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

**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

**Table of contents**

[Abstract 4](#_Toc439251630)

[1 Introduction 5](#_Toc439251631)

[2 Concept 6](#_Toc439251632)

[3 WebRTC 7](#_Toc439251633)

[3.1 Overview 7](#_Toc439251634)

[3.2 API components 7](#_Toc439251635)

[3.2.1 MediaStream 7](#_Toc439251636)

[3.2.2 PeerConnection 8](#_Toc439251637)

[3.2.3 DataChannel 8](#_Toc439251638)

[3.3 Connection setup 8](#_Toc439251639)

[3.3.1 The WebRTC triangle 8](#_Toc439251640)

[4 Prototype 10](#_Toc439251641)

[4.1 Management server 10](#_Toc439251642)

[4.2 Web interface 10](#_Toc439251643)

[4.3 Remote support drawing feature 10](#_Toc439251644)

[5 Evaluation 11](#_Toc439251645)

[6 Conclusion 12](#_Toc439251646)

[List of figures 13](#_Toc439251647)

[List of tables 14](#_Toc439251648)

[Bibliography 15](#_Toc439251649)

Abstract

# 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 stream. 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.

# Concept

# WebRTC

### Overview

## 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.

### 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 from a laptop or a smartphone or a microphone. 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. (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 November 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. (REFERENCE?)

### 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).

### 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 (REFERENCE). 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).

## Connection setup

### The WebRTC triangle

In the WebRTC architecture, 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 2015, p. 3).



Figure 1: 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).

# Prototype

## Management server

## Web interface

## Remote support drawing feature

# Evaluation

# Conclusion

List of figures

[Figure 1: The WebRTC triangle 9](#_Toc439250562)

List of tables

Bibliography

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