The Philips Remote AI Streaming (PRAIS) platform

PDEng THESIS REPORT

The Philips Remote AI Streaming (PRAIS) platform

Robin Mennens
October 2020
Department of Mathematics & Computer Science

PDEng SOFTWARE TECHNOLOGY

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

The Philips Remote AI Streaming (PRAIS) platform

Robin Mennens

October 2020

Eindhoven University of Technology
Stan Ackermans Institute – Software Technology

PDEng Report: 2020/060

Confidentiality Status: Public

Partners

Philips

Eindhoven University of Technology

Steering Group Marcel Quist
Zoran Stankovic
Alexander Serebrenik

Date October 2020

Composition of the Thesis Evaluation Committee:

Chair: Mark van den Brand

Members: Reinder Bril

Ihor Kirenko

Marcel Quist

Zoran Stankovic

Alexander Serebrenik

The design that is described in this report has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct.

Date October, 2020

Contact address
Eindhoven University of Technology
Department of Mathematics and Computer Science
Software Technology
MF 5.086
P.O. Box 513
NL-5600 MB
Eindhoven, The Netherlands
+31 (0)40 247 9111

Published by Eindhoven University of Technology

PDEng Report 2020/060

Abstract

An extremely relevant topic for Philips (and healthcare in general), is Artificial Intelligence (AI), which has the potential to improve many aspects of people's lives. Relatively new AI data sources include audio/video/data streams that deliver data in real time and enable many new AI use cases. While the combination of AI and streaming has significant potential, there are several obstacles to tackle: care providers do not always have the required (expensive) hardware to use AI algorithms, care providers rarely have the required knowledge and infrastructure to develop and maintain streaming technology, AI algorithms are not always easy to integrate because they are developed using different technologies, and AI algorithms are hard to replace once integrated. Aiming to tackle these obstacles and to enable AI streaming use cases, Philips Research is maturing *remote AI streaming*: the remote (in the cloud or on premise) execution of AI algorithms that take an audio/video/data stream as input and/or output. In this work, we present the Philips Remote AI Streaming (PRAIS) platform, which allows developers to easily build applications that require real-time audio/video/data streaming functionality. With such functionality, PRAIS enables AI streaming use cases and tackles the above-listed obstacles. As a platform, PRAIS benefits both Philips and its open innovation partners. We evaluated PRAIS during two collaborations. Firstly, a group of bachelor computer science students used PRAIS to develop demonstrators that show how PRAIS enables the sharing of AI algorithms among hospitals and the real-time analysis of Neonatal Intensive Care Unit (NICU) video and sensory data. A usability study with the students shows that PRAIS is considered easy to use. Secondly, in a collaboration with Maxima Medisch Centrum we explored how PRAIS can be used to record NICU baby footage. Such recordings are used for research purposes.

Keywords	AI, streaming, real-time, cloud, healthcare, platform
Preferred reference	The Philips Remote AI Streaming (PRAIS) platform. Eindhoven University of Technology, PDEng Report 2020/060, October 2020.
Partnership	This project was supported by Eindhoven University of Technology and Philips

Abstract

An extremely relevant topic for Philips (and healthcare in general), is Artificial Intelligence (AI), which has the potential to improve many aspects of people's lives. Relatively new AI data sources include audio/video/data streams that deliver data in real time and enable many new AI use cases. While the combination of AI and streaming has significant potential, there are several obstacles to tackle: care providers do not always have the required (expensive) hardware to use AI algorithms, care providers rarely have the required knowledge and infrastructure to develop and maintain streaming technology, AI algorithms are not always easy to integrate because they are developed using different technologies, and AI algorithms are hard to replace once integrated. Aiming to tackle these obstacles and to enable AI streaming use cases, Philips Research is maturing *remote AI streaming*: the remote (in the cloud or on premise) execution of AI algorithms that take an audio/video/data stream as input and/or output. In this work, we present the Philips Remote AI Streaming (PRAIS) platform, which allows developers to easily build applications that require real-time audio/video/data streaming functionality. With such functionality, PRAIS enables AI streaming use cases and tackles the above-listed obstacles. As a platform, PRAIS benefits both Philips and its open innovation partners. We evaluated PRAIS during two collaborations. Firstly, a group of bachelor computer science students used PRAIS to develop demonstrators that show how PRAIS enables the sharing of AI algorithms among hospitals and the real-time analysis of Neonatal Intensive Care Unit (NICU) video and sensory data. A usability study with the students shows that PRAIS is considered easy to use. Secondly, in a collaboration with Maxima Medisch Centrum we explored how PRAIS can be used to record NICU baby footage. Such recordings are used for research purposes.

Foreword

It is such a privilege that we have been able to witness the growth path of the PDEng education from multiple angles. First of all, being a PDEng student myself in the past, I learned to appreciate the multi-disciplinary technology design approaches. Secondly, with the PDEng Software Technology international team we worked closely on a fun Artificial Intelligence (AI) demonstrator assignment with 18 fellow international PDEng students and our team, and that was where we first met. And now, over the last 10 months, we witnessed your personal growth from closeby within our team. A warm and positive experience!

Where the first team assignment was about exploration for a fun and inspiring demonstrator, showing what you can do with already available AI in the cloud, your final assignment has been a major step in technical complexity, while adapting to the stakeholders' expectation levels. You were able to do that in a Philips research settings where we seek and design health solutions based on customer insights and needs. Where the requirements are typically in flux and never carved in stone. You actually supported and drove the process to get these requirements clear and focused by rapid prototyping, inspiring by tangible examples and – most of all nowadays – empowering co-creation to unleash the talents of many others. This is in high level terms our envisioned assignment.

Now more specifically on the topic, Robin turned a preliminary proof of concept into a real 'Access to AI' platform - to stream audio and video data sources (e.g. from camera, screen share, or communication apps) from wherever in the world, to an AI algorithm wherever in the world (e.g. Microsoft, Google, and Amazon clouds, as well as dedicated Philips Healthcare AI solution components).

- **This impacts people.** Experts, e.g. physicians who own the video data sources can work together with global experts owning or developing AI algorithms.

- **This impacts resources.** The ubiquitous streaming enables free design of resource distributions on-premise or in the cloud instead of inevitable embedded approaches.

- **This impacts speed of innovation.** The streaming connection to AI is simplified, so researchers and innovators can rely on provided initial set-up and can dedicate more time to their targeted use cases. (e.g. PhD students with an assignment of 3 years may otherwise spend significant amount of time e.g. approximately 10 months with the setup alone).

I observed with pleasure how you merged naturally in our Philips Research 'Scalable Service Delivery' team, with other interns coming and going. And step-by-step grew your contributions and position in the team. How you drove and supported the assignment of the TU/e SEP students team of

12, as a first validation of ‘unleashing talent of many others’, resulting in inspiring demonstrators. Up to the moment we pulled you in a customer facing position to co-create with PhD’s, researching new video use cases.

In a report that is published in 2020, at least one reference to COVID-19 must appear somewhere, and where better than in this foreword? The team circumstances changed considerably in your 3rd month of the assignment (March 2020) when the offices closed down and working from home became the new norm. Thank you for the quick and easy adaption to this new reality – in keeping up the team spirit alongside impressive contributions to the ‘Scalable Service Delivery’ team and Philips.

As said, I consider it a privilege knowing you personally and being in the position to ‘add your name’ to the annals of our joint working history in Philips!

Thank you!

Eindhoven, September 25, 2020
Marcel Quist and Zoran Stankovic

Preface

This document is the main deliverable of the Philips Remote AI Streaming (PRAIS) platform project and describes the process of designing and implementing PRAIS. PRAIS is a platform that allows developers to easily build applications that require real-time audio/video/data streaming functionality. With such functionality, PRAIS enables AI streaming use cases and tackles many technical challenges. Example use cases include:

- Real-time analysis of Intensive Care Unit (ICU) video and vital sign data can provide faster and more accurate detection of anomalies, such as apnea in neonates.
- Real-time pose detection of patients can be used to quickly detect seizures.
- Speech to text transcription enables features such as real-time (translated) subtitles, automatic transcription of a doctors consult, and real-time sentiment analysis.

As a platform, PRAIS benefits both Philips and its open innovation partners.

This project was carried out by Robin Mennens as part of his ten-month Software Technology (ST) Professional Doctorate in Engineering (PDEng) graduation project. The project was carried out within Philips Research.

The target audience of this document mainly includes people with a technical (computer science) background with an interest in PRAIS. Chapters 1, 2, and 6 are recommended reading material for people with a non-technical background.

Eindhoven, September 25, 2020
Robin Mennens

Acknowledgements

This project would not have been possible without numerous people I was happy to work with.

Firstly, I would like to thank my company supervisors Marcel Quist and Zoran Stankovic. From day one I felt welcome and part of the team. I learned a lot and was really inspired by your enthusiasm, creativity, and teamwork. This project would not have been possible without your support and supervision. Marcel, thank you for showing me around and inspiring me with the amazing things happening within Philips Research. Zoran, thank you for taking the time to explain and introduce me to the complex domain of real-time streaming. I definitely learned a lot on a technical level. Also a big thank you to Arjan, who was always able to help me out with technical issues. It was a pleasure working with you.

I would like to thank my TU/e supervisor Alexander Serebrenik for his supervision, guidance, and extensive feedback throughout. This work would not have been possible without your supervision. A special thank you for the valuable teachings regarding usability studies. I definitely learned a lot there.

A special thanks to the other team members. Melis and Livia, while our time together was short, thank you for the warm welcome and nice dinner we had together. I want to thank Roel and Ralitsa for their amazing work on the apnea and pose detection algorithms. Without your help, the SEP project definitely would have been much more of a struggle. Thank you Priyanka for your support and the valuable discussions that we had. Last but not least, I want to thank Hubrecht for his inspiring work and humor.

I want to thank my fellow PDEng trainees for the interesting, valuable, and especially fun times we had. A special thanks to Yanja Dajsuren and Désirée van Oorschot for their guidance and support throughout the last two PDEng years.

A word of appreciation goes to the external Thesis Evaluation Committee members Reinder Bril, Ihor Kirenko, and Mark van den Brand. Thank you for taking the time to read my thesis and to grade my work.

Finally, I would like to thank my family and friends for supporting and encouraging me throughout.

Eindhoven, September 25, 2020
Robin Mennens

Executive Summary

Artificial Intelligence (AI) has the potential to improve many aspects of people's lives, and thereby coincides perfectly with the Philips ambition to improve the lives of three billion people per year by 2030. Relatively new AI data sources include audio/video/data streams that deliver data in real time and enable many new AI use cases. For example:

- Real-time analysis of Intensive Care Unit (ICU) video and vital sign data can provide faster and more accurate detection of anomalies, such as apnea in neonates.
- Real-time pose detection of patients can be used to quickly detect seizures.
- Speech to text transcription enables features such as real-time (translated) subtitles, automatic transcription of a doctors consult, and real-time sentiment analysis.

While the combination of AI and streaming has significant potential, the harmonized platform services at Philips do not yet provide out-of-the-box streaming functionality. Aiming to fill this technological gap, we developed the Philips Remote AI Streaming (PRAIS) platform, which enables *remote AI streaming*: the remote (in the cloud or on premise) execution of AI algorithms that take an audio/video/data stream as input and/or output. As an AI co-creation platform, PRAIS benefits both Philips and its open innovation partners. In particular:

- PRAIS allows AI algorithms to run in the cloud or on premise, providing easy access to AI.
- By using PRAIS, care providers do not need complex technical knowledge and expensive infrastructure to use AI.
- PRAIS enables many valuable use cases that involve AI streaming algorithms.
- As an AI co-creation platform, PRAIS allows AI developers to easily expose their algorithms.

We designed and implemented PRAIS based on four envisioned future use cases and have been able to validate twice during two collaborations. Firstly, a group of ten bachelor computer science students used PRAIS to develop demonstrators. By abstracting away the complexities of real-time streaming, PRAIS enabled the students to build complex streaming applications in just six weeks. A usability study with the students shows that PRAIS is considered easy to use. Secondly, in an open innovation collaboration with Maxima Medisch Centrum we explored how PRAIS can be used to record NICU baby footage. Such recordings are used for AI research purposes.

We recommend to further develop and mature PRAIS such that more use cases can be implemented. In particular, we recommend integrating PRAIS with the Philips Realtime Communications Platform [50], which would make AI even more accessible.

Glossary

AI	**Artificial Intelligence**
PDEng	**Professional Doctorate in Engineering**
ST	**Software Technology**
TU/e	**Eindhoven University of Technology**
PSG	**Project Steering Group**
PMP	**Project Management Plan**
SDK	**Software Development Kit**
API	**Application Programming Interface**
PR	**Philips Research**
S2S	**Screen to Screen**
P2P	**Peer to Peer**
UI	**User Interface**
RTC	**Real Time Communication** is the real-time exchange of information, e.g., video, audio, and/or data streams, from a sender to a receiver over any low-latency telecommunications connection.
WebRTC	**Web Real-Time Communications** is a collection of standards, protocols, and Javascript APIs. It enables P2P audio, video, and data RTC. With webRTC, latency is minimal because it uses real-time protocols and P2P connections.
Peer	An entity that implements WebRTC and is thereby able to join a conference.
Conference	A remote communication session between one or more peers using WebRTC.
Participant	A peer in a conference that is typically controlled/used by a human.
Algorithm	A peer in a conference that is not a participant. It is typically a computer program that performs some calculations or executes other problem-solving operations. It can be an AI, for example.
PRAIS	The **Philips Remote AI Streaming** platform is the system that was designed and developed during this project.
IRM	The **Innovation Rack Manager** is a peer that adds/removes algorithms to/from conferences.

HSRA	The **HealthSuite Reference Architecture** is the Philips architecture that guides and governs the individual solution architectures, platform architectures, and product architectures in all Philips healthcare domains.
HSDP	The **HealthSuite Digital Platform** is the practical manifestation of the HSRA. It is essentially a repository containing all technologies described in the HSRA.
CAO	The **Chief Architect Office** governs the HSRA.
Javascript RTC API	The (mature) API developed by the PR team that got transferred to the CAO.
Prototype C# RTC API	The API developed within the PR team for demo purposes. As a proof of concept, however, it was much less mature than the Javascript RTC API. In particular, it was not designed as a platform and does not contain a vendor abstraction layer.
PRAIS C# API	The API that was designed and developed during this project.
NAT	**Network Address Translation**
STUN	**Session Traversal Utilities for NAT**
TURN	**Traversal Using Relays around NAT**
Research RTC Backend	The set of entities that enable webRTC for the Javascript RTC API, Prototype C# RTC API, and PRAIS C# API. In particular: the STUN/TURN servers, the signaling and messaging servers, and the orchestration service.
ICE	**Interactive Connectivity Establishment** is a technique used in computer networking to find ways for two computers to talk to each other as directly as possible in peer-to-peer networking.
SDP	**Session Description Protocol** is a format for describing streaming media communications parameters.
Secondary Storage	A collection of storage components not consisting of random access memory or the part designated as swapping pool.
SSO	**Single Sign On**
IP	**Internet Protocol**
PaaS	**Platform as a Service**
CI/CD	**Continuous Integration/Continuous Development**
TAM	**Technology Acceptance Model**
mTAM	**modified TAM**
NPS	**Net Promotor Score**
PU	**Perceived Usefulness**
PEU	**Perceived Ease-of-Use**

List of Tables

List of Figures

Contents

1 Introduction

In this chapter, we introduce the context of the problem statement (Section 1.1). Afterwards, in Section 1.2, we define the project scope and goals. Finally, in Section 1.3, we provide an outline of the report.

1.1 Project context

Philips [21], one of the leading health technology companies in the world, strives to make the world healthier and more sustainable. In particular, its goal is to improve the lives of three billion people per year by 2030 [21]. To this end, Philips Research (PR) [19], which is part of Philips, aims to provide meaningful innovations that improve people's lives. Also, being at the forefront of innovation puts Philips in a stronger and more beneficial business position. Therefore, PR actively stimulates open innovation, in which PR leverages its knowledge, intellectual property, and technologies to collaborate and innovate together with selected organizations. In practice, open innovation often manifests in a situation in which Philips provides the technical infrastructure that enables others to innovate faster. The leading architecture of such infrastructure is defined by the Philips HealthSuite Reference Architecture (HSRA) [20]: a consistent, unified, and company-wide approach to architecture and platform development. In particular, the HSRA provides a framework of shared rules, guidelines, APIs, data models, and technology choices that centralize customer experience and stimulate innovation. The practical manifestation of the HSRA is Philips' HealthSuite Digital Platform (HSDP) [49]. Since these two refer to the same concept, we only refer to the HSRA in the remainder of this work. While the Philips Chief Architect Office (CAO) is responsible for managing and updating the HSRA, one of the main goals of PR is to mature and validate new technologies such that they can be adopted in the HSRA. Overall, the HSRA is used within Philips itself and by open innovation partners.

An extremely relevant topic for Philips (and healthcare in general), is Artificial Intelligence (AI), which has the potential to improve many aspects of people's lives. Philips is already actively using AI in the medical data interpretation domain, in which AI is used to detect/interpret medical data. Because Philips sees significant potential in this domain, many new research projects are being started. For example, to improve aspects such as availability and deployability of AI, PR is maturing technologies to bring AI to the cloud. In addition, relatively new data sources in the medical data interpretation domain are audio/video/data streams. Such streams deliver data in real time and enable many new AI use cases. For example, an AI algorithm that monitors the vitals and a video of a prematurely born baby can be used to detect apnea faster and more reliably [Mon20, SBM+15].

The combination of AI in the cloud and streaming is what the PR Scalable Service Delivery team (hereafter referred to as PR team) is investigating. This project took place in the PR team, which is maturing the technology and developing the infrastructure required for *remote AI streaming*: the

remote (in the cloud or on premise) execution of AI algorithms that take an audio/video/data stream as input and/or output. Reasons for developing such technology include:

- The technology enables remote AI streaming use cases (see Section 2.4 and Appendix C).
- Hospitals or other care providers do not always have the required hardware to run certain AI algorithms. Such hardware is often expensive and requires maintenance.
- Hospitals or other care providers rarely have the required knowledge and infrastructure to develop and maintain streaming technology.
- Algorithms are not always developed using the same technologies. Consequently, in a typical scenario where algorithms are directly integrated with the systems that use them, integration can be non-trivial.
- In a setting where an algorithm has been integrated into existing systems, whenever the algorithm is modified/improved, then all systems using that algorithm need to be updated.
- In addition to PR, there are other parties such as academic hospitals that also develop AI. For them, there are several technical and non-technical reasons (see Section 2.4.3) to not share their developed algorithms.

The above listed issues indicate a need for technical infrastructure that allows the easy setup of audio/video/data streams between remote peers. Such technology would drive open innovation and would be a valuable addition to the HSRA. In this work, we present the Philips Remote AI Streaming (PRAIS) platform, which aims to fill this newly identified technological gap.

1.1.1 Philips Remote AI Streaming platform

During this project, aiming to tackle the above-mentioned issues, we developed the Philips Remote AI Streaming (PRAIS) platform (see Figure 1.1). At the foundation of PRAIS, lies Web Real-Time Communications (webRTC) [6], which is the streaming technology that enables real-time communication between peers. Built on top of that, is a set of easy-to-use Application Programming Interfaces (APIs) that enable developers to easily set up streams between AI algorithms and other peers. Combined with the provided infrastructure, PRAIS is a platform with which we aim to tackle the above-mentioned issues and turn these into innovation opportunities:

- **Streaming use cases:** PRAIS enables many valuable use cases that involve streaming AI algorithms (see Section 2.4 and Appendix C).
- **Open innovation:** As a platform, PRAIS can be used by other organizations, allowing them to more quickly develop and use streaming AI algorithms. They do not need the technical knowledge/infrastructure anymore to do this.
- **Technical obstacles:** With PRAIS, we tackle several technical obstacles:
  - **Accessibility:** By running algorithms remotely (on Philips hardware), the algorithm functionality becomes much more accessible. In theory it could be used from any device, e.g., smartphone, tablet. Furthermore, hospitals or other healthcare entities can save money because they do not have to buy/maintain expensive hardware.
  - **Deployability and Replaceability:** Individual algorithms can be deployed/updated/replaced without significantly affecting other algorithms/systems.

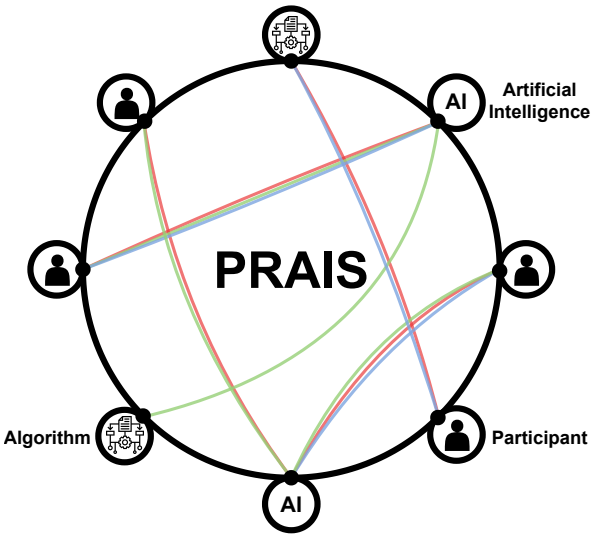


Figure 1.1: A conceptual overview of the Philips Remote AI Streaming (PRAIS) platform. Via audio (blue), video (red), and data (green) streams, users (called participants) can easily connect to AIs or any other algorithm.

- **Composability:** Algorithm functionality can be composed for different purposes.
- **Scalability:** New algorithm instances can be spawned depending on the system load.
- **Technology Heterogeneity:** PRAIS standardizes the communication between algorithms, meaning that the algorithms themselves can be implemented using different technologies.

At the start of this project, the PR team already had a system in place, which formed the starting point of PRAIS. In Section 1.1.2 we describe this system.

1.1.2 The origin of PRAIS

As described before, PR matures and validates technologies to be incorporated in the HSRA. To this end, PR employs three research phases:

1. Exploration phase: Explore whether a concept be interesting. Read, talk, sketch, and validate.
2. Proof of concept: Build demos and/or do in-house pilots in a safe environment.
3. Advanced development: Develop a pilot at an actual customer/partner. Further advertise the technology within PR using demonstrators/pilots/proofs of concept.

These phases are all about maturing technology to a level where it can be adopted into the HSRA. This means that others (the CAO, for example) need to be convinced that the technology is mature enough. In practice, this entails the building of demonstrators/pilots/proofs of concept. Furthermore, an important driver behind such convincing is whether the technology serves a valuable business case.

At the start of the project, the PR team already had a system in place (see Figure 1.2). This system formed the basis of PRAIS and it had already gone partially through the PR research phases. More specifically, webRTC [6] is a technology that the PR team matured through the advanced development

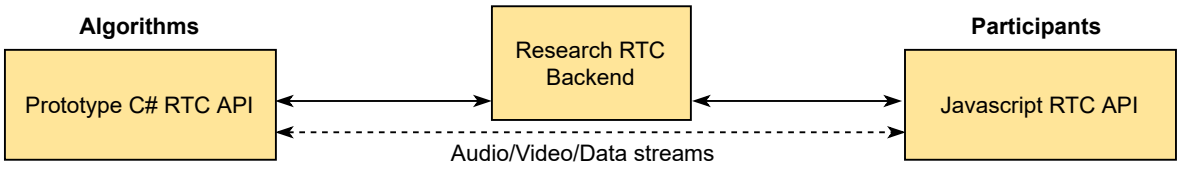


Figure 1.2: An overview of the existing system at the start of the project. The Prototype C# RTC API was only a prototype, meaning that it served as a proof of concept. In contrast, the more mature Javascript RTC API had already been transferred to the CAO to be incorporated into the HSRA. The Research RTC back-end mostly consists of infrastructure required to set up audio/video/data streams.

phase, to be incorporated into the HSRA. WebRTC functionality is exposed via the Javascript RTC API (see Figure 1.2) to participants, who get access to RTC functionality by using web applications. In particular, the API contains a vendor abstraction layer that makes sure the same functionality can be provided using different implementations. This is an important requirement from the HSRA, which aims to prevent vendor lock-in as much as possible.

To enhance the Javascript RTC API and to prove the concept of remote AI streaming, the PR team developed the Prototype C# RTC API. This API brings webRTC functionality to C# algorithms (this was a given for this project), enabling audio/video/data streaming between algorithms and participants. With the Prototype C# RTC API, the PR team proved the potential of remote AI streaming and filed a patent on it. As a proof of concept, however, the Prototype C# RTC API was much less mature than the Javascript RTC API. It was not designed as a platform and does not contain a vendor abstraction layer. All in all, the Prototype C# RTC API had only reached the end of the proof of concept phase and during this project we aim to mature it through the advanced development phase.

1.2 Scope and Goal

Given the project context (Section 1.1) and the existing system (Section 1.1.2), the scope, main goal, and the sub-goals of this project are:

G1 Mature remote AI streaming such that it reaches a maturity level that is suitable for the advanced development phase. This means that:
   (a) PRAIS is ready to be used by open innovation partners.
   (b) Demonstrators that show the potential of PRAIS have been implemented.

In the remainder of this work, we refer to these (sub)goals as: (G*X*), where *X* indicates the goal number. In Section 3.1, we describe a set of technical goals that follow from these (sub)goals.

1.3 Outline

In the remainder of this document, we first provide a more in-depth problem analysis in Chapter 2. After that, in Chapter 3, we describe the requirements gathering process and provide an overview of the requirements. Then, in Chapter 4, we describe the architecture and design of PRAIS. Following that, we describe how we verified and validated PRAIS in Chapter 5. Then, in Chapter 6, we provide a conclusion and directions for possible future work. Lastly, in Chapter 7, we describe the project management process and provide a retrospective on this project.

2 Problem analysis

To better understand the problem at hand, we investigated both technical and non-technical aspects, which we describe in this chapter. In Section 2.1, we describe and explain webRTC, which is the streaming technology that forms the foundation of PRAIS. After that, in Section 2.2, we describe and compare different webRTC providers that were used during the project. Then, in Section 2.3, we describe an analysis of the existing system to understand its functionality and limitations. After that, we describe a use case analysis conducted to find suitable real-time AI streaming use cases (Section 2.4). Then, in Section 2.5, we describe a technical analysis that we did to better understand the typical inputs and outputs of algorithms/AIs. Finally, to better understand the stakeholders and their concerns, we conducted a stakeholder analysis (Section 2.6).

2.1 WebRTC

Web Real-Time Communications (webRTC) [6] is a collection of standards, protocols, and Javascript APIs. It enables Peer-to-Peer (P2P) Real-Time Communication (RTC) capabilities in a browser and was introduced by Google. Nowadays, it is actively supported by many companies such as Apple, Microsoft, and Mozilla. With webRTC, it is possible to add real-time audio/video/data streaming capabilities to an application via JavaScript APIs that are present in all major browsers.

While webRTC provides the streaming functionality itself, it does not provide a default implementation for signaling, which is the discovery and negotiation process prior to setting up the actual connection. More specifically, two peers that wish to connect over a webRTC connection typically reside in different networks, which means that they first have to locate one another. After that, the two peers negotiate the media format that they shall use in their communication.

Setting up a webRTC connection to a peer is not straightforward for several reasons: firewalls may need to be bypassed, Network Address Translation (NAT) typically hides devices behind routers, and if the router does not allow direct connections, then a relay may be required. To tackle these issues, the Interactive Connectivity Establishment (ICE) framework was introduced. ICE uses Session Traversal Utilities for NAT (STUN) servers to discover the public Internet Protocol (IP) address of a peer and to determine whether the peer's router would prevent a direct connection. If this is the case, then ICE uses Traversal Using Relays around NAT (TURN) servers to bypass the router's restrictions by opening a connection with a TURN server that relays all information to the other peer (see Figure 2.1).

Given the issues described above, webRTC requires some extra infrastructure to make it work properly. Because of this, there exist a number of companies that provide this infrastructure and (typically) their own API layer built on top of webRTC. Furthermore, some of those companies also implement their own native webRTC stack such that native applications can also use webRTC. In Section 2.2, we describe the webRTC providers relevant to this project.

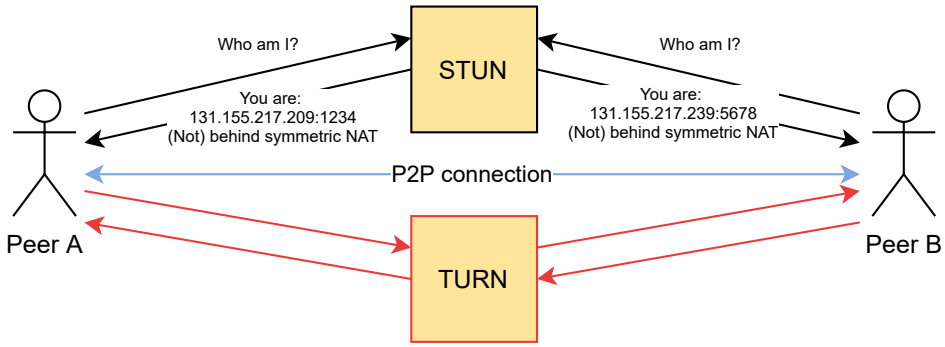


Figure 2.1: Peers use a STUN server to discover their public IP. If a peer is behind a symmetric NAT, then the peer uses a TURN server as a relay (in red). Otherwise, a direct P2P connection is set up (in blue).

2.2 WebRTC providers

In this project, the choice for using webRTC was given, but which webRTC provider to use was still an open question. As mentioned in Section 1.2, vendor (a webRTC provider is a vendor in this case) abstraction is an important requirement for the HSRA, and at the start of this project, PR already had licenses for two webRTC providers: ICELink (Section 2.2.1) and LiveSwitch (Section 2.2.2). The PR team, however, indicated that there may be other (better) webRTC providers out there, which is why we investigated and compared several other webRTC providers using the following criteria:

- C# client library: The algorithm RTC API is written in C# (which was a given for this project).
- Back-end flexibility: The system should be deployable both in the cloud and on premise.
- Licensing: PR already had licenses for ICELink and LiveSwitch, making it possible to use them immediately.

Our investigation showed that ICELink and LiveSwitch were indeed the best choice. An overview of the other investigated webRTC providers can be found in Appendix A.

2.2.1 ICELink

ICELink [2] allows developers to easily add webRTC functionality to their applications by providing a cross-platform API that can be used in web applications, native mobile applications, and native desktop applications. ICELink, like webRTC, is signaling-agnostic, so it requires a separate signaling mechanism. FrozenMountain [4], the developer of ICELink, also develops WebSync [5], which can be used to implement signaling both in the cloud and on premise. Furthermore, at the start of this project, the PR team was already using the JavaScript and C# ICELink SDKs in the JavaScript RTC API and Prototype C# RTC API, respectively.

2.2.2 LiveSwitch

LiveSwitch [1] extends ICELink and is also being developed by FrozenMountain. The additional LiveSwitch server adds features such as out-of-the-box signaling, peer presence management, and

flexibility to combine Peer-to-Peer (P2P), Selective Forwarding Unit (SFU), and Multipoint Control Unit (MCU) connections (see Appendix B for more details). The LiveSwitch server can be deployed both in the cloud and on premise.

2.2.3 Summary

At the start of the project, the PR team was already familiar with ICELink while LiveSwitch was relatively new. There were, however, no customers that used Philips applications built using ICELink, which is why Philips did not renew the ICELink license. On the other hand, LiveSwitch was used more actively, which is why Philips did renew the LiveSwitch license. Consequently, LiveSwitch was chosen initially as the preferred webRTC provider. Later on in the project, when the PR team sold an application built using ICELink, the ICELink license got renewed again. To make PRAIS work with this application, we also added ICELink as a webRTC provider (see Chapter 4).

From a technical point of view, we deem it easier and less work to simply use a single webRTC provider instead of implementing multiple. The story above, however, shows that there are also non-technical forces that influence which technology can be used. This is exactly why vendor abstraction is so important.

The starting point of PRAIS is an existing system, which we have to understand in depth. We describe an analysis of this system in Section 2.3.

2.3 The existing system

We analyzed the system that the PR team already had in place to understand its structure and limitations. A conceptual overview is shown in Figure 2.2 while the deployment diagram for the same system is shown in Figure 2.3. Note that the blue, yellow, and orange colors in both figures identify the same parts of the system. For both algorithms and participants there is a layered structure.

Participants use the TeleHealth web application, which was built using the Javascript RTC API, to join webRTC conferences: remote RTC sessions between one or more peers. As mentioned before, the Javascript RTC API provides a vendor abstraction layer such that different webRTC providers can be used. Initially, there were two webRTC providers: ICELink (see Section 2.2.1) and Vidyo [13] (also see Appendix A). Vidyo, however, only provided its service in the cloud, which is why ICELink was the preferred webRTC provider.

Algorithms join webRTC conferences by using the Prototype C# RTC API. At the start of this project, the API was tightly coupled to ICELink (note the dotted line in Figure 2.2), did not contain a vendor abstraction layer, and was designed as a proof of concept.

Using the Research RTC back-end, participants and algorithms are able to connect to each other over webRTC audio/video/data streams. The required components for setting up such a connection include: the WebSync [5] signaling server, the ICELink STUN/TURN server, and the Amazon Web Services (AWS) [22] authentication lambda function that verifies authentication tokens provided by peers that wish to connect to the STUN/TURN or signaling server. The other two lambda functions and the DynamoDB are used for logging purposes.

An important entity within the existing system is the Innovation Rack Manager (IRM). The IRM is essentially an algorithm that lives in its own webRTC conference and has the responsibility of

adding/removing algorithms to/from other conferences. It is identifiable by name and participants use this name when sending messages over the signaling channels to instruct the IRM. The signaling server also knows the name of the IRM and can therefore forward the message correctly. The IRM would spawn algorithms as separate processes on the same machine and instruct the algorithm to connect to the correct conference.

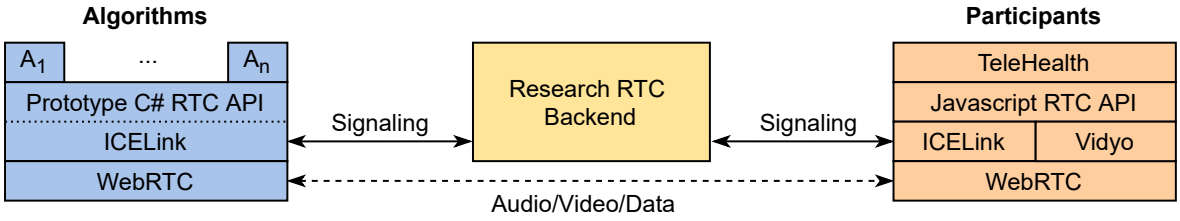


Figure 2.2: A high-level overview of the system the PR team had in place at the start of this project. The blue, yellow, and orange colors map to the similarly colored components in Figure 2.3. Algorithms and participants use the Research RTC back-end to set up webRTC audio/video/data connections among each other.

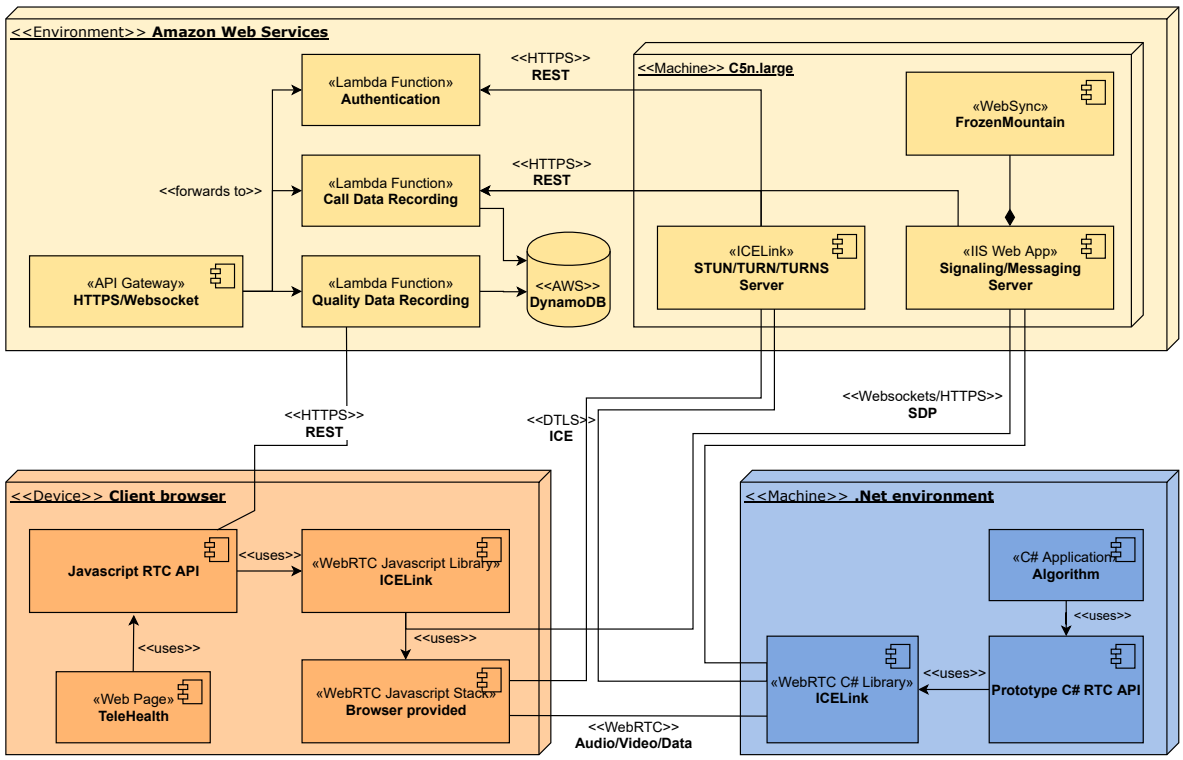


Figure 2.3: An overview of how the system the PR team had in place at the start of this project was deployed. The blue, yellow, and orange colors map to the similarly colored components in Figure 2.2.

Following the analysis of the existing system, we identified the following issues, all of which are motivators for the requirements described in Chapter 3:

- An IRM is identified by name, which is not necessarily unique. Furthermore, participants need to know the IRM names and which algorithms can be spawned by which IRM.

- The default WebSync server was extended to make the sending of messages to IRMs work. This creates a mix of responsibilities where the WebSync server is used for both signaling and messaging to the IRM.
- The system only allowed the creation of a single default webRTC connection between two peers.
- Algorithms could only broadcast their results to the whole conference as there was no mechanism to selectively send it to a single peer.
- Tokens were stored in browser applications/algorithms because they could not be generated dynamically. This is not secure, because anyone who manages to get such a token gets access to the system.
- Peers only have a (not necessarily unique) name to identify themselves.
- The Prototype C# RTC API did not have any documentation and was part of a large Visual Studio [23] solution that also contains other (unnecessary) elements than the API.

2.4 Use Cases

The PRAIS platform should be generic in the sense that it should work for different applications in as many use cases as possible. Therefore, during several brainstorming sessions and discussions with domain experts and PR team members, we explored many different use cases. A select few were eventually chosen to be implemented (see Sections 2.4.1-2.4.4) while the others are described in Appendix C. By implementing applications for the use cases (see Sections 4.2 and 4.3.1), we tested PRAIS and obtained valuable demonstrators (G1b). Most of the use cases involve a Neonatal Intensive Care Unit (NICU).

A NICU is an intensive care unit that is typically used for prematurely born babies. The parents of such babies cannot stay with their baby indefinitely, which makes it emotionally very tough. The NICU Screen-to-Screen (S2S) application, which was developed by the PR team, aims to tackle this problem. By placing a camera inside the NICU, parents can remotely view their baby. In addition to helping with the emotional burden of the parents, it also creates the opportunity to use the baby footage and other sensory data for AI (and other) use cases. Two of such use cases are described in Sections 2.4.1 and 2.4.2.

2.4.1 Neonatal pose detection

Cerebral Palsy (CP) is a group of neurological disorders that permanently affect body movement and muscle coordination. It appears in infancy and is therefore a highly relevant topic in NICU research. Recent work [EBK+19] has shown that the absence of fidgety movements at three to five months of age is a strong indicator for developing CP. Nowadays, a doctor has to look at a baby for approximately ten minutes to assess whether or not fidgety movements are present. Furthermore, such an assessment often occurs remotely or at the parent's home because at three months of age, a baby is typically already home. To shorten this time-intensive task and to make remote assessment easier, the PR team is investigating whether AI can be used to analyze the movements of a baby. As a first step, the team developed a pose detection AI that is able to detect the baby's skeletal structure (see Figure 2.4) and visualize the movement patterns.

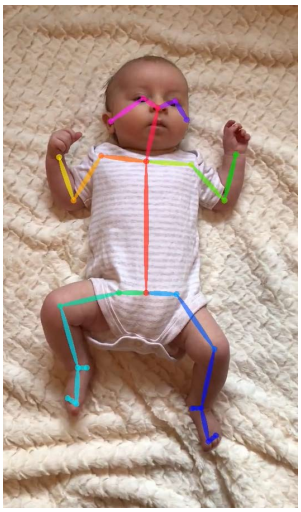


Figure 2.4: By using AI, we can detect the skeletal structure of a baby in an image. Colors are used to more easily distinguish body parts.

2.4.2 Neonatal apnea detection

Apnea is a disorder where breathing temporarily (approximately 20+ seconds) pauses or is very shallow, resulting in a dangerous drop of oxygen saturation. It occurs during sleep and those affected are typically sleepy or feel tired during the day. Apnea can be obstructive, in which breathing is interrupted by a blockage of airflow, it can be central, in which the brain simply stops sending signals to breathe, or it can be a combination of the two.

While apnea typically only occurs during sleep, the brain of a prematurely born baby is not fully developed yet, which can result in apnea occurring while the baby is awake. Nowadays, the sensors in a NICU trigger an alarm whenever apnea is detected (i.e., breathing stops for 20+ seconds and oxygen saturation is too low). Nurses then have to make sure the apnea goes away by stimulating the baby, e.g., touching/shaking the baby, or by administering some caffeine. The moment an alarm goes off, it is basically too late because oxygen saturation is already too low. Therefore, every second counts, which is why the PR team is investigating whether AI can be used to detect apnea faster and more reliably, as well as maybe even predict it [Mon20]. In particular, the team developed an AI that, given video and sensory data, can detect and classify apnea. In addition, the AI estimates the baby's heart rate and breathing rate, which removes the need for a sensor that measures these values. Considering that a neonatal is very small and vulnerable, this is a major improvement.

2.4.3 Algorithm sharing

In an academic hospital, in addition to the standard medical care, the staff is also involved in medical research. In the Netherlands, for example, we have academic hospitals in Amsterdam, Utrecht, Maastricht, and other cities. With the rise of AI, such hospitals are focusing more on developing AI (or any other algorithm) that can be used for medical purposes. While hospitals have the desire to share their algorithms to improve healthcare, there are several reasons why this is troublesome. First, on the academic level, there is the precious balance between sharing for better healthcare versus academic competition (who publishes first and protection of intellectual property). Second, privacy and security

are sensitive topics in the healthcare domain and often introduce constraints. Also, hospitals do not always have the technical knowledge to share algorithms and even if they do, setting up the technological infrastructure and arranging all contracts between the hospitals can take months. Finally, we believe it to be more practical to keep the algorithm with the experts such that they can maintain and develop it further while others use it. Overall, hospitals would greatly benefit from technology that lets others make use of their (AI) algorithms without sharing the actual algorithm itself. PRAIS is a first step towards providing such technological infrastructure that has been verified and is easy to work with. From a security perspective, the usage of P2P streaming technology means that there is no server between the peers and that there is no need to store sensitive data on secondary storage outside the hospital (assuming the algorithm does not store the data it receives).

2.4.4 Audio/video recording

To train an AI model, researchers typically use prerecorded audio/video. Obtaining such data is mostly a labor-intensive task and there are often privacy/security constraints to tackle. In a use case that we encountered in the past, a PhD student spent about five months tackling privacy/security constraints and setting up the technological infrastructure for recording audio/video from a NICU camera, to eventually change directions because it turned out to be too much work.

Maxima Medisch Centrum (MMC) Veldhoven bought the NICU S2S application and hired a PhD student to develop AIs that use the NICU camera footage. Similar to the case described above, in a collaboration with MMC, recording of NICU camera footage is again needed. This time, however, with PRAIS, the technological infrastructure works out-of-the-box, making it possible to record NICU footage.

This use case is a nice example of how PRAIS enables open innovation (G1a), where researchers can focus on the research instead of having to bother with technological infrastructure and privacy/security constraints.

2.4.5 Summary

The use cases described above are the main drivers behind the PRAIS requirements. All use cases require some form of audio, video, and/or data streaming between peers in a fast and secure manner. This means that topics such as peer authentication, peer-to-peer streaming, and media source diversity have high priority. More detailed requirements are defined and discussed in Chapter 3.

During a Software Engineering Project (SEP) with computer science bachelor students (see Section 4.2), we implemented applications that realize the pose (Section 2.4.1), apnea (Section 2.4.2), and algorithm sharing (Section 2.4.3) use cases. In a collaboration with MMC (see Section 4.3), we explored the recording use case (Section 2.4.4) and built the PRAIS Recorder Application (see Section 4.3.1).

In Section 2.5, we describe an analysis that we performed to better understand typical algorithm/AI inputs/outputs.

2.5 Algorithm and AI analysis

On a high level, algorithms and AI have input and output, i.e., they can be seen as a black box. Since PRAIS should be able to stream those inputs and outputs, we conducted a technical analysis in which we explored the most common inputs and outputs. In particular, we did the analysis by doing online research, by leveraging the knowledge within the PR team, by leveraging the algorithm/AI experience of the trainee, and by considering the algorithms required for the use cases described in Section 2.4.

Since algorithms are developed in C#, we first considered the C# built-in types [16]: bool, (s)byte, char, decimal, double, float, (u)int, (u)long, (u)short, object, and string. While this list is quite long, note that all of these types are stored as bytes and that they can be encoded as strings, e.g., Json [18] (which is more human readable). Therefore, PRAIS should be able to stream both bytes and strings. While byte/string streaming is already quite enabling, note that typical algorithms use/provide collections/data structures [17] that consist of bytes and strings. For example, a typical image classification AI takes an image (a multidimensional array of ints) as input and provides an array of probabilities (doubles, for example) as output. In particular, considering the use cases described in Section 2.4, there is also a need to stream audio/video. Video is essentially a sequence of images, which are multidimensional arrays of numbers. Audio is encoded as a sequence of numeric samples taken at a certain frequency. While audio and video could be streamed using bytes/strings, it would require quite some additional logic, which is not user-friendly. Therefore, PRAIS should be able to stream audio and video. These insights were used in the requirements gathering process, which we describe in Chapter 3.

2.6 Stakeholder Analysis

We conducted a stakeholder analysis to identify stakeholders and to understand their concerns. The stakeholders belong to three organizations: Eindhoven University of Technology (TU/e), Philips, and Maxima Medisch Centrum (MMC). An overview of the stakeholders is provided in Table 2.1.

Two important stakeholders actually used PRAIS. The first was a group of ten TU/e bachelor students who worked together with us during their final Software Engineering Project (SEP). More details on this collaboration are described in Section 4.2. The second PRAIS user was a PhD student at Maxima Medisch Centrum, who worked on the recording use case (Section 2.4.4). More information on this collaboration is described in Section 4.3. In Section 7.1 we describe how we communicated with each of the stakeholders.

Lastly, observe that the scope of this project does not include the actual transfer of PRAIS to the HSRA, i.e., the CAO. This is a process that typically takes months and is therefore envisioned to be done after this project is completed. Consequently, the CAO is not a stakeholder for this project.

In the next chapter, we describe the requirement gathering process and provide an overview of the requirements.

Eindhoven University of technology		
Name	**Role**	**Interest in**
Robin Mennens	PDEng trainee	Gaining architectural/design/development knowledge and experience. Successfully graduating.
Alexander Serebrenik	TU/e supervisor	Successful project delivery, report quality, and PRAIS verification and validation.
Yanja Dajsuren	PDEng ST manager	Quality of project results, relationship with Philips, successful graduation of trainee.
SEP students	PRAIS user	Usability and functionality of PRAIS (see Section 4.2).
Philips		
Name	**Role**	**Interest in**
Marcel Quist	PR team lead and Project owner	Successful demos using PRAIS, increasing awareness around PRAIS, and further maturing PRAIS concepts.
Zoran Stankovic	PR team software architect and Project mentor	Successful demos using PRAIS, increasing awareness around PRAIS, and further maturing PRAIS concepts. Usability/extensibility/maintainability of PRAIS.
Arjan Draisma	PR team research engineer	Usability/extensibility/maintainability of PRAIS.
Ralitsa Kehayova	PR team intern: algorithm developer	Successful integration of the pose detection algorithm with PRAIS.
Roel Montree	PR team intern: algorithm developer	Successful integration of the apnea detection algorithm with PRAIS.
Maxima Medisch Centrum		
Name	**Role**	**Interest in**
Ilde Lorato	PhD Researcher and PRAIS user	Functionality of PRAIS Recorder Application (see Section 4.3.1).
Carola van Pul	Supervisor PhD Researcher	Functionality of PRAIS Recorder Application (see Section 4.3.1).

Table 2.1: The identified stakeholders for this project.

3 System Requirements

In this chapter, we describe a set of technical goals (Section 3.1) that more precisely describe what we aimed for during this project on a technical level. After that, in Section 3.2, we describe the requirement gathering process. In Section 3.3, we give an overview of the PRAIS use cases. Finally, in Section 3.4, we provide an overview of the requirements using a requirements model.

3.1 Technical goals

The main goals of this project (see Section 1.2) mostly focus on business aspects. To better understand what was technically required to achieve these goals, we provide a set of technical project goals:

TG1 Design and implement the PRAIS C# API. In particular, a vendor abstraction layer should be added.
TG2 Integrate the PRAIS C# API with the existing Javascript RTC API.
TG3 Address the technical limitations of the existing system (see Section 2.3).
TG4 Investigate and prototype visualization streaming (see Section 3.1.1).

In the remainder of this work, we refer to these goals as: (TG*X*), where *X* indicates the goal number. The first technical goal (TG1) follows from the fact that the Prototype C# RTC API was designed as a proof of concept. In particular, it does not contain a vendor abstraction layer. With TG2, we aimed to combine the strengths of the Javascript RTC API and the PRAIS C# API. By making sure they integrate, developers can easily connect participants in the browser to algorithms. TG3 directly follows from the limitations identified in the existing system (see Section 2.3). We aimed to address most of these limitations during this project. Lastly, TG4 is a technical exploration that is of interest to the PR team (see Section 3.1.1).

3.1.1 Visualization Streaming

In a typical scenario, as shown in the top part of Figure 3.1, an algorithm streams its output over a data channel to a participant (who uses a web application). The web application then visualizes the algorithm results. In this setup, there is a dependency because the web application needs to know the algorithm output format and how to visualize it. With visualization streaming, we aim to remove this dependency by moving the responsibility of result visualization to the algorithm itself (see the bottom part of Figure 3.1). The algorithm sends the code required for visualizing the output to the web application. By running this code in a placeholder, the web application is able to visualize the algorithm output. Overall, with visualization streaming, we aim to have algorithms that are compatible with any web application that supports visualization streaming and vice versa. During this project, we aimed to explore the technical feasibility and to implement a prototype of visualization streaming.

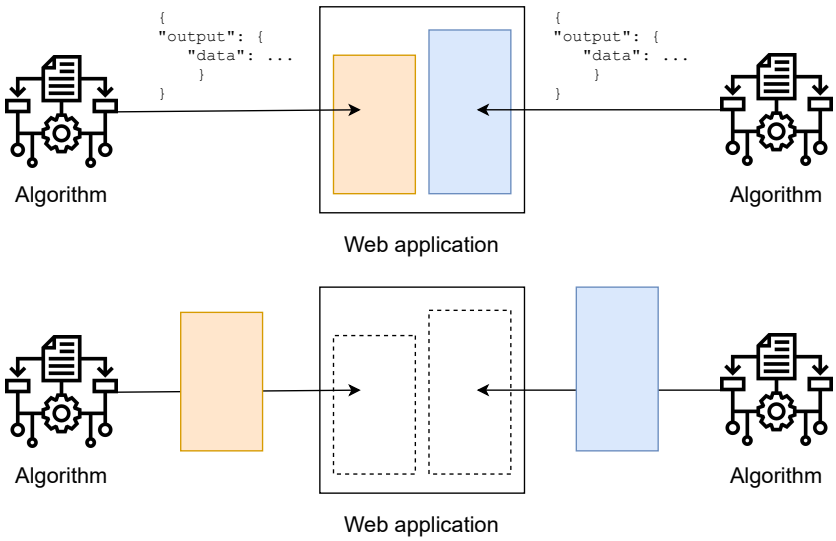


Figure 3.1: A conceptual view of visualization streaming. In a typical setup (top), algorithms stream their output over a data channel to a web application (which is used by a participant) that then visualizes the output. In contrast, when using visualization streaming (bottom), the algorithm first streams the visualization code to the web application, which has placeholders for such visualizations. After that, the algorithm streams its output to the web application, which can then visualize the output.

3.2 Requirement gathering

Throughout the project, there were three requirement gathering iterations, resulting in three project phases (also see Section 7.2). The first two phases mainly relate to the fact that at the start of the project, there were no customers who used Philips applications built using ICELink, which is why LiveSwitch was chosen initially as the preferred webRTC provider (see Section 2.2.3). Later on in the project, when the PR team sold the ICELink-powered NICU S2S application, the ICELink license got renewed again, resulting in the second phase. Lastly, we iterated through the requirements with the specific recording use case (see Section 2.4.4) in mind. The three phases are described in detail below.

In the first phase, we started gathering those requirements that would comprise the core of PRAIS, i.e., the functionality that would definitely be required. We did this during several activities:

- As described in Section 2.3, we analyzed the existing system to identify limitations. We defined requirements to address these limitations.
- Since PRAIS should support several use cases, we analyzed several of them (see Section 3.3) to better understand what functionality should be provided by the PRAIS C# API.
- Within the PR team, we held several brainstorming sessions around the whiteboard. In particular, the experience of the PR team with the Prototype PRAIS C# API offered valuable insights/knowledge.
- We consulted several literature sources:
  - Remote AI results in a setting that is very similar to a microservices architecture [NN15], in which an application is structured as a collection of independently deployable services,

i.e., algorithms. Consequently, literature [NN15] on best practices regarding microservices architectures provided inspiration for important aspects to include in the requirements.

- HSRA guidelines are provided by the CAO on Platform as a Service (PaaS) [vDWvZ18] development. Since PRAIS is essentially a PaaS, we consulted this document on topics such as logging and security. In particular, the document prescribes the usage of OAuth2.0 [42] with (optionally) OpenID Connect [38] for access management.
- We consulted Philips privacy and security officers to discuss related aspects around PRAIS. We quickly found out that privacy concerns the processing of data. PRAIS itself, as a platform, only transfers data and does not really process it. The processing is done by the algorithms developed by PRAIS users. Consequently, privacy concerns did not play a major role in this project. Security, on the other hand, did play a major role. In fact, one of the Philips security officers provided us with a document [Prob] that lists about 400 security and privacy requirements that should be fulfilled by Philips services. For this project, this list is too long to fully consider. Therefore, we used the document as a source of inspiration for important security requirements that we could take into account.

By combining all the knowledge/insights from the above-mentioned activities, we collected an initial set of requirements that resulted in the PRAIS C# API. Important criteria in prioritizing these requirements were as follows:

- Whether the functionality is essential to get things working, i.e., minimum viable product.
- Whether the functionality is required in one of the use cases described in Section 2.4.
- Lastly, we first wanted to make sure that the PRAIS C# API is usable in a simple but common setting, which we considered to be the scenario where a developer uses the API on his/her computer and from there also runs the algorithms he/she is developing. This means that topics such as advanced security, centralized logging, and containerization have lower priority.

Given the initial set of requirements, the PRAIS C# API was designed and developed using LiveSwitch as a webRTC provider (see the darker green boxes in the LiveSwitch rectangle in Figure 3.4). To connect algorithms (built using the PRAIS C# API) to participants, we developed a simple LiveSwitch-powered web app [33]. The outcome of this phase was used by the SEP students (see Section 4.2), providing us with a first iteration of user experience (see Section 5.2.1).

For the second phase, the plan was initially to implement the Javascript RTC API using LiveSwitch. This would make the PRAIS C# API and Javascript RTC API compatible. As described above, however, the ICELink-powered NICU S2S application required algorithm functionality, which is why ICELink was added as a webRTC provider to the PRAIS C# API (see the darker green boxes in the ICELink rectangle in Figure 3.4). This allowed us to verify the vendor abstraction layer built into the PRAIS C# API. Furthermore, after modifying the existing JavaScript RTC API, both APIs became compatible.

In the last phase, now that both APIs were compatible, we defined and prioritized the requirements for the recording use case, which was to be implemented at Maxima Medisch Centrum (MMC). We did this by having meetings with the PhD student who is doing a NICU-based research at MMC. Based on her input, we would further define the requirements during brainstorming sessions within the PR team.

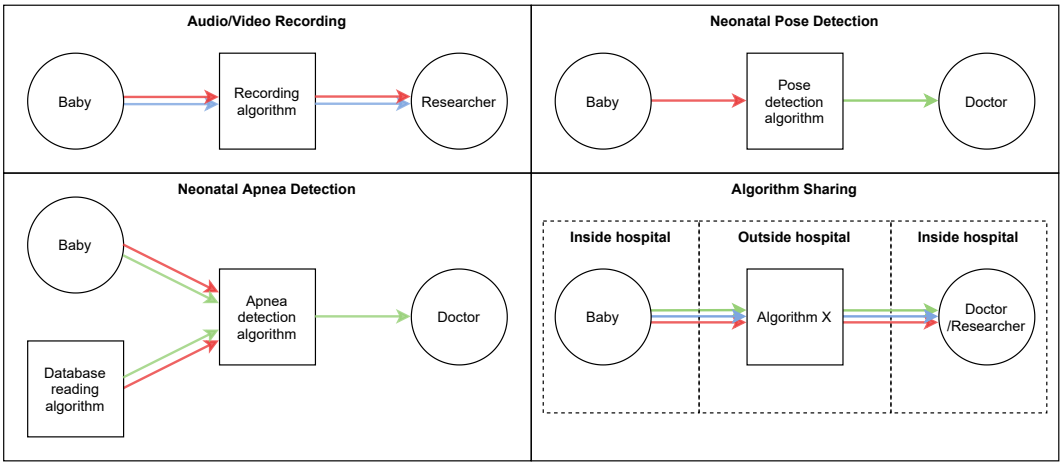


Figure 3.2: Illustrated are the streaming infrastructures of the four use cases described in Section 2.4 (top left: Section 2.4.4, top right: Section 2.4.1, bottom left: Section 2.4.2, and bottom right: Section 2.4.3). Circles represent participants while squares represent algorithms. Audio, video, and data streams are represented by blue, red, and green arrows respectively.

As described above, an important factor in prioritizing the requirements was whether or not a piece of functionality was required in one of the use cases. To this end, we analyzed the use cases in depth, which we describe in Section 3.3.

3.3 PRAIS use cases

At the start of phase one, in order to determine the set of requirements that comprise the core functionality of PRAIS, we conducted a use case analysis. We analyzed the use cases described in Section 2.4 to determine what functionality should be offered by the PRAIS C# API. Note the distinction between use cases and what we call PRAIS use cases. A use case essentially describes an application that a developer would build using PRAIS. A PRAIS use case describes a piece of PRAIS functionality that is used by such a developer. So, by understanding use cases we aim to identify and understand the required PRAIS use cases.

For each use case, we determined the required streaming infrastructure, as shown in Figure 3.2. Note that each of these use cases can be implemented in different ways. The ones shown in Figure 3.2 are just examples that helped us understand the PRAIS use cases. In the Audio/Video Recording use case (top left and see Section 2.4.4), the audio and video footage of a baby is streamed to an algorithm that records all incoming media. Afterwards, a researcher can watch/use the footage by replaying the recording. The Neonatal Pose Detection use case (top right and see Section 2.4.1) covers the situation in which video footage of a baby is streamed in real time to a pose detection algorithm that sends the output over a data channel to a doctor, who can then view the results. The Algorithm Sharing use case (bottom right and see Section 2.4.3), represents a more generic scenario. Baby media/data is streamed from inside a hospital to an external algorithm X that then streams the output media/data back to a doctor/researcher in the hospital again. Finally, for the Neonatal Apnea Detection use case (bottom left and see Section 2.4.2), we considered two scenarios. The first is a real-time scenario where baby video and sensor data is directly streamed to the apnea detection algorithm. The second is a scenario

where recorded baby video (that was, for example, recorded using a recording algorithm) and sensor data is streamed from a database to the apnea detection algorithm. In both scenarios, the algorithm streams the output to a doctor who analyzes the results.

An overview of other use cases that we considered in our analysis, but did not dive into deeply, is provided in Appendix C. The output of the use case analysis is a set of PRAIS use cases, which are shown in Figure 3.3. These PRAIS use cases form the basis of the requirements, which are described in Section 3.4.

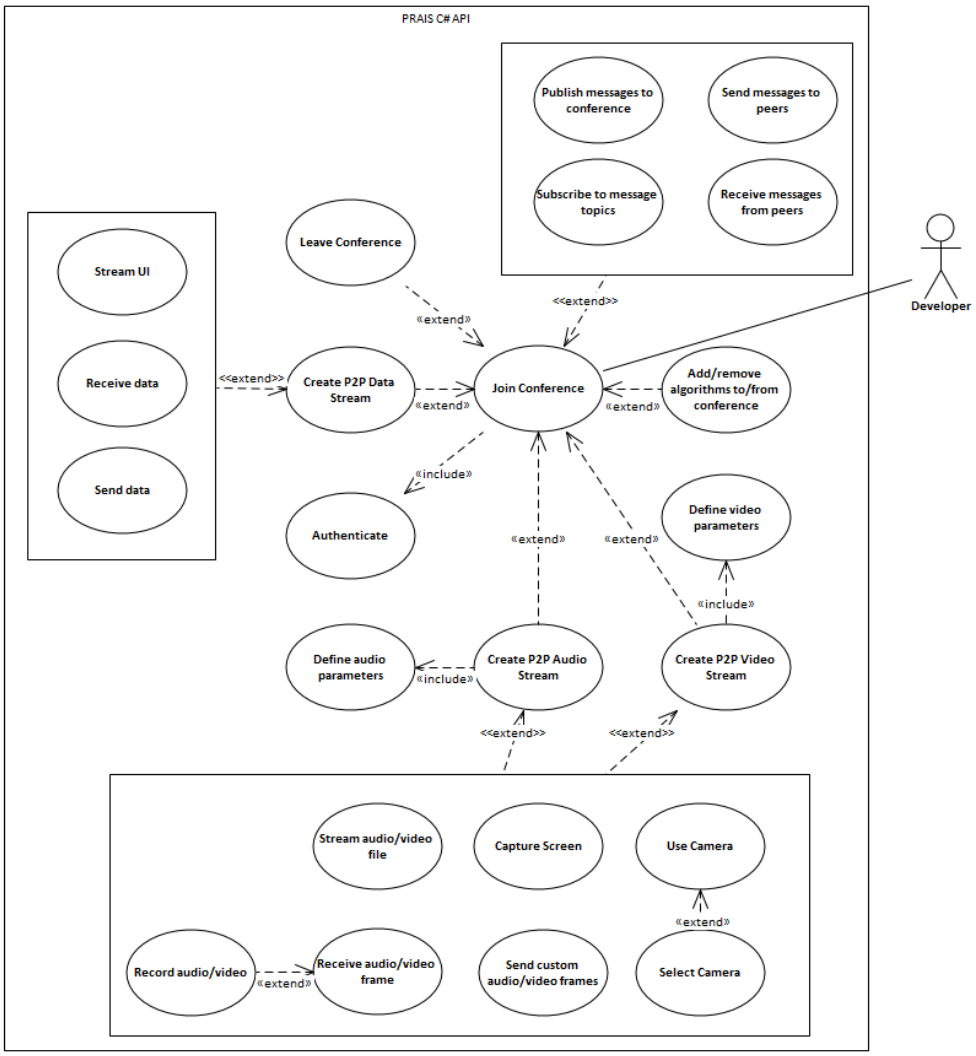


Figure 3.3: A visual overview of the PRAIS use cases. The Developer actor is shown to be only associated with the Join Conference PRAIS use case, which is done to make the diagram readable. In fact, a Developer can be involved in any of the PRAIS use cases.

3.4 Requirement overview

The outcome of the requirement gathering process includes two sets of MoSCoW [25] (*Must, Should, Could, Won't*) prioritized functional and non-functional requirements. The functional requirements are described in Section 3.4.1 while the non-functional requirements are described in Section 3.4.2.

3.4.1 Functional requirements

A complete list of all 150 functional requirements is provided in Appendix D. Since we deem this list too long to discuss in the main text, we group the requirements into categories, of which a visual overview is shown in Figure 3.4 (except for the *won't* requirements). Each category represents a set of requirements that form a piece of functionality and colors indicate priority (*must, should*, and *could*, are dark green, green, and yellow respectively). Observe that the LiveSwitch and ICELink requirement categories have quite some overlap because they mostly cover the same functionality that is implemented using different webRTC providers. The requirement categories shown under 'Common' are those requirements that describe functionality that is generic to the system and is thus webRTC provider agnostic. Each requirement category is described below (where overlapping LiveSwitch and ICELink requirements categories are discussed together).

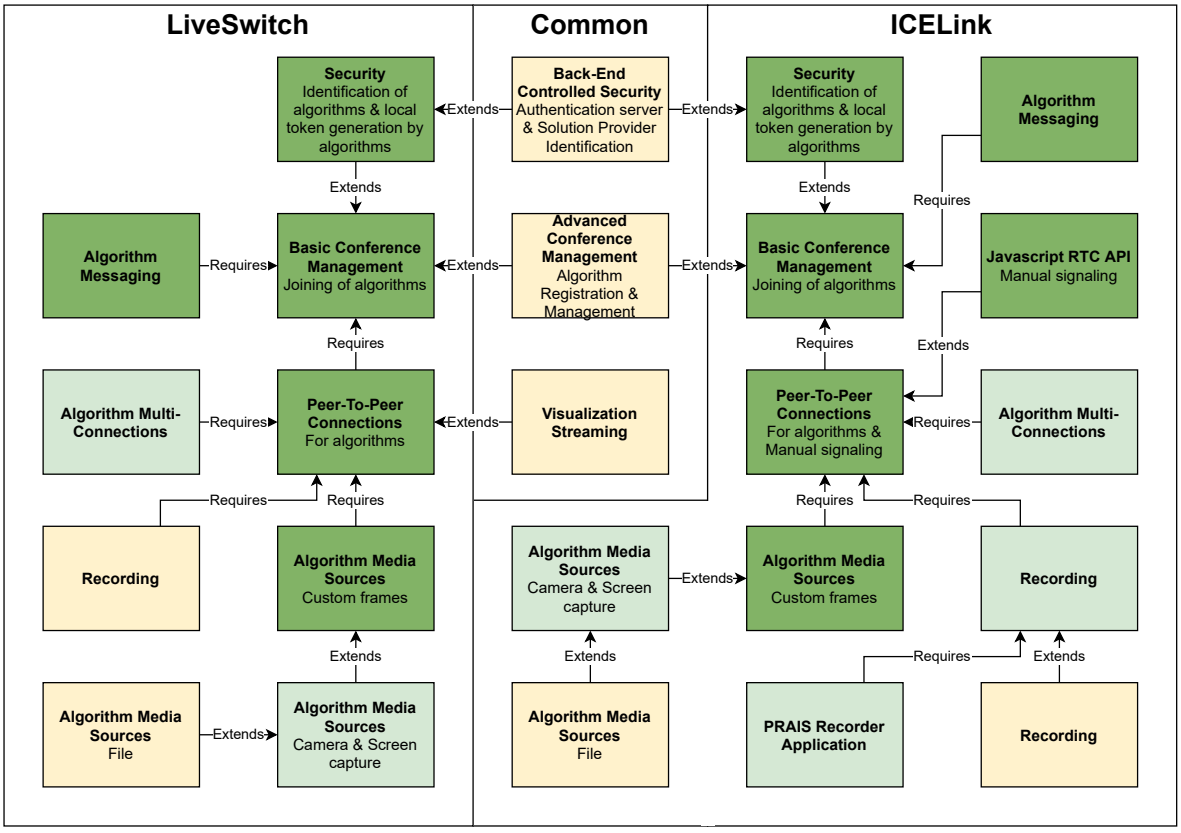


Figure 3.4: A visual overview of the requirement categories (except for requirements with *won't* priority) and their relations. Colors indicate MoSCoW priorities where *must*, *should*, and *could*, are dark green, green, and yellow respectively.

3.4.1.1 Basic Conference Management

Must: Before peers can connect to each other, they need to join a conference. Once joined, algorithms should detect the joining/leaving of other peers such that connections can be set up.

3.4.1.2 Security

Must: We want to control who can join a certain conference. So, participants and algorithms shall authenticate themselves before joining a conference. Authentication shall be done using tokens that have an expiration date.

3.4.1.3 Peer-To-Peer Connections

Must: In order to connect peers, algorithms shall set up P2P connections to other peers. This includes both audio/video and data connections.

3.4.1.4 Algorithm Media Sources

Must: When an algorithm sets up an audio/video P2P connection, then the algorithm shall send custom audio/video frames to the other peer. With custom audio/video frames, basically any frame can be sent. This offers a lot of flexibility and therefore has *must* priority.

Should: When an algorithm sets up an audio/video P2P connection, then the algorithm shall use screen capture or a webcam as a media source.

Could: When an algorithm sets up an audio/video P2P connection, then the algorithm shall use a local file as a media source.

3.4.1.5 Algorithm Messaging

Must: Audio/video/data connections can already be used to stream media/data. Streaming connections, however, are quite resource intensive and are therefore excessive when, for example, simple event triggers need to be sent. Therefore, algorithms shall have a lightweight method to send messages to other peers.

3.4.1.6 Javascript RTC API

Must: The signaling implementation of the Javascript RTC API had to be changed to manual signaling (see Section 4.5.2) to make the API integratable with the PRAIS C# API.

3.4.1.7 Algorithm Multi-Connections

Should: Algorithms shall set up multiple audio/video/data connections between the same peer. This is especially useful for data channels, where different channels can be used for different purposes. Many use cases can already be implemented with only a single connection, however, which is why this has *should* priority.

3.4.1.8 Recording

Should: Algorithms shall record incoming audio/video to a local file. This has *should* priority for ICELink because it was actually used at MMC.

Could: For LiveSwitch, we have the same requirements as above, but there was no direct need to have this functionality with LiveSwitch, which is why this has *could* priority. Additional *could* requirements include the recording of additional metadata and making sure that recorded video frames have the same resolution (which was desirable but not required by MMC).

3.4.1.9 PRAIS Recorder Application

Should: The PRAIS Recorder Application was used at MMC and is essentially a user interface on top of the PRAIS recording functionality. In Section 4.3.1, we describe its functionality and design.

3.4.1.10 Back-end Controlled Security

Could: In the basic security setup (see Section 3.4.1.2), peers authenticate themselves with the back-end using locally stored or generated tokens (see Section 4.5.1.1). With this extended security, algorithms have to first authenticate themselves with an authentication server to obtain such a token.

3.4.1.11 Advanced Conference Management

Could: With basic conference management (see Section 3.4.1.1), algorithms have control over when they join a conference and which conference that is. This is in contrast to the original IRM functionality (see Section 2.3), where other peers can decide when algorithms join/leave a certain conference. With advanced conference management, we aim to again provide such functionality. In particular, algorithms shall register themselves with a server upon startup and other peers shall then instruct algorithms (via the server) to join/leave certain conferences.

3.4.1.12 Visualization Streaming

Could: These requirements cover the visualization streaming exploration/prototype (TG4).

3.4.2 Non-Functional Requirements

The list of non-functional requirements is shown in Table 3.1. Each non-functional requirement is placed into a context, i.e., an *ility*. Note that a lot of security requirements are already included in the functional requirements. NF-9, in practice, means that PRAIS should support OAuth2.0 [42] and OpenID Connect [38]. NF-10, in practice, means that the PRAIS C# API should be implemented as a .NET Standard library [26].

In the next chapter, we describe the architecture and design of PRAIS, which followed from the requirements described in this chapter.

Table 3.1: An overview of the non-functional requirements. Each non-functional requirement is placed into a context, i.e., an *ility*.

ID	Priority	Context	Description
NF-1	Must	Usability	The PRAIS C# API shall be documented for developers.
NF-2	Must	Usability	The PRAIS C# API shall be easy to use.
NF-3	Must	Vendor abstraction	The system shall abstract away the webRTC provider.
NF-4	Must	Installability	The PRAIS C# API shall be easy to install by users.
NF-5	Must	Installability	Philips shall control who gets access to the PRAIS C# API.
NF-6	Must	Deployability	The system shall run in the cloud.
NF-7	Must	Deployability	The system shall run on premise.
NF-8	Must	Integratability	The PRAIS C# API shall integrate with the Javascript RTC API.
NF-9	Could	Security	The system shall use the access control technology prescribed by the Philips PaaS document [vDWvZ18].
NF-10	Could	Compatibility	The PRAIS C# API shall be usable from any .NET implementation.

4 PRAIS Architecture & Design

In this chapter, we describe the design and architecture of PRAIS by using the 4+1 view model of architecture [Kru95] (see Section 4.1). First, we describe how we cover the use cases (i.e., the *+1* view) described in Section 2.4 during the two collaborations that took place during this project. In particular, in Section 4.2, we describe a project that we did together with a group of computer science students. After that, in Section 4.3, we describe the collaboration with a hospital that does NICU research. Then, the logical, process, physical, and development views are described in Sections 4.4, 4.5, 4.6, and 4.7. Since the architecture and design follow from the requirements, we use (FR-*X*) and (NF-*Y*) notation to refer to functional requirement category *X* and non-functional requirement *Y*.

4.1 The 4+1 View model of architecture

The 4+1 view model of architecture, originally introduced by Kruchten [Kru95], is a model for describing the architecture of software-intensive systems, based on multiple views[Kru95]:

- *The **logical view**, which is the object model of the design (when an object-oriented design method is used).*
- *The **process view**, which captures the concurrency and synchronization aspects of the design.*
- *The **physical view**, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect.*
- *The **development view**, which describes the static organization of the software in its development environment.*

The description of an architecture – the decisions made – can be organized around these four views and then illustrated by a few selected use cases or scenarios which become a fifth view.

Before diving into the design of PRAIS, it is important to understand the contexts of the projects in which we brought the use cases/scenarios described in Section 2.4 into practice (i.e., the *+1* view). In Section 4.2, we describe a Software Engineering Project (SEP) that we did together with a group of computer science bachelor students. After that, in Section 4.3, we describe how we collaborated in an open innovation project with Maxima Medisch Centrum (MMC).

4.2 Software Engineering Project (SEP)

At the end of their last computer science bachelor year, students of Eindhoven University of Technology participate in a SEP project. For ten weeks, they work with a real customer to obtain their first

real software development experience. For us, SEP provided the perfect opportunity to let a group of students work with the PRAIS C# API and thereby assess its usability (see Section 5.2.1). The SEP project started on April 20 and lasted until July 3. Relating this to the timeline of this project, this means that the SEP students were provided the PRAIS C# API with LiveSwitch as webRTC provider. Figure 4.1 provides a high level overview of what the SEP students worked on. In Appendix E, a detailed description of the SEP project can be found. The SEP group represented two academic hospitals that want to share NICU AI algorithms with each other. Hospital A developed a pose detection algorithm for neonates while hospital B developed an apnea detection algorithm. Observe that the pose detection (Section 2.4.1), apnea detection (Section 2.4.2), and algorithm sharing (Section 2.4.3) use cases are covered here. We already possessed the pose and apnea algorithms, which we provided to the students. Given Figure 4.1, the students developed the following:

- The pose detection, apnea detection, and database bots, which were developed using the PRAIS C# API. The pose and apnea detection algorithms that we provided to the students are written in Python, so the students developed a Python wrapper that enables running a Python algorithm from C#.
- The hospital A [37] and B [36] web applications, which are used to simulate doctors and NICUs. Since we did not have LiveSwitch as a webRTC provider in the Javascript RTC API, the students had to use LiveSwitch directly in these web applications.
- The hospital A authentication back-end, which consists of an OpenID Connect [38] authentication server and an OpenID Connect identity provider (see Section 4.5.1.2 for more details).
- A simulated database for hospital A that contained (simulated) NICU sensor data. This was done because Philips has NICU sensor data on an internal network drive. By having a database bot that connects to this drive and then streams the data to an algorithm, we aimed to demonstrate that algorithm input can come from different/multiple sources (G1b).

All in all, with the SEP project, we aimed to achieve the following:

- Assess the usability of PRAIS, which we did using a usability study (see Section 5.2).
- Assess the stability of PRAIS, i.e., to discover bugs/other issues.
- To explore the options regarding combining OpenID Connect and PRAIS. This relates closely to the security non-functional requirement (NF-9).
- To explore and prototype visualization streaming (TG4 and see Section 3.1.1).
- To build demonstrators that show the potential of PRAIS (G1b).

In the following sections, where relevant, we include parts of the design of the SEP project.

4.3 Maxima Medisch Centrum

MMC is a hospital located in Veldhoven that has several NICUs. To allow parents to remotely view their babies, MMC is using the NICU S2S application that was built using the Javascript RTC API. A PhD student at MMC is doing a NICU-based research and her project required NICU video recordings, i.e., the recording use case described in Section 2.4.4. This use case required the integration of PRAIS with the NICU S2S application. On a technical level, this meant we had two options:

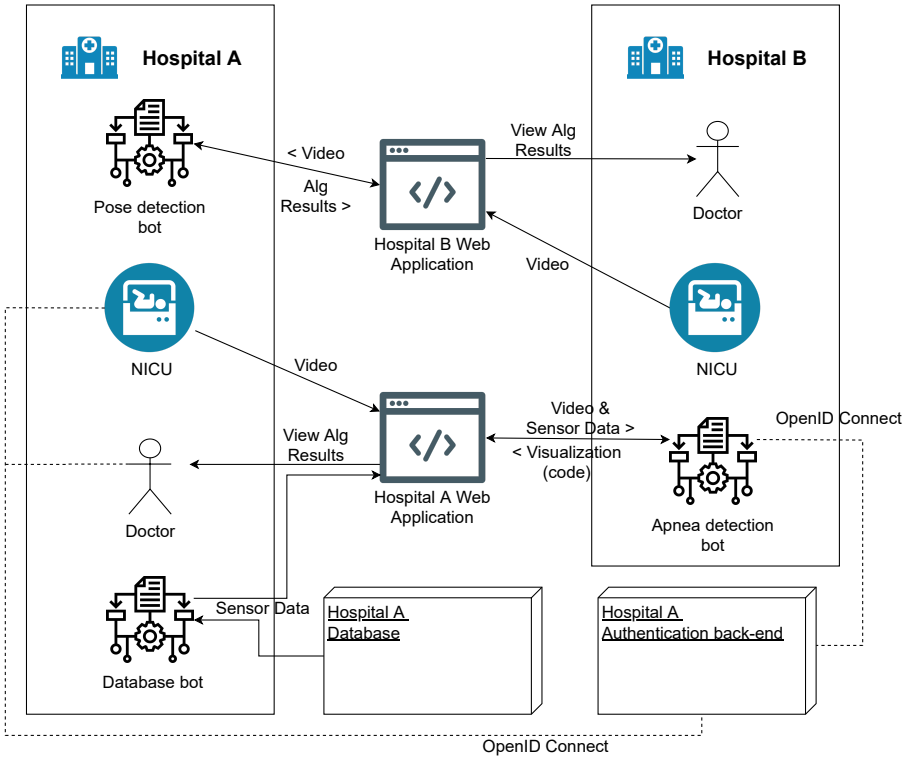


Figure 4.1: A conceptual overview of the system developed by the SEP students.

1. Add ICELink as a webRTC provider to the PRAIS C# API.
2. Add LiveSwitch as a webRTC provider to the Javascript RTC API.

Eventually, we chose option 1 for two reasons. The first is that adding ICELink to the PRAIS C# API allowed us to test whether the vendor abstraction layer was properly designed and implemented (TG1). It turned out that this was the case because we were able to add ICELink as a webRTC provider without having to change the design. Secondly, the NICU S2S application was already running at MMC (indirectly on ICELink). Switching it to LiveSwitch while it was already live was deemed too much of a risk.

Since the PhD student is not very technically proficient in C#, we developed the PRAIS Recorder Application (see Section 4.3.1), which is essentially a UI layer on top of the PRAIS C# API. All in all, with the MMC collaboration, we aimed to achieve the following:

- Integration of the PRAIS C# API and Javascript RTC API was already an important non-functional requirement (NF-8) and goal (TG2). The MMC collaboration provided the perfect opportunity to test the integration.
- To demonstrate how PRAIS can be used to stimulate open innovation (G1a). In particular, PRAIS essentially provides an academic layer on top of the NICU S2S application. Any hospital that uses the NICU S2S application can now use PRAIS to connect algorithms to their NICUs.
- To demonstrate the recording functionality of PRAIS (G1b).

4.3.1 PRAIS Recorder Application

In addition to being a demonstrator, the PRAIS Recorder Application is a tool that was built for pragmatic purposes: to enable recording of NICUs at MMC. Its functionality mostly follows from the requirements specified by MMC (FR-3.4.1.9). For example, features include: log-in before a researcher can connect to a NICU, only connect to NICUs that have informed consent, and recording of a specified number of video chunks of specified duration. By using PRAIS as a basis, most features were trivial to implement (because PRAIS already provides the actual recording functionality), which is why we provide a more elaborate description of the PRAIS Recorder Application in Appendix F.

A more difficult requirement to satisfy relates to the fact that the video recordings are used for research purposes. In this case, an accurate timestamp per recorded video frame was required, which turned out to be difficult to obtain. More specifically, the timestamp should represent the moment a video frame was generated, should describe a UTC date and time, and should be millisecond accurate. Figure 4.2 provides a visual overview of the recording flow at MMC. A webcam is attached via a USB cable to a tablet that runs the NICU S2S web application. The PRAIS Recorder Application, which runs on the researcher's computer, connects to the NICU (the tablet) and then retrieves the video to be recorded. As is shown in Figure 4.2, the ideal moment to generate a timestamp for a video frame is directly after it has been generated. Note, however, that dashed rectangles indicate steps that are executed by software tools/components that we do not control. Consequently, we can only generate a timestamp when we actually record a video frame. After doing some tests, we found that the delay between generating a video frame and actually recording it is about 700 milliseconds. Unfortunately, this delay heavily depends on the network bandwidth availability and CPU availability of the tablet and computer. All in all, aiming to satisfy this requirement, we explored different solutions (see Table 4.1) but concluded that, given the available time, this is the best we can do. In the future, when more time is available, we deem it worthwhile to implement one of the potential solutions listed in Table 4.1.

With the SEP and MMC collaborations, we cover all the use cases described in Section 2.4. Furthermore, the collaborations combined required the implementation of all *must* and *should* requirements (see Section 5.1 for more details). In the next sections, we describe each of the views of the 4+1 view model of architecture [Kru95].

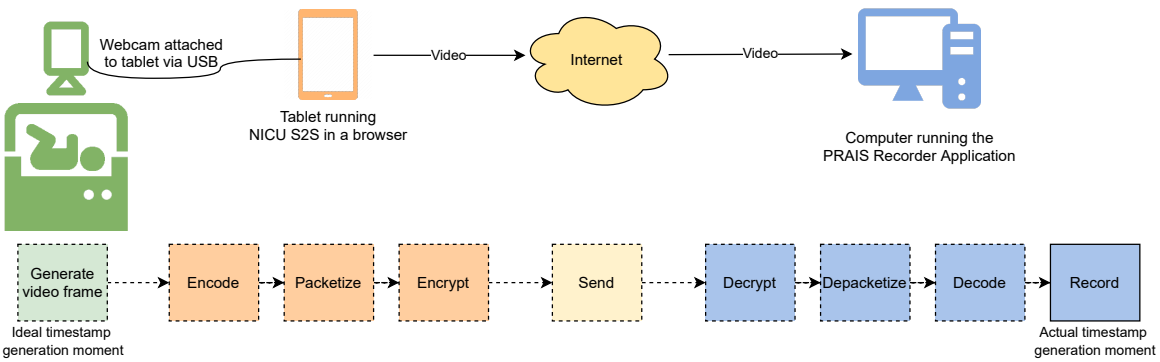


Figure 4.2: A visual overview of the recording flow at MMC. The top part of the figure illustrates which entities are involved in the recording pipeline while the bottom part details which steps are executed by each entity (note the color mapping). In the bottom part, dashed rectangles represent steps that we do not control, i.e., they are part of software tools/components that we use.

Description	Pros	Cons
Generate frame timestamps upon receiving them (chosen solution)	Easy to implement. Only requires changes in the PRAIS C# API.	Timestamps are not accurate.
Embed a timestamp in the video frame itself	Timestamps are accurate.	Both NICU S2S and the PRAIS C# API need to be changed. Unsure about feasibility. Higher coupling: extra dependency between NICU S2S and PRAIS (embedding of timestamps should somehow be toggled from PRAIS).
Send timestamps in parallel over a data channel	Timestamps are accurate.	Both NICU S2S and the PRAIS C# API need to be changed. Generation of video frames is done by the browser, so we are not sure whether this is feasible. Higher coupling: extra dependency between NICU S2S and PRAIS (extra datachannel needs to be opened).
Replace NICU S2S in the browser with a native version of NICU S2S	Enables control over video frame generation and thus enables accurate timestamp generation at the sending side.	Requires a significant amount of development.

Table 4.1: A comparison of the different solutions we considered regarding the generation of timestamps for recorded video frames.

4.4 Logical View

In Figure 4.3, an overview of all the packages and components of PRAIS is shown. On the highest level, we have a C# package and a Javascript package.

The C# package contains the PRAIS C# API NuGet package [30]. A NuGet package is the Microsoft-supported mechanism for sharing code and thereby makes installation of the PRAIS C# API easy (NF-4). As we can see, the NuGet package includes four components that follow a layered architectural pattern. *AlgorithmCore* contains the core functionality of PRAIS. It defines all API functions but does not implement these functions itself because that is the responsibility of the webRTC implementors: *LiveSwitch WebRTC Implementor* and *ICELink WebRTC Implementor*. The only responsibility and functionality of the *PRAIS* component is to connect and instantiate *AlgorithmCore* with one of the implementors at runtime. With this setup, we allow the user to pick a webRTC implementor at runtime and thereby achieve a vendor abstraction layer (NF-3).

Observe that the components of the NuGet package use different .NET variants: .NET Framework 4.7.2 and .NET Standard 2.0. Another .NET variant (that we did not use) is .NET Core. Both .NET Framework and .NET Core are implementations of .NET, while .NET Standard is a formal specification of the APIs that are common across all .NET implementations [26]. In practice, this means that .NET Standard libraries can be used from any .NET implementation. This is exactly why *AlgorithmCore* is implemented as a .NET Standard library, making the core of PRAIS compatible with

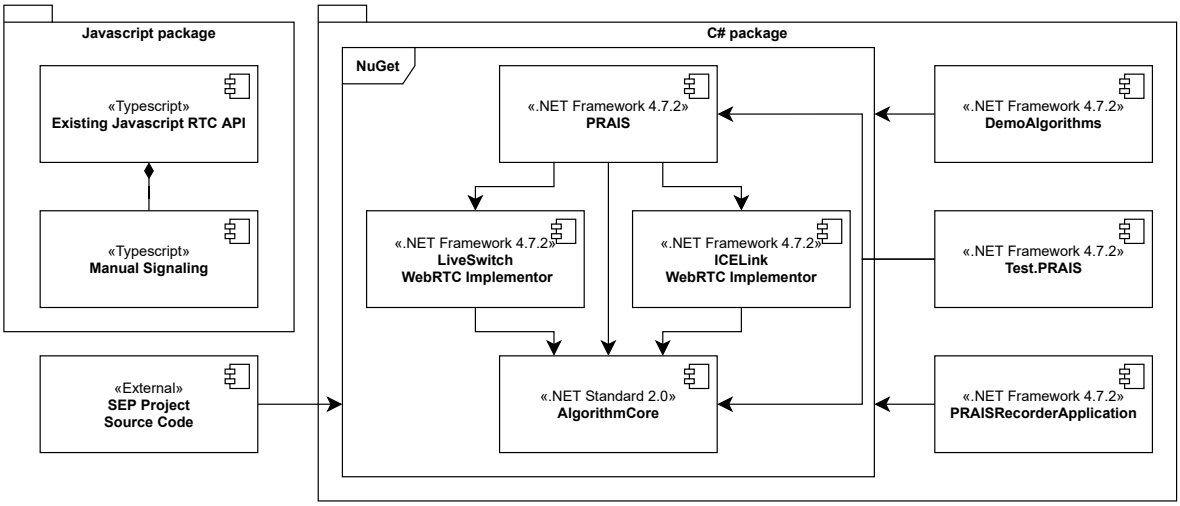


Figure 4.3: An overview of all the packages and components of PRAIS. All arrows represent a *uses* relationship. Overall, we have a Javascript package and a C# package that each contain different components. Furthermore, the *NuGet* frame represents which components are part of the PRAIS C# NuGet package.

essentially any .NET webRTC implementor (NF-10). Table 4.2 shows a comparison of .NET Framework and .NET Core. Given this comparison, we would ideally implement the webRTC implementors in .NET Core (or .NET Standard) because it works cross-platform, is faster, and does not need to run in a Windows docker container. Unfortunately, at the start of this project, ICELink was only provided as a .NET Framework library. LiveSwitch, on the other hand, did also come as a .NET Standard library, which did unfortunately not include relevant features such as camera/screen/microphone media capture and the H.264 video codec on MacOS [39]. Because of this, we used the LiveSwitch and ICELink .NET Framework libraries.

Other components shown in Figure 4.3 include:

- *DemoAlgorithms*: A collection of demo algorithms built using the PRAIS C# API (G1b).
- *Test.PRAIS*: A collection of automated system tests that verify the functionality of the PRAIS C# API. See Section 5.1 for more details.
- *PRAISRecorderApplication*: The recorder application developed for MMC (see Section 4.3.1).
- *SEP Project Source Code*: The SEP students worked with the PRAIS C# API NuGet package.

	.NET Framework	**.NET Core**
Performance	Lower	Higher
Cross-platform	Windows only	Windows, Linux, and MacOS
Maturity	Released in 2002	Released in 2016
Owner	Microsoft	Open source
Docker support	Windows docker containers only	Linux docker containers. More optimized for containerization.

Table 4.2: Comparison of .NET Framework and .NET Core

- *Existing Javascript RTC API*: This is the Javascript RTC API that was already there at the start of this project.
- *Manual Signaling*: This represents an extension of the Javascript RTC API which was required for the integration of the PRAIS C# API and the Javascript RTC API (NF-8).

In Section 4.4.1, we discuss the NuGet package, i.e., the design of the PRAIS C# API in more detail.

4.4.1 The PRAIS C# API

As shown in Figure 4.3, the PRAIS C# API consists of four components. Recall that the *PRAIS* component is a simple layer that connects and instantiates the other components. Therefore, in this section, we only describe the designs of *AlgorithmCore* (Section 4.4.1.1), and *ICELink WebRTC Implementor* and *LiveSwitch WebRTC Implementor* (Section 4.4.1.2). The documentation that belongs to the PRAIS C# API (NF-1) can be found in Appendix G and online [29].

4.4.1.1 Algorithm Core

The class diagram representing the design of *AlgorithmCore* is shown in Figure 4.4. Observe the rectangles in green that represent interfaces that are implemented by the *ICELink WebRTC Implementor* and *LiveSwitch WebRTC Implementor*. With this bridge design pattern [40], we realize the vendor abstraction layer (NF-3). To make the API easily accessible, all core functionality resides in the *AlgorithmImpl* class (NF-2). After instantiating such a class with the correct configuration, an algorithm can perform actions such as *Join()*/*Leave()* conferences (FR-3.4.1.1), *SendMessageToPeer()* (FR-3.4.1.5), *OpenDataChannels()*, and *OpenMediaStream()* (FR-3.4.1.3, FR-3.4.1.7) with a certain media source when desired (FR-3.4.1.4). By using an observer design pattern [Proa], several events that are triggered by the API can be observed. For example, after joining a conference, the *OnPeerConnected* event is triggered whenever a peer joins the same conference. During this project, we reviewed the design of *AlgorithmCore* in a design/code review session with Zoran and Arjan (see Section 2.6). Overall, there were six important design decisions to consider.

Firstly, both ICELink and LiveSwitch use generic connection concepts that bundle audio, video, and data streams. In contrast, webRTC splits these concepts into separate media and data streams. Conceptually, both approaches make sense and technically there also is not a very big difference. In the end, we decided to go for the webRTC approach because of three reasons. Firstly, we noticed that in ICELink/LiveSwitch, a connection has a direction, which makes sense for audio and video but not so much for data. In their documentation, they even write that data connections are always bi-directional. Secondly, we believe that conceptually, audio and video always go well together while data is a more separate concept. Thirdly, the functionality that goes with audio/video and data streams is different. Simply put, a data stream is used to send/receive strings/bytes while an audio/video stream is used to send/receive audio and video frames. Because of these reasons, we decided to split the concepts of media and data streams (NF-2).

Secondly, in addition to a unique *Id* (TG3), Peers also have a *DisplayName*, *Tag*, and list of *Roles*. We decided to add the *Tag* because it allows developers to add any custom (json) string to its peers. Furthermore, we also saw value in enabling algorithms to reason based on a peer's role, which is why we added the *Roles* attribute (NF-2).

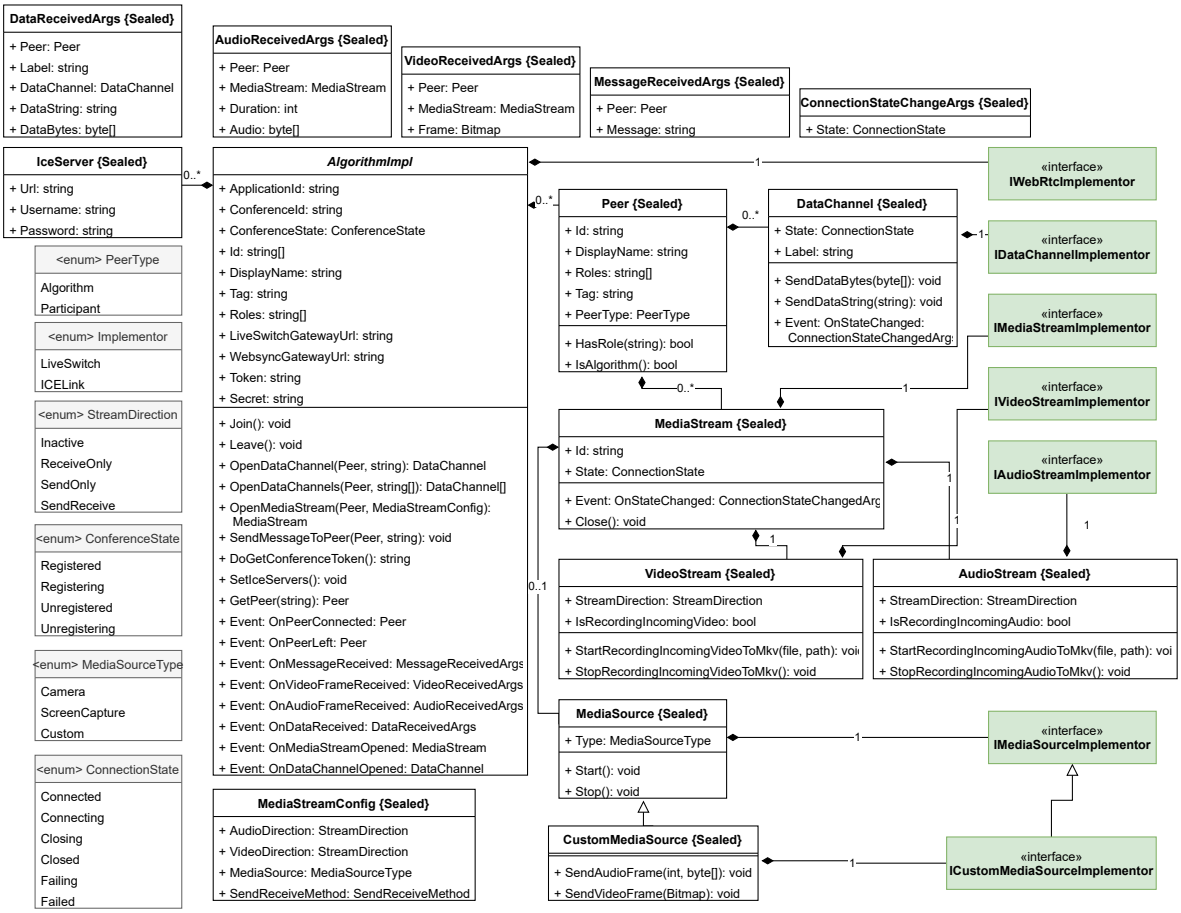


Figure 4.4: The class diagram that represents the design of *AlgorithmCore*. Rectangles in green represent interfaces that are implemented by the *ICELink WebRTC Implementor* and *LiveSwitch WebRTC Implementor*. Note that, to make the diagram readable, not all methods/properties are included. In particular, several asynchronous versions of methods are left out. The complete API specification can be found in Appendix G and online [29].

The *LiveSwitchGatewayUrl* and *WebsyncGatewayUrl* define the address of the back-end server. Depending on which webRTC provider is used, one of the fields should be defined. The *Secret* and *Token* fields are used for authentication (FR-3.4.1.2), which we describe in more detail in Section 4.5.1.

Another design decision relates to the consideration of having a default (inactive) connection to every peer that joins the same conference (which was the behavior in the Prototype C# API). A user would then change the direction of this default connection instead of creating new ones. While, from a design perspective we preferred not having default connections (NF-2, FR-3.4.1.7), it also turned out that LiveSwitch has a bug that prevents the changing of connection directions. At this point, this bug still has not been solved, which made the decision regarding this design trade off easy.

We considered whether there is a requirement for an algorithm instance to be present in multiple conferences at the same time, which is possible in the Javascript RTC API. After some discussion and use case investigation, we could not identify a use case where this functionality is really needed. Therefore, we concluded that the extra technical effort, complexity, and uncertainty (in terms of technical

feasibility) did not outweigh the envisioned benefits.

Lastly, observe that all classes (except *AlgorithmImpl*) are sealed, which means they cannot be extended. This is mainly done from the consideration that in the future, the PRAIS C# API may change. By preventing extension, there is a smaller chance that application code that uses the PRAIS C# API breaks, i.e., there is better backwards compatibility (NF-2).

4.4.1.2 ICELink and LiveSwitch WebRTC Implementors

The class diagram representing the design of *ICELink WebRTC Implementor* is shown in Figure 4.5. The design of *LiveSwitch WebRTC Implementor* is very similar, which is why it is included in Appendix H. The main difference between the two is that *ICELink WebRTC Implementor* also contains *ManualSignaling* (NF-8, FR-3.4.1.6). The green rectangles represent the interfaces that are part of *AlgorithmCore*. By implementing them here, and by connecting them to ICELink functionality (orange rectangles), we realize the bridge design pattern [40], i.e., the vendor abstraction layer.

An important design decision, regarding the *LiveSwitch WebRTC Implementor*, relates to the way in which we implement the connections. Recall that peers can only handle about four simultaneous P2P connections (depending on the hardware) and that LiveSwitch offers support for SFU and MCU connections (see Section 2.2.2). While it would be nice to use such SFU/MCU connections under the hood, it also adds quite some complexity to the system. Furthermore, considering our use cases, there is no use case that really requires more than four active simultaneous connections. Therefore, we decided to use P2P connections (FR-3.4.1.3).

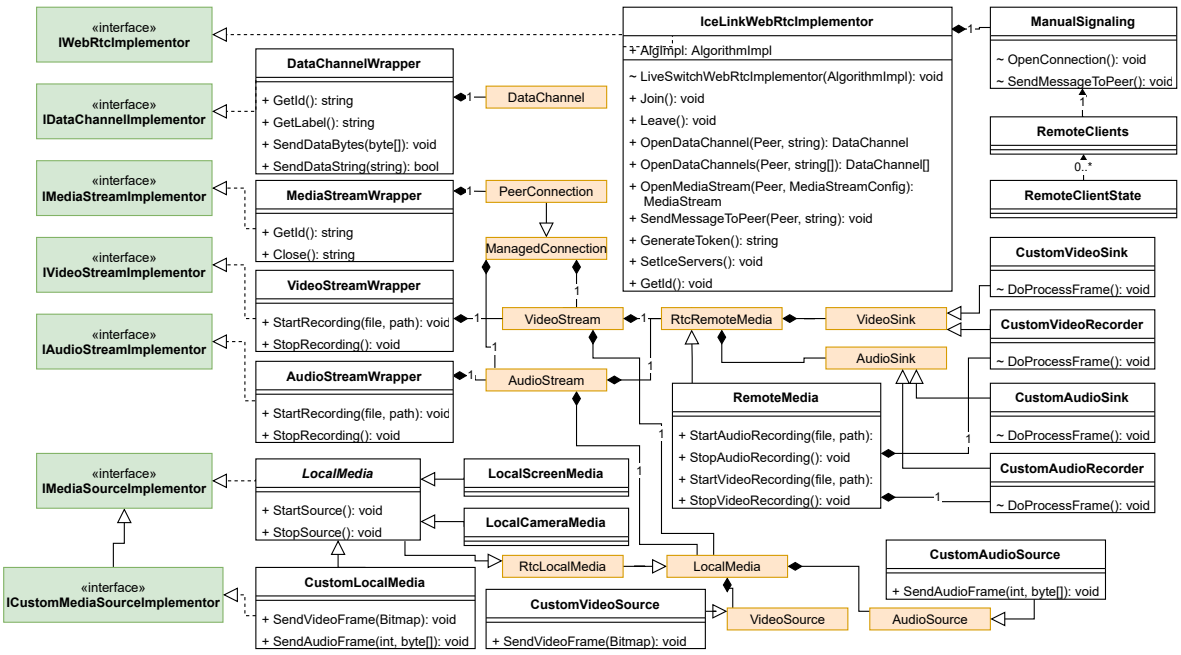


Figure 4.5: The class diagram that represents the design of *ICELink WebRTC Implementor*. Rectangles in green represent the interfaces of *AlgorithmCore*. Rectangles in orange represent ICELink classes. Note that, to make the diagram readable, not all methods/properties are included. In particular, several asynchronous versions of methods are left out.

4.5 Process View

Within the process view, there are several important aspects to consider. In Section 4.5.1, we describe how we use tokens to authenticate peers. After that, in Section 4.5.2, we describe manual signaling. Then, in Section 4.5.3, we discuss some implementation aspects regarding connection setup. Lastly, we describe the design of the visualization streaming prototype developed during the SEP project in Section 4.5.4.

4.5.1 Authentication

PRAIS enables the easy streaming of audio/video/data, which makes security, i.e., authentication, a very important topic. In general, we note the following regarding security in PRAIS:

- As mentioned before, the Javascript RTC API has already been transferred to the CAO. From a security perspective, this means that webRTC has been checked, verified, and approved as a technology to be incorporated into the HSRA (FR-3.4.1.2).
- At the start of this project, the PR team was using token-based authentication [38, 42], in which a peer is authenticated using a token (see Section 4.5.1.1 for more details).
- WebRTC streams are always encrypted. Furthermore, webRTC only works over HTTPS, which means that also the signaling channels are encrypted.

Recall non-functional requirement NF-9, which essentially states that PRAIS should support OAuth2.0 [42] and OpenID Connect [38] (which both work with tokens). These technologies are the leading standard for Single Sign-On (SSO) and identity provision on the internet, which is why Philips prescribes using them [vDWvZ18]. Furthermore, since the existing system was already using token-based authentication and since LiveSwitch also uses tokens [43], we decided to use token-based authentication in PRAIS.

4.5.1.1 Token based authentication

A peer uses a token to authenticate with the back-end (FR-3.4.1.2). For ICELink, this means that a peer requires a token to connect to the WebSync/TURN server. For LiveSwitch, a peer requires a token to connect to the LiveSwitch server. With token-based authentication, there are two important considerations: token generation and token distribution.

Token generation
A token is essentially a piece of data that the back-end uses to authenticate a peer. A token is encoded using a secret that should only be known to the back-end because the back-end uses the same secret to decode the token. Ideally, the tokens used to connect to the ICELink and LiveSwitch back-ends are the same, i.e., there is just one token type for PRAIS. Since the PR team already defined what tokens look like for ICELink and since LiveSwitch uses its own tokens, we have two types of tokens, which are shown in Table 4.3. In the future, it is possible to change one of the systems to make sure one token type can be used by both back-ends.

As we can see in Table 4.3, both tokens use the same core concepts. At the highest level, there is an application for which conferences are created (FR-3.4.1.1). The difference is that LiveSwitch tokens

ICELink	LiveSwitch [43]	Description
Application Salt	Application ID	A unique identifier for the application
-	Client ID	A unique identifier associated with a specific LiveSwitch instance
User ID	User ID	A unique username
-	Device ID	A unique identifier associated with a particular device
-	Roles	A list of roles belonging to the peer
-	Conference ID	Conference that may be connected to
Not valid before time	-	Earliest time at which the token is considered valid
Expiration time	Expiration time	The token is not valid anymore after this time
Type	-	Is either TRN or MSG. Indicates whether the token should be used for connecting to the TURN server or WebSync server respectively.

Table 4.3: Comparison of the ICELink in LiveSwitch token contents

explicitly specify which conference may be joined (which is safer). The application salt in an ICELink token maps to an application ID that is only known by the back-end. An organization key that maps to this application ID is used to encode and decode ICELink tokens. Similarly, LiveSwitch uses a secret for encoding and decoding. Lastly, the most important field in both tokens is the expiration time. With this field, we make sure that tokens are not valid anymore after some point in time, which is vital in case a token is lost/illegally shared (FR-3.4.1.2).

Token distribution

Peers require a token to connect to the back-end, which means they somehow have to obtain this token. Recall that we first want to make things work in a simple but common setting, which we consider to be the scenario where a developer uses the API on his/her computer and from there also runs the algorithms he/she is developing. To this end, in the requirements, we already split token distribution into two levels, resulting in the two requirement categories: *Security* (FR-3.4.1.2) and *Back-end controlled security* (FR-3.4.1.10). *Security* concerns the common setting in which the back-end authenticates using tokens but does not generate those tokens yet. For ICELink, this means that peers simply have a locally stored token. For LiveSwitch, peers generate tokens locally using a secret (see the first alternative in Figure 4.6). Note that this is not safe, but that it is practical for developers. For both back-ends, we have test applications for which these tokens can be used. In applications such as NICU S2S and the PRAIS Recorder Application, *Back-end controlled security* should be used. In such a setting, we use SSO, which we explain in Section 4.5.1.2.

4.5.1.2 Single Sign-On: OAuth2.0 and OpenID Connect

SSO allows a user to log in with a single username and password to different software systems. In our case, this means that a peer obtains ICELink/LiveSwitch tokens by authenticating with an external identity provider (NF-9, FR-3.4.1.10). The biggest advantage of this setup is that the PRAIS back-end does not need to know any credentials, it just needs to be connected to identity providers. Regarding SSO technologies, the Philips HSRA prescribes [vDWvZ18] OAuth2.0 [42] and/or OpenID

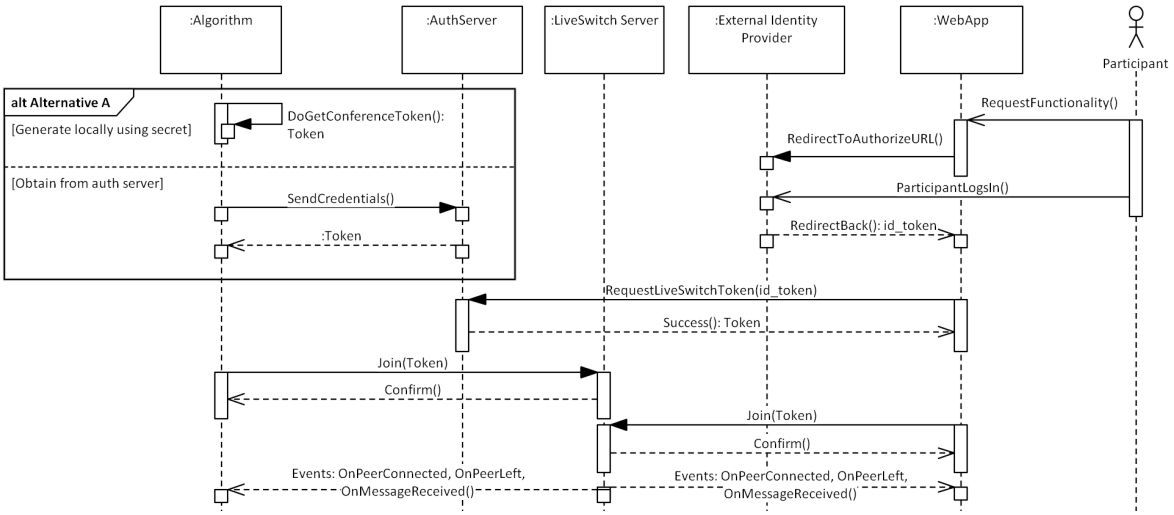


Figure 4.6: A sequence diagram representing the authentication flow implemented by the SEP students for algorithms and participants. Algorithms use the client credentials grant type [45] while participants use the implicit flow grant type [44]. The two bottom-most event calls indicate how the LiveSwitch server notifies peers of certain events.

Connect [38]. OAuth2.0 was developed as a standard for secure authorization for external parties, and later the OpenID Connect specification was defined to add secure authentication as well. All in all, both frameworks define different grant types that can be used to obtain access tokens.

During the SEP project, we explored integrating OAuth2.0 and OpenID Connect with PRAIS. To this end, the students developed an authentication server and simulated an identity provider that does the actual authentication. Figure 4.6 illustrates how algorithms and participants obtain LiveSwitch tokens from the authentication server. Algorithms are not human, meaning they do not really have an identity and they can not interact with a login screen. Because of this, algorithms follow the client credentials grant type [45], in which they directly send locally stored credentials to the authentication server. If they are correct, the authentication server returns a LiveSwitch token. Participants, on the other hand, can interact with a login screen, which is why they follow the implicit flow grant type [44]. In this flow, participants are redirected to the identity provider where they log in. This results in an *id_token* that tells the authentication server the user was authenticated.

All in all, the SEP results show that PRAIS and OAuth2.0/OpenID Connect nicely integrate. For now, the SEP results serve as a prototype and should still be integrated with PRAIS. In the future, the authentication server should be extended to also generate ICELink tokens (or tokens should be standardized). By supporting OAuth2.0/OpenID Connect, PRAIS will allow organizations to use their own authentication mechanism, giving them control over which peers get access to PRAIS.

4.5.2 Manual Signaling

As described in Section 2.1, webRTC enables peer to peer connections but still requires signaling [41] to set up such connections. While the signaling protocol is predefined, the means used to transfer signaling messages is not. In our case, LiveSwitch provides out-of-the-box signaling but ICELink does not. In particular, for ICELink, we have to set up our own signaling server, which the PR

team already did by using WebSync [5]. Since WebSync is the ICELink recommended approach for signaling, ICELink provides a default signaling implementation called *AutoSignaling*. The JavaScript RTC API uses *AutoSignaling* internally, so ideally, the PRAIS C# API also uses *AutoSignaling*. We identified some limitations, however:

- *AutoSignaling* always sets up a default connection to a newly connected peer. This does not work for us because we want to control how and to which peers we connect (NF-2).
- *AutoSignaling* does not provide functionality to easily connect over additional connections to already connected peers. Since we want to potentially have multiple connections to the same peer, *AutoSignaling* does not suffice (FR-3.4.1.7).

For these reasons, we decided to implement signaling ourselves, which allows us to get the flexibility that we require. ICELink already provides a very basic template for implementing signaling, which they call *ManualSignaling*. We extended and implemented *ManualSignaling* both in the Javascript RTC API (FR-3.4.1.6) and the PRAIS C# API (which is required to make them integrate).

Since we already had the WebSync server in place and since the signaling protocol itself [41] is predefined, we mostly had to define how peers exchange (signaling) messages. Figure 4.7 illustrates how an ICELink-powered algorithm obtains a token and connects to the WebSync server. WebSync provides a publish-subscribe messaging mechanism [5], where peers publish and subscribe to channels. In Figure 4.7, for example, the WebSync server creates the channels and the algorithm/participant subscribes to the relevant channels. Overall, we use the following channels (*italic* text represents literal strings while {text} represents a placeholder):

- /*ClientId*/{ClientID} is used to send messages to a peer that is not in the same conference (which is required by one of the Javascript RTC API calls). Any peer can publish to this channel (for which they need to know {ClientID}) and only the peer with {ClientID} subscribes to this channel.
- /*ConfId*/{ConferenceID} is used to send messages to the whole conference. All peers that are part of the conference publish and subscribe to this channel. Furthermore, the WebSync server publishes events whenever a peer subscribes/unsubscribes. This allows the peers to keep track of which peers are present in the conference.
- /*SigClient*/{ConferenceID}/{ClientID} is used to send messages to a peer that is in the same conference. Furthermore, this channel is used as the messaging channel for a specific peer. All peers part of {ConferenceID} can publish while only the peer with {ClientID} subscribes to this channel.

As we can see, we not only require a channel for signaling messages but also channels for peer management (FR-3.4.1.1) and message exchange (FR-3.4.1.5). Note that each of the channels has a certain prefix. We use these prefixes in the WebSync server to filter and log conference subscribe/unsubscribe events (FR-3.4.1.6). Overall, *ManualSignaling* provides a lot of flexibility and enables the integration of the Javascript RTC API and the PRAIS C# API (NF-8). In Section 4.5.2.1, we describe how we achieved this integration.

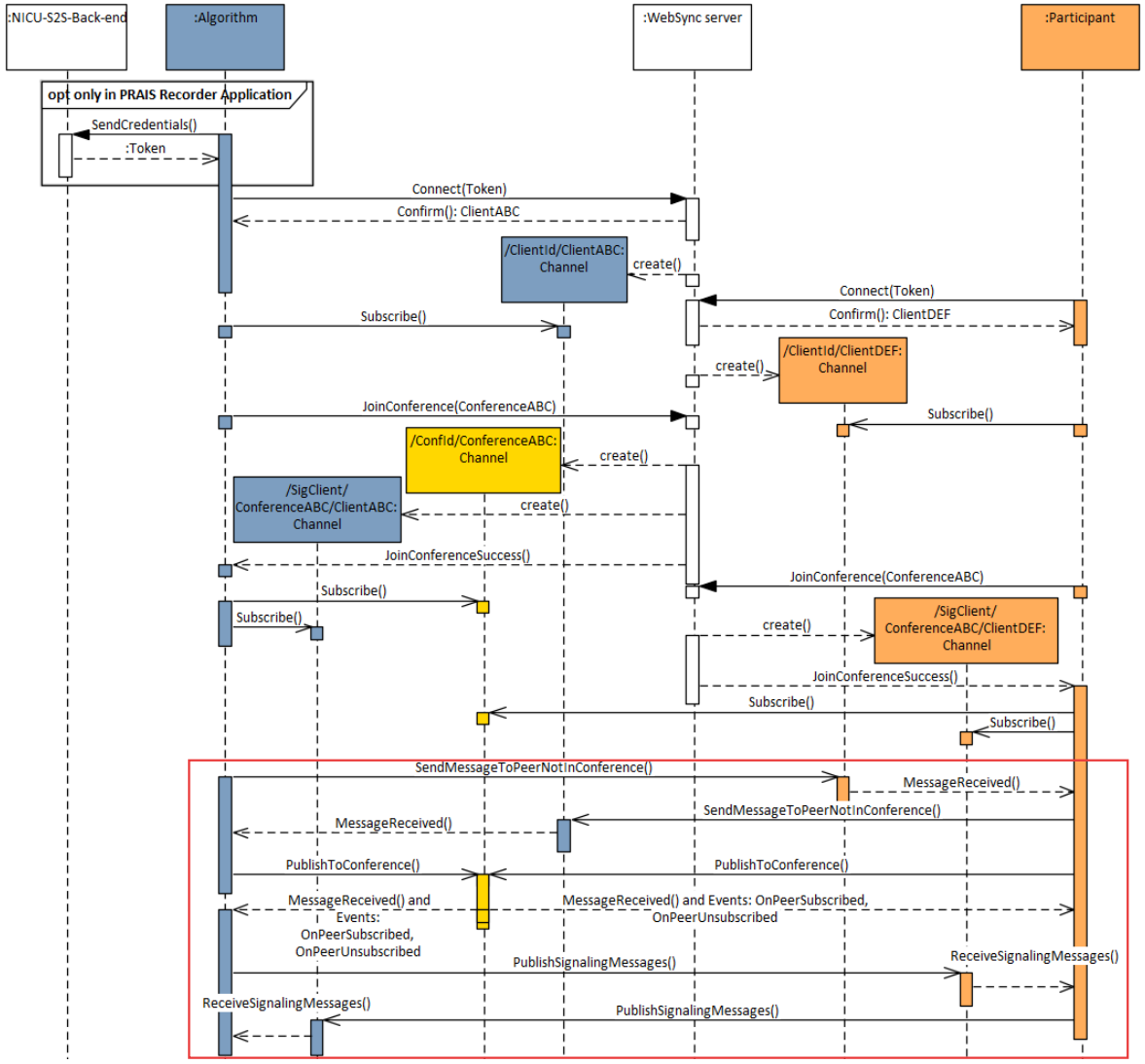


Figure 4.7: A sequence diagram that illustrates the typical *ManualSignaling* flow of an algorithm and participant. Furthermore, the diagram shows how the PRAIS Recorder Application obtains a token (see Section 4.3.1). Blue lifelines represent the algorithm and the channels that the algorithm subscribes to while orange lifelines represent the participant and the channels that the participant subscribes to. The yellow lifeline represents the conference channel that both the algorithm and participant subscribe and publish to. The messages/events in the red rectangle can essentially happen in any order. They are shown here to illustrate what types of messages/events are sent over which channels.

4.5.2.1 Manual Signaling integration

As described in Section 4.5.2, we added manual signaling to make the integration of the Javascript RTC API and the PRAIS C# API possible. Furthermore, at MMC, the NICU S2S application was already being used by nurses and parents. Consequently, we had to deploy the modified Javascript RTC API without breaking the existing system. To this end, we used the following procedure:

1. Implement manual signaling in both the Javascript RTC API and the PRAIS C# API.
2. Verify the implementation by using the automated system tests with the ICELink test app (see Section 5.1.1).
3. In a code review session with Zoran and Arjan (see Section 2.6), we inspected and verified the design and implementation of manual signaling.
4. Set up a test deployment of the NICU S2S application that uses the modified Javascript RTC API.
5. Manually test the application in combination with PRAIS, and fix bugs where needed.
6. Deploy the modified Javascript RTC API to the NICU S2S deployment running at MMC.

By following these steps, we successfully deployed the modified APIs to the NICU S2S application at MMC. By doing so, we realized the connection between NICU S2S and PRAIS, enabling MMC to use NICU footage for research. All in all, we see this as the first example of how PRAIS enables open innovation (G1a).

4.5.3 Connection setup implementation details

There are several implementation aspects to consider regarding the setting up of a connection. In Section 4.5.2, we explained how some of the manual signaling channels include a {ConferenceID}. Also, recall that we always create conferences that belong to a certain application (see Section 4.5.1.1). Consequently, it is possible for two applications to have conferences with the same ID. Therefore, to prevent this, the {ConferenceID} in a manual signaling channel is actually a concatenation of the ICELink application salt and conference ID.

Another problem to tackle when setting up a data channel(s), is that both peers must use the same label for the data channel(s). Unfortunately, both ICELink and LiveSwitch do not have a built-in mechanism to exchange such a label(s). Luckily, for both, when setting up a connection, a tag can be added. We use this tag to include a json string that lists the data channel label(s). By doing so, both peers know which data channel label(s) should be used.

Lastly, in our manual signaling implementation, we use a similar approach for the signaling messages that we exchange. More specifically, we tag every signaling message with a json string that contains: *DataChannelLabels*, *ConnectionID*, and a *Tag*. Depending on the signaling message type (offer, answer, or candidate), not all fields are required. We include a *ConnectionID* because multiple connections may be set up simultaneously. The *Tag* field has value *offer*, *answer*, or *candidate*, to indicate the signaling message type.

4.5.4 Visualization Streaming

During the SEP project, we worked with the students on a visualization streaming (see Section 3.1.1) prototype (FR-3.4.1.12). The visualization streaming process and design is illustrated in Figure 4.8. By using a data channel, the algorithm first streams the Javascript visualization code to the participant, who then loads the script into the browser. The visualization code should implement a simple interface with just three functions: *initializeVisualization()*, *updateVisualization(data)*, and *removeVisualization()*. On the participant side, we can then simply call these functions to load, update, and delete the visualization from a placeholder. Afterwards, another data channel is used to continuously send data to be visualized.

For this project, the goal was to prototype visualization streaming (TG4). In the future, the Javascript RTC API and PRAIS C# API can be extended to provide easy access to visualization streaming.

4.6 Physical View

A visual overview of how different entities are deployed is shown in Figure 4.9. To the back-end, we added an Amazon Elastic Container (EC2) machine *T3a.medium* that runs the LiveSwitch server. This server can essentially run anywhere, meaning that it will also work on premise when needed (NF-6, NF-7). For now, by deploying it on Amazon, we make sure it is available in the cloud (the same is true for *C5n.large)*. We added an Amazon S3 bucket (*healthrtc.org*) that contains the files for the PRAIS documentation (NF-1) and several web apps: ICELink Test App, LiveSwitch Test App, and SEP Web Apps. Observe that these web apps are also shown in the orange *Client browser* to illustrate how they connect to other components. An S3 bucket that already existed at the start of this

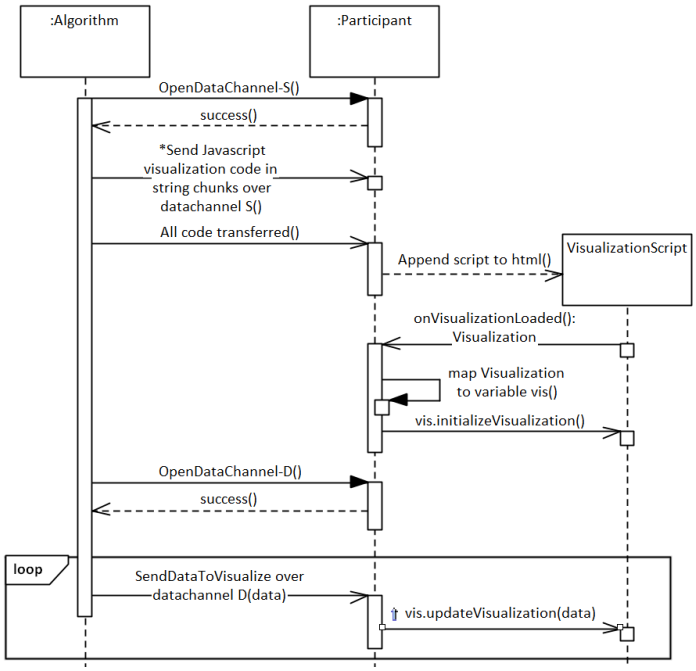


Figure 4.8: A sequence diagram that represents the envisioned visualization streaming design

project (*healthrtc.net*) was extended by adding Telehealth PRAIS and NICU S2S PRAIS. Telehealth PRAIS is a modified telehealth application that uses the Javascript RTC API that contains manual signaling. NICU S2S PRAIS is a test deployment of the NICU S2S application that also runs the modified Javascript RTC API. Both were used to verify the integration of manual signaling with the existing NICU S2S application (NF-8); see Section 4.5.2.1 for more details.

As described before, the PRAIS C# API only runs on .NET Framework. Therefore, the blue *.Net environment* machine needs to be a Windows machine. When developing, .NET Framework 4.7.2 also needs to be installed. Since webRTC is supported on all major browsers, there is quite some flexibility in which *Client browser* to use.

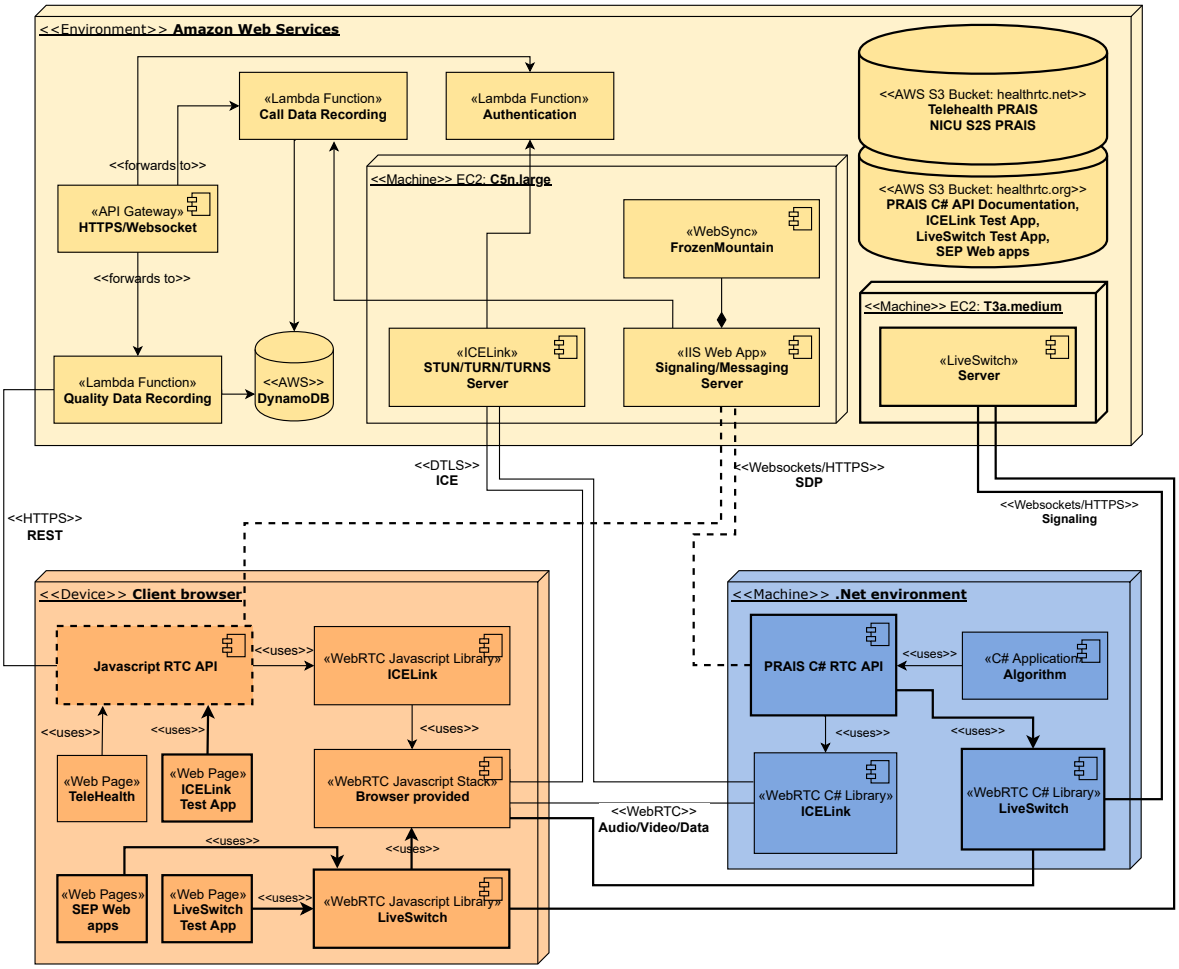


Figure 4.9: A visual overview of how different entities are deployed. This figure is an extended version of Figure 2.3, which shows the deployment of the system at the start of this project. Recall that yellow, orange, and blue colors refer to back-end, participant side, and algorithm side entities, respectively. Lines/rectangles with a bold border are new entities/connections. Dashed lines/rectangles represent connections/entities that were modified.

4.7 Development View

In this section, we describe how the software is organized in its development environment. Non-technical aspects such as project planning, risk management, and way of working are described in Chapter 7. Figure 4.10 provides a visual overview of how the software development is organized. All code is present in the following Philips Gitlab repositories:

- **PRAIS:** Contains all PRAIS source code including the Extensible Markup Language (XML) comments that are used to autogenerate the API documentation, for which we use Docfx [46]. Docfx automatically detects the XML comments included in the source code and compiles them to readable API documentation. In addition, Docfx allows the inclusion of handwritten documentation. We use this, for example, to also include content such as the conceptual PRAIS documentation and the installation instructions. Docfx can be used locally (through the *build_docs* and *serve_docs* batch files), to check whether the documentation is generated correctly. Alternatively, Docfx can be used in the Continuous Integration/Continuous Development (CI/CD) pipeline that automatically builds and uploads the documentation to AWS [29]. The PRAIS repository also contains all automatic system tests, which we describe in more detail in Section 5.1.1. Since some of these tests require a webcam, which is not available in a CI/CD pipeline machine, we decided that for now, it is sufficient to run these tests on the developer's machine before pushing to the main branch. Lastly, whenever a developer builds the PRAIS project, the PRAIS C# API NuGet package is automatically generated locally (see Appendix K).
- **HealthRtcOrg:** Contains the source code of several web apps that are hosted on *healthrtc.org*. In particular, it contains the LiveSwitch Test App [33], ICELink Test App [32], and the web apps [36, 37] created by the SEP students. Similar to the PRAIS repository, whenever a developer pushes to the main branch, all code is automatically deployed to AWS.
- **PRAISExamples:** This is a private repository that we aim to selectively share with developers in the future. It contains PRAIS examples (including the PRAIS Recorder Application) and the PRAIS NuGet package(s). By providing access, we essentially allow the developer to use PRAIS (NF-5).
- **uc-pal-private:** Contains the Javascript RTC API. During this project, a branch was created for the version that contains manual signaling. We use this new version by manually copying it to the Telehealth repository and the AWS S3 bucket that hosts the NICU S2S test deployment. We used this deployment to test whether everything still worked after adding the new Javascript RTC API; more details can be found in Section 4.5.2.1.
- **Telehealth:** To test whether the Telehealth application still works with the new Javascript RTC API (that contains manual signaling), we manually add the new API version, build the required web app files, and then copy the build output to an S3 bucket that is hosted on *healthrtc.net*.

On AWS, Route 53 is Amazon's Domain Name System (DNS) service that we use to route users to the correct domains. CloudFront is also an AWS service that we use to host two AWS S3 buckets as websites: *healthrtc.org* and *healthrtc.net*. The reason for splitting the two is that the .net domain was already live at the start of the project while the .org domain was not. To make sure we could safely work without affecting the existing S3 bucket, we added the .org domain (and corresponding S3 bucket).

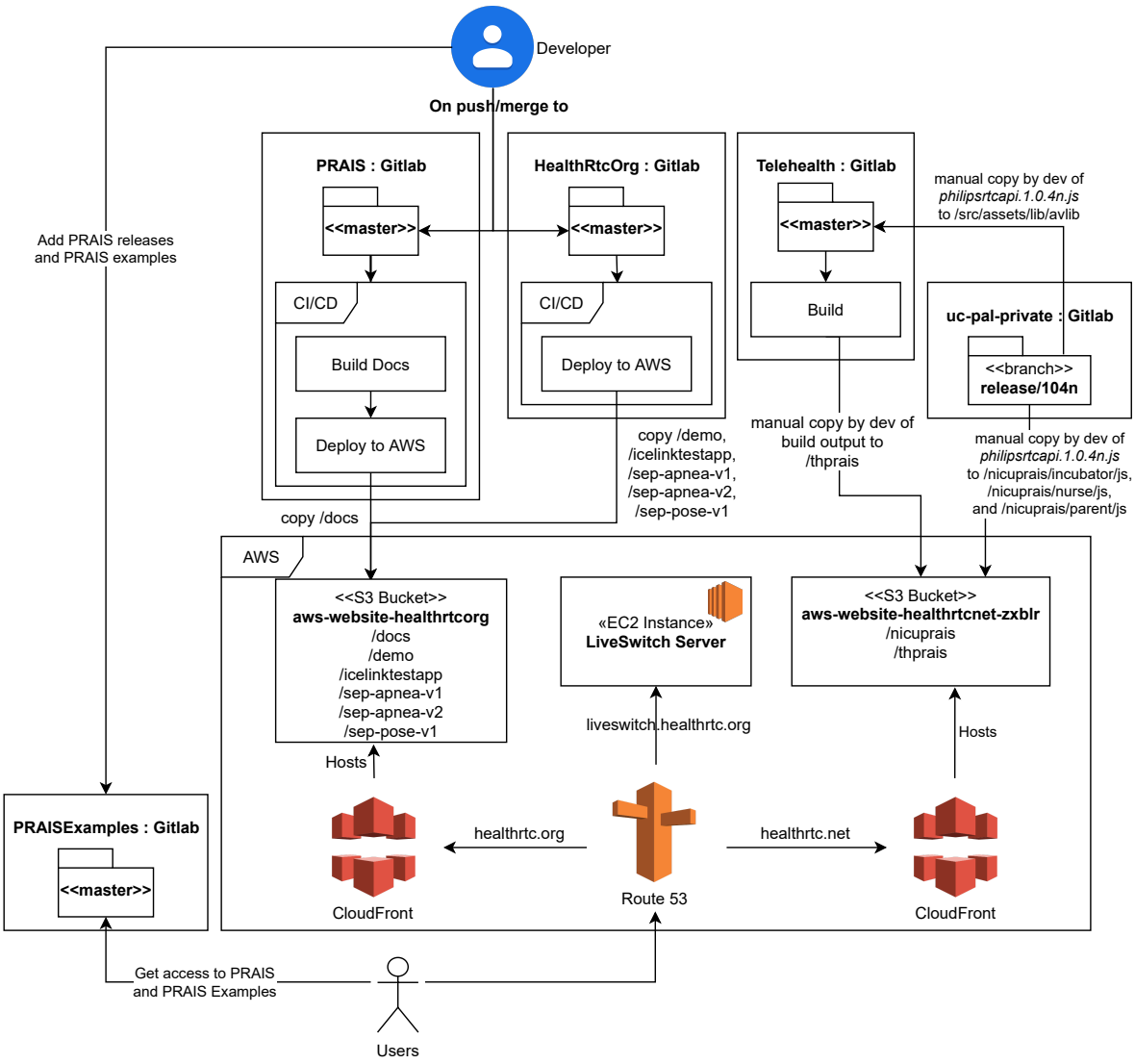


Figure 4.10: A visual overview of how the software development is organized.

5 Verification and Validation

In this chapter, we describe how we verified and validated different aspects of PRAIS. In particular, in Section 5.1, we describe which of the functional requirements were actually implemented and how we tested the functionality. Then, in Sections 5.2 through 5.8, we address each of the non-functional requirements and describe how we validated them.

5.1 Functional evaluation

In Chapter 3, we provided an overview of all 150 functional requirements (see Appendix D) by splitting the requirements into 24 MoSCoW prioritized categories. All requirement categories with *won't* priority were not implemented during this project because of time limitations. The requirement categories with *must*, *should*, or *could* priority are listed in Table 5.1 with their implementation status. As we can see, all requirements with *must* and *should* priorities were implemented successfully. Some requirement categories with *could* priority were not implemented. Regarding *Back-end controlled security*, a first design/prototype was made during the SEP project (see Section 4.5.1.2) and thereby provides a starting point for future development. For the *IL: Recording* requirements, the technical obstacle described in Section 4.3.1 impeded a proper design/implementation. To validate the requirements that were implemented, we used automated system tests (where possible), which are described in Section 5.1.1.

5.1.1 Automated system testing

In general, there are four levels at which software can be tested: unit, integration, system, and acceptance testing [BCS+10]. In a typical development process, unit tests are added first to test the functionality of individual components. Then, the integration of components is tested by adding integration tests. After that, the system as a whole is tested by adding system tests. Finally, the system is provided to actual users, who test the system by using it. Ideally, the first three test types are automated such that changes in the software can be quickly verified. In our case, for example, C# test frameworks such as MSTest [27] provide the tools to easily automate testing.

While developing the PRAIS C# API, we also implemented automated tests, but we did not follow the typical development process because of the nature of PRAIS. More specifically, most algorithm functionality only becomes available after joining a conference and involves other peers. This makes it very difficult to only test individual components or sets of components. Consequently, we decided to only implement tests on the system level. In particular, we used functional testing [BCS+10] and implemented automated functional tests. In practice, this means that all our tests only use PRAIS C# API calls, i.e., the API is treated as a black box. By doing so, in a way, we are indirectly also doing

Table 5.1: An overview of all functional requirement categories (except the requirements with *won't* priority) and their implementation status at the end of the project. M, S, and C stand for Must, Should, and Could respectively. IL and LS refer to ICELink and LiveSwitch. Categories without IL or LS are the categories generic to the system.

Category	Prio.	Impl.	Comment
IL: Algorithm Media Sources	M	✓	
IL: Algorithm Messaging	M	✓	
IL: Basic Conference Management	M	✓	
IL: Javascript RTC API	M	✓	
IL: Peer-to-peer Connections	M	✓	
IL: Security	M	✓	
LS: Algorithm Media Sources	M	✓	
LS: Algorithm Messaging	M	✓	
LS: Basic Conference Management	M	✓	
LS: Peer-to-peer Connections	M	✓	
LS: Security	M	✓	
IL: Algorithm Media Sources	S	✓	
IL: Algorithm Multi Connections	S	✓	
LS: Algorithm Media Sources	S	✓	
LS: Algorithm Multi Connections	S	✓	
PRAIS Recorder Application	S	✓	
IL: Recording	S	✓	
IL: Recording	C	×	See the technical obstacle described in Section 4.3.1.
LS: Recording	C	✓	
Back-end Controlled Security	C	×	During the SEP project, an OpenID Connect authentication server that generates LiveSwitch tokens was implemented (Section 4.5.1.2). To make this server compatible with PRAIS, ICELink token generation should also be added, which is potential future work (see Section 6.1).
LS: Algorithm Media Sources	C	×	Not required for any of the implemented use cases.
IL: Algorithm Media Sources	C	×	Same as above.
Advanced Conference Management	C	×	Due to time limitations, we did not implement this functionality.
User Interface Streaming	C	✓	During the SEP project, a prototype of user interface streaming was implemented (see Section 4.5.4). This proves the feasibility of the concept and serves as a starting point for future development.

unit and integration testing. Furthermore, it allows us to easily test the implementation of different webRTC providers. This was especially useful when adding ICELink as a webRTC provider. For the tests that involve a participant, we created two test webapps, one for LiveSwitch [33] and one for ICELink [32]. By using Selenium [28], we automatically open a browser window with a test webapp (depending on which webRTC implementor we are testing), resulting in a participant that joins a conference. In total, we implemented 80 functional tests using the MSTest framework to verify the functionality of the PRAIS C# API.

In the following sections, we discuss and validate each of the non-functional requirements.

5.2 Usability

Regarding usability, we defined two non-functional requirements in Section 3.4.2:
"*The PRAIS C# API shall be documented for developers.*"
"*The PRAIS C# API shall be easy to use.*"

For the first requirement, the documentation written for PRAIS and the PRAIS C# API can be found online [29] and in Appendix G. In the usability study with the SEP students (see Section 5.2.2), we asked the students what they thought of the documentation. Overall, they were very positive (see Section 5.2.4).

Assessing whether the PRAIS C# API is easy to use is more difficult, because it is a very subjective matter. It requires us to understand the human activities of the PRAIS C# API users. To this end, we did a usability study with the SEP students (see Section 5.2.1).

5.2.1 Usability Study Goal

At the time of executing this project, the SEP students were the only PRAIS users, which is why we aimed to assess the usability of PRAIS using their experience and insights. There exist many different research methods to study and understand the human activities [ESSD08, KLL97] regarding the usage of a system. To pick an applicable research method, we considered the following:

- We have a limited user population (only ten SEP students).
- While the SEP project lasted for ten weeks, we only had two weeks to execute the usability study.
- The benefits of using PRAIS are hard to quantify. More specifically, there are no PRAIS alternatives (that we know of) that were used by the SEP students in the past. The only possible comparison we can do is within the project itself, where LiveSwitch was used for participants in the browser and the PRAIS C# API was used for algorithms.

Considering these factors, we decided to do an exploratory case study [ESSD08], in which we collect quantitative data using a questionnaire and qualitative data through interviews. With the questionnaire results, we aim to understand how the SEP students experienced the usability of PRAIS. Furthermore, by using the Net Promotor Score (NPS) [31], we aim to understand whether or not the students would recommend PRAIS to others. The interview results serve to understand why the students had certain experiences and to understand the strong/weak points of PRAIS. All in all, this user study serves as

an initial investigation of the usability of PRAIS. We aim to derive new hypotheses and build theories regarding PRAIS' usability. The complete study is described in detail in Section 5.2.2.

5.2.2 Methodology

There exist many different tools for assessing the usability of software systems. The System Usability Scale (SUS) [BJ96] was designed to take a quick measurement of how people perceive the usability of software systems they were using. It consists of ten five-point questions with alternating positive and negative tone. A tool that was designed to provide similar results to those obtained with the SUS, is the Usability Metric for User Experience (UMUX) [Fin10]. UMUX consists of four seven-point questions where two are positively and two are negatively toned. In an attempt to even further reduce the number of questions, UMUX-LITE [LUM13] only contains two out of the four UMUX questions. Lastly, another tool we considered and compared against the above-mentioned tools is the Technology Acceptance Model (TAM) [Dav89, VD00, VB08, LLŠ20]. All in all, since TAM nicely splits and defines usability as a combination of perceived usefulness and perceived ease-of-use we decided to use it (see Section 5.2.2.1).

5.2.2.1 The Technology Acceptance Model (TAM)

The Technology Acceptance Model [Dav89] aims to measure perceived usefulness (PU) and perceived ease-of-use (PEU). PU is defined as *"The degree to which a person believes that using a particular system would enhance his or her job performance."* [Dav89]. PEU, in contrast, refers to *"The degree to which a person believes that using a particular system would be free of effort."* [Dav89]. The TAM questionnaire consists of 12 questions, six for the measurement of PU and six for PEU. In later works [VD00, VB08], TAM got extended into TAM2 and TAM3 respectively. In the extended models, aspects such as social influence, cognitive instrumental processes, trust, and perceived risk on system use are also included.

Observe that the purpose of TAM is to predict future use instead of rating the experience of actual use. Lah *et al.* [LLŠ20] address this shortcoming by introducing modified TAM (mTAM). In mTAM, respondents indicate agreement with statements regarding actual user experience instead of anticipated experience. Similar to TAM, mTAM still consists of 12 items, six for PU and six for PEU. We used these 12 items as a basis for defining the PRAIS questionnaire items. In Section 5.2.2.2, we describe the complete study and the questionnaire in more detail.

5.2.2.2 The usability study

As described above, the PRAIS usability study consist of a questionnaire and an interview, which can both be found in Appendix I. The questionnaire consists of 12 seven-point statements about PRAIS that are based on mTAM. More specifically, we modified each statement to specifically talk about PRAIS. Furthermore, the first statement was modified to explicitly talk about LiveSwitch to make the statement more specific: "*Using PRAIS in my job enables me to accomplish tasks more quickly than LiveSwitch.*" In addition to these 12 statements, the questionnaire includes an NPS [31] question: "*How likely is it that you would recommend PRAIS to a friend or colleague?*" With this question, we aim to understand whether or not the students would recommend PRAIS to others. By following interview guidelines [34], we designed the interview questions to understand the experience of the

SEP students and to understand the strong/weak points of PRAIS.

Since the SEP students were the only PRAIS users thus far, we did not have a group to test the usability study with. Instead, we asked the PR team to review the study. Their feedback was used to improve the initial version of the study. Furthermore, the study was reviewed and approved by the Eindhoven University of Technology Ethical Review Board of the Mathematics and Computer Science department (Reference ID: ERB2020MCS6).

All ten SEP students were asked via email to participate (voluntarily) in the usability study and five ended up actually participating. Since the students have different responsibilities within their team, we checked with the participants whether they actually worked with PRAIS during their ten-week project. The five students that did not participate in the study did not work a lot with PRAIS because they had other responsibilities such as testing, documentation, and front-end development. All of the students are in the last year of their computer science bachelor's program. The questionnaire was distributed via email as a Word file that the students could fill in. The interview was done online via the LiveSwitch test app [33]. This allowed us to add an algorithm that recorded the audio of the individual speakers, which made the audio to text transcription much easier. Every interview took approximately half an hour. In Section 5.2.2.3, we discuss aspects to consider regarding the reliability and validity of the study. The results of the questionnaire and interviews are described in Section 5.2.3.

5.2.2.3 Reliability and validity

We assess the reliability and validity of our study by using the following four aspects [KLL97, ESSD08]: construct validity, internal validity, external validity, reliability. Each of these aspects is discussed in more detail below.

Construct validity is used to determine how well a test measures what it is supposed to measure. In our case, this refers to the question whether or not our study actually measures usability. Firstly, usability is a very broad term that can be interpreted in many ways. Because of this, we defined usability more specifically by following the TAM [Dav89] model and we split it into PU and PEU. TAM has been shown to properly measure PU and PEU [Dav89]. A risk to consider here is that we modified the first statement of the mTAM model to talk about LiveSwitch, which may affect the reliability of the results. Secondly, the NPS question aims to determine likelihood to recommend and not usability. Consequently, we cannot use its result to determine usability. Still, it provides an indication of how the students experience working with PRAIS. Lastly, there is no guarantee that the questions asked during the interview are the right ones.

Internal validity considers the study design itself, i.e., whether the results really follow from the data. A factor of influence is the fact that the author of this document was the customer for the students. This introduces the risk of the students being biased in their feedback because there is a hierarchy between the researcher and the participants. In particular, we had to grade the SEP students, which may lead to the students giving mostly positive feedback. To reduce this bias as much as possible, we did the usability study only after grading the SEP project.

Another factor to consider is the comparison to LiveSwitch. The students used the LiveSwitch typescript API while for PRAIS they used the PRAIS C# API. This difference in programming language and the fact that students may be more adept at certain languages may affect how the students feel about the usability of PRAIS compared to LiveSwitch.

External validity considers whether claims regarding the generality of the results are justified. In

our case, the envisioned users of PRAIS are software developers, i.e., professionals. The usability study, however, was done with bachelor level computer science students. In general, students and professionals have different characteristics, e.g., differences in skill/motivation. Consequently, we cannot claim with certainty that the SEP students are good proxies for the envisioned professional users [FZB+18].

Reliability considers whether the study can be repeated with the same results. In our case, the questionnaire and interview can be reused as is in a different study. The biggest challenge in reproducing the results would be the setting of the SEP project.

To test the internal consistency of the PU and PEU measurements, we computed Cronbach's alpha values for both. For PU and PEU, we have Cronbach's alpha values of 0.90 and 0.29 respectively. While the first is acceptable, the second is not. Note that these values do not say much because our sample size is small, even smaller than the number of questions asked, which is a risk to the reliability of this study. Given that the SEP students are the only PRAIS users so far, however, this is the best we could do.

5.2.3 Results

The detailed questionnaire results and interview transcriptions can be found in Appendix J. In Section 5.2.3.1, we describe our findings based on the questionnaire results. After that, in Section 5.2.3.2, we describe how we analyzed the questionnaire transcriptions and report our findings.

5.2.3.1 Quantitative Results

Lah *et al.* [LLŠ20] describe in their mTAM work how to compute mTAM scores for PU and PEU: *"To get mTAM scores that, like the SUS and UMUX, range from 0 to 100, for PU and PEU separately, compute the mean of the item scores, subtract one from that mean, then multiply by 100/6. To get an overall mTAM score, compute the mean of PU and PEU."* [LLŠ20]. Using the questionnaire results, we computed mTAM scores for PU and PEU, which are both 72.77. This appears to be a coincidence as the results for PU and PEU differ. Unfortunately, Lah *et al.* [LLŠ20] do not specify how to interpret these values. Luckily, since mTAM scores were designed to be similar to SUS scores, we can use other resources [BKM09] that provide guidelines on how to interpret such scores. In particular, Bangor *et al.* [BKM09] describe three scales to interpret SUS scores. On an acceptable/not acceptable scale, our 72.77 score would be rated as acceptable. On a classical American grading scale, we would get a C, and on an adjective scale, the score can be interpreted as *good*.

When we do not convert the quantitative results to mTAM scores, we can do some statistical analysis on the original answers. Recall that these values range from 1 to 7, representing strongly disagree to strongly agree respectively. For PU, we have a mean of 5.36 with a standard deviation of 0.91. Similarly, for PEU, we have a mean of 5.36 with a standard deviation of 0.75. This indicates that, on average, participants are positive with respect to PU and PEU of PRAIS.

Regarding the NPS question, all participants answered with either seven or eight, which translates to all participants being neither promoters nor detractors [31]. This results in an NPS score of zero, which is the lowest possible score. Considering that all participants are essentially neutral, we do not deem this as a bad result.

5.2.3.2 Qualitative Results

To analyze and interpret the interview results, we used card sorting [35]. We first transcribed the five interview audio recordings to text. After that, we selected all sentences/statements made about PRAIS and turned these into 166 so-called cards (which can be found in Appendix J). Lastly, all cards were put into one Excel sheet and during a card sorting session, we grouped the cards into categories. The output of this grouping is a hierarchical structure (see Figure 5.1) that essentially represents the mental model of the participants regarding PRAIS. At the highest level, we found three main categories: tips, tops, and neutral statements. Within those categories, we again have subcategories and even sub-subcategories, which can be seen in Figure 5.1. In Section 5.2.4, we discuss these results more in depth.

5.2.4 Discussion and Conclusion

Given the quantitative and qualitative results, we make the following observations regarding the usability of PRAIS.

- Overall, participants are happy with PRAIS (96 tops compared to 67 tips and mTAM scores of 72.77). They feel that PRAIS is simple and easy to use, and that most required functionality is already there.
- There appears to be more consensus when it comes to tops. More specifically, there is no tip that was mentioned by all five participants.

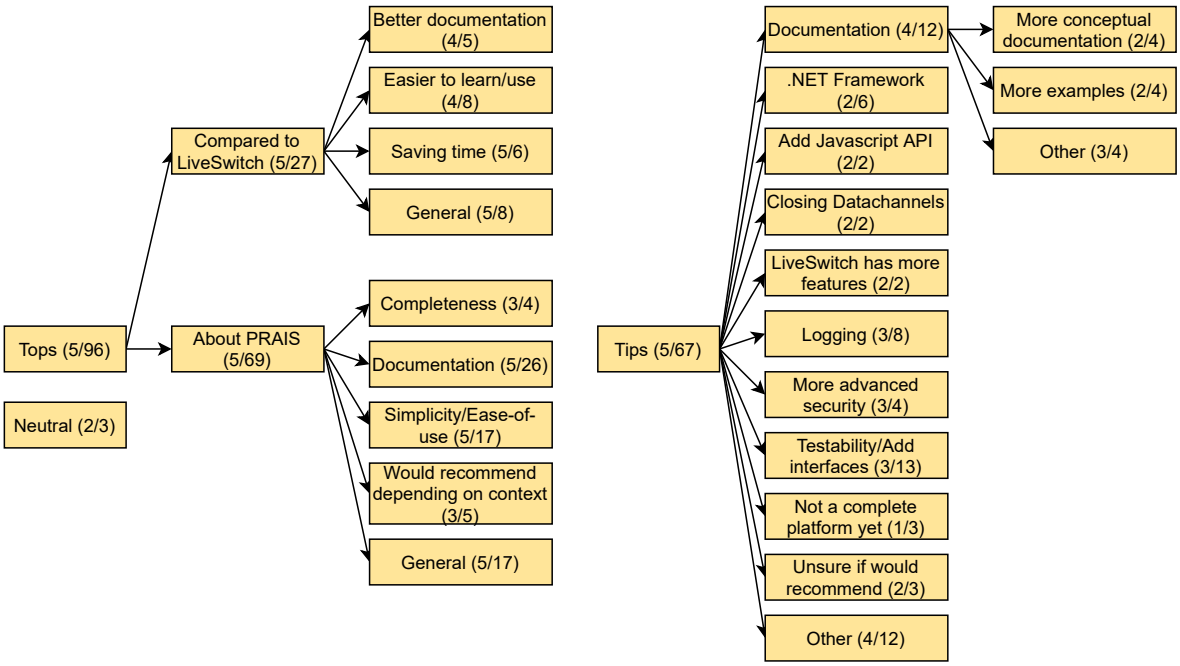


Figure 5.1: A visual overview of the hierarchical structure and categories that were identified after card sorting. The numbers in parentheses represent: *number of participants/number of statements*. For example, all five participants provided tops about PRAIS using 69 statements.

- Participants are happy with PRAIS' documentation, especially compared to LiveSwitch's documentation. They would like some more examples and conceptual documentation, however.
- Compared to LiveSwitch, participants find PRAIS easier to learn, that PRAIS saves time, and that PRAIS is better in general. They also note, however, that LiveSwitch has more features which makes it more suitable in complex use cases.
- Most participants would recommend PRAIS if a friend or colleague were to build a streaming application.
- Most tips refer to additional nice-to-have features such as closing of data channels, better logging, and more interfaces to make testing easier.
- The tip regarding .NET Framework is more fundamental to PRAIS and even directly relates to NF-10 (also see Section 5.8). In essence, .NET Framework is Windows-only and the students encountered some issues with Visual Studio. They recommend providing PRAIS as .NET Core/Standard to make PRAIS also available on Linux/MacOS.
- The tip regarding adding a Javascript API is something we had already solved by implementing ICELink as a webRTC provider in the PRAIS C# API.

Overall, the usability study results provide very valuable insights into the usability of PRAIS. In the future, we definitely recommend considering adding the features that were identified as missing (see Section 6.1). Given the scope and goals of this project, we can conclude that PRAIS satisfies the usability requirements. It is to be seen whether these results can also be generalized to a broader audience. Therefore, we hypothesize: *"PRAIS is perceived as useful and easy to use by software developers."*

5.3 Vendor Abstraction

Regarding vendor abstraction, we defined the following non-functional requirement in Section 3.4.2: "*The PRAIS C# API shall abstract away the webRTC provider.*"

In Section 4.4, we described how we achieve vendor abstraction in our design with a bridge pattern [40]. By adding both LiveSwitch and ICELink as webRTC providers, we have shown that different providers can be added. Furthermore, the automated system testing (see Section 5.1.1) at API level makes it very easy to check whether a new webRTC provider implementation works correctly. The only place where vendor specific infrastructure is required is the back-end, where we have both a WebSync and a LiveSwitch server. All in all, we deem that this requirement is satisfied.

5.4 Installability

Regarding installability, we defined the following non-functional requirements in Section 3.4.2:
"*The PRAIS C# API shall be easy to install by users.*"
"*Philips shall control who gets access to the PRAIS C# API.*"

As described in Section 2.3, the Prototype C# RTC API was part of a large Visual Studio [23] solution that also contained other (unnecessary) elements than the API. To make the installation easier, the PRAIS C# API is distributed as a NuGet [30] package. The NuGet package is internally available within a Philips repository. Where needed, Philips can grant access to others if they wish to use

PRAIS. The PRAIS installation instructions are written in the online documentation [29]. Furthermore, after asking, the SEP students indicated that all of them were able to install the PRAIS C# API without any issues.

5.5 Deployability

Regarding deployability, we defined the following non-functional requirements in Section 3.4.2:
"*The system shall run in the cloud.*"
"*The system shall run on premise.*"

In Section 4.6, we described the deployment of PRAIS. Whether or not the system can run in the cloud or on premise depends on the back-end. Currently, both the WebSync and LiveSwitch server run in the cloud. Each of these servers can also be deployed on an on-premise machine, making it possible to also run the system on premise.

5.6 Security

Regarding security, we defined the following non-functional requirement with *could* priority in Section 3.4.2: "*The system shall use the access control technology prescribed by the Philips PaaS document [vDWvZ18].*"

As described in Section 4.5.1, during the SEP project, we successfully prototyped the integration of Auth2.0/OpenID Connect and PRAIS. Due to time limitations, however, we did not manage to fully integrate the authentication server with PRAIS. Still, the provided design (see Section 4.5.1.2) and implemented authentication server provide a good starting point for future integration.

5.7 Integratability

Regarding integratability, we defined the following non-functional requirement in Section 3.4.2: "*The PRAIS C# API shall integrate with the Javascript RTC API.*"

As described in Section 4.5.2, with manual signaling, we realized the integration of the Javascript RTC API and the PRAIS C# API. At MMC we tested this integration with the PRAIS Recorder Application (see Section 4.3.1) in a real-life scenario where recording functionality was added to the NICU S2S application being used at MMC.

5.8 Compatibility

Regarding compatibility, we defined the following non-functional requirement with *could* priority in Section 3.4.2: "*The PRAIS C# API shall be usable from any .NET implementation.*"

In Section 4.4, we described the four core components of the PRAIS C# API and explained our choice for different .NET variants. Currently, the PRAIS C# API is only available in .NET Framework. The core of the API, however, is implemented as a .NET Standard library, meaning that in the future, any .NET webRTC provider can be added.

6 Conclusion and Future Work

There are two fronts on which Philips leverages its Health Suite Reference Architecture (HSRA). Firstly, the HSRA provides a consistent, unified, and company-wide approach to software architecture and development within Philips itself. Secondly, the HSRA provides architectural building blocks that are leveraged in open innovation collaborations. Philips Research (PR) is mainly responsible for maturing and validating new technologies such that they can be adopted in the HSRA. *Remote AI streaming*, is one of such technologies, and with it, we aim to address several issues (as listed in Section 1.1): to enable remote AI streaming use cases, to remove the need for care providers to buy/-maintain expensive hardware, to remove the need for care providers to develop/maintain streaming technology, to make AI algorithms more available/replaceable, and to enable sharing of AI algorithms.

In this work, we presented the Philips Remote AI Streaming (PRAIS) platform, which aims to address the above-listed issues. The design and development of PRAIS was driven by a set of technical and non-technical project goals, which are repeated below:

G1 Mature remote AI streaming such that it reaches a maturity level that is suitable for the advanced development phase. This means that:
   (a) PRAIS is ready to be used by open innovation partners.
   (b) Demonstrators that show the potential of PRAIS have been implemented.

TG1 Design and implement the PRAIS C# API. In particular, a vendor abstraction layer should be added.

TG2 Integrate the PRAIS C# API with the existing Javascript RTC API.

TG3 Address the technical limitations of the existing system (see Section 2.3).

TG4 Investigate and prototype visualization streaming (see Section 3.1.1).

At the core of PRAIS lies the PRAIS C# API, which supports both ICELink and LiveSwitch as webRTC providers (TG1). We verified the usability of the API in a collaboration with SEP students, during which PRAIS demonstrators were built (G1b). Furthermore, we did a usability study of which the results show that users find the API easy to use. During the SEP project, we also established the feasibility of visualization streaming. This proof of concept provides the basis for further exploration (TG4). After adding manual signaling, we successfully integrated the PRAIS C# API with the Javascript RTC API (TG2). By doing so, we connected PRAIS to the NICU S2S application running at MMC. By using the PRAIS Recorder application (G1b), MMC is able to record NICU video footage, which they use for research purposes. This is a first example of how PRAIS can stimulate open innovation (G1a). In addition to designing and implementing the PRAIS C# API, we addressed most of the identified limitations in the existing system, either by implementing functionality or by providing a design for an envisioned solution (TG3). All in all, the project results provide strong evidence that PRAIS is becoming mature (G1). In the future, more demonstrators should be built to further prove PRAIS' value and we envision adding several features, as described in Section 6.1.

6.1 Recommendations and Future Work

In this section, we list possible directions for future work.

LiveSwitch and .NET Standard
Overall, based on our experience with LiveSwitch and ICELink, we would recommend LiveSwitch. While ICELink provides more flexibility in the sense that any signaling implementation can be used, LiveSwitch provides out-of-the-box signaling, which saves a lot of development/maintenance time. Furthermore, LiveSwitch supports features such as SFU/MCU connections, enabling use cases that require more than four simultaneous connections. The main drawback of LiveSwitch is that the LiveSwitch server must be used. This means that FrozenMountain [4] also requests payment for P2P connections (which is not the case with ICELink).

On a technical level, we already mentioned in Section 4.4 that LiveSwitch is also provided in .NET Standard. In addition, around the end of this project, ICELink was also released in .NET Standard. Since .NET Standard works cross-platform, is faster, and can run in a Linux container, we recommend implementing PRAIS with .NET Standard. Do note, however, that not all features are supported in .NET Standard [39].

Authentication back-end and Standardized tokens
As mentioned in Section 4.5.1.1, we use different tokens for ICELink and LiveSwitch. In the future, we recommend standardizing the token format such that both back-ends can use the same tokens.

In parallel, we recommend integrating the authentication server built during the SEP project with PRAIS. By doing so, PRAIS will support OpenID Connect, which is in line with the Philips PaaS guidelines [vDWvZ18]. Once integrated, the authentication server can be extended to also support other SSO protocols such as LDAP [47].

Visualization streaming
During the SEP project, we successfully prototyped visualization streaming. To make visualization streaming more usable/accessible, we recommend extending the Javascript RTC API and the PRAIS C# API with functionality that enables the easy setting up of visualization streaming. The design described in Section 4.5.4 provides a starting point for such an extension.

Recording timestamps
As described in Section 4.3.1, we considered several solutions regarding the generation of timestamps per recorded video frame. In the future, to improve upon the current approach, we recommend to implement one of the considered solutions.

7 Project Management

In this chapter, we discuss several aspects regarding the project management of this project. In Section 7.1, we discuss our way of working. After that, in Section 7.2, we provide a complete overview of the project planning. Then, in Section 7.3, we describe how we managed risks during this project. Lastly, in Section 7.4, we provide a retrospective by the author of this document. In this chapter, we refer to the main author as *the trainee*.

7.1 Way of working

In total, this project lasted for ten months, during which the trainee was part of the PR team. Because of the corona virus, this was a very interesting project from a project management perspective. The first 2.5 months of the project took place in the office, but the remainder was done almost completely remotely. This also implied a change in the way of working.

Within the PR team, a Scrum/Agile [48] working methodology is used. Originally, sprints lasted for two weeks and stand-ups were held two times per week. When we started working remotely, however, we increased the stand-up frequency to five times per week. Other recurring meetings such as the sprint planning and sprint demo provided a structured way to keep the team up to date. Furthermore, where needed, ad hoc meetings were scheduled.

In addition to the Scrum/Agile process, weekly meetings were held with the TU/e supervisor to keep track of the project's short-term progress. On a monthly basis, during the Project Steering Group (PSG) meetings, the trainee, TU/e supervisor, Project owner, and Project mentor came together to discuss the project planning on a higher level. During the SEP project, weekly demo and planning meetings were held to monitor and steer the SEP project. During the collaboration with MMC, we planned remote meetings on an ad-hoc basis when needed.

7.2 Planning

On a high level, this project can be split into three phases:

1. LiveSwitch phase in which we gathered requirements, defined relevant use cases, prepared the SEP project, and designed and implemented the PRAIS C# API using LiveSwitch as a webRTC provider.
2. ICELink phase in which we implemented the PRAIS C# API using ICELink as a webRTC provider, added manual signaling, and integrated the Javascript RTC API and the PRAIS C# API. In parallel, we executed the SEP project.

3. Recording phase in which we deployed the new API versions to MMC, added recording features to PRAIS, and built the PRAIS Recorder Application.

Throughout these phases, other activities also took place such as report writing and the comeback day presentation. In Appendix L, we provide a Gantt chart that illustrates the complete project planning. A milestone trend analysis chart that shows how we planned/achieved different milestones over time is shown in Figure 7.1. In this chart, the User Doc V1, System V2, and PRAIS Recorder App milestones represent the ends of the LiveSwitch, ICELink, and Recording phases, respectively. Overall, the planning of the first phase went smoothly, mainly because the start of the SEP project resulted in a very strict deadline. Consequently, almost all effort went into designing and developing the PRAIS C# API. During the second and third phases, the planning was more flexible, especially when it came to the report (note how the report milestones change a lot). The goal was to finish all main chapters of the report before a big review session (and holiday) at the beginning of August.

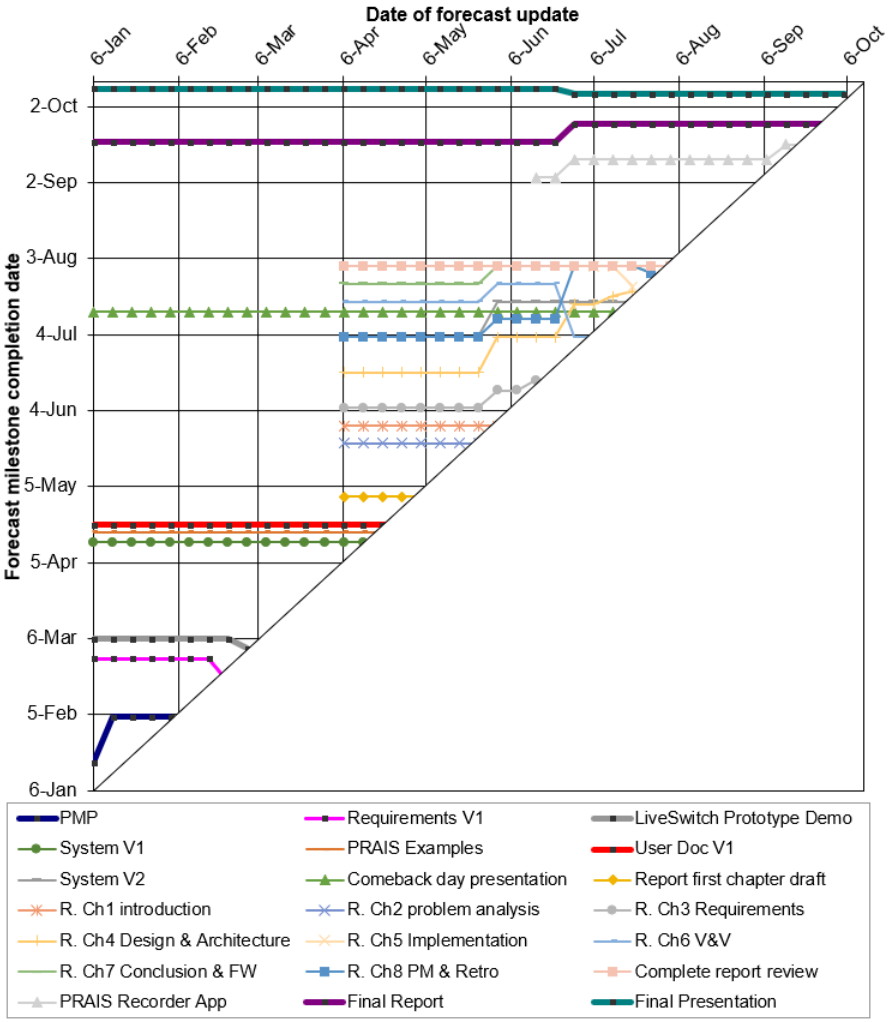


Figure 7.1: A milestone trend analysis chart that shows how milestones were planned and achieved over time. *R.* stands for Report.

7.3 Risk Management

In any project, there are risks, and to prevent/mitigate them as well as possible, we kept track of the risks that we were aware of. For each risk, we defined an: identifier, status, description, likelihood, impact, priority, mitigation action (reduce likelihood), and contingency action (reduce impact). The likelihood value ranges from 1 to 5 and represents: extremely unlikely, remote possibility, possibly occur, will probably occur, and almost certain, respectively. The impact value ranges from 1 to 5 and represents: insignificant, minor, moderate, major, and catastrophic, respectively. To get a risk priority, we multiply the likelihood and impact values. During every PSG meeting, we would discuss newly identified risks and check if there were unidentified risks. The complete list of identified risks is too large to include in the main text, which is why Table 7.1 lists only one risk to show how we managed them. The complete risk table can be found in Appendix L.

ID	Status	Description	L	I	P	Mitigation action (reduce likelihood)	Contingency action (reduce impact)
12	Mitigated, all required features were implemented on time.	The SEP students will have to work with the system, which means that when they start, there should be a minimal viable version that works well enough for them to complete the project. So, there is a risk that the system would not be ready yet.	0	4	0	First, defining and agreeing on what is included in the minimal viable system is essential. Robin shall do this by defining priorities of the requirements. Second, finishing the minimal viable system shall have the highest priority until the SEP project begins.	Assuming there is at least a minimal workable system, i.e., the minimal viable system is not complete but there is at least some functionality. Then, it is essential to come up with a project that is still doable and also has value to this project. In the worst case, when there is no working system at all, we may have to cancel the SEP project.

Table 7.1: A risk that we identified during the project. The ID, L, I, and P columns represent Identifier, Likelihood, Impact, and Priority, respectively. The P column is color coded with a gradient from red to yellow that represent high to low priority respectively.

7.4 Retrospective

In this section, I reflect on different project aspects and discuss some lessons learned.

Philips

In addition to being the longest project that I ever worked on, this was also my first project in a very large company. In the beginning, this was quite challenging because I had to figure out what my position was in this enormous organization. Luckily, after some time, I learned who is a relevant stakeholder and who is not. In addition, I learned that one can greatly benefit from the connections of colleagues. For example, I had to get in touch with privacy/security officers. Instead of going around looking for them myself, I simply asked my supervisor who already knew whom to contact.

SEP
Something I really enjoyed during this project was supervising the SEP project. For me, it was the first time that I was a customer. It was amazing to experience a collaboration where both parties really benefit. The students indicated that they really enjoyed the project and I ended up with useful demonstrators and PRAIS feedback. It was amazing to see how a group of enthusiastic students picks up your project definition and delivers a working product after just ten weeks. On a personal level, I learned that I really enjoyed managing a group and feel confident in saying that I was also good at it.

MMC
The collaboration with MMC was also very motivating and rewarding. After working on PRAIS for approximately seven months, it was very rewarding to see it finally being used in practice by researchers. Especially the knowledge that PRAIS enables innovation that is eventually going to help babies is very exciting. In addition, it was my first collaboration with an actual hospital. Throughout the project, I heard several stories within the team about how privacy and security are important topics. Only until I actually talked to the people in the hospital did I realize how important such topics are. All in all, I see the collaboration with MMC as a very valuable experience.

Changing Environment
This project was pretty dynamic for two main reasons. The first is the corona virus, which resulted in suddenly having to work full-time from home. The second lies in the switch from LiveSwitch to ICELink. All in all, this required quite some flexibility and I believe that I managed quite well. With the Scrum/Agile way of working, it was fairly easy to react to the changing environment. Furthermore, in Philips Research, it is quite normal to encounter unexpected/new circumstances, meaning that the team was also very supportive in adapting to new situations.

It is all about money
As software engineers, we love to stay in our technical bubble and simply want to develop software. When money is involved, however, which is the case in any company, business aspects play an important role. In this project, this became especially apparent for the webRTC providers. Vendor abstraction was an important design aspect to prevent vendor lock-in, and obtaining a new ICELink license turned out to be quite a long process. All in all, this taught me two things. Firstly, no matter how perfect a technical solution is, if it is too expensive, it will not matter. Secondly, in a big company, it often takes some time to get what you need because there are quite some people/steps to go through.

Bibliography

[1] Liveswitch. https://help.frozenmountain.com/docs/liveswitch. Accessed: 2020-04-30. 6

[10] Spitfire. https://github.com/RainwayApp/spitfire. Accessed: 2020-05-12. 69

[11] Webrtc for the universal windows platform. https://webrtc-uwp.github.io/. Accessed: 2020-05-12. 69

[12] Mixedreality-webrtc project. https://microsoft.github.io/MixedReality-WebRTC/manual/introduction.html. Accessed: 2020-05-12. 69

[13] Vidyo.io. https://developer.vidyo.io/#/documentation. Accessed: 2020-05-12. 7, 70

[14] Opentok platform. https://tokbox.com/developer/guides/basics/. Accessed: 2020-05-12. 70

[16] C# built-in types. https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types. Accessed: 2020-05-19. 12

[17] C# data structures. https://docs.microsoft.com/en-us/dotnet/standard/collections/. Accessed: 2020-05-19. 12

[18] Json. https://www.json.org/json-en.html. Accessed: 2020-05-19. 12

[19] Philips research. https://www.philips.com/a-w/research/about-philips-research.html. Accessed: 2020-05-26. 1

[2] Icelink. https://help.frozenmountain.com/docs/icelink3. Accessed: 2020-04-30. 6

[20] Healthsuite reference architecture. https://www.philips.com/a-w/research/blog/20191209-reference-architectures-and-guardrails-burden-or-opportunity-for-innovation.html. Accessed: 2020-05-26. 1

[21] Philips. https://www.philips.com/a-w/about/company/our-strategy/our-strategic-focus. Accessed: 2020-05-27. 1

[22] Amazon web services. https://aws.amazon.com/. Accessed: 2020-06-04. 7

[23] Microsoft visual studio. https://visualstudio.microsoft.com/. Accessed: 2020-06-04. 9, 52

[25] Moscow. https://www.volkerdon.com/pages/moscow-prioritisation. Accessed: 2020-06-16. 20

[26] .net standard. https://docs.microsoft.com/en-us/dotnet/standard/net-standard. Accessed: 2020-06-16. 22, 29

[27] Mstest framework. https://docs.microsoft.com/en-us/visualstudio/test/getting-started-with-unit-testing?view=vs-2019. Accessed: 2020-06-22. 45

[28] Selenium. https://www.selenium.dev/. Accessed: 2020-06-22. 47

[29] Prais online documentation. https://healthrtc.org/docs/articles/conceptual.html. Accessed: 2020-06-24. xvi, 31, 32, 42, 47, 53, 99

[30] Nuget packages. https://docs.microsoft.com/en-us/nuget/what-is-nuget. Accessed: 2020-06-24. 29, 52, 173

[31] Net promotor score (nps). https://hbr.org/2003/12/the-one-number-you-need-to-grow. Accessed: 2020-06-26. 47, 48, 50

[32] Icelink test webapp. https://healthrtc.org/icelinktestapp/testapp.html?&channel=ConferenceId&mode=P2P&username=Anonymous&tag=tag&roles=roleone&token=ENTERTOKENHERE. Accessed: 2020-06-22. 42, 47

[33] Liveswitch test webapp. https://healthrtc.org/demo/index.htm. Accessed: 2020-06-22. 17, 42, 47, 49

[34] Survey best practices & design guidelines. https://www.surveymonkey.com/mp/survey-guidelines/. Accessed: 2020-06-26. 48

[35] Card sorting. https://github.com/ds4se/chapters/blob/master/zimmermann/card-sorting.md. Accessed: 2020-07-08. 51

[36] Sep hospital b web application. https://healthrtc.org/sep-pose-v1/index.html. Accessed: 2020-07-15. 26, 42

[37] Sep hospital a web application. https://healthrtc.org/sep-apnea-v1/index.html. Accessed: 2020-07-15. 26, 42

[38] Openid connect. https://openid.net/connect/. Accessed: 2020-07-15. 17, 22, 26, 34, 36

[39] Liveswitch library versions. https://help.frozenmountain.com/docs/liveswitch/clients#.NET. Accessed: 2020-07-15. 30, 56

[4] Frozenmountain. https://www.frozenmountain.com/. Accessed: 2020-04-30. 6, 56

[40] Bridge design pattern. https://refactoring.guru/design-patterns/bridge. Accessed: 2020-07-15. 31, 33, 52

[41] Webrtc signaling. https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Signaling_and_video_calling. Accessed: 2020-07-20. 36, 37

[42] Oauth 2.0. https://oauth.net/2/. Accessed: 2020-07-20. 17, 22, 34, 35

[43] Liveswitch tokens. https://help.frozenmountain.com/docs/liveswitch/server/advanced-topics#CreatinganAuthServer. Accessed: 2020-07-22. 34, 35

[44] Implicit flow. https://openid.net/specs/openid-connect-core-1_0.html#ImplicitFlowAuth. Accessed: 2020-07-22. xvi, 36

[45] Client credentials flow. https://oauth.net/2/grant-types/client-credentials/. Accessed: 2020-07-22. xvi, 36

[46] Docfx. https://dotnet.github.io/docfx/. Accessed: 2020-07-23. 42

[47] Ldap. https://ldap.com/. Accessed: 2020-07-27. 56

[48] Scrum. https://www.scrum.org/resources/what-is-scrum. Accessed: 2020-07-28. 57

[49] Healthsuite digital platform. https://www.hsdp.io/. Accessed: 2020-08-13. 1

[5] Websync. https://help.frozenmountain.com/docs/websync4. Accessed: 2020-04-30. 6, 7, 37

[50] Philips realtime communications platform. https://share.philips.com/sites/STS020170418141503/architecture/Lists/Asset%20Category1/Category.aspx?ID=56&Source=https%3A%2F%2Fshare%2Ephilips%2Ecom%2Fsites%2FSTS020170418141503%2Farchitecture%2FSiteAssets%2FPages%2FPlatform%2520Details%2Easpx%3FPF%3DHSRA%2520%28CAO%29&ContentTypeId=0x010054F21D110376634C81B77A41A6A257C60072939D0FAFE0D84EA18279A5E38F37 Accessed: 2020-09-14. ix

[6] Webrtc. https://webrtc.org/. Accessed: 2020-04-30. 2, 3, 5

[7] Kurento. https://doc-kurento.readthedocs.io/en/6.13.2/index.html. Accessed: 2020-05-12. 69

[8] Janus. https://janus.conf.meetecho.com/docs/. Accessed: 2020-05-12. 69

[9] Openvidu. https://docs.openvidu.io/en/2.13.0/. Accessed: 2020-05-12. 69

[BCS+10] Paulo Borba, Ana Cavalcanti, Augusto Sampaio, Jim Woodcook, Patrícia Machado, Auri Vincenzi, and José Maldonado. *Testing Techniques in Software Engineering*, volume 6153. 2010. 45

[BJ96] BROOKE and J. SUS : A 'quick and dirty' usability scale. *Usability Evaluation in Industry*, 1996. 48

[BKM09] Aaron Bangor, Philip T. Kortum, and James T. Miller. Determining what individual SUS scores mean: adding an adjective rating scale. *undefined*, 2009. 50

[Dav89] Fred D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly: Management Information Systems*, 13(3):319–339, 9 1989. 48, 49

[EBK+19] Einspieler, Bos, Krieber-Tomantschger, Alvarado, Barbosa, Bertoncelli, Burger, Chorna, Del Secco, DeRegnier, Hüning, Ko, Lucaccioni, Maeda, Marchi, Martín, Morgan, Mutlu, Nogolová, Pansy, Peyton, Pokorny, Prinsloo, Ricci, Saini, Scheuchenegger, Silva, Soloveichick, Spittle, Toldo, Utsch, van Zyl, Viñals, Wang, Yang, Yardımcı-Lokmanoğlu, Cioni, Ferrari, Guzzetta, and Marschik. Cerebral Palsy: Early Markers of Clinical Phenotype and Functional Outcome. *Journal of Clinical Medicine*, 8(10):1616, 10 2019. 9

[ESSD08] Steve Easterbrook, Janice Singer, Margaret Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer London, 2008. 47, 49

[Fin10] Kraig Finstad. The Usability Metric for User Experience. *Interacting with Computers*, 22(5):323–327, 9 2010. 48

[FZB+18] Robert Feldt, Thomas Zimmermann, Gunnar R. Bergersen, Davide Falessi, Andreas Jedlitschka, Natalia Juristo, Jürgen Münch, Markku Oivo, Per Runeson, Martin Shepperd, Dag I.K. Sjøberg, and Burak Turhan. Four commentaries on the use of students and professionals in empirical software engineering experiments, 12 2018. 50

[KLL97] Barbara Kitchenham, Stephen Linkman, and David Law. DESMET: A methodology for evaluating software engineering methods and tools. *Computing and Control Engineering Journal*, 8(3):120–126, 1997. 47, 49

[Kru95] Philippe B. Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, 1995. 25, 28

[LLŠ20] Urška Lah, James R. Lewis, and Boštjan Šumak. Perceived Usability and the Modified Technology Acceptance Model. *International Journal of Human–Computer Interaction*, pages 1–15, 2 2020. xiii, 48, 50, 123

[LUM13] James R. Lewis, Brian S. Utesch, and Deborah E. Maher. UMUX-LITE - When there's no time for the SUS. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 2099–2102, New York, New York, USA, 2013. ACM Press. 48

[Mon20] R. J. H. Montree. Apnea detection and classification in neonates using non-invasive electromyography and video analysis. Master's thesis, Eindhoven University of Technology, the Netherlands, 2020. 1, 10

[NN15] Sam Newman and Sam Newman. *Building microservices : designing fine-grained systems*. O'Reilly Media, 2015. 16, 17

[Proa] Programming in the Large with Design Patterns - Eddie Burris - Google Books. 31

[Prob] Product Security and Services Office. Services Security and Privacy Requirements - Revision 5. Technical report, Philips (Confidential). 17

[SBM+15] Shashank Sharma, Sourya Bhattacharyya, Jayanta Mukherjee, Parimal Kumar Purkait, Arunava Biswas, and Alok Kanti Deb. Automated detection of newborn sleep apnea using video monitoring system. In *ICAPR 2015 - 2015 8th International Conference on Advances in Pattern Recognition*. Institute of Electrical and Electronics Engineers Inc., 2 2015. 1

[VB08] Viswanath Venkatesh and Hillol Bala. Technology Acceptance Model 3 and a Research Agenda on Interventions. *Decision Sciences*, 39(2):273–315, 5 2008. 48

[VD00] Viswanath Venkatesh and Fred D. Davis. Theoretical extension of the Technology Acceptance Model: Four longitudinal field studies. *Management Science*, 46(2):186–204, 2000. 48

[vDWvZ18] Ronald van Driel, Klaas Wijbrans, and Jan van Zoest. CAO - Philips PaaS Reference Architecture Design. Technical report, Philips (Confidential), 2018. 17, 23, 34, 35, 53, 56

About the author

Robin Mennens received his bachelor's degree in Software Science (2016) and his master's degree in Computer Science and Engineering (2018) from Eindhoven University of Technology (The Netherlands). During his master's program, he had an internship and carried out his master's thesis project at ProcessGold in Eindhoven. His master's thesis, "Graph layout stability in process mining," was in the fields of process mining and graph drawing and involved the development and implementation of a graph layout algorithm. Still eager to learn more, he started the Software Technology PDEng program in 2018, also at Eindhoven University of Technology. During the program, he was involved as a project manager, developer, SCRUM master, and product owner in AI, agriculture, and software quality management projects for Philips, Precision Agrotech Center, and ASML.

A WebRTC Providers

During this project, we investigated several webRTC providers and compared them. In this chapter, we describe the webRTC providers that we investigated but did not end up working with. Each of the providers has some kind of limitation, which we describe below.

Kurento [7]
Kurento provides a webRTC media server and a set of client-side APIs to simplify the development of audio/video applications for the web and smartphone platforms. By defining media pipelines, the media server can be configured to handle media elements in a certain order/way. Similar to our use case, they have modules (which are essentially algorithms) that run on the media server (while in our case we want to run the algorithms as webRTC clients). Also, Kurento only provides JavaScript and Java client-side APIs.

OpenVidu[9]
OpenVidu is built on top of Kurento and therefore suffers from the same drawbacks. It aims to further wrap and hide all the low-level technicalities such that users are provided with a simple, effective, and easy to use API.

Janus [8]
Janus is a general-purpose webRTC server written in C. Extra functionality can be added by adding plug-ins to the server. Unfortunately, there are no client-side APIs and we found the documentation to be poor.

Spitfire [10]
Spitfire is a library that wraps the webRTC native code such that .NET applications can take advantage of data channels. Considering our use case, Spitfire did not suffice because we also required audio/video streaming.

WebRTC for the Universal Windows Platform [11]
Microsoft launched this project in an effort to port the webRTC codebase to Universal Windows Platform (UWP). It is open source and available as a NuGet package. On the API level it is very similar to the standard webRTC API, which means that it is still quite low level. Because of this, we preferred other providers such as ICELink and LiveSwitch, which abstract away some of the details.

MixedReality WebRTC Project [12]
The MixedReality webRTC project consists of a set of components designed to help mixed reality app developers integrate peer-to-peer audio, video, and data real-time communication into their application. Like the name suggests, the project mainly focuses on features that enhance collaborative experiences in mixed reality apps. Since we require a more general-purpose webRTC implementation, this option did not really suit our needs.

Vidyo.io [13]
Vidyo.io is a platform as a service (PaaS) that makes it easy for developers to integrate real-time video communication capabilities into their application. It mainly consists of client-side SDKs that provide APIs for integrating such communication capabilities. All of the clients connect to the Vidyo cloud service, which means that there is no on-premise option. For the same reason, we did not use Vidyo.

OpenTok platform [14]
OpenTok is similar to Vidyo. In addition to providing client-side SDKs, it also provides server-side SDKs that give more control over RTC sessions and authentication. Like Vidyo, all clients connect to the cloud (OpenTok Cloud), which means that there is no on premise option.

B SFU and MCU

Figure B.1 visually compares a Peer-to-Peer (P2P), Selective Forwarding Unit (SFU), and Multipoint Control Unit (MCU) setup. While P2P connections are the fastest in terms of transmission speed, they also require quite some CPU power because encoding audio/video is an expensive operation (while decoding is not). A typical Personal Computer (PC), for example, can only manage three P2P connections at the same time. By placing an SFU or MCU between the peers (see Figure B.1), peers only need to encode their audio/video signal once; the SFU/MCU then takes care of the distribution to other peers. A disadvantage of an MCU is that it needs to decrypt, decode, encode, and encrypt each incoming stream (in that order) to merge them into one outgoing stream. This means that an MCU needs a decryption/encryption key, which is less secure than a P2P/SFU connection.

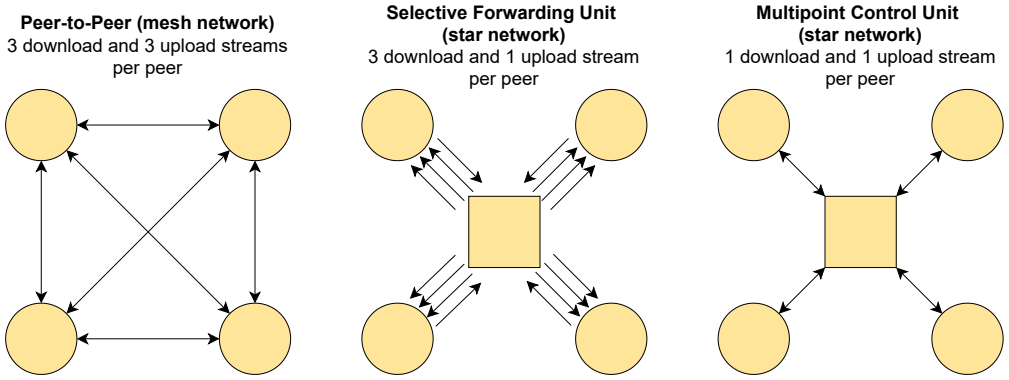


Figure B.1: A visual representation of P2P/SFU/MCU. Circles represent peers while squares represent a server. When using P2P, all peers connect to each other, resulting in a mesh network. By using an SFU/MCU, a star network is created in which all traffic goes via the server. An MCU differs from an SFU in the sense that it merges all incoming streams into a single outgoing stream, which is done per peer.

C Use Case Analysis

During the use case analysis, we explored many different use cases, out of which some were picked to be investigated in more detail (see Section 2.4). In the sections below, we describe the other use cases and explain why we did not investigate them in more detail. Figure C.1 shows the output of one of the brainstorming sessions we had while exploring use cases.

Bilingual conversation
Two actors that speak different languages are having a remote audio/video session. To make sure that the actors can understand each other, the system does the following:

1. Convert the spoken audio into text.
2. Translate the text to the other language.
3. Show the translated text as subtitles.
4. (Optional) synthesize the translated text into audio again.

Before this project, the PR team already implemented a demonstrator for this use case in a project done together with a group of PDEng Software Technology trainees. As a first test for PRAIS, we implemented this use case again to verify the PRAIS functionality. Currently, the involved algorithm is part of the PRAISExamples repository (see Section 4.7).

Multilingual conversation
The same as bilingual conversation but now there are three or more actors that speak two or more different languages. Since this is simply a more complex case of the bilingual conversation, we did not investigate it further.

Condition monitoring
In a scenario where a doctor and a patient have regular remote sessions, the system automatically keeps track of different aspects of these sessions. Think of:

- The emotional state of the patient (can be based on audio and/or video).
- A (translated) transcription of the conversation.
- A (translated) summary of the conversation.
- A history of transcriptions and summaries.
- A sentiment score per participant based on the transcription.
- Live sentiment analysis to give live feedback to the doctor/patient.
- Video snapshots of the conversation (of the patient's face, for example).

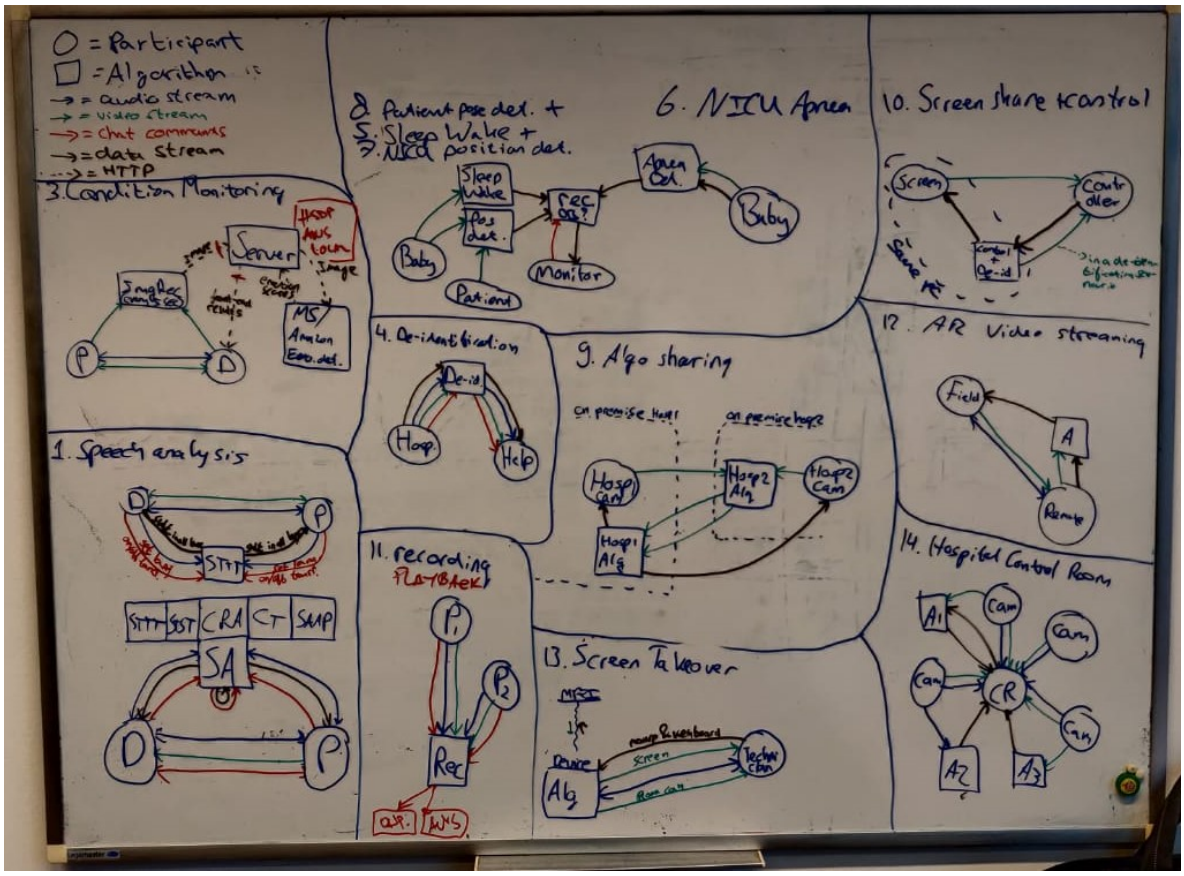


Figure C.1: During use case brainstorming, we would draw how participants and algorithms connect among each other. This helped us understand how use cases differ/overlap and provided insight into what functionality should be part of PRAIS.

- An AI can detect topics of interest/entities during the conversation. If someone, for example, mentions *paracetamol* then the system can provide more information on this medication. This can be done live during conversation or as a summary afterwards.
- Multilingual instant messaging, i.e., translate chat messages.

Before this project, the PR team had already implemented a demonstrator (containing most of the features of this use case) in a project done together with a group of PDEng Software Technology trainees. Therefore, use-case-wise, there was not much value to gain anymore.

De-identification of private data

Whenever video is shared between two or more actors (screen share, for example), the system detects private data in the video and automatically blurs it. The same can be done for audio, i.e., one can replace sensitive data with beeps.

Before this project, the PR team had already implemented (the video part of) this use case in a project done together with a group of PDEng Software Technology trainees. Therefore, use-case-wise, there was not much value to gain anymore.

Neonatal Intensive Care Unit (NICU) sleep-wake detection
With an AI algorithm and a camera that is constantly filming a prematurely born baby, the system can automatically detect whether the baby is awake or sleeping.

The PR team had already managed to prototype this use case in the past. Also, it is very similar to the apnea detection (Section 2.4.2) and pose detection (Section 2.4.1) use cases (for which no prototype existed yet).

Patient pose detection
With a camera that is constantly filming a patient and an AI algorithm, the system is able to detect the position/pose of the patient. This information can be used to detect seizures, for example, i.e., to detect the shaking of the body. Since one of our team members was actively working on neonatal pose detection (which is a similar use case), we preferred that use case instead of this one.

Controlling a remote device
In a screenshare session, one of the clients takes over control of the other's machine. During use case analysis, we did not have a stakeholder who would directly benefit from such a system, which is why we did not investigate it further.

Take over screen of remote machine (that is broken)
By attaching a device that runs an algorithm to a hospital machine (MRI for example), it would be possible to see the MRI screen and even control it. More specifically, the device can read the video signal of the machine and stream it to a remote service engineer. At the same time, the remote service engineer can send mouse/keyboard input to the machine via the attached device. During use case analysis, we did not have a stakeholder who would directly benefit from such a system, which is why we did not investigate it further.

Augmented Reality (AR) video streaming
A field service engineer (FSE) who is unable to solve a problem and who is wearing an AR headset can get remote help from a remote service engineer (RSE). The RSE sees whatever the AR headset is filming and he or she can point to places on the screen. An algorithm then converts this 2d pointer to 3d space and shows it in AR to the FSE. We did not have such an algorithm/AR headset directly available, which is why we did not investigate this use case further.

Hospital control room
In a central hospital location, many RTC streams come in from different cameras/algorithms. This gives the crew watching these streams a means to have an overview of the complete hospital. While hospitals have shown interest in such a system in the past, during use case analysis, we did not have a concrete stakeholder that would directly benefit from such a system, which is why we did not investigate it further.

D Requirements

Table D.1 lists all the functional requirements. The requirements are categorized and priorities follow the MoSCoW model: Must, Should, Could, Would.

ID	Priority	Category	Description
F-13	Must	ICELink /LiveSwitch: Security	Algorithms shall generate tokens locally
F-36	Must	ICELink /LiveSwitch: Security	The system shall uniquely identify applications
F-51	Must	ICELink /LiveSwitch: Security	The system shall uniquely identify conferences
F-52	Must	ICELink /LiveSwitch: Security	The system shall uniquely identify peers
F-53	Must	ICELink /LiveSwitch: Security	Peers shall authenticate themselves using tokens before joining a conference
F-54	Must	ICELink /LiveSwitch: Security	A conference shall uniquely map to an application
F-57	Must	ICELink /LiveSwitch: Security	An algorithm instance shall be present in at most one conference
F-148	Must	ICELink /LiveSwitch: Security	The system shall encrypt streams
F-149	Must	ICELink /LiveSwitch: Security	Tokens shall have an expiration date

F-9	Must	ICELink: Algorithm Media Sources	An ICELink algorithm shall use a custom media source to send audio/video to other peers
F-111	Must	ICELink: Algorithm Messaging	An ICELink algorithm shall send messages to specific peers in the same conference.
F-112	Must	ICELink: Algorithm Messaging	An ICELink algorithm shall receive messages from specific peers in the same conference.
F-116	Must	ICELink: Basic Conference Management	An ICELink algorithm shall join a conference
F-117	Must	ICELink: Basic Conference Management	An ICELink algorithm shall leave a conference
F-118	Must	ICELink: Basic Conference Management	An ICELink algorithm shall terminate itself
F-127	Must	ICELink: Javascript RTC API	The Javascript RTC API shall use manual signaling to settle peer-to-peer connections
F-119	Must	ICELink: Peer-to-peer Connections	An ICELink algorithm shall send audio peer-to-peer to specific peers over audio stream when specified by the user
F-120	Must	ICELink: Peer-to-peer Connections	An ICELink algorithm shall send video peer-to-peer to specific peers over video stream when specified by the user
F-121	Must	ICELink: Peer-to-peer Connections	An ICELink algorithm shall send data peer-to-peer to specific peers over data stream when specified by the user
F-122	Must	ICELink: Peer-to-peer Connections	An ICELink algorithm shall disconnect from specific peers over any stream when specified by the user
F-123	Must	ICELink: Peer-to-peer Connections	An ICELink algorithm shall receive audio peer-to-peer from specific peers over audio stream when specified by the user
F-124	Must	ICELink: Peer-to-peer Connections	An ICELink algorithm shall receive video peer-to-peer from specific peers over video stream when specified by the user

F-125	Must	ICELink: Peer-to-peer Connections	An ICELink algorithm shall receive data peer-to-peer from specific peers over data stream when specified by the user
F-126	Must	ICELink: Peer-to-peer Connections	An ICELink algorithm shall use manual signaling to set up a peer to peer connection
F-150	Must	ICELink: Peer-to-peer Connections	Manual Signaling shall integrate with the existing back-end WebSync call data record logging functionality
F-79	Must	LiveSwitch: Algorithm Media Sources	A LiveSwitch algorithm shall use a custom media source to send audio/video to other peers
F-69	Must	LiveSwitch: Algorithm Messaging	A LiveSwitch algorithm shall send messages to specific peers in the same conference.
F-83	Must	LiveSwitch: Algorithm Messaging	A LiveSwitch algorithm shall receive messages from specific peers in the same conference.
F-34	Must	LiveSwitch: Basic Conference Management	A LiveSwitch algorithm shall join a conference
F-35	Must	LiveSwitch: Basic Conference Management	A LiveSwitch algorithm shall leave a conference
F-38	Must	LiveSwitch: Basic Conference Management	A LiveSwitch algorithm shall terminate itself
F-1	Must	LiveSwitch: Peer-to-peer Connections	A LiveSwitch algorithm shall send audio peer-to-peer to specific peers over audio stream when specified by the user
F-2	Must	LiveSwitch: Peer-to-peer Connections	A LiveSwitch algorithm shall send video peer-to-peer to specific peers over video stream when specified by the user
F-3	Must	LiveSwitch: Peer-to-peer Connections	A LiveSwitch algorithm shall send data peer-to-peer to specific peers over data stream when specified by the user
F-4	Must	LiveSwitch: Peer-to-peer Connections	A LiveSwitch algorithm shall disconnect from specific peers over any stream when specified by the user

F-70	Must	LiveSwitch: Peer-to-peer Connections	A LiveSwitch algorithm shall receive audio peer-to-peer from specific peers over audio stream when specified by the user
F-71	Must	LiveSwitch: Peer-to-peer Connections	A LiveSwitch algorithm shall receive video peer-to-peer from specific peers over video stream when specified by the user
F-72	Must	LiveSwitch: Peer-to-peer Connections	A LiveSwitch algorithm shall receive data peer-to-peer from specific peers over data stream when specified by the user
F-65	Should	ICELink: Algorithm Media Sources	An ICELink algorithm shall use screen capture as a media source to send audio/video to other peers
F-110	Should	ICELink: Algorithm Media Sources	An ICELink algorithm shall use a camera/microphone (connected to the device the algorithm runs on) as a media source to send audio/video to other peers
F-113	Should	ICELink: Algorithm Multi Connections	An ICELink algorithm shall connect over multiple video streams to the same peer when specified by the user
F-114	Should	ICELink: Algorithm Multi Connections	An ICELink algorithm shall connect over multiple audio streams to the same peer when specified by the user
F-115	Should	ICELink: Algorithm Multi Connections	An ICELink algorithm shall connect over multiple data streams to the same peer when specified by the user
F-67	Should	ICELink: Recording	An ICELink algorithm shall record incoming audio from peers specified by the user
F-68	Should	ICELink: Recording	When an ICELink algorithm records audio and video from a peer, then the algorithm shall synchronize the audio and video
F-131	Should	ICELink: Recording	An ICELink algorithm shall record incoming video from peers specified by the user
F-132	Should	ICELink: Recording	An ICELink algorithm shall store a recording as mkv format
F-133	Should	ICELink: Recording	An ICELink algorithm shall store a recording with a filename specified by the user
F-134	Should	ICELink: Recording	An ICELink algorithm shall store a recording in a folder specified by the user

F-135	Should	ICELink: Recording	An ICELink algorithm shall store an absolute millisecond accurate frame receival timestamp time for every recorded video frame
F-80	Should	LiveSwitch: Algorithm Media Sources	A LiveSwitch algorithm shall use a camera/microphone (connected to the device the algorithm runs on) as a media source to send audio/video to other peers
F-81	Should	LiveSwitch: Algorithm Media Sources	A LiveSwitch algorithm shall use screen capture as a media source to send audio/video to other peers
F-5	Should	LiveSwitch: Algorithm Multi Connections	A LiveSwitch algorithm shall connect over multiple data streams to the same peer when specified by the user
F-6	Should	LiveSwitch: Algorithm Multi Connections	A LiveSwitch algorithm shall connect over multiple video streams to the same peer when specified by the user
F-7	Should	LiveSwitch: Algorithm Multi Connections	A LiveSwitch algorithm shall connect over multiple audio streams to the same peer when specified by the user
F-136	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall record a number of video chunks specified by the user
F-137	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall record video chunks of a duration specified by the user
F-138	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall authenticate a user before allowing connections to incubators
F-139	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall only record incubators at MMC that have consent
F-140	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall terminate a connection to an incubator when the nurse disables the same incubator
F-141	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall show the video of a connected incubator
F-142	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall connect to exactly one incubator at the same time
F-143	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall stop showing the video of a connect incubator when specified by the user via the UI

F-144	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall configure the file name of the recording based on what the user specified via the UI
F-145	Should	PRAIS Recorder Application	The PRAIS Recorder Application shall store the recorded video in a local folder specified by the user
F-39	Could	Advanced Conference Management	An algorithm shall register with the system before joining a conference
F-40	Could	Advanced Conference Management	The system shall keep track of all registered algorithms
F-41	Could	Advanced Conference Management	An algorithm shall be identified by name within a solution provider
F-42	Could	Advanced Conference Management	The system shall instruct an algorithm to join a specific conference
F-43	Could	Advanced Conference Management	When a peer is in a conference, then the peer shall add algorithms to the same conference
F-66	Could	Advanced Conference Management	When a peer is in a conference, then the peer shall remove algorithms from the same conference
F-109	Could	Advanced Conference Management	The system shall instruct an algorithm to leave a specific conference
F-10	Could	Back-end Controlled Security	Peers shall be authenticated by solution providers using open ID connect
F-11	Could	Back-end Controlled Security	The system shall generate tokens and provide them to authenticated peers
F-37	Could	Back-end Controlled Security	An application shall uniquely map to a solution provider
F-50	Could	Back-end Controlled Security	The system shall uniquely identify solution providers
F-55	Could	Back-end Controlled Security	An algorithm shall uniquely map to a solution provider
F-56	Could	Back-end Controlled Security	An algorithm shall join a conference if the algorithm and conference belong to the same solution provider

F-58	Could	Back-end Controlled Security	The system shall encrypt all security related messages
F-8	Could	ICELink: Algorithm Media Sources	An ICELink algorithm shall use a local file as a media source to send audio/video to other peers
F-146	Could	ICELink: Recording	An ICELink algorithm shall store an absolute millisecond accurate frame created timestamp time for every recorded video frame
F-147	Could	ICELink: Recording	An ICELink algorithm shall record all video frames at the same resolution
F-82	Could	LiveSwitch: Algorithm Media Sources	A LiveSwitch algorithm shall use a local file as a media source to send audio/video to other peers
F-128	Could	LiveSwitch: Recording	A LiveSwitch algorithm shall record incoming audio from peers specified by the user
F-129	Could	LiveSwitch: Recording	When a LiveSwitch algorithm records audio and video from a peer, then the algorithm shall synchronize the audio and video
F-130	Could	LiveSwitch: Recording	An ICELink algorithm shall record incoming video from peers specified by the user
F-63	Could	User Interface Streaming	An algorithm shall send a user interface to other peers
F-64	Could	User Interface Streaming	A peer shall display a user interface that was sent by another peer
F-44	Won't	Algorithm Deployment	The system shall run algorithms as docker containers
F-45	Won't	Algorithm Deployment	The system shall orchestrate the docker containers
F-46	Won't	Algorithm Deployment	The system shall load balance the algorithms
F-14	Won't	Algorithm Media Sources	When an algorithm uses a camera as media source and when there are multiple cameras connected to the device the algorithm runs on, then the algorithm shall select which camera shall be used as media source.
F-15	Won't	Algorithm Media Sources	When an algorithm uses a microphone as media source and when there are multiple microphones connected to the device the algorithm runs on, then the algorithm shall select which microphone shall be used as media source.

F-24	Won't	Audio Stream Parameters	An algorithm shall specify the required sample rate of the audio stream
F-25	Won't	Audio Stream Parameters	An algorithm shall specify the desired sample rate of the audio stream
F-26	Won't	Audio Stream Parameters	An algorithm shall specify the required sample size of the audio stream
F-27	Won't	Audio Stream Parameters	An algorithm shall specify the desired sample size of the audio stream
F-28	Won't	Audio Stream Parameters	An algorithm shall specify the required channelCount of the audio stream
F-29	Won't	Audio Stream Parameters	An algorithm shall specify the desired channelCount of the audio stream
F-30	Won't	Audio Stream Parameters	An algorithm shall specify the required audio bandwidth of the audio stream
F-31	Won't	Audio Stream Parameters	An algorithm shall specify the desired audio bandwidth of the audio stream
F-32	Won't	Audio Stream Parameters	An algorithm shall check whether required constraints are supported by the device it runs on
F-33	Won't	Audio Stream Parameters	An algorithm shall check whether required constraints are supported by the peer it connects to
F-94	Won't	LiveSwitch: Basic Conference Management	A LiveSwitch participant shall join a conference
F-95	Won't	LiveSwitch: Basic Conference Management	A LiveSwitch participant shall leave a conference
F-89	Won't	LiveSwitch: Participant Media Sources	When A LiveSwitch participant uses a camera as media source and when there are multiple cameras connected to the device the participant runs on, then the participant shall select which camera shall be used as media source.
F-90	Won't	LiveSwitch: Participant Media Sources	When A LiveSwitch participant uses a microphone as media source and when there are multiple microphones connected to the device the participant runs on, then the participant shall select which microphone shall be used as media source.
F-91	Won't	LiveSwitch: Participant Media Sources	A LiveSwitch participant shall use a local file as a media source to send audio/video to other peers

F-92	Won't	LiveSwitch: Participant Media Sources	A LiveSwitch participant shall use a camera/microphone (connected to the device the participant runs on) as a media source to send audio/video to other peers
F-93	Won't	LiveSwitch: Participant Media Sources	A LiveSwitch participant shall use screen capture as a media source to send audio/video to other peers
F-84	Won't	LiveSwitch: Participant Messaging	A LiveSwitch participant shall receive messages from specific peers in the same conference.
F-85	Won't	LiveSwitch: Participant Messaging	A LiveSwitch participant shall send messages to specific peers in the same conference.
F-86	Won't	LiveSwitch: Participant Multi Connections	A LiveSwitch participant shall connect over multiple data streams to the same peer in the same conference
F-87	Won't	LiveSwitch: Participant Multi Connections	A LiveSwitch participant shall connect over multiple video streams to the same peer in the same conference
F-88	Won't	LiveSwitch: Participant Multi Connections	A LiveSwitch participant shall connect over multiple audio streams to the same peer in the same conference
F-102	Won't	LiveSwitch: Peer-to-peer Connections	A LiveSwitch participant shall send audio peer-to-peer to specific peers over audio stream
F-103	Won't	LiveSwitch: Peer-to-peer Connections	A LiveSwitch participant shall send video peer-to-peer to specific peers over video stream
F-104	Won't	LiveSwitch: Peer-to-peer Connections	A LiveSwitch participant shall send data peer-to-peer to specific peers over data stream
F-105	Won't	LiveSwitch: Peer-to-peer Connections	A LiveSwitch participant shall disconnect from specific peers over any stream when specified by the user
F-106	Won't	LiveSwitch: Peer-to-peer Connections	A LiveSwitch participant shall receive audio peer-to-peer from specific peers over audio stream
F-107	Won't	LiveSwitch: Peer-to-peer Connections	A LiveSwitch participant shall receive video peer-to-peer from specific peers over video stream

F-108	Won't	LiveSwitch: Peer-to-peer Connections	A LiveSwitch participant shall receive data peer-to-peer from specific peers over data stream
F-73	Won't	LiveSwitch: SFU Connections	A LiveSwitch algorithm shall send audio via SFU to specific peers over audio stream when specified by the user
F-74	Won't	LiveSwitch: SFU Connections	A LiveSwitch algorithm shall send video via SFU to specific peers over video stream when specified by the user
F-75	Won't	LiveSwitch: SFU Connections	A LiveSwitch algorithm shall send data via SFU to specific peers over data stream when specified by the user
F-76	Won't	LiveSwitch: SFU Connections	A LiveSwitch algorithm shall receive audio via SFU from specific peers over audio stream when specified by the user
F-77	Won't	LiveSwitch: SFU Connections	A LiveSwitch algorithm shall receive video via SFU from specific peers over video stream when specified by the user
F-78	Won't	LiveSwitch: SFU Connections	A LiveSwitch algorithm shall receive data via SFU from specific peers over data stream when specified by the user
F-96	Won't	LiveSwitch: SFU Connections	A LiveSwitch participant shall send audio via SFU to specific peers over audio stream
F-97	Won't	LiveSwitch: SFU Connections	A LiveSwitch participant shall send video via SFU to specific peers over video stream
F-98	Won't	LiveSwitch: SFU Connections	A LiveSwitch participant shall send data via SFU to specific peers over data stream
F-99	Won't	LiveSwitch: SFU Connections	A LiveSwitch participant shall receive audio via SFU from specific peers over audio stream
F-100	Won't	LiveSwitch: SFU Connections	A LiveSwitch participant shall receive video via SFU from specific peers over video stream
F-101	Won't	LiveSwitch: SFU Connections	A LiveSwitch participant shall receive data via SFU from specific peers over data stream

F-59	Won't	Monitoring	The system shall have a central log store
F-60	Won't	Monitoring	The system shall log separately for every solution provider
F-61	Won't	Monitoring	The system shall display the solution provider's logs to the solution provider
F-62	Won't	Monitoring	A peer shall send log entries to the central log store
F-47	Won't	Peer Async Communication	A peer shall publish events to the conference via a message broker
F-48	Won't	Peer Async Communication	A peer shall subscribe to events that are published by peers in the same conference
F-49	Won't	Peer Async Communication	The system shall use a centralized message broker
F-12	Won't	Security	A LiveSwitch participant shall generate tokens locally
F-16	Won't	Video Stream Parameters	An algorithm shall specify the required aspect ratio of the video stream
F-17	Won't	Video Stream Parameters	An algorithm shall specify the desired aspect ratio of the video stream
F-18	Won't	Video Stream Parameters	An algorithm shall specify the required resolution (width,height) of the video stream
F-19	Won't	Video Stream Parameters	An algorithm shall specify the desired resolution (width,height) of the video stream
F-20	Won't	Video Stream Parameters	An algorithm shall specify the required framerate of the video stream
F-21	Won't	Video Stream Parameters	An algorithm shall specify the desired framerate of the video stream
F-22	Won't	Video Stream Parameters	An algorithm shall specify the required video bandwidth of the video stream
F-23	Won't	Video Stream Parameters	An algorithm shall specify the desired video bandwidth of the video stream

Table D.1: A list of all the functional requirements. The requirements are categorized and priorities follow the MoSCoW model: Must, Should, Could, Would.

E SEP project description

Philips Remote AI Streaming (PRAIS)

SEP project description

Eindhoven University of Technology

Philips Research

Mennens, R.J.P.

16 April, 2020
Version: 1.0

Table of Contents

Context

Philips aims to rapidly transform from a company that sells products to a company that co-creates strategic partnerships with customers and delivers solutions and services over a long term. By standardizing and optimizing the building blocks of healthcare – hardware, software, and services – Philips aims to connect consumers to their caregivers and enable health systems to deliver better outcomes at lower cost.

Providing services involves multiple touchpoints with customers. For efficiency, some of these interactions can be done remotely. Additionally, part of providing a service is to help an organization become more efficient and to facilitate remote collaboration inside the hospital and beyond. Overall, Philips sees a wide variety of opportunities for remote collaboration throughout Philips businesses and is currently implementing the Philips Remote AI Streaming (PRAIS) platform.

In an academic hospital, in addition to the standard medical care, the staff is also involved in medical research. In the Netherlands, for example, we have academic hospitals in Amsterdam, Utrecht, Maastricht, and other cities. With the rise of artificial intelligence (AI), such hospitals are focusing more and more on developing AI that can be used for medical purposes. In this project, we shall, in particular, focus on Neonatal Intensive Care Unit (NICU) AI algorithms. Like the name suggests, a NICU is an intensive care unit that is typically used for prematurely born babies. The parents of such babies cannot stay 24/7 with their baby, which makes it emotionally very tough. Therefore, there is ongoing research into placing a camera in a NICU, allowing the parents to view their baby remotely (see Figure 1). Furthermore, the video and other sensory data can be used for AI use cases.

Figure 1 - A camera in a NICU enables remote viewing of the baby.

Collaboration between the academic hospitals is significant, and we see an increasing need to safely share AI algorithms between the hospitals. This means that aspects such as privacy and security, which are very important in the healthcare domain, need to be taken care of. In this project, we shall explore the sharing of AI algorithms between hospitals and investigate security, privacy, and visualization aspects.

Assignment description

Your group represents two academic hospitals that want to share NICU AI algorithms with each other. Academic hospital A developed a pose detection algorithm for neonates that detects the skeletal structure of the baby (see Figure 2). AI pose detection enables many interesting use cases. For example, by analyzing the movement patterns of a baby, a doctor can assess whether there is brain damage. Academic hospital B has developed an apnea detection algorithm that, given video and sensory input, detects apnea and its type (central, obstructed, mixed). By using AI, fewer sensors are required to detect apnea, the monitorability of the babies is improved, and in the future, apnea prediction may even be possible.

Figure 2 - Baby pose detected

Overall, the project is split into two phases. At first, there is a “feature-first” phase, which focuses mostly on development and integration with PRAIS. The second phase, “architecture-first”, is more explorative and focuses on different ways to implement certain features (see below).

Feature-first

In this project, we do not want to focus on algorithm development but rather on building applications (using the PRAIS platform) that use the algorithms. Therefore, we shall provide both algorithms. It is up to you to integrate the algorithms with PRAIS such that they can be used remotely in a streaming fashion. Figure 3 shows an overview of the complete use case. In both hospitals, there are doctors who want to see the results of the algorithms, and there are NICUs that provide the video (and sensory) inputs for the algorithms. Since we cannot provide you with a NICU camera, we leave it up to you to simulate one. You can either use your own web cam or you can consider implementing functionality for streaming a video file. Regarding the data, we shall provide you with a CSV file that contains the sensory data.

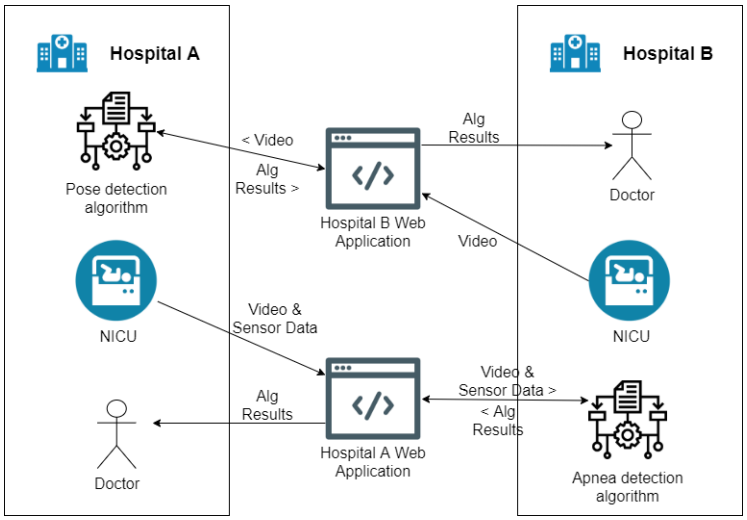


Figure 3 - Use case overview

Architecture-first

After implementing the setup as shown in Figure 3, we want to investigate different options (and implement one of them) regarding three issues we identified.

The first issue is related to security and shall be implemented in hospital B. With the current implementation, basically anyone can use their web application, which is of course not what we want. Therefore, at Hospital B, they decided to add an additional layer of security, i.e., participants and algorithms first need to authenticate themselves before they can join a conference. To this end, we want to explore how open ID connect should be used in combination with PRAIS.

The second issue is related to the visualization of the apnea detection algorithm results. In the current implementation, the Hospital A Web Application developers depend on the apnea detection algorithm in the sense that they need to know the exact output format in order to properly visualize the results. We want to remove this dependency by generating the visualization (code) at the algorithm side. Hospital A Web Application would then only require a placeholder/container for the visualization.

Finally, the CSV file that we provided you contains fake data. The real data is stored in a database (also CSV file). Since this is data about actual patients, it is very privacy sensitive. More specifically, the data should never be stored anywhere else except for the database. So, instead of using the CSV that we provided, we want to investigate whether it is possible for the NICU in Hospital A to stream the real data directly from the database to the apnea detection algorithm. By doing this, the data is only stored in-memory and not copied to disk.

Figure 4 visually shows the use case with the extra features described above. Red indicates the changed/new elements.

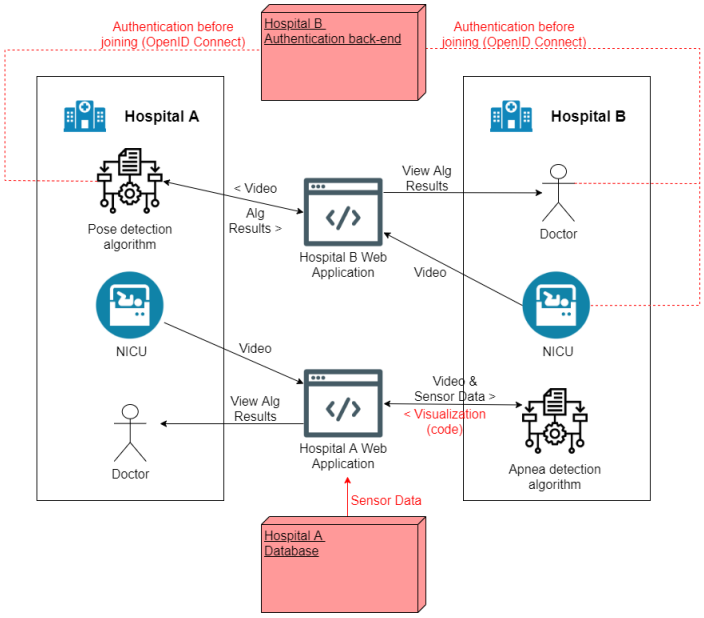


Figure 4 - Use case with extra features. Red indicates changes compared to Figure 3

Goals of the Project

- Verify the usability and stability of PRAIS.
- Explore the options regarding combining OpenID Connect and PRAIS (capture learnings).
- Explore the options regarding user interface streaming (capture learnings).
- Build demonstrators.

Deliverables

The following elements are of interest to us:

- Report, including:
  - Architectural Design
  - Learnings and Recommendations
  - Demo Script
- Demo(s)

We are aware that you already have to write a lot of documentation for SEP itself. So, delivering a short readme that describes where we can find these topics is fine.

Technical Guidance

All important concepts are explained in the PRAIS conceptual documentation. In short, PRAIS is a platform that allows developers to easily build applications that require real-time audio/video/data streaming functionality. At this point, it only contains a C# SDK which you will have to use to integrate the provided algorithms (note that the algorithm we provide are written in Python, so you will have to find a way to run Python inside/from C#). In the future, we also envision having a JavaScript SDK such that it becomes

easier to add participants to conferences via a browser. Since we do not have this yet, for this project, we shall provide you with two JavaScript applications (one for each hospital) that directly use LiveSwitch (the C# SDK also uses LiveSwitch under the hood). You will have to extend/adapt these applications such that they fit better in the use cases. More specifically, when a doctor uses the application, then he/she should be able to see the algorithm results and when a NICU connects to the application, then it should simply send its video (and sensory data) to the algorithm.

We will provide you with CSV files that contain the sensory data required for the apnea detection algorithm. Unfortunately, we do not have a NICU camera available. Therefore, you can simply simulate a NICU camera (how, is up to you). As stated before, the focus of this project is not on the algorithms themselves but on the integration with PRAIS. So, it does not matter too much if the inputs to the algorithms are not perfect, as long as there is some output that we can visualize/use.

Required software

- Visual Studio community edition 2019
- PRAIS installation instructions, examples, conceptual and API documentation can be found here: https://healthrtc.org/docs/index.html
- We prepared two client-side browser applications that use LiveSwitch such that they work with PRAIS. You should use these applications as a basis and extend/modify them where needed. Both applications are hosted using GitHub pages at:
  HospitalA: XXX
  HospitalB: XXX
  Please send Robin your GitHub email addresses such that he can add you to the repository.

We already created two applications (and their secrets) in the PRAIS back-end for you. The two client-side applications are already configured with these values:
ApplicationId: XXX
Secret: XXX

ApplicatoinId: XXX
Secret: XXX

Required hardware

The required hardware consists of bring-your-own-device hardware such as laptops and smart phones. The built-in cameras and microphones on your devices should be sufficient. All the envisioned client-side software is compatible on both PC and mobile phone. Finally, you will need a Windows machine to run the algorithms (the PRAIS C# SDK runs on .NET Framework).

Required expertise

On a non-technical level:

- Cross discipline, with a common denominator in Software technology.
- AI, audio-video/multimedia communications
- Requirements collection
- Systems engineering, requirements management
- Prototyping, demo building

On a technical level, we recommend having a look at the following:

- PRAIS documentation
- LiveSwitch documentation
- Open ID connect
- Running Python in/from C#

Contact Information

Several people are involved on the end of the client, see Table 1. To keep things simple for you, Robin is your main customer and point of contact. We prefer communication via email or Microsoft teams. For short and quick questions, feel free to add us to a WhatsApp group.

Table 1 Philips Research stakeholders

Name	**Project Role (Official role)**	**E-mail**	**Phone**
Robin Mennens	Project customer (PDEng trainee Software Technology)	XXX	XXX
Marcel Quist	Project support (Concept Business Architect)	XXX	XXX
Zoran Stankovic	Project support (Software Architect)	XXX	XXX
Arjan Draisma	Project support (Software developer)	XXX	XXX

F PRAIS Recorder Application

The User Interface (UI) of the PRAIS Recorder Application is shown in Figure F.1. The four red rectangles indicate the following parts of the UI:

1. A user first needs to log in before any functionality of the application becomes available. When the PRAIS Recorder Application is started, only the UI elements in rectangle 1 are enabled.
2. Provides an overview of all NICUs that have informed consent, i.e., those incubators that may be connected to. The *Refresh* button can be used to refresh the list of NICUs while the *Connect to incubator* button is used to connect to one of the incubators.
3. After connecting to a NICU, the live camera feed is shown. The *Show video* checkbox can be used to toggle the showing of the camera feed.
4. Allows the user to configure different recording parameters such as: *Recording Folder*, *Recording File Name*, *Recording Chunk Duration*, and *Number of Chunks* to be recorded.

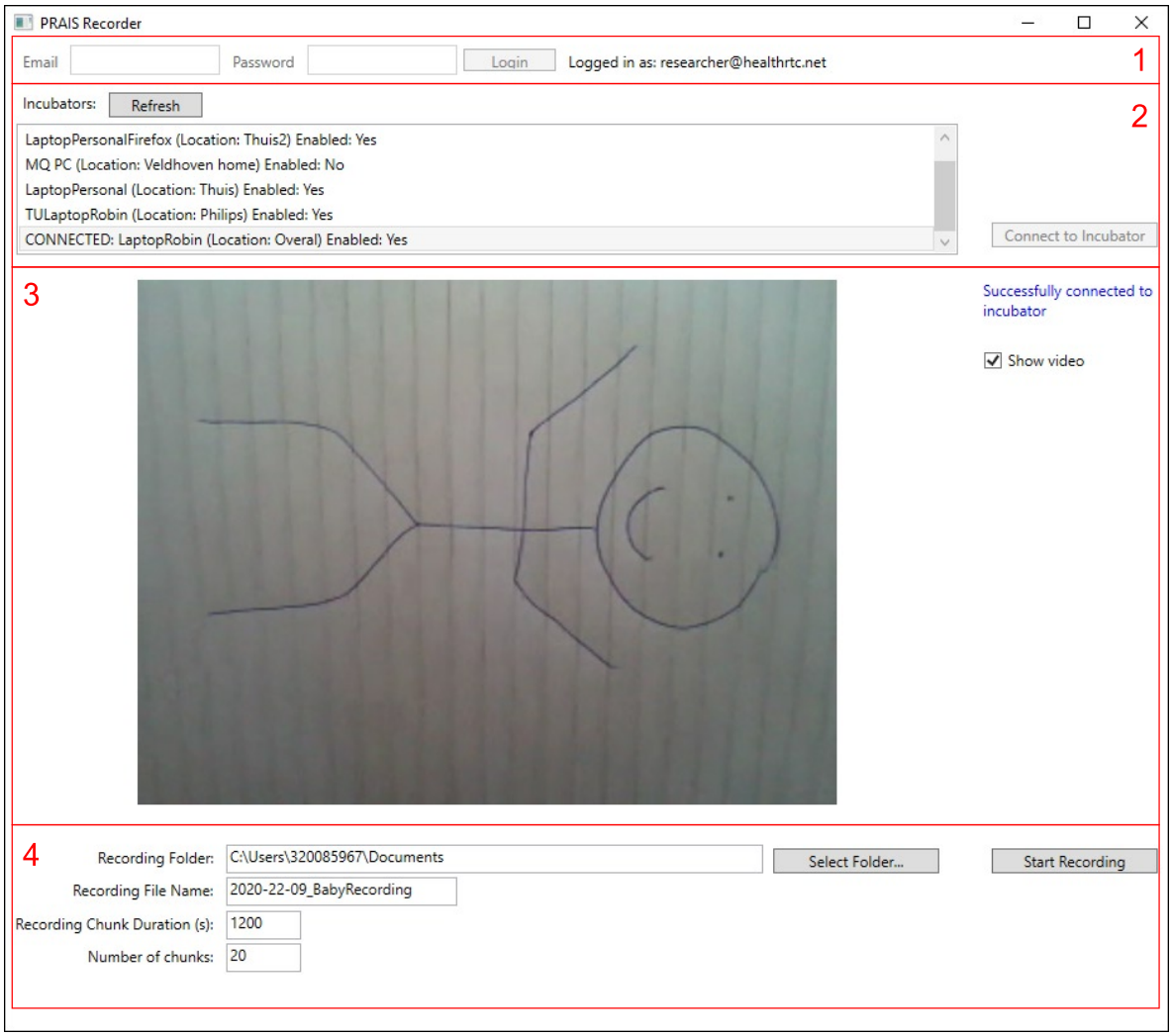


Figure F.1: The user interface of the PRAIS Recorder Application. The red rectangles indicate four parts of the user interface that each have their own purpose.

G PRAIS Documentation

Below, to show what the documentation looks like, we include some PRAIS conceptual and API documentation. The complete documentation can be found online [29].

The PRAIS platform

The Philips Remote AI Streaming (PRAIS) platform is a platform that allows developers to easily build applications that require real-time audio/video/data streaming functionality.

The platform consists of:

- A C# .Net Framework SDK (called the PRAIS C# API) that allows you to easily connect your (AI) algorithms to other peers.
- The Javascript RTC API that allows you to easily integrate the streaming functionality in your browser applications.
- (not yet implemented) A management console that gives you control over and insight in your applications.

The documentation of the Javascript RTC API can be found here (https://healthrtc.net/philipsrtcapibackup/#/) and is not further discussed in this documentation. Important to know is that the Javascript RTC API uses the ICELink webRTC provider. This means that you should also use the ICELink webRTC provider in the PRAIS C# API.

C# API Prerequisites

Before you begin, ensure you have met the following requirements:

- You have installed the latest version of .NET Framework
- You have a Windows machine

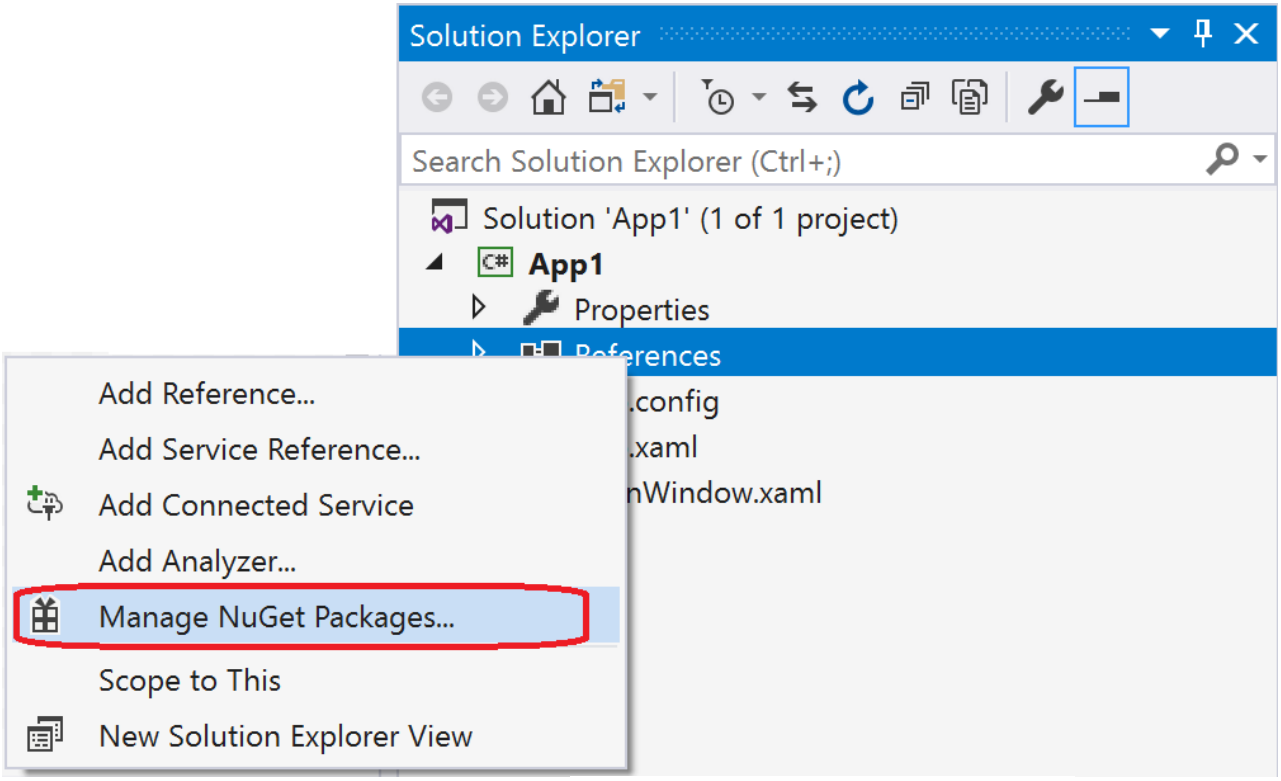
Installing the PRAIS C# API

Prerequisites:

- You are using Visual Studio 2019 (any edition).
- You already have a C# project.
- You already have the PRAIS C# NuGet package. The package is located in the PRAIS examples repository, to which you get access via us.

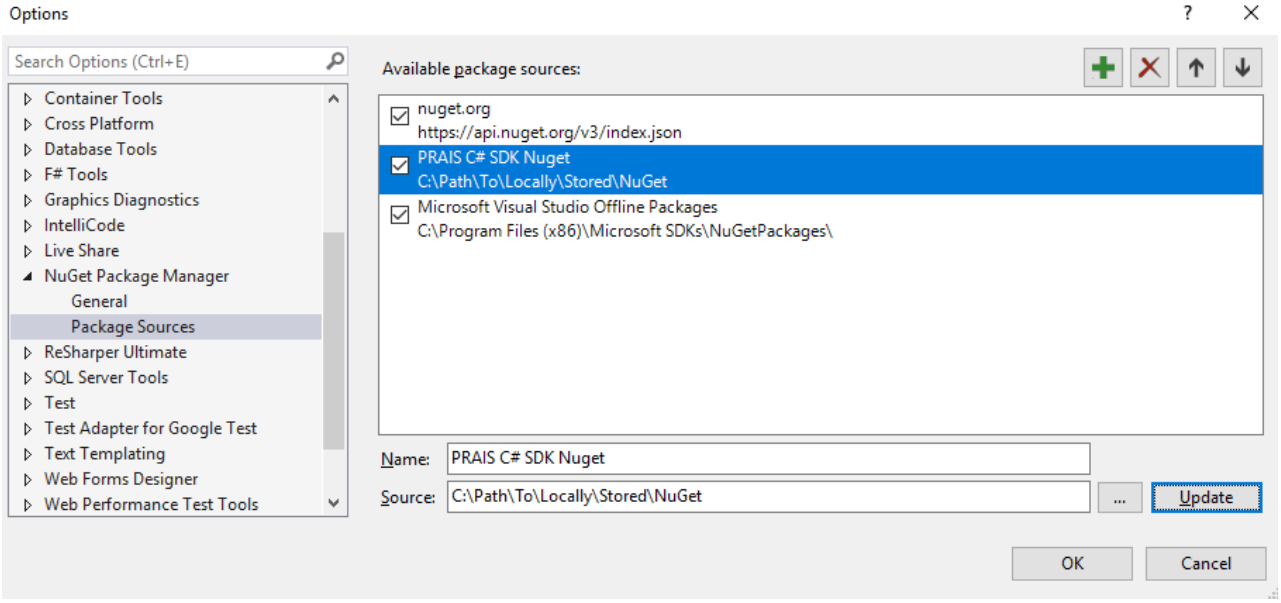
The PRAIS C# API is distributed as a NuGet package (https://docs.microsoft.com/en-us/nuget/what-is-nuget). All you have to do is install the NuGet package in your project:

1. In Solution Explorer, right-click References and choose Manage NuGet Packages.

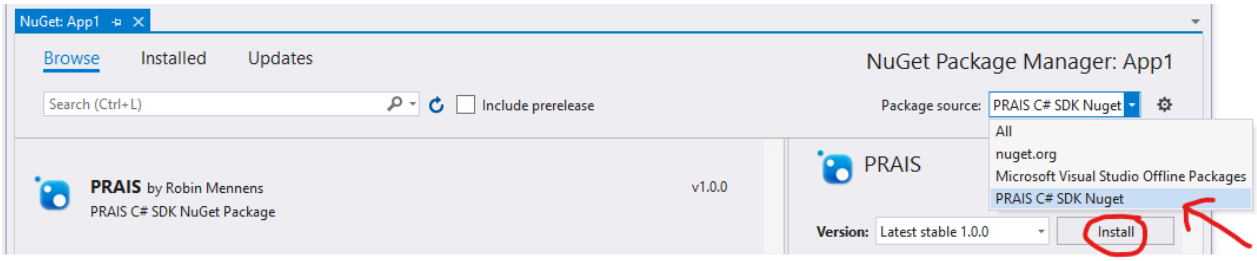


2. Click the gear icon next to the Package Source.

3. Click on the + icon to add a new NuGet package source. Give it an appropriate Name, set the Source to the folder location where you stored the PRAIS C# NuGet package, and click OK.



4. Change the Package Source to the source you just created. The PRAIS C# API Nuget package should now be listed. Click on it and then click on Install.



Using the PRAIS C# API

After installation, add the PRAIS using directives to the C# code files in which PRAIS is used.

```
using PRAIS;
using PRAIS.EventModels;
```

- See the conceptual documentation (articles/conceptual.html) for explanations about how PRAIS works conceptually.
- See the examples (articles/examples.html) page for example usages of PRAIS.
- See the API documentation (api/PRAIS.html).

License

Class AlgorithmImpl

The main point of algorithm functionality. Do not use this class directly. Instead, extend AlgorithmBase to access all functionality.

Inheritance

↳ System.Object
  ↳ AlgorithmImpl
    ↳ AlgorithmBase (AlgorithmAPI.AlgorithmBase.html)

Implements

System.IDisposable

Inherited Members

System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.ToString()

Namespace: AlgorithmAPI (AlgorithmAPI.html)

Assembly: AlgorithmCore.dll

Syntax

```
public abstract class AlgorithmImpl : IDisposable
```

Properties

ApplicationId

Gets the application identifier.

Declaration

```
public string ApplicationId { get; }
```

Property Value

Type	Description
System.String	The application identifier.

ConferenceId

Gets the conference identifier.

Declaration

```
public string ConferenceId { get; }
```

Property Value

Type	Description
System.String	The conference identifier determines which conference shall be joined.

ConferenceState

The registration state of the algorithm in the conference.

Declaration

```
public ConferenceState ConferenceState { get; }
```

Property Value

Type	Description
ConferenceState (AlgorithmAPI.ConferenceState.html)	The state of registration.

DisplayName

Gets the display name of the Algorithm.

Declaration

```
public string DisplayName { get; }
```

Property Value

Type	Description
System.String	A human-readable display name.

Id

Gets the identifier of the Algorithm.

Declaration

```
public string Id { get; }
```

Property Value

Type	Description
System.String	A unique identifier.

LiveSwitchGatewayUrl

Gets the LiveSwitch gateway URL.

Declaration

```
public string LiveSwitchGatewayUrl { get; }
```

Property Value

Type	Description
System.String	The URL at which the LiveSwitch gateway is present, only used when the LiveSwitch implementor is used.

Peers

Gets the peers currently present in the conference.

Declaration

```
public ReadOnlyDictionary<string, Peer> Peers { get; }
```

Property Value

Type	Description
System.Collections.ObjectModel.ReadOnlyDictionary<System.String, Peer (AlgorithmAPI.Peer.html)>	The peers.

Roles

Gets the roles of this Algorithm.

Declaration

```
public string[] Roles { get; }
```

Property Value

Type	Description
System.String[]	

Secret

Gets the secret that is required to generate tokens for joining a conference of an application.

Declaration

```
public string Secret { get; }
```

Property Value

Type	Description
System.String	The string representing the secret.

Remarks

Tokens should be generated by a secure server that has authenticated your algorithm's identity and is authorized to allow you to register with the LiveSwitch server. It is strongly recommended to not store your Secret here but to store it securely with you authentication server.

Tag

Gets the tag of the Algorithm.

Declaration

```
public string Tag { get; }
```

Property Value

Type	Description
System.String	An (optional) tag that can be used to identify the Algorithm.

Methods

Dispose()

Leaves the conference asynchronous, and then removes the algorithm from memory.

Declaration

```
public void Dispose()
```

DoGetConferenceToken(String)

Generates a token for a conference with `conferenceId`.

Declaration

```
public virtual string DoGetConferenceToken(string conferenceId = null)
```

Parameters

Type	Name	Description
System.String	*conferenceId*	(optional) The conference identifier. If not provided, then ConferenceId (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_ConferenceId) is used.

Returns

Type	Description
System.String	The token that can be used to join the conference.

Remarks

The default implementation of this method generates a token locally using Secret (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_Secret). This approach is not recommended. Tokens should be generated by a secure server that has authenticated the algorithm identity and is authorized to allow you to register with the LiveSwitch server. It is strongly recommended to not store your Secret here but to store it securely with you authentication server.

This method is called by Join(String) (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_Join_System_String_) and JoinAsync(String) (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_JoinAsync_System_String_) so it does not need to be called manually.

GetPeer(String)

Gets the peer with the specified id.

Declaration

```
public Peer GetPeer(string id)
```

Parameters

Type	Name	Description
System.String	*id*	The identifier of the peer.

Returns

Type	Description
Peer (AlgorithmAPI.Peer.html)	The Peer (AlgorithmAPI.Peer.html) with provided id.

Exceptions

Type	Condition
System.Exception	Peer with id: `id` does not exist

Join(String)

Joins a conference synchronous.

Declaration

```
public void Join(string conferenceId = null)
```

Parameters

Type	Name	Description
System.String	*conferenceId*	(optional) The identifier of the conference to join. If not provided, the value of ConferenceId (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_ConferenceId) is used.

Remarks

This method calls DoGetConferenceToken(String) (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_DoGetConferenceToken_System_String_) and sets ConferenceId (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_ConferenceId) if `conferenceId` is provided.

JoinAsync(String)

Joins a conference asynchronous.

Declaration

```
public Task JoinAsync(string conferenceId = null)
```

Parameters

Type	Name	Description
System.String	*conferenceId*	(optional) The identifier of the conference to join. If not provided, the value of ConferenceId (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_ConferenceId) is used.

Returns

Type	Description
System.Threading.Tasks.Task	awaitable Task

Remarks

This method calls DoGetConferenceToken(String) (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_DoGetConferenceToken_System_String_) and sets ConferenceId (AlgorithmAPI.AlgorithmImpl.html#AlgorithmAPI_AlgorithmImpl_ConferenceId) if `conferenceId` is provided.

Leave()

Leaves the conference synchronous.

Declaration

```
public void Leave()
```

LeaveAsync()

Leaves the conference asynchronous.

Declaration

```
public Task LeaveAsync()
```

Returns

Type	Description
System.Threading.Tasks.Task	Awaitable Task

OpenDataChannel(Peer, String)

Opens a send/receive peer-to-peer data channel to the specified peer with specified label.

Declaration

```
public DataChannel OpenDataChannel(Peer peer, string label)
```

Parameters

Type	Name	Description
Peer (AlgorithmAPI.Peer.html)	*peer*	The peer.
System.String	*label*	The label of the data channel to be created. The label should be unique among all data channel labels related to this peer.

Returns

Type	Description
DataChannel (AlgorithmAPI.DataChannel.html)	The created DataChannel (AlgorithmAPI.DataChannel.html).

OpenDataChannelAsync(Peer, String)

Opens a send/receive peer-to-peer data channel to the specified peer with specified label asynchronous.

Declaration

```
public Task<DataChannel> OpenDataChannelAsync(Peer peer, string label)
```

Parameters

Type	Name	Description
Peer (AlgorithmAPI.Peer.html)	*peer*	The peer.
System.String	*label*	The label of the data channel to be created. The label should be unique among all data channel labels related to this peer.

Returns

Type	Description
System.Threading.Tasks.Task< DataChannel (AlgorithmAPI.DataChannel.html)>	The created DataChannel (AlgorithmAPI.DataChannel.html).

OpenDataChannels(Peer, String[])

Opens multiple send/receive peer-to-peer data channels to the specified peer.

Declaration

```
public DataChannel[] OpenDataChannels(Peer peer, string[] labels)
```

Parameters

Type	Name	Description
Peer (AlgorithmAPI.Peer.html)	*peer*	The peer.
System.String[]	*labels*	The labels of the data channels to be created. The labels should be unique among all data channel labels related to this peer.

Returns

Type	Description
DataChannel (AlgorithmAPI.DataChannel.html)[]	An array of DataChannel (AlgorithmAPI.DataChannel.html)s with specified `labels`.

OpenDataChannelsAsync(Peer, String[])

Opens multiple send/receive peer-to-peer data channels to the specified peer asynchronous.

Declaration

```
public Task<DataChannel[]> OpenDataChannelsAsync(Peer peer, string[] labels)
```

Parameters

Type	Name	Description
Peer (AlgorithmAPI.Peer.html)	*peer*	The peer.
System.String[]	*labels*	The labels of the data channels to be created. The labels should be unique among all data channel labels related to this peer.

Returns

Type	Description
System.Threading.Tasks.Task< DataChannel (AlgorithmAPI.DataChannel.html)[]>	An array of DataChannel (AlgorithmAPI.DataChannel.html)s with specified `labels`.

OpenMediaStream(Peer, MediaStreamConfig)

Opens a MediaStream (AlgorithmAPI.MediaStream.html) to the specified peer.

Declaration

```
public MediaStream OpenMediaStream(Peer peer, MediaStreamConfig config = null)
```

Parameters

Type	Name	Description
Peer (AlgorithmAPI.Peer.html)	*peer*	The peer.
MediaStreamConfig (AlgorithmAPI.MediaStreamConfig.html)	*config*	(optional) The media stream configuration. If not provided, then a default config is used.

Returns

Type	Description
MediaStream (AlgorithmAPI.MediaStream.html)	A MediaStream (AlgorithmAPI.MediaStream.html)

Exceptions

Type	Condition
System.Exception	Cannot open media stream: peer does not exist.

OpenMediaStreamAsync(Peer, MediaStreamConfig)

Opens a MediaStream (AlgorithmAPI.MediaStream.html) to the specified peer asynchronous.

Declaration

```
public Task<MediaStream> OpenMediaStreamAsync(Peer peer, MediaStreamConfig config = null)
```

Parameters

Type	Name	Description
Peer (AlgorithmAPI.Peer.html)	*peer*	The peer.
MediaStreamConfig (AlgorithmAPI.MediaStreamConfig.html)	*config*	(optional) The media stream configuration. If not provided, then a default config is used.

Returns

Type	Description
System.Threading.Tasks.Task< MediaStream (AlgorithmAPI.MediaStream.html)>	A MediaStream (AlgorithmAPI.MediaStream.html)

Exceptions

Type	Condition
System.Exception	Cannot open media stream: peer does not exist.

SendMessageToPeer(Peer, String)

Sends a message to specified peer.

Declaration

```
public void SendMessageToPeer(Peer peer, string message)
```

Parameters

Type	Name	Description
Peer (AlgorithmAPI.Peer.html)	*peer*	The peer.
System.String	*message*	The message.

Exceptions

Type	Condition
System.Exception	algorithm is not registered in conference yet. or message cannot be null or provided peer is not in the conference (anymore).

Events

OnAudioFrameReceived

Occurs when an audio frame is received on any of the media streams. Runs asynchronous.

Declaration

```
public event EventHandler<AudioReceivedArgs> OnAudioFrameReceived
```

Event Type

Type	Description
System.EventHandler<AudioReceivedArgs (AlgorithmAPI.EventModels.AudioReceivedArgs.html)>	

OnDataChannelOpened

Occurs when another algorithm opens a data channel to this algorithm. Runs asynchronous.

Declaration

```
public event EventHandler<DataChannel> OnDataChannelOpened
```

Event Type

Type	Description
System.EventHandler<DataChannel (AlgorithmAPI.DataChannel.html)>	

OnDataReceived

Occurs when data is received on any of the data channels. Runs asynchronous.

Declaration

```
public event EventHandler<DataReceivedArgs> OnDataReceived
```

Event Type

Type	Description
System.EventHandler<DataReceivedArgs (AlgorithmAPI.EventModels.DataReceivedArgs.html)>	

OnMediaStreamOpened

Occurs when another algorithm opens a media stream to this algorithm. Runs asynchronous.

Declaration

```
public event EventHandler<MediaStream> OnMediaStreamOpened
```

Event Type

Type	Description
System.EventHandler<MediaStream (AlgorithmAPI.MediaStream.html)>	

OnMessageReceived

Occurs when a message is received from another peer in the conference. Runs asynchronous.

Declaration

```
public event EventHandler<MessageArgs> OnMessageReceived
```

Event Type

Type	Description
System.EventHandler<MessageArgs (AlgorithmAPI.EventModels.MessageArgs.html)>	

OnPeerConnected

Occurs when a Peer (AlgorithmAPI.Peer.html) connects to the conference. Is called for peers already present when joining and called for peers that join after you. Runs asynchronous.

Declaration

```
public event EventHandler<Peer> OnPeerConnected
```

Event Type

Type	Description
System.EventHandler<Peer (AlgorithmAPI.Peer.html)>	

OnPeerLeft

Occurs when a Peer (AlgorithmAPI.Peer.html) leaves the conference. Runs asynchronous.

Declaration

```
public event EventHandler<Peer> OnPeerLeft
```

Event Type

Type	Description
System.EventHandler<Peer (AlgorithmAPI.Peer.html)>	

OnVideoFrameReceived

Occurs when a video frame is received on any of the media streams. Runs asynchronous.

Declaration

```
public event EventHandler<VideoReceivedArgs> OnVideoFrameReceived
```

Event Type

Type	Description
System.EventHandler<VideoReceivedArgs (AlgorithmAPI.EventModels.VideoReceivedArgs.html)>	

Implements

System.IDisposable

See Also

System.IDisposable

H Additional Design

The class diagram representing the design of *LiveSwitch WebRTC Implementor* is shown in Figure H.1.

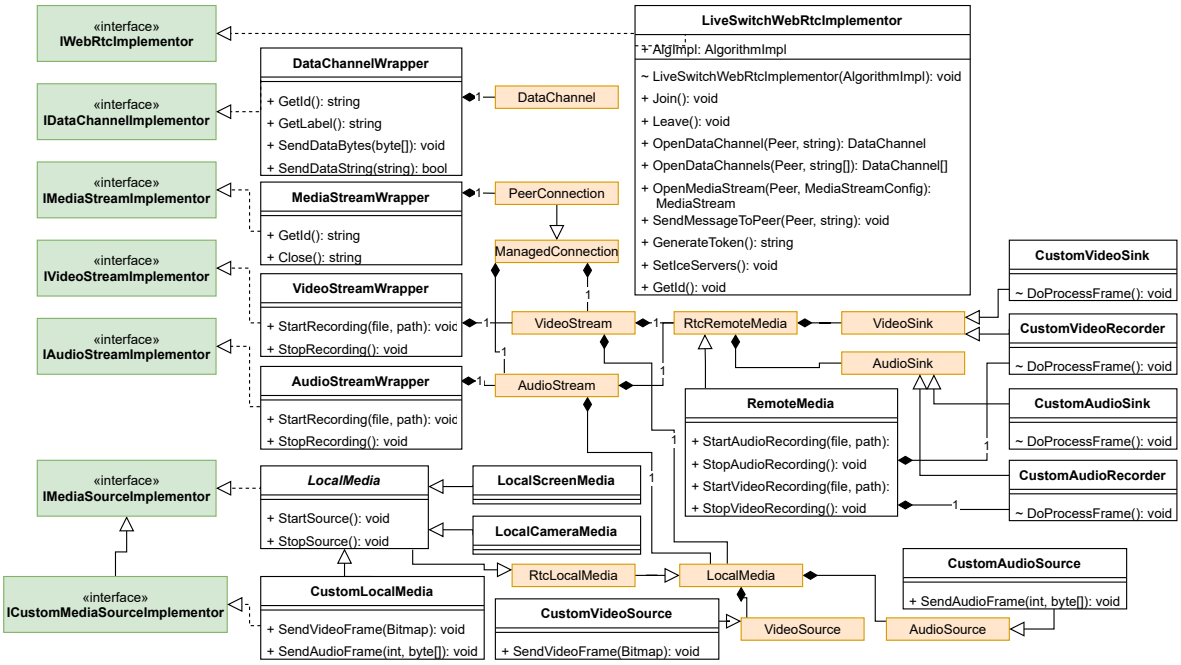


Figure H.1: The class diagram that represents the design of the *LiveSwitch WebRTC Implementor*. Rectangles in green represent interfaces part of *AlgorithmCore*. Rectangles in orange represent LiveSwitch classes. Note that, to make the diagram readable, not all methods/properties are included. In particular, several asynchronous versions of methods are left out.

I Usability Study Files

Below, included are the informed consent form and the questionnaire (including the interview questions) the participants had to fill in.

Information form for participants

This document gives you information about the study "PRAIS usability evaluation". Before the study begins, it is important that you learn about the procedure followed in this study and that you give your informed consent for voluntary participation. Please read this document carefully.

Aim and benefit of the study

The aim of this study is to measure the usability and ease-of-use of PRAIS. This information is used to determine the strong/weak points of PRAIS. More specifically, PRAIS is in development and the outcomes of the study help us improve PRAIS and provide input for the training of future PRAIS users.

This study is performed by Robin Mennens, a Software Technology PDEng Trainee under the supervision of dr. Alexander Serebrenik (Mathematics and Computer Science department).

Procedure

This study consists of two parts that should be executed in this order:
1. A questionnaire mostly consisting of (closed) usability/ease-of-use questions. This questionnaire shall take place using a word file via email.
2. An online semi-structured interview with Robin to dive deeper into the "Why/how/what/which". This questionnaire shall be done via Microsoft Teams.

Risks

The study does not involve any risks, detrimental side effects, or cause discomfort.

Note that Robin will not take the study results into account when grading the SEP project as a whole. To this end, this study will take place after Robin graded the SEP project.

Duration

The instructions, measurements, and debriefing will take approximately 40 (10 for the questionnaire and 30 for the interview) minutes.

Participants

You were selected because you are part of the Philips Healthcare Exchange (PHEX) SEP group that worked with PRAIS. Please note that in order to properly take part in this study, you need to have worked with PRAIS in any possible way: actually used it or maybe just used its documentation.

Voluntary

Your participation is completely voluntary. You can refuse to participate without giving any reasons and you can stop your participation at any time during the study. You can also withdraw your permission to use your data up to 24 hours after they were recorded. None of this will have any negative consequences for you whatsoever.

Confidentiality and use, storage, and sharing of data.

This study has been approved by the Ethical Review Board of Eindhoven University of Technology.

In this study, no personal data is explicitly recorded. Experimental data (your responses to the questionnaire and an audio recording + transcription of the interview) will be recorded, analyzed, and stored. Only the audio recording of the interview may contain personal information. We will anonymize this information during transcription and remove the original audio recording. The goal of collecting, analyzing, and storing this data is to answer the research question and to describe the experiment results in Robin's final report. To protect your privacy, all data will be stored on a protected PDEng Software Technology server that is only accessible by Robin and the PDEng Software Technology manager (Yanja Dajsuren). No information that can be used to personally identify you will be shared with others.

We will not share personal information about you or your responses in this study with anyone outside of the research team (Robin Mennens, Marcel Quist, Zoran Stankovic, Arjan Draisma, Alexander Serebrenik, Yanja Dajsuren). Only Robin will know your identity because he is the interviewer.

The anonymized data collected in this study that will be released to the public will (to the best of our knowledge and ability) not contain information that can identify you. It will include all answers you provide during the study.

The audio recordings that are made will not be distributed and will not be played back in the presence of persons other than the researchers. The material will be used only for scientific analysis and deleted after transcribing the data.

Further information

If you want more information about this study, the study design, or the results, you can contact Robin (contact email: r.j.p.mennens@tue.nl).

If you have any complaints about this study, please contact the supervisor, Alexander Serebrenik (a.serebrenik@tue.nl). You can report irregularities related to scientific integrity to confidential advisors of the TU/e.

Informed consent form

PRAIS usability evaluation

- I have read and understood the information of the corresponding information form for participants.

- I have been given the opportunity to ask questions. My questions are sufficiently answered, and I had sufficient time to decide whether I participate.

- I know that my participation is completely voluntary. I know that I can refuse to participate and that I can stop my participation at any time during the study, without giving any reasons. I know that I can withdraw permission to use my data up to 24 hours after the data have been recorded.

- I agree to voluntarily participate in this study.

- I know that no information that can be used to personally identify me or my responses in this study will be shared with anyone outside of the research team.

- I □ **do**
  □ **do not**
  give permission to make my anonymized recorded data available in Robin's final report.

Certificate of consent

I, (NAME) ……
want and provide consent to participate in this study.

_____ _____

Participant's Signature Date

Part 1 - Questionnaire

To as great an extent as possible, think about all the tasks that you performed with PRAIS while you answer these questions. Note that some statements refer to a job. For those statements, we would like you to interpret SEP as a job. Please read each statement carefully and indicate how strongly you agree or disagree with the statement (where **1 = extremely disagree** and **7 = extremely agree**) by putting an X in the appropriate table cell**.** If you cannot answer a statement because you, for example, did not work with LiveSwitch, then leave the answer empty.

	1	**2**	**3**	**4**	**5**	**6**	**7**
Using PRAIS in my job enables me to accomplish tasks more quickly than LiveSwitch.							
Using PRAIS improves my job performance.							
Using PRAIS in my job increases my productivity.							
Using PRAIS enhances my effectiveness on the job.							
Using PRAIS makes it easier to do my job.							
I have found PRAIS useful in my job.							
Learning to operate PRAIS was easy for me.							
I found it easy to get PRAIS to do what I want it to do.							
My interaction with PRAIS has been clear and understandable.							
I found PRAIS to be flexible to interact with.							
It was easy for me to become skillful at using PRAIS.							
I found PRAIS easy to use.							

Please answer the following question, where **0 = not likely** and **10 = very likely**.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
How likely is it that you would recommend PRAIS to a friend or colleague?											

Part 2 – Interview

The second part of this study consists of an interview with Robin. He will plan a 30-minute timeslot with you in which he will ask some open questions about your experience with PRAIS. For your reference, the questions are already included below.

1. Overall, how was your experience with PRAIS?
2. Which things did you like the best about PRAIS? Why?
3. Which things did you like the least about PRAIS? Why?
4. If you could change something about PRAIS, what would it be? Why?
5. What do you think is still missing in PRAIS? Why?

6. What do you think about PRAIS' documentation?

7. How much time do you think you saved by using PRAIS, instead of having to use LiveSwitch?

8. Why would you or would you not recommend PRAIS to a friend or colleague?

J PRAIS Usability Study Results

Table J.1 lists the results of the questionnaire. Each column refers to one of the questionnaire questions. In particular:

- Q1: Using PRAIS in my job enables me to accomplish tasks more quickly than LiveSwitch.
- Q2: Using PRAIS improves my job performance.
- Q3: Using PRAIS in my job increases my productivity.
- Q4: Using PRAIS enhances my effectiveness on the job.
- Q5: Using PRAIS makes it easier to do my job.
- Q6: I have found PRAIS useful in my job.
- Q7: Learning to operate PRAIS was easy for me.
- Q8: I found it easy to get PRAIS to do what I want it to do.
- Q9: My interaction with PRAIS has been clear and understandable.
- Q10: I found PRAIS to be flexible to interact with.
- Q11: It was easy for me to become skillful at using PRAIS.
- Q12: I found PRAIS easy to use.
- Q13: How likely is it that you would recommend PRAIS to a friend or colleague?

Table J.2 lists the 166 cards used during card sorting and their assigned categories. After that, the audio transcriptions of the 5 interviews are shown. In those transcriptions, sentences in bold are the statements that were converted into cardsorting cards.

Participant	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13
A	5	6	6	5	5	6	7	5	5	5	6	6	8
B	6	5	6	6	6	6	5	5	5	5	5	5	7
C	7	6	6	6	7	6	5	6	7	4	5	6	7
D	5	4	5	4	4	5	6	6	6	4	6	6	8
E	6					6	4	5	6	5	5	5	7

Table J.1: The results of the questionnaire. Participant E mentioned he could not fill in questions 2, 3, 4, and 5, which is why he left them empty. In the analysis, we replaced these empty values with the middle value 4, which is recommended by the mTam [LLŠ20] model.

Who	Category	SubCategory	SubSubCategory	Statement
E	Neutral			There is authentication to a degree using the secrets, it's not the most secure thing in the world. But it's definitely better than nothing.
E	Neutral			Seven or 8/10, like I wouldn't be able to actually convince them to use PRAIS if they have already chosen something else. Because I don't have experience with any other real-time communications API.
A	Neutral			But after I got that done (learning liveswitch) I don't think it would've been that much harder to copy that side over to the bots which use PRAIS.
C	Tips	.NET Framework		.net framework. Very clearly .net framework. We have all vowed to never use Visual Studio ever again, ever.
C	Tips	.NET Framework		because it was only Windows. I think that does set back mostly the algorithm side. When you make algorithms, most servers run Linux.
C	Tips	.NET Framework		So, for a hospital, deploying a C# app on Windows may be a stretch. So, I would really consider moving to .net core. If possible.
C	Tips	.NET Framework		.net framework is very integrated with Windows and the entire tooling system. And they just set us back a lot, to have to use Visual Studio sometimes. That was the thing we like the least I guess.
C	Tips	.NET Framework		I think the .net core, the .net framework it kind of captures all the problems we had. If you just switch from .net framework it would solve nearly all the problems. Because most things where just due to that.
A	Tips	.NET Framework		Well more about developing with PRAIS was in using Visual Studio which was not really a success.
C	Tips	Add API	Javascript	I think I also answered question four, what would you change. I would probably go with Javascript..

A	Tips	Add Javascript API		Well the problem still is that there is no PRAIS for the web apps side of things so we still had to learn about LiveSwitch which was unfortunate.
E	Tips	Closing data channels		I believe at some point we ran into a thing where we wanted to reliably open and close datachannels but there was no method for that.
A	Tips	Closing data channels		Oh yeah there is the thing for terminating datachannels, but that something we already talked about that was a LiveSwitch limitation
C	Tips	Documentation	More Conceptual Docs	maybe the only part that I would improve is the general concepts. So even though there is a diagram and it's in English so there is a data channel class what would probably create a data channel surprise. It may be a bit hard for some developers to understand. I think we did most of our understanding through just creating some random demos. And I think that's a good way to learn but not everybody wants to learn that way. So they want to know what can I do with a media source. Not even what is a media source but what could I theoretically do with it? What kind of things are media sources? And I think that mostly I think we also struggled with that on the front-end, we thought it would be really easy to send video, it wasn't. In that sense like what is a media source, is that dependent on LiveSwitch? For instance it says the type of media source, media source type, okay camera screen and custom, but what could a custom media source be? Is it anything that sense just frames? So maybe go into detail on what you can expect more. That would be an improvement.
C	Tips	Documentation	More Conceptual Docs	(Documentation:) You explain what it does, but not what I could do with it. That would be the only improvement you could make.
A	Tips	Documentation	More Conceptual Docs	Yeah. And there is also, I'm looking at the documentation now, there is the, you have to join async and a regular join method and maybe some think about what it means to join asynchronously. And why would use it over something else.
A	Tips	Documentation	More Conceptual Docs	Yeah I guess maybe we need some more docs about how this works conceptually. If you know that the frame is not really safe to work with then you can copy it that would be nice maybe a good addition. Interviewee: yes
E	Tips	Documentation	More Examples	Yes, it might be good to have maybe one or two more examples that are a bit more elaborate.

E	Tips	Documentation	More Examples	I suppose the only thing that I would add would be something like maybe another example like I said before that really shows you more how to use a specific methods because that's not always clear from just the documentation
A	Tips	Documentation	More Examples	But maybe some example things because well there's not a lot of examples out there for PRAIS other than the, what is it, echo bot and the translation bots that were provided
A	Tips	Documentation	More Examples	Okay, so more examples would help in understanding the concepts better? Interviewee: Yes, exactly. And especially also like when to use which type of construct.
E	Tips	Documentation	Other	Yes it's a really basic example. It shows the idea behind PRAIS very well but I think leaves a bit too much to the imagination to really give you an idea of like this is what it is really capable of.
D	Tips	Documentation	Other	Only the thrown exceptions that you added at some point they aren't documented or not completely I think but I think that's the only thing that I would change.
A	Tips	Documentation	Other	But yeah for PRAIS itself the thing I like least was at the start I think that the documentation could be a little bit more extensive.
E	Tips	Documentation	Other	Except for the Microsoft Azure one.
D	Tips	LiveSwitch has more features		Well because PRAIS contains a lot of sealed classes and not a lot of interface or at least not a lot of interfaces that are public within the library it doesn't really allow to modify PRAIS so when you use PRAIS you should really stick with the PRAIS functionality mostly. So, if you have to do stuff that is a bit different than what PRAIS does then, I think it easier to just use LiveSwitch instead.
A	Tips	LiveSwitch has more features		Let's see on a technical level it maybe would be nice for I noticed in LiveSwitch you have datachannels you can see a status whether they are still setting up or something like that maybe it would be nice to get something like that and PRAIS as well I think we found ourselves needing that exact functionality at some point I don't know if it was still required in the final product but it would be nice to have some of those things as well.

D	Tips	Logging	The two things that were a bit less when using PRAIS were I think the exceptions sometimes LiveSwitch throws an exception and PRAIS catches it but does not propagate it. This makes it somewhat uneasy to deal with exceptions and
D	Tips	Logging	On the one hand there is the exceptions, they are sometimes catched by PRAIS but not propagated I would really like to see them propagated in some way either just propagate the actual exception or throw a different one something like that.
D	Tips	Logging	So one thing that I would change I already mentioned is the exceptions.
C	Tips	Logging	Because all the LiveSwitch logging is kind of not really taken into the abstraction layer. So I think we can really improve on that. Interviewee: yeah
C	Tips	Logging	(missing things:) But in general it's PRAIS logging I think. Just have PRAIS log useful messages because LiveSwitch does not.
B	Tips	Logging	So, more logging? Interviewee: Uhuh, yes.
B	Tips	Logging	So you think improving like the things PRAIS tells you about on what's going on, that would help you in debugging what's going wrong? Interviewee: uhuh
B	Tips	Logging	But you already mentioned that the logging would already help a lot there. Interviewee: yeah
E	Tips	More Advanced Security	Well I briefly mentioned this changing the way secrets are handled.
E	Tips	More Advanced Security	But support for a more advanced way of authentication would be a good idea.
C	Tips	More Advanced Security	Add the authentication server into PRAIS, make it like a normal part of how it functions
B	Tips	More Advanced Security	okay so you mentioned authentication would be something to help by default like an authentication server you made. Interviewee: Uhuh

C	Tips	Not a complete platform yet	(Recommendation:) The only reason I wouldn't yet is because it doesn't feel complete.
C	Tips	Not a complete platform yet	here it seems that the product works but I can't download it yet, I can't create an account yet, I have no panel to register my app I just get the C# code from some out of band communication and I think that the only thing that's.
C	Tips	Not a complete platform yet	No I think PRAIS itself is because you have the, you don't have some, yes some missing parts so I can't create my own secret I can't create my own conference at the moment because that is missing it's more of a technical demo even for us as users. Than it is a full platform yet.
D	Tips	Other	Something else that I'm not really sure about but I had some doubts about it. To get the PRAIS functionality you have to extend algorithm base. I'm not really sure if that's like the best way to do it or whether there is any design choice behind this.
C	Tips	Other	I do think that the one thing that we did experience was, its not really the fault of PRAIS but it is remarkable that you still needed to know LiveSwitch in a way. At the very basic level you can just do things with PRAIS, but you very quickly came into like oh why is this error thrown, because the log entries are still LiveSwitch. Why did this happen, is it my fault or LiveSwitch's fault? Then you still had to go to LiveSwitch documentation to find out what happened and then the abstraction kinda got in the way because you couldn't diagnose if it was your part, PRAIS, or LiveSwitch.
C	Tips	Other	yeah I actually think that if you understand LiveSwitch, you using the front-end was sometimes easier to diagnose problems than through PRAIS. Because when you did it through PRAIS you were under the impression that you did it right and most of the time you did do it right, but some very specific use case wasn't the main focus of the method in PRAIS and then it bugs LiveSwitch or something.
C	Tips	Other	So you could find where the problem was because you called LiveSwitch. So that's the only downside of abstraction layers. But I don't think that a PRAIS thing specifically.
C	Tips	Other	That you had to know some things about LiveSwitch, but I already mentioned that before, just for the clarity to included here.

C	Tips	Other	We had dependencies that were unclear or nuget packages, I don't think it's the best method of packaging a thing.
C	Tips	Other	But it would've been nicer if there was just a PRAIS server for which you have a SDK in every language to talk with.
C	Tips	Other	Yeah it's more of a very very long proof of concept technical demo, at least what it seems to me from the outside and I don't think that's bad, but before you sell it you do need some more things around it.
B	Tips	Other	Like after that you had like a question is there is something missing... yeah a library could always have like more functionality and we looked into the authentication and adding a database and I think those are functionalities that a company would also use. So some interface for that would also be nice.
A	Tips	Other	Yeah because it's I think with the target audience of PRAIS It does not make sense to have to learn both LiveSwitch and PRAIS
A	Tips	Other	okay so basically more state information about datachannels and also media streams I guess, that would help? Interviewee: yes, that would be nice.
A	Tips	Other	Yes there is one thing with especially with the focus on real-time it would be nice if there is a construct to easily say group or link some data to a certain set of video frames so say like you send one second of video and it gets a specific tag and you send one second of data over datachannels and it also get that specific tag and on the other side you can then easily piece the two back together and see oh yeah the data is corresponds to that video
E	Tips	Testability/Add interfaces	we had to create some interfaces to perform some class construction so we can not like have all our classes depend on PRAIS which makes them pretty much impossible to unit test.
E	Tips	Testability/Add interfaces	So we had to write some interfaces for several things so we could use more classes and generally make testing possible at all.
D	Tips	Testability/Add interfaces	As you may have noticed, we didn't really use PRAIS the same way anymore as you prescribed in the documentation mainly because it allowed us to be able to test PRAIS with unit testing

D	Tips	Testability/Add interfaces	testing was a bit less easy.
D	Tips	Testability/Add interfaces	It's not really easy to test,
D	Tips	Testability/Add interfaces	But having interfaces for these classes and somehow being able to substitute them that would be useful.
D	Tips	Testability/Add interfaces	One thing, something that I would change or something that I would add are the interfaces.
D	Tips	Testability/Add interfaces	So if you extend algorithm base it's not testable.
D	Tips	Testability/Add interfaces	But, the testing is a bit more difficult
C	Tips	Testability/Add interfaces	Yes, so in a sense returning interface types, or allowing interface types to be sent would be useful
C	Tips	Testability/Add interfaces	It should just accept something that implements the interface instead of requiring to send a peer for instance. You should require it to send a peer interface, not an object instance. That would've saved us, I think some time because now we have classes that just take our peer and convert it to some fake version of the PRAIS peer or mocked version of the PRAIS peer that would not have been necessary if we could just send the fake one directly.
C	Tips	Testability/Add interfaces	But some interfaces would've been nice.
C	Tips	Testability/Add interfaces	Maybe add more interfaces but we already talked about it.
D	Tips	Unsure if would recommend	When you need to do things that PRAIS cannot do at this moment then I think I would recommend not to use PRAIS but use something that's easier modifiable. Yeah.

D	Tips	Unsure if would recommend		So mostly I would just recommend PRAIS, but when you have to modify the behavior or do something different then probably not.
B	Tips	Unsure if would recommend		So, I'm not sure if I would recommend it.
E	Tops	Compared to LiveSwitch	Better Documentation	(the documentation) It's better than LiveSwitch, I can tell you that much.
D	Tops	Compared to LiveSwitch	Better Documentation	Which was really nice also I think that the documentation of PRAIS is quite good especially when you try to compare it with LiveSwitch.
C	Tops	Compared to LiveSwitch	Better Documentation	(Documentation:) But I think it's more than sufficient now and it's amazing compared to what LiveSwitch did.
A	Tops	Compared to LiveSwitch	Better Documentation	Because for LiveSwitch as well you don't have any proper documentation that explains this function actually does this. For PRAIS it's definitely better and you can actually get stuff working.
A	Tops	Compared to LiveSwitch	Better Documentation	it's better documented than LiveSwitch that also a bonus
D	Tops	Compared to LiveSwitch	Easier to learn/use	There was not a lot of when you look at LiveSwitch you have like lots of functions and some functions do similar things but you're not sure how they differ and that kind of stuff when you look at PRAIS you just have a couple of functions they are really clear in what they're supposed to do and it's really easy to use.
C	Tops	Compared to LiveSwitch	Easier to learn/use	And even within the classes there aren't that many functions. There are enough functions to do what you want enough methods but they are not too many. So for instance for LiveSwitch you have some classes that have way too many overwriters and functions and methods I think in LiveSwitch there are 10 ways to do somethings.
B	Tops	Compared to LiveSwitch	Easier to learn/use	But it was way clearer than LiveSwitch, so that's nice.

A	Tops	Compared LiveSwitch	to	Easier learn/use	to	I liked using it, once I got the hang of it, it was I think definitely easier than LiveSwitch.
A	Tops	Compared LiveSwitch	to	Easier learn/use	to	after I got into the LiveSwitch thing, it was way more of a hassle than learning to work with PRAIS.
A	Tops	Compared LiveSwitch	to	Easier learn/use	to	Especially over LiveSwitch where you have to really go digging into the code and have to hook your code into some way and it kind of feels a bit hacky and this has a nice interface for doing the real-time communication.
A	Tops	Compared LiveSwitch	to	Easier learn/use	to	Because you're not actually, you're not doing the in my opinion needless things. For LiveSwitch you have to write a lot of boilerplate code yourself still in. For PRAIS it basically look at PRAIS and you start actually doing your business logic.
A	Tops	Compared LiveSwitch	to	Easier learn/use	to	So I definitely would use PRAIS for that. I remember getting like a basic example to run in like, I don't know under an hour and for LiveSwitch it was constantly digging into documentation some mild frustration and then only after quite some time you get stuff to work the way you want to.
E	Tops	Compared LiveSwitch	to	General		what I like the best about PRAIS is not having to use LiveSwitch.
E	Tops	Compared LiveSwitch	to	General		So, using PRAIS is definitely preferable.
E	Tops	Compared LiveSwitch	to	General		Well the first thing is I would say please don't use LiveSwitch.
D	Tops	Compared LiveSwitch	to	General		Okay, so if my friend or colleague has to do relatively simple things with LiveSwitch with not too many peers I think or just send receive some data processes it nothing really fancy then I would really recommend PRAIS because it's very simple easy to use.
C	Tops	Compared LiveSwitch	to	General		And I don't know if everybody noticed that but compared to LiveSwitch on the front-end we spent more time getting LiveSwitch to work even though the app was already there than building things with PRAIS I think

B	Tops	Compared to LiveSwitch		General	And I also think you can work with PRAIS without understanding LiveSwitch in depth.
A	Tops	Compared to LiveSwitch		General	And also I think in the long run it's nicer because I think the code for PRAIS is more maintainable because its directly almost directly business logic with PRAIS instead of LiveSwitch where you have just a lot of stuff that deals with having LiveSwitch work correctly. So I would say it's way faster.
A	Tops	Compared to LiveSwitch		General	so yeah it's I think in a lot of aspects it's superior..
E	Tops	Compared to LiveSwitch		Saving Time	Yeah, sounds good, I guess the main point here was to check if you think PRAIS actually saves you time. Interviewee: I definitely think so
D	Tops	Compared to LiveSwitch		Saving Time	I looked a little bit into the LiveSwitch documentation and it was really horrible. So, yeah, like LiveSwitch has lots of functions, lots of functionality but looking at the documentation you have no clue what it does. So, I think that this would save me quite a bit of time. Maybe about 30 hours in this project I guess.
D	Tops	Compared to LiveSwitch		Saving Time	It did save me time, I'm sure about that. How much, I don't know exactly but it did save me time.
C	Tops	Compared to LiveSwitch		Saving Time	(Time saving:) Uhm, infinite probably.
B	Tops	Compared to LiveSwitch		Saving Time	There was a lot of difficulty and also myself understanding LiveSwitch because of the limited documentation so I'm really sure that PRAIS saves a lot of time because the documentation was a lot clearer
A	Tops	Compared to LiveSwitch		Saving Time	Yes, I think it definitely saves us a lot of time, especially in the initial phase PRAIS is way easier to get the hang of than LiveSwitch.
E	Tops	PRAIS		Completeness	Pretty much all the functions you want from it are already there.

B	Tops	PRAIS	Completeness	I think it's quite, the main functionalities are there and there is also not much more that you would want. You have like peers and algorithm and different streams. That kind of what you need. So I think that sufficient
B	Tops	PRAIS	Completeness	and the main functionalities were already implemented for us. That was definitely the advantage of PRAIS.
A	Tops	PRAIS	Completeness	No at the moment I don't really, I think that it is pretty well rounded already
E	Tops	PRAIS	Documentation	the thing that really helped me understand how PRAIS works are the examples that were on the PRAIS website.
E	Tops	PRAIS	Documentation	I played around with those a bit. I got those to work pretty quickly.
E	Tops	PRAIS	Documentation	I really like the fact that there were actually like examples that are like well explained.
E	Tops	PRAIS	Documentation	There is a at least a simple example that as well explained. It helped me understand it better.
E	Tops	PRAIS	Documentation	But, no, in general the documentation for PRAIS is quite clear.
E	Tops	PRAIS	Documentation	I think everything is present, I haven't found any functions that do exist but aren't in the documentation.
E	Tops	PRAIS	Documentation	Most of them are self-explanatory. Something like media source.stop(), I just pulled it open right now the documentation
E	Tops	PRAIS	Documentation	Yeah, most things are pretty much everything is like very well specified. Every method has a description of every class has a description. It says how to use the methods, says what a method returns. With even a description of the type that it returns which is something you don't see all the time. Especially helpful if you have a lot of custom classes custom records custom datatypes.
E	Tops	PRAIS	Documentation	No, (the documentation) they are very complete as far as I can tell. That's all
D	Tops	PRAIS	Documentation	Yeah, good documentation also helps that you added these XML comments really helps the IDE to highlight them and just allows for faster development.
D	Tops	PRAIS	Documentation	Yeah, I think the documentation is really nice.

D	Tops	PRAIS	Documentation	So, it's well-documented functions have their descriptions about what they do and what they return.
D	Tops	PRAIS	Documentation	They have XML comments which is useful in the IDE.
C	Tops	PRAIS	Documentation	Very clear, nicely auto generated. That's always a plus because that also tells me that you made proper documentation in your code. Because it can auto generate this. I think it's clear,
C	Tops	PRAIS	Documentation	So yes I think that's already pretty great and also the examples helped a lot.
C	Tops	PRAIS	Documentation	(Documentation:) It also shows how easy it is to create a thing.
B	Tops	PRAIS	Documentation	It's quite easy to use because of the documentation is clear.
B	Tops	PRAIS	Documentation	I also like that there was already an example present, the echo bot.
B	Tops	PRAIS	Documentation	by starting with such an example it was way easier to extend and understand how PRAIS was intended.
B	Tops	PRAIS	Documentation	Uhuh. Okay so the examples were really helpful? Interviewee: Yes.
B	Tops	PRAIS	Documentation	(Documentation) I think it's good.
B	Tops	PRAIS	Documentation	I also liked the conceptual figure. You get quickly and overview of what kind of functions there are in PRAIS.
B	Tops	PRAIS	Documentation	Yeah it's not really a large model so I don't think you need additional explanation.
A	Tops	PRAIS	Documentation	I think it's nice documentation,
A	Tops	PRAIS	Documentation	to me at least it explains pretty well what everything should do how we can use it. Especially going along with the examples.
A	Tops	PRAIS	Documentation	Very clear, nicely auto generated. That's always a plus because that also tells me that you made proper documentation in your code. Because it can auto generate this. I think it's clear,

E	Tops	PRAIS	General	I didn't work with it that much but generally pretty positive. I noted that the functions that it's supposed to do, it does them very well.
E	Tops	PRAIS	General	I think audio quality is generally very accurate, the video quality, there is pretty much no deterioration in that as far as I can tell, images are clear images and crisp.
E	Tops	PRAIS	General	Not really, there weren't really any things that jumped out to me is like this is bad. This needs to be improved or changed by like significant amount
D	Tops	PRAIS	General	Overall I think that I had a good experience with PRAIS.
D	Tops	PRAIS	General	Like if you want to create simple applications with web RTC then this is really the way to go if you ask me
D	Tops	PRAIS	General	Yes, overall I'm really happy with PRAIS
D	Tops	PRAIS	General	when you want to do basic things, PRAIS is great and it covers most scenarios which is really nice.
C	Tops	PRAIS	General	I could explain why all these classes exist to someone who doesn't know about video streaming. And I think that's the biggest strength of having like a low amount of classes that are very clearly for one goal.
C	Tops	PRAIS	General	I think it's a good testament that a group with some programmers who have never programmed before could actually build a web streaming thing in less than 10, maybe only five weeks.
C	Tops	PRAIS	General	our progress was really nice. I think that's, yeah, a lot of that has to do with PRAIS.
B	Tops	PRAIS	General	I enjoyed working with PRAIS.
B	Tops	PRAIS	General	So yeah NuGet package, you followed the installation instructions I guess. Any issues with that? Interviewee: That worked fine.
B	Tops	PRAIS	General	No, not really. I was quite happy with how it works overall.
B	Tops	PRAIS	General	Which is also a good thing because they depend on each other but if you should be comfortable with both libraries completely then I would be a bit more hesitant to

B	Tops	PRAIS	General	I liked using PRAIS so I would use it again if that would be possible.
A	Tops	PRAIS	General	(Experience with PRAIS:) I would say it's positive.
A	Tops	PRAIS	General	Well nothing more I already said that, it's really been nice to use.
E	Tops	PRAIS	Simplicity/Ease-of-use	It's not that hard to use.
E	Tops	PRAIS	Simplicity/Ease-of-use	I would say that as an overall experience it was fairly easy to set up and install. Especially for someone who has no experience in both C# and typescript which are the two languages that we used.
D	Tops	PRAIS	Simplicity/Ease-of-use	PRAIS is really easy to use and I don't think we encountered any real bugs with PRAIS.
D	Tops	PRAIS	Simplicity/Ease-of-use	I think that what I like the best about PRAIS was the simplicity.
D	Tops	PRAIS	Simplicity/Ease-of-use	Just you are looking for something, something easy like send datastring. There you have some function send it synchronously send it asynchronously just really straightforward to use. I think that was what I liked most about PRAIS
D	Tops	PRAIS	Simplicity/Ease-of-use	Well simplicity and the documentation. I think those are really the things that were the best
C	Tops	PRAIS	Simplicity/Ease-of-use	I think PRAIS is pretty clear,
C	Tops	PRAIS	Simplicity/Ease-of-use	you tried to keep the amount of classes low and I think that really shows in the clarity of our code. Because if you look at our code it doesn't seem that hard, it seems to be very easy to do things.
C	Tops	PRAIS	Simplicity/Ease-of-use	it just seems to function in a more streamlined way than you would maybe expect on a technical level. That was my first impression.

C	Tops	PRAIS	Simplicity/Ease-of-use	Alright so you would say the simplicity is nice. That it's very easy to understand. Interviewee: Yes.
C	Tops	PRAIS	Simplicity/Ease-of-use	The clarity in explaining everything. So we had a team where some people haven't ever worked with C at all or C# and we also had people that were experienced. And someone could write some code and show it to someone else and they understood what it should do. Like the business logic part I think that's very powerful.
C	Tops	PRAIS	Simplicity/Ease-of-use	So, the parts on the server where that didn't happen, because it just worked, it just easily was readable, solutions were clear and it worked. I think that saved us a lot of time.
B	Tops	PRAIS	Simplicity/Ease-of-use	as a library I understood it quite fast.
A	Tops	PRAIS	Simplicity/Ease-of-use	While I like that it was just subscribing to events and then PRAIS handles everything. That was really nice.
A	Tops	PRAIS	Simplicity/Ease-of-use	So it's the simplicity. Interviewee: Yes, exactly.
A	Tops	PRAIS	Simplicity/Ease-of-use	I think this is really where the simplicity of PRAIS really shines.
A	Tops	PRAIS	Simplicity/Ease-of-use	Well I would recommend PRAIS because it again going to save time with developing and like I said before it also in my opinion makes the code look neater and more understand able at that.
E	Tops	PRAIS	Would recommend depending on context	The second thing I would say is hey I worked with PRAIS this one time it was pretty good you should try looking into that.
E	Tops	PRAIS	Would recommend depending on context	if they just mention like hey I'm trying to build some application how would you suggest I approaches this? I would definitely at least mention PRAIS that they should look into it

C	Tops	PRAIS	Would recommend depending on context	For starters I would recommend PRAIS if someone was building an AI platform.
B	Tops	PRAIS	Would recommend depending on context	I think I would recommend it to a programmer friend if they would use a similar project since I found it quite easy to use but its quite specific for media streaming, that's not really something that you use in every project.
B	Tops	PRAIS	Would recommend depending on context	Okay so I assuming that somebody says hey I'm working on a streaming web app/application something then you would say okay consider using PRAIS? Interviewee: Yeah, uhuh.

Table J.2: A list of all the 'cards' used during card sorting.

Interviewer	Then let's just start with the question list. So something I would like to know before actually going into the questions is you each probably had a role or job within the team so how did you work with PRAIS? What was your responsibility with respect to PRAIS?
Participant A	At the start of the project I, well, did some experimenting with PRAIS and seeing exactly like how we can nicely use it to get images into python afterwards. I did a small mockup with that and afterwards it was mainly working on the bot side of things. So yeah that is just normal functionality of PRAIS getting the video stream video frames and getting data from the datachannels and well processing that.
Interviewer	So you also worked on the python wrapper, is that correct?
Participant A	Yes I partially worked on that as well
Interviewer	Okay, so mostly on processing of incoming data
Participant A	Yes
Interviewer	and anything on LiveSwitch that you worked on?
Participant A	Yeah LiveSwitch I also very slightly worked on. At least I have an idea on how to where in the demo code if you will it where to begin with your own code
Interviewer	All right okay, that's nice so you actually worked on both sides
Participant A	Yes
Interviewer	Which kind of allows you to compare LiveSwitch and PRAIS?
Participant A	Yes
Interviewer	Very good, nice. Yeah then let's go to the first question. Overall, how was your experience with PRAIS?
Participant A	**(Experience with PRAIS:) I would say it's positive.**
	I liked using it, once I got the hang of it, it was I think definitely easier than LiveSwitch.
	Well the problem still is that there is no PRAIS for the web apps side of things so we still had to learn about LiveSwitch which was unfortunate.
Interviewer	Yes that's indeed something we are working on. We actually just for your info we do have an API for the browser as well. That one works with ICELink, however, that's also by FrozenMountain the same developer as LiveSwitch. And during the last few weeks I was implementing PRAIS in ICELink. So now the browser API and PRAIS they work together. So you used LiveSwitch while ICELink now actually works with both.
Participant A	Nice
Interviewer	But that's good feedback, indeed its a limitation of the LiveSwitch side.
Participant A	**Yeah because it's I think with the target audience of PRAIS It does not make sense to have to learn both LiveSwitch and PRAIS**
	because
	after I got into the LiveSwitch thing, it was way more of a hassle than learning to work with PRAIS.
	But after I got that done (learning liveswitch) I don't think it would've been that much harder to copy that side over to the bots which use PRAIS.
Interviewer	Okay, that makes sense thank you
Interviewer	I guess we can just continue to the more specific questions then. So, which things that you like the best about praise and why?

Participant A	**While I like that it was just subscribing to events and then PRAIS handles everything. That was really nice.**
	Especially over LiveSwitch where you have to really go digging into the code and have to hook your code into some way and it kind of feels a bit hacky and this has a nice interface for doing the real-time communication.
Interviewer	**So it's the simplicity**
Participant A	**Yes exactly (refers to above)**
Interviewer	All right
Participant A	**Because you're not actually, you're not doing the in my opinion needless things. For LiveSwitch you have to write a lot of boilerplate code yourself still in. For PRAIS it basically look at PRAIS and you start actually doing your business logic.**
Interviewer	Uhuh, nice that's good to hear. All right, very good. Anything else you want to add to that things you like about PRAIS?
Participant A	**Well nothing more I already said that, it's really been nice to use.**
Interviewer	Okay. I think it's clear. Simplicity is indeed something we are really aiming for. To hide all of this LiveSwitch complexity in setting up all the streams and like you say putting in all the different pieces in some weird way. Instead we like you said I think it's a nice way to say it we want to immediately go with the business logic instead of going for the complete infrastructure and all that other stuff you have to do as well.
Participant A	Yes
Interviewer	Uhm okay, so of course which things that you like the least about PRAIS and why?
Participant A	Ahh well, the least thing I like about PRAIS..
	Well more about developing with PRAIS was in using Visual Studio which was not really a success.
	But yeah for PRAIS itself the thing I like least was at the start I think that the documentation could be a little bit more extensive.
	Because for LiveSwitch as well you don't have any proper documentation that explains this function actually does this. For PRAIS it's definitely better and you can actually get stuff working.
	But maybe some example things because well there's not a lot of examples out there for PRAIS other than the, what is it, echo bot and the translation bots that were provided
Interviewer	**Okay, so more examples would help in understanding the concepts better?**
Participant A	**(Refers to above) Yes, exactly. And especially also like when to use which type of construct.**
Interviewer	Uhuh
Participant A	So like yeah what is maybe a good instance for using a data channel and maybe something also like on what format are we going to put stuff on a data channel. But that might be a little bit out of scope for the documentation actually.
Interviewer	Okay, so it was not really clear when to use a media stream and went to use a data channel for example?
Participant A	No that's to me was clear. It would be nice to see some example for when to use something.
Interviewer	All right, okay
Participant A	I don't have a concrete example of that, I'm sorry

Interviewer	No, that's fine, it's indeed while making this I was like let's add the most simple example you can have, the echo bot and let's make a more complicated one but yeah the Azure one you need this Microsoft key. Which again makes it kind of hard to redo it yourself. So, it is indeed I think a good point to add some more examples that work out-of-the-box and show the concepts more in depth.
Participant A	**Yeah. And there is also, I'm looking at the documentation now, there is the, you have to join async and a regular join method and maybe some think about what it means to join asynchronously. And why would use it over something else.**
Interviewer	Okay yeah so these things are already technical I guess multithreading using stuff in the background but I must say this is like the most difficult part of PRAIS to make sure all the threads are safe and do not break each other. Which is a real pain. I agree that it's not really clear when an async is better than sync. I could definitely add something to the docs on a conceptual level what it means to do it in the background and what you have to be careful with when you do run it in background all right okay I think that clear
Participant A	Yes, good
Interviewer	Anything else that you would say this is something I don't really like about PRAIS?
Participant A	No, I pretty much liked it for the rest of PRAIS
Interviewer	Okay, the next one then. So if you could change something about PRAIS, what would it be and why? I guess we already discussed some things like documentation and examples but it or anything else that you maybe want to change on a technical level anything?
Participant A	**Let's see on a technical level it maybe would be nice for I noticed in LiveSwitch you have datachannels you can see a status whether they are still setting up or something like that maybe it would be nice to get something like that and PRAIS as well I think we found ourselves needing that exact functionality at some point I don't know if it was still required in the final product but it would be nice to have some of those things as well.**
Interviewer	Like the state of the channel
Participant A	Yes like a state of the channel
Participant A	So then, yeah you can actually also see like oh something before sending on a data channel before like whether it still setting up and you should wait instead
Interviewer	Yes in principle it should you should only get this data channel when it has been set up completely but I guess there are some scenarios where that's not always the case
	okay so basically more state information about datachannels and also media streams I guess, that would help?
Participant A	**Yes that would be nice**
Interviewer	Okay, that's a good one. Anything else you would like to see different in PRAIS?
Participant A	No not really
Interviewer	Okay yes the next question is very similar again but now more what do you think is still missing in PRAIS and why?
Participant A	**Yes there is one thing with especially with the focus on real-time it would be nice if there is a construct to easily say group or link some data to a certain set of video frames so say like you send one second of video and it gets a specific tag and you send one second of data over datachannels and it also get that specific**

	tag and on the other side you can then easily piece the two back together and see oh yeah the data is corresponds to that video
Interviewer	Okay so I guess on the data side it's already possible but on the video there's no tag attribute for example
Participant A	Yeah it would be nice to have something where you can easily get data and video of the same tag and group that together because I can imagine that being important when you because you actually need stuff to be synchronized to some extend
Interviewer	Yes that is a good one, I've been thinking about this indeed and it's not there but it would definitely be of value, good that you mention it, thanks.
Interviewer	Something else, so what else is there anything else that you feel like hey adding this would be really of value to PRAIS?
Participant A	**No at the moment I don't really, I think that it is pretty well rounded already**
Interviewer	Okay, thanks for that. Very useful insights. It's also nice to see that you are the third one and the other people already mentioned other things so it's really nice to get a different insights into what you felt is missing so I can really like implement in very specific things to address this
Participant A	Okay, good.
	Oh yeah there is the thing for terminating datachannels, but that something we already talked about that was a LiveSwitch limitation
Interviewer	Yes that is something. Yes I also discuss this with the other, it's a trade-off that I'm aware of and it's good that you mention this. It's a nice feature to have so I will definitely rethink this. Thanks for that.
Interviewer	We also discussed the documentation a bit already, but just to get back with what do you think about the documentation, you already mentioned examples are missing but may be the general impression, how is that?
Participant A	**I think it's nice documentation,**
	to me at least it explains pretty well what everything should do how we can use it. Especially going along with the examples.
Interviewer	All right, okay good to hear think you. I guess that covers all documentation.
Participant A	Yes
Interviewer	Next question is a tricky one you used LiveSwitch and PRAIS so you can compare them I guess. Maybe this question is not so much about how much time that you saved because it's like impossible to estimate of course but comparing the two, do you think that PRAIS saves you a lot of time?
Participant A	**Yes, I think it definitely saves us a lot of time, especially in the initial phase PRAIS is way easier to get the hang of than LiveSwitch.**
	So I definitely would use PRAIS for that. I remember getting like a basic example to run in like, I don't know under an hour and for LiveSwitch it was constantly digging into documentation some mild frustration and then only after quite some time you get stuff to work the way you want to.
	I think this is really where the simplicity of PRAIS really shines.
	And also I think in the long run it's nicer because I think the code for PRAIS is more maintainable because its directly almost directly business logic with PRAIS instead of LiveSwitch where you have just a lot of stuff that deals with having LiveSwitch work correctly. So I would say it's way faster.

Interviewer	Great great, that's nice. I feel the same of course. Good to see it also others have this struggle with LiveSwitch and that PRAