contrast to the previously described application types, web-based applications do not need to be locally installed on the device, but are rather accessed through a web browser. They consist of HTML pages, which can either be static files or dynamically created ones, with additional JavaScript and CSS files to extend and improve the web page's layout and functionality. Typically, web-based applications are built on the client-server principle. Clients request the web page from the server, and displays it in the browser upon response arrival.

Compared to desktop applications, web-based applications offer the advantage that they can be maintained with less effort, since the software only needs to be changed once on the server it runs on, instead of each device that has installed it. Furthermore, web-based applications can be used from anywhere as long as the device is connected to the internet, while desktop applications are physically constrained to the device they are installed on. On the other hand, due to their confined environment, desktop applications are less vulnerable to security risks (cf. Smith n.d.).

One approach that has received a substantial amount of attention over the last years is responsive web design. Its aim is the „creation of web sites that take into account different types of devices, usually from mobile phones to desktops, and optimize viewing experience for the device at hand“ (Voutilainen & Salonen 2015, p. 1f). This is achieved through the use of flexible grids and images in addition to CSS3 media queries. With media queries, it is possible to create different CSS rules depending on specific attributes of the device, like screen size or the current orientation of the device (cf. Johansen, Pagani Britto & Cusin 2013, p. 1). This enables developers to write web applications that can be used on devices with large screens as well as device with very small screens, like smartphones or even wearables like smart glasses, without considering the device's operating system.

2.7 Research conclusions

The research at hand has led to the following conclusions about the prototype development of a remote support application: In order to target a maximum possible number of devices with a single code base, the prototype will be developed as as web-based application that can be accessed through browsers. With this choice, the application can be used on desktop computers with large screens as well as smartphones or even smaller devices like smart glasses or possibly smart watches, provided they are equipped with a web browser application.

W
[illegible]

For the connection architecture, the application will use the client-server model for the user discovery and the connection establishment, and peer-to-peer connections using WebRTC after finishing this process. This removes the necessity of implementing the complex logic behind video chat applications regarding network connections and security as well as video and audio streaming, since that is already part of WebRTC's default functionality and can be accessed through the JavaScript API directly in the web browser.

To implement vision enhancement, mixed reality will be used. With HTML5 video and canvas elements, it is possible to render helping indications on top the video feed without any complex logic behind it. Consequently, no computer vision algorithms will be needed to recognize objects in the workspace to perform enhancement with Augmented Reality, thus reducing the technical complexity of the prototype application considerably.

Why WebRTC?
Alternatives!

3 WebRTC

Loreto & Romano (2014, p. vii) describe WebRTC as a „new standard that lets browsers communicate in real time using a peer-to-peer architecture“. This is especially interesting because it enables developers to build real-time audio and video streaming applications without any external plugins or software.

This chapter will give an overview of WebRTC, its API components and the essential elements necessary to set up a peer-to-peer connection. The history of WebRTC will be discussed as well as advantages and limitations of this technology, together with its current development status.

3.1 Overview

Loreto & Romano (2014, p. vii) imply that WebRTC joins together two technologically related fields, which were still considered separately in the past: Web development and telecommunication applications. This might stem from the mutlitude of facets that have to be considered in the telecommunication industry, while web development is a more self-contained environment.

3.1.1 Architecture

WebRTC is built on a C++ API for managing peer-to-peer connections. Web applications can interact with it through the web API, written in JavaScript. It is designed in a way that developers do not need to handle network transport, audio and video engines. The overall architecture is depicted in the ~~figure~~ Figure 1 below (cf. WebRTC Architecture n.d.).

maybe also explain what is important in the figure, what can we learn from the image

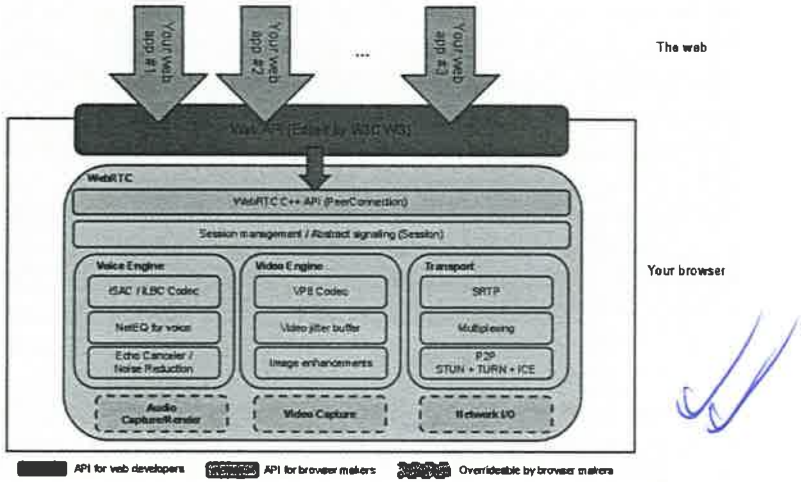


Figure 1: Overall WebRTC architecture

3.1.2 Functionality and features

Essentially, WebRTC enables users to establish secure audio and video streams to other peers, directly in a web browser. This is achieved without the use of external software or plugins, which was not possible before the arrival of WebRTC. It is important to note that the connections are established peer-to-peer, meaning that there are no third-party servers involved in the data traffic, only in order to set the connection up. This reduces network latency and provides an additional layer of security (cf. Azevedo et al. 2015).

In WebRTC, the well-known client-server model of the Internet is extended with peer-to-peer connections between browsers. Loreto & Romano (2014, p. 2) describe the usual scenario „to be the one where both browsers are running the same web application, downloaded from the same web page“. This is illustrated in the figure below (Loreto & Romano 2014, p. 3).

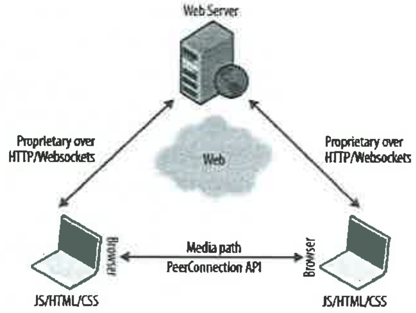


Figure 2: The WebRTC triangle

WebRTC web applications use web browsers as a communication interface between users. Developers implement the desired functionality using the standardized WebRTC API, written in JavaScript. This API handles the functions that are vital for a real-time communication application, like connection management, audio and video stream access and encodings as well as data encryption (cf. Loreto & Romano 2014, p. 3f).

3.1.3 History

At the end of May 2011, Google announced for the first time that they were working on WebRTC and made it available to the public (cf. Alvestrand 2011). In November 2011 the first implementation of WebRTC was added to Google Chrome (version 23). A little more than a year later, at the beginning of 2013, Mozilla Firefox added their first version of WebRTC, although at that stage it only supported the *navigator.getUserMedia* function (cf. WebRTC Tutorial 2014).

An important milestone was reached in February 2013, when it was possible for the first time to use WebRTC across browser borders together on Google Chrome and Mozilla Firefox. In addition, over summer and fall of 2013, the first mobile browsers supported WebRTC, when first Google Chrome for Android and a short time afterwards Mozilla Firefox for Android added their first implementations (cf. WebRTC Tutorial 2014).

3.1.4 Advantages

As Grégoire (2015, p. 1) points out, WebRTC comes with the substantial advantage of platform independence. There is no need for any external plugins to be installed, it can be used directly in a web browser. This is convenient for developers, since they do not have to ensure that their applications run on various operating systems, like Windows, Linux or Mac OS. On top of that, browser applications will also automatically work on most handheld devices, although some features from desktop computers might only be available to a limited extent, and the web page might need a responsive design to exploit its full potential.

Furthermore, encryption is mandatory for all WebRTC components, including the signaling necessary to initiate a peer-to-peer connection. As a result, the entire data transfer of WebRTC can be considered securely encrypted. For media streams, the Secure Real-time Transport Protocol (SRTP) is used, for other data the Datagram Transport Layer Security (DTLS) protocol is used. Both protocols are standardized and commonly used (cf. A Study of WebRTC Security 2015).

Aktuality ✓

Another interesting topic is WebRTC's media model. It is designed in a way that developers do not need any knowledge about audio and video codecs. It was decided by the IETF that a minimum set of audio and video codecs must be implemented by browsers to ensure a common ground between applications running on different platforms (cf. Loreto & Romano 2014, p. 6). However, in the process of setting up a peer-to-peer connection, WebRTC tries to find the best-fitting codec that both parties' browser has implemented or access to. Consequently, this could be a different, non-mandatory audio or video codec.

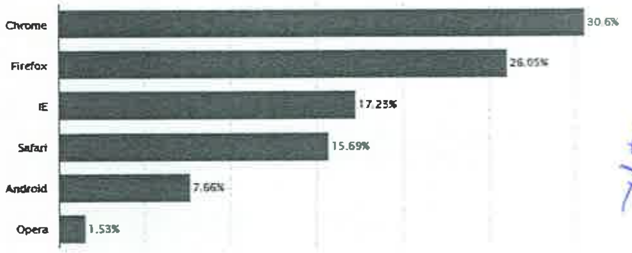
By design, WebRTC supports two audio codecs, OPUS and G.711. This decision was made with regard to the fact that these are free of royalties. However, most common telecommunication systems use different codecs, for instance AMR or AMR-WB for mobile devices, therefore the possibility of creating an interoperability application is severely aggravated, which could lead to the addition of more mandatory codecs in the future (cf. Bertin et al. 2013, p. 2)

For a long time, it was undecided which video codec would be mandatory to implement. It was a choice between the VP8 codec developed by Google or the more commonly used H.264. Similar to the audio codec situation, VP8 is free of royalties, while H.264 is more widely used by standard telecommunication applications, for instance by Skype (cf. Bertic et al. 2013, p. 2). Finally, in November 2014, an agreement was reached that both VP8 and H.264 would be mandatory video codecs in WebRTC (cf. Levent-Levi 2014).

3.1.5 Limitations ✓✓

A significant limitation to WebRTC currently is that it is still under development and although there is a valid W3C standard, it is still possible that some functions might be added, changed or removed. As a result, many browsers still use their own vendor prefixes for methods. For instance, the RTCPeerConnection in the W3C standard is called webkitRTCPeerConnection in Google Chrome and mozRTCPeerConnection in Mozilla Firefox. This introduces additional sources of errors for developers. However, Google maintains the open source library adapter.js that helps programmers solve this problem. A more detailed explanation to adapter.js can be found in chapter 4.2.4.

Furthermore, not all web browsers support WebRTC yet. For now, only Google - Chrome, Mozilla Firefox and Opera are able to establish interoperable WebRTC connections. As shown in the figure below, these three browsers accounted for roughly 58% of the market share in Austria in 2014 (Statista 2015). Although this is a promising quota that will likely rise further in the near future, it cannot yet be expected that an arbitrary device is capable of using WebRTC.



[illegible] ✓
[illegible] ✓
[illegible] ✓

Figure 3: Web browser market share in Austria in 2014

It has to be noted that Microsoft's new web browser, Edge, is missing in this chart. In October 2015, it was for the first time possible to set up a peer-to-peer connection between Microsoft Edge and other WebRTC-capable browsers. In its current state, however, Microsoft Edge is not able to open DataChannel connections and the mandatory video codec implementations are also not supported yet (cf. Hancke 2015).

The fact that WebRTC is running natively in web browsers has many advantages, although it also introduces some downsides, according to Grégoire (2015, p. 1). A substantial limitation is the restricted access to storage media on the device through the browser. A remedy to this disadvantage could be the use of the browser's local storage, a feature that was introduced in HTML5 (cf. Ranganathan & Sicking 2015). Additionally, a new File API is currently under development that will enable web browsers to access the local file system from the web browser, in accordance with a thorough security concept (cf. Hickson 2015).

3.1.6 Current status

For now, according to What's next for WebRTC? 2015, around 720 companies use WebRTC in some way in their products. Popular applications like Google Hangouts use it for creating DataChannels between users in chats and secure session description mechanisms (cf. Hancke 2014).

Currently, the developers behind WebRTC at Google are working on implementing the new version 1.2 of the encryption protocol DTLS and other enhancements regarding improvement of video and audio on mobile devices (cf. What's next for WebRTC? 2015).

3.2 API components

WebRTC consists of three main components, which developers have to implement and connect together in order for the application to work as intended. These components are called MediaStream, PeerConnection and DataChannel. The functionality and details of all three will be explained in the following section.

3.2.1 MediaStream

To broadcast audio and video streams over the Internet, *MediaStream* objects are used. They enable the developer to interact with the streams, like displaying it in the browser window, taking snapshots or sending it to other users (cf. Loreto & Romano 2014, p. 6).

Before using a MediaStream object, it is necessary to get access to a media stream from a local media-capture device. This could be a camera or a microphone from a laptop or a smartphone. Developers can request access to these *LocalMediaStreams* through the function *navigator.getUserMedia()*. It is possible to specify the type of LocalMediaStream to be requested, audio, video or both. This is done in a configuration object that can be passed upon object initialization (cf. Loreto & Romano 2014, p. 6).

In JavaScript, the access to local media-capture devices is handled via opt-in approval from the user. When developers call the navigator.getUserMedia() function for the first time, a pop-up window asks the users if they want to grant the application access to the specified media-capture devices. This approval can be revoked by both users and developers at any time so the application no longer has access to the camera or microphone.

In December 2015, Google Chrome removed the possibility to use navigator.getUserMedia() on web pages that do not support HTTP Secure (HTTPS). With HTTPS, all data transfer around the web page connection is encrypted with Transport Layer Security (TLS), thus ensuring that the data is not transferred in plain text (What's next for WebRTC? 2015). This acts as an additional layer of security, because developers are actively encouraged to use encryption in all parts of their applications.

3.2.2 PeerConnection

Instances of *PeerConnections* allow users to communicate with each other peer-to-peer, i.e. directly from one browser to another, without any web servers involved. It has to be noted, however, that a web server is always necessary for setting up a PeerConnection between two users, in order for them to find each other. This normally happens when both users visit the same web page, running on a web server which handles the peer connection setup between users. Typically, the coordination of the connection setup is handled with XMLHttpRequests or WebSockets (cf. Loreto & Romano 2014, p. 7).

3.2.3 DataChannel

While the two previous components were mandatory for a successful WebRTC connection, the third one, *DataChannel*, is optional. It offers the possibility to send arbitrary data between users connected via a PeerConnection. The DataChannel API was modeled after the WebSocket API, with similar function calls. Like WebSockets, DataChannels also offer a bidirectional connection. Developers can open an unlimited number of DataChannels within one PeerConnection, as long as each DataChannel is specified with a unique name (cf. Loreto & Romano 2014, p. 8f).

3.3 Connection setup

3.3.1 Signaling

In the WebRTC design process, it was decided to „fully specify how to control the media plane, while leaving the signaling plane as much as possible to the application layer“ (Loreto & Romano 2014, p. 5). As a result, developers do not need to handle components like video and audio formats and encodings. They do, however, have to implement the signaling in order to set up a successful WebRTC connection themselves. In practice, this means that they have to use the right API methods in the right order (cf. Loreto & Romano 2014, p. 5).

3.3.2 NAT problem

Initially, Internet Protocol (IP) version 4 (Ipv4) was used to deliver network packets over the Internet from one host to another. It uses 32-bit addresses, thus limiting the number of possible hosts to 2^{32}, or 4 294 967 296. Due to the constantly increasing demand of new Internet-capable devices and applications, one popular method to avoid IPv4 address space exhaustion was the introduction of Network address translation (NAT). NAT enables networks to map multiple hosts inside a network to use one public IP address, therefore reducing the usage of public IPv4 addresses.

For WebRTC, however, it is vital to know the public IP address of all parties in order to set up a peer-to-peer connection between them. This functionality is achieved through the use of the Session Traversal Utilities for NAT (STUN) protocol. It enables an application to detect the usage of NAT in a host's network, and to retrieve the allocated IP address and port if that is the case. A third-party STUN server is necessary in order to obtain the host's public IP address (cf. Loreto & Romano 2014, p. 8). There are publicly available STUN servers for developers to use in this case, for example from Google (cf. Dutton 2012).

Additionally, the Traversal Using Relays around NAT (TURN) protocol extends the functionality of STUN by allowing a host inside a network that uses NAT to receive a public IP address from a relay server (cf. Loreto & Romano 2014, p. 8). Consequently, the host is able to „receive media from any peer that can send packets to the public Internet“ (Loreto & Romano 2014, p. 8).

3.3.3 ICE candidates

As described above, WebRTC uses PeerConnection objects to establish connections between two users. To do so, it uses the Interactive Connectivity Establishment (ICE) protocol. It facilitates peers to detect information about their network's topology in order to find one or more connection paths between them, by using a variety of network protocols (cf. Loreto & Romano 2014, p. 117). Dutton (2012) points out that ICE starts with the User Datagram Protocol (UDP) first, as it has the lowest network latency. In the case that the UDP connection attempt remains unsuccessful, the Transmission Control Protocol (TCP) is used, HTTP and lastly HTTPS.

In the WebRTC JavaScript API, it is necessary to set a valid ICE server Uniform Resource Locator (URL), called the ICE Agent, in a configuration object whenever a new PeerConnection object is created (cf. Loreto & Romano 2014, p. 117). Below is a minimal example for doing so, with the URL of the publicly available server from Google (cf. Dutton 2012).

```
var config = {“iceServers“: [{“url“: “stun:stun.l.google.com:19302“}]};
var peerConnection = new RTCPeerConnection(config);
peerConnection.onicecandidate = onIceCandidate;
```

Listing x

Super wäre auch Zeilennummern

Each time a new ICE candidate is found, the ICE Agent updates the PeerConnection object and calls its *onicecandidate* callback function, in the example above *onIceCandidate* (cf. Loreto & Romano 2014, p. 117).

For this ICE candidate negotiation process, a server is always needed. Its sole purpose, however, is to relay the ICE candidate messages from one peer to another. The whole process is illustrated in the figure below (cf. Loreto & Romano 2014, p. 118).

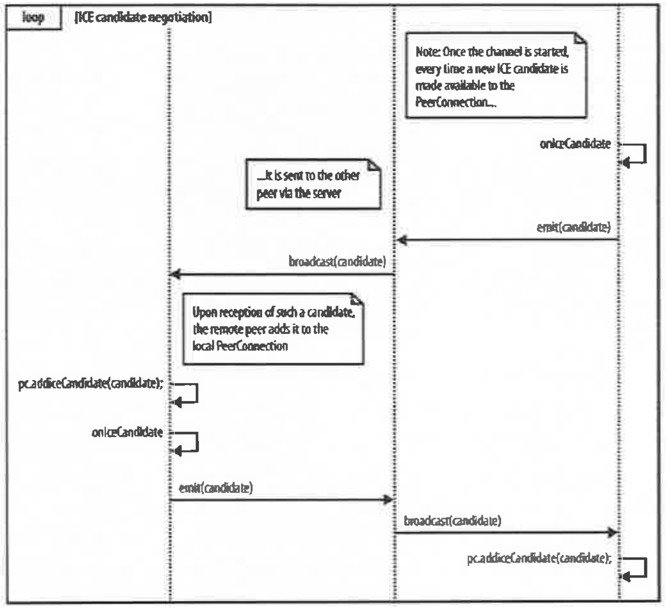

loop
[ICE candidate negotiation]
Note: Once the channel is started, every time a new ICE candidate is made available to the PeerConnection...
onIceCandidate
...It is sent to the other peer via the server
emit(candidate)
broadcast(candidate)
Upon reception of such a candidate, the remote peer adds it to the local PeerConnection
pc.addIceCandidate(candidate);
onIceCandidate
emit(candidate)
broadcast(candidate)
pc.addIceCandidate(candidate);

Figure 4: ICE candidate negotiation process

3.3.4 Session description offers and answers

It is is customary in telecommunication applications that one user calls another user. In the WebRTC architecture, this is accomplished by creating a PeerConnection object and, consequently, creating an offer and sending it to the user who is being called. The creation of the PeerConnection was already described above, an offer is created by calling its *createOffer* method (cf. Dutton 2012).

```
peerConnection.createOffer(setLocalAndSendMessage, onSignalingError,
mediaConstraints);
```

The parameters of the *createOffer* method are a success callback handler, an error callback handler as well as possible constraints regarding the media encoding and quality. One simple implementation of the success callback handler is described below. The session description object is stored locally in the PeerConnection object and, additionally, serialized and sent to the remote peer (cf. Dutton 2012).

```
function setLocalAndSendMessage(sessionDescription) {
    peerConnection.setLocalDescription(sessionDescription);
```

```
    // send session description to peer
    sendMessage(sessionDescription);
}
```

Upon arrival of the session description message, the remote peer registers it in its PeerConnection object. Similar to the process of creating the offer, the remote peer answers by calling the *createAnswer* method. It takes the same three parameters as the *createOffer* method (cf. Loreto & Romano 2014, p. 122).

```
peerConnection.setRemoteDescription(new RTCSessionDescription(message));
peerConnection.createAnswer(setLocalAndSendMessage, onSignalingError,
mediaConstraints);
```

Now, both users have exchanged session descriptions and details on how they can be located over the Internet, with the help of the management server. They are now directly connected, thus no longer need the management server to communicate with each other (cf. Loreto & Romano 2014, p. 122).

3.3.5 Data channels

Up to now, it is only possible for two users to communicate via audio and video streams. To extend this to the possibility to send arbitrary data from peer-to-peer, a DataChannel is needed. Sending data in this way, directly, without third-party servers involved, comes with substantial benefits, thanks to the low network latency and high troughput (c.f. Dutton 2012). Dutton (2012) further points out that there are many use cases for this functionality, like remote desktop applications, file transfer, gaming or real-time text chat.

The API methods of DataChannels were deliberately modeled after those from WebSockets, therefore most web developers should be fairly familiar with the syntax. Additionally, the DataChannel API brings more advantages for applications, like the usage of multiple prioritizable channels within one PeerConnection, mandatory, automatic encryption as well as the support of reliable and unreliable message delivery (cf. Dutton 2012).

One user – in most cases the one creating the PeerConnection – also creates a DataChannel. There can be an unlimited number of DataChannels within one PeerConnection, identified by unique names. Afterwards, three event handlers are attached to the DataChannel, which will be called each time this event fires (cf. Loreto & Romano 2014, p. 125).

```
var sendChannel = peerConnection.createDataChannel("sendChannel",
{"reliable": "true"});
sendChannel.onopen = handleSendChannelStateChange;
sendChannel.onmessage = handleDataChannelMessage;
sendChannel.onclose = handleSendChannelStateChange;
```

Because DataChannels are bidirectional, the other user does not have to create one himself. In contrast, he only has to bind an *ondatachannel* event handler to the PeerConnection object, which in turn attaches the same three event handlers to this receive DataChannel (cf. Loreto & Romano 2014, p. 125ff).

```
peerConnection.ondatachannel = gotReceiveChannel;

function gotReceiveChannel(event) {
    receiveChannel = event.channel;
    receiveChannel.onmessage = handleDataChannelMessage;
    receiveChannel.onopen = handleReceiveChannelStateChange;
    receiveChannel.onclose = handleReceiveChannelStateChange;
}
```

It is important to note, however, that unlike MediaStream and PeerConnection, DataChannel is optional to a WebRTC connection and does not necessarily have to be implemented by the developer if it is not needed.

X roter Faden X

Summ, Überleitung

warum geht es wie weiter

(warum nächste Kapitel)

roter Fad. ✓

Intro ✓

4 Prototype

With the insights and findings of the research, a prototype application was developed. The application consists of two core parts: The management server and the web interface. A vital component of the application is the overlay indicator feature, which is part of the web interface. These three most important components of the prototype will be discussed in detail in the following section. The complete source code can be found in the appendix of this thesis.

4.1 Management server

The management server is the main component of the prototype application. It performs the following tasks: First, it serves the web page and related static files, like JavaScript source files and Cascading Style Sheets (CSS). Second, it manages WebSockets for full-duplex communication to each connected browser client. Third, it carries out management and control tasks in order to set up peer-to-peer connections between users. The implementation and tasks of the management server will be described in more detail below.

4.1.1 Implementation

It was decided to use Node.js as a platform for the management server. Node.js brings the significant advantage of using the same programming language, JavaScript, on the backend and the frontend. As a result, a considerable amount of programming code could be used for both parts.

4.1.2 Web server

The web page, which has the role of the main user interface of the application, is served by the management server as well as all static source files. This includes JavaScript and CSS files of the application and the two external JavaScript source files, jQuery and adapter.js.

4.1.3 WebSockets

When a user opens the web page and enters a user name, a secure WebSocket connection to the management server is established. As a consequence, both parties then share a full-duplex connection and are able to exchange data at any time. This WebSocket connection stays alive until the user decides to leave the web page.

4.1.4 Management and control tasks

Each time a new user connects to the web page via WebSocket for the first time, the management server assigns a unique id to the user. This id is used as identification when control messages are sent to users. On the management server, there are two types of control messages: First, there are server messages. These serve the sole purpose of administration tasks, like assigning user ids or broadcasting available users, which happens every time a new user connects to the web page. Second, there are relay messages, which are used to set up peer-to-peer connections between users.

4.2 Web interface

The web interface enables users to interact with each other. Currently, users only need to provide a user name to use the application, no password is required. This is due to the fact that it is only a prototype for now. In a production environment, however, this would raise serious security issues.

4.2.1 HTML5 elements

Modern HTML5 elements were used on the web page where it was possible. This includes header, footer and section elements for the main page structure as well as video and canvas elements to display the video feeds and enable users to draw overlay indicatiors without the use of external plugins.

4.2.2 User interaction

The complete programming logic and user interaction was implemented with JavaScript. This includes all WebRTC functionality and the signaling communication with the management server necessary to set up connections between two peers. These functions are triggered by the users‘ page interactions, like clicks on buttons.

4.2.3 Responsive design

Research showed that remote support applications are most commonly used on mobile devices, like smartphones, tablets or smart glasses. Accordingly, an important focus in the development process was the possibility to use the application on mobile devices. It was decided to use CSS media queries to achieve a valuable user experience for all device types and sizes. Currently, there are five size breakpoints in the main CSS file, which could be easily extended to support a larger number of different screen sizes if necessary.

4.2.4 Facilitating libraries

One important principle for the prototype development was to use as few external libraries and plugins as possible. However, it was economically reasonable to use some external code to reduce the programming effort in certain areas. In the end, only two external JavaScript libraries were used: jQuery and adapter.js.

jQuery (2016) was used because it is currently a de-facto standard in web development, thanks to its essential features for HTML document manipulation, event handling and useful AJAX functions. Additionally, it enables developers to write code for all common web browsers, without needing to worry about syntax differences between them.

adapter.js is an open source library maintained by Google that helps developers abstract browser prefixes and API changes in the WebRTC specification. The use of adapter.js significantly reduces the amount of code necessary to implement WebRTC functionality in multiple browsers. This is due to the fact that WebRTC is currently still under development, and therefore each browser uses different function prefixes. Additionally, some API functions might be added, renamed or removed in the future. Through the use of adapter.js, developers do not need to regularly check if these parts have changed (cf. Loreto & Romano 2014, p. 96).

4.3 Overlay indicators feature

An essential feature of the prototype application is the overlay indicators feature. It enables two users, sharing a peer-to-peer connection, to assist each other through drawings on the other user's screen. For now, it is possible to track the user's movements on the canvas with the mouse or with the finger or stylus on handheld devices.

4.3.1 Implementation

For the implementation of this feature it was decided to use two canvases on top of each other. One canvas displays the current frame of the video element, which is bound to the media stream of the WebRTC connection. This first canvas is updated 24 times per second, appearing as a constant video stream to the user's eye. The second canvas lies exactly on top of the first one, and is used to display the overlay indicators of the remote user. Two canvases have to be used in this case because the first canvas must be cleared each time it displays the current frame of the video stream and, consequently, the drawn path would also be erased. Therefore, it is necessary to use two canvases for the implementation of this feature.

4.3.2 Mouse events

The difficult part of this feature was to track the user's movement with the mouse on an HTML canvas element. Cabanier et al. (2015) showed how geometric figures can be drawn on a canvas. Essentially, all that needs to be done is to draw a line from the previous touch point to the current touch point.

```
function drawPath() {
    var canvas = document.getElementById(“drawing-canvas“);
    var context = canvas.getContext(“2d“);
    context.beginPath();
    context.moveTo(previousX, previousY);
    context.lineTo(currentX, currentY);
    context.stroke();
    context.closePath();
}
```

In JavaScript, it is possible to add event listeners to DOM elements, for instance when the mouse moves over the element, when a mouse button is clicked on the element or when the mouse leaves the element. In order to achieve the desired drawing functionality, it is important to not only track the movement of the mouse over the element, but also track the state whether the left mouse button is currently being clicked or not. If that is the case, the code snippet from above gets executed each time the *mousemove* event handler is called and the path of the mouse is drawn on the canvas.

4.3.3 Touch events

While the implementation of the support drawing feature with mouse events was not tremendously difficult, the solution did not immediately work on handheld devices like smartphones or tablets. This is due to the fact that they are operated with fingers and styluses instead of a mouse. Some handheld devices emulate mouse events when DOM elements are touched, however, the functionality of the feature on smartphones and tablets was unpredictable and unacceptable in terms of user experience.

Eventually, a solution to this problem could be found by extending the previously described logic with touch events. Touch events are the counterparts to mouse events on desktop devices. They are, however, more complex than mouse events,

because while there is always only one mouse pointer on desktop devices, „a user may touch the screen with multiple fingers at the same time“ (Jenkov 2014).

For the feature at hand, it was decided to ignore multiple, simultaneous touches because it is assumed that the majority of users will not use more than one finger at a time to draw on the screen. Using this presumption, it was possible to implement the same drawing functionality on handheld devices by adding the same event handlers for touch events (*touchstart*, *touchmove* and *touchend*), and in them, dispatch the corresponding mouse event (*mousedown, mousemove* and *mouseup*) with the position of the touch, without any additional logic.

4.3.4 Data transfer

To transfer the drawing path from one user to another, the DataChannel component of WebRTC was used. As described in chapter 3.2.3 above, it supports the sending of arbitrary data directly from peer to peer, which exactly fulfills the requirements for the implementation of this feature.

To send the drawing path to the remote peer, in addition to the track path method being called by the canvas event listeners, a message containing the path info is sent to the connected user. On the side of this other user, the same *drawPath* method is executed with the received information.

4.3.5 Text chat

In addition to the possibility to communicate via audio and video stream and send support drawings to another, it is also possible to send text messages to the connected peer, which are displayed next to the video stream. This is especially helpful in loud environments, where it is not possible to converse with somebody. Similar to the drawing path data transfer, the text chat also uses the DataChannel to send the text messages.

5 Evaluation

Google Chrome Developer Tools Network Throttling

GPRS (50 kb/s, 500 ms latency)
Regular 3G (750 kb/s, 100 ms latency)
Regular 4G (4 Mb/s, 20 ms latency)
WiFi (30 Mb/s, 2 ms latency)

Video resolution

1280 x 720
640 x 360
320 x 180

Frame rate

30
60

User tests

WebRTC behavior under changing network conditions (connection breakdown, adaption?)

6 Possible extensions

At the time of writing, the prototype application is not suitable for use in a production environment. Several improvements and extensions would be necessary in order to remove the current limitations of the prototype. A few suggestions for possible enhancements will be outlined in the following chapter.

6.1 Screenshots

A simple, though useful improvement would be the option to save screenshots from the video chat session. Additionally, these screenshots could be also sent to the other user via the web application. This functionality could be used versatilely, for the purpose of documentation or for easily assembling user guides for the repair of malfunctioning components.

6.2 User authentication

One substantial improvement to the application would be user authentication. So far, users are not required to provide a password, anyone can use the application. While this is fine and in fact convenient during the development process, it is incongruous for live operation and raises severe security issues. One possibility to implement such a functionality without signifcant effort would be to use OAuth, an open standard which „allows secure authorization in a simple and standard method“ (OAuth 2015). With OAuth, users do not have to create a new account for using the application, but can instead use an existing account from popular web sites like Facebook or Twitter for authentication without giving away personal information about themselves.

6.3 E-mail invitations

At present, users can only call other users via the web interface. Consequently, the called user must have the web page opened in order to be notified about the incoming call. One useful extension would be the possibility to invite other users to a session by entering their e-mail address. On the management server, it would be necessary to implement some logic to generate a session id, save it along with other meta data about the session and send an e-mail with a clickable link to the invited user that leads them directly to chat session on the web page.

6.4 Cross-platform application

So far, the prototype application is only working in web browsers. While this offers flexibility, it would be useful to have a native app, especially for smartphones. Native app development, however, brings the disadvantage of having to implement the same application logic on multiple platforms. An economic solution to this problem would be the development of a cross-platform app, with a framework like Apache Cordova.

6.5 Sessions with more than two users

While there can technically be an infinite number of chats running simultaneously, the number of conversational partners per chat is limited to two. This is due to the fact that WebRTC does not natively support multi-user chats. A Multipoint Control Unit (MCU) would be necessary to provide the possibility to talk to more than one person at a time. To minimize the programming effort, it would be possible to use an open-source plugin like Janus (2016) that performs this task, altough this would introduce a vast number of external plugins and dependencies to the application.

7 Conclusion

List of figures

List of abbreviations

AJAX	Asynchronous JavaScript And XML
AMR	Adaptive Multi-Rate
AMR-WB	AMR Wideband
API	Application Programming Interface
CSS	Cascading Style Sheets
DTLS	Datagram Transport Layer Security
HTTP	Hypertext Transfer Protcol
HTTPS	HTTP Secure
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IP	Internet Protocol
MCU	Multipoint Control Unit
NAT	Network Address Translation
SRTP	Secure Real-time Transport Protocol
STUN	Session Traversal Utilities for NAT
TLS	Transport Layer Security
TURN	Traversal Using Relays around NAT
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WebRTC	Web Real-Time Communication
XML	Extensible Markup Language

ToDo: readable!

?

Bibliography

Sort by author

A Study of WebRTC Security, 2015. Available from: < http://webrtc-security.github.io/>. [4 January 2016]

Alvestrand H. 2011, *Google release of WebRTC source code*. Availble from: <http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html>. [5 January 2016]

Apache Cordova, computer software 2016. Available from: <https://cordova.apache.org/>. [11 January 2016]

Azevedo J, Lopes Pereira R & Chainho P 2015, 'An API proposal for integrating Sensor Data into Web Apps and WebRTC'. Available from: ACM. [6 January 2016]

Bertin E, Cubaud S, Tuffin S & Cazeaux S 2013, 'WebRTC, the day after'. Available from: ACM. [5 January 2016] mehr INFO ++

Bonsor K n.d., *How Augmented Reality Works*. Available from: <http://computer.howstuffworks.com/augmented-reality.htm>. [13 January 2016]

Cabanier R, Mann J, Munro J, Wiltzius T & Hickson I 2015, *HTML Canvas 2D Context*, Available from: <http://www.w3.org/TR/2dcontext/>. [3 January 2016]

Chen S, Chen M, Kunz A, Yantaç AE, Bergmark M, Sundin A & Fjeld M 2013, 'SEMarbeta: Mobile Sketch-Gesture-Video Remote Support for Car Drivers'. Available from: ACM. [11 January 2016] + Old

Ciman M, Gaggi O & Gonzo N 2014, 'Cross-Platform Mobile Development: A Study on Apps with Animations'. Available from: ACM. [15 January 2016]

Davies M, Zeiss J & Gabner R 2012, 'Evaluating two approaches for browser-based real-time multimedia communication'. Available from: ACM. [15 January 2016]

Dutton, S 2012, *Getting Started with WebRTC*. Available from: <http://www.html5rocks.com/en/tutorials/webrtc/basics/>. [2 January 2016]

WebRTC Architecture, n.d. Available from: <https://webrtc.org/architecture/>. [5 January 2016]

Grégoire JC 2015, 'On Embedded Real Time Media Communications'. Available from ACM. [4 January 2016]

Hancke P 2014, *How does Hangouts use WebRTC? Webrtc-internals analysis*. Available from: < https://webrtchacks.com/hangout-analysis-philipp-hancke/>. [6 January 2016]

Hancke P 2015, *Hello Chrome and Firefox, this is Edge calling*. Available from: <https://webrtchacks.com/chrome-firefox-edge-adapterjs/>. [6 January 2016]

Hickson I 2015, *Web Storage*, 2nd edn. Available from: <http://www.w3.org/TR/webstorage/>. [5 January 2016]

Huang W, Alem L, Nepal S, Thilakanathan D 2013, 'Supporting Tele-Assistance and Tele-Monitoring in Safety-Critical Environments'. Available from: ACM. [11 January 2016]

Janus, computer software 2016, Available from: <https://github.com/meetecho/janus-gateway>. [2 January 2016]

Jenkov J 2014, *Touch Event Handling in JavaScript*. Available from: <http://tutorials.jenkov.com/responsive-mobile-friendly-web-design/touch-events-in-javascript.html>. [4 January 2016]

Johansen RD, Pagani Britto TC, Cusin CA 2013, 'CSS Browser Selector Plus: A JavaScript Library to Support Cross-browser Responsive Design'. Available from: ACM. [15 January 2016]

Johnson E 2015, *Choose Your Reality: Virtual, Augmented or Mixed*. Available from: <http://recode.net/2015/07/27/whats-the-difference-between-virtual-augmented-and-mixed-reality/>. [13 January 2016]

jQuery, computer software 2016. Available from: <http://jquery.com/>. [15 January 2016]

Levent-Levi T 2014, *Who are the Winners and Losers of the WebRTC Video Codec MTI Decision?* Available from: <https://bloggeek.me/winners-losers-webrtc-video-mti/>. [5 January 2016]

Loreto, S & Romano SP, 2014, *Real-Time Communication with WebRTC*, 1st edn., O'Reilly, Sebastopol.

Viele Papers aktuell ✓✓

OAuth, computer software 2015. Available from: <http://oauth.net/>. [30 December 2015]

Ranganathan A & Sicking J 2015, *File API*, W3C Working Draft 21 April 2015. Available from: <http://www.w3.org/TR/FileAPI/>. [5 January 2016]

Sinha A 1992, 'Client-Server Computing', *Communications of the ACM*, vol. 35, no. 7, pp. 77-98. Available from: ACM [15 January 2016]

Skype, computer software 2016. Available from :<http://www.skype.com/en/>. [11 January 2016]

Smith J n.d., *Desktop Applications vs. Web Applications*. Available from: <http://www.streetdirectory.com/travel_guide/114448/programming/desktop_applications_vs_web_applications.html>. [15 January 2016]

Statista 2015, *Market share of web browsers in Austria in 2014*. Available from: <http://www.statista.com/statistics/421152/wbe-browser-market-share-in-austria/>. [5 January 2016]

Tehrani K & Andrew M 2014, *Wearable Technology and Wearable Devices: Everything You Need to Know*. Available from: <http://www.wearabledevices.com/what-is-a-wearable-device/>. [13 January 2016]

Vuforia, computer software 2016. Available from: <https://www.qualcomm.com/products/vuforia>. [13 January 2016] ?

WebRTC Tutorial, 2014 (video file). Available from: < https://www.youtube.com/watch?v=5ci91dfKCyc>. [5 January 2016]

What's next for WebRTC, 2015 (video file). Available from: < https://www.youtube.com/watch?v=HCE3S1E5UwY>. [5 January 2016]

Wikitude, computer software 2016. Available from: <http://www.wikitude.com/>. [13 January 2016] ¿ ?

?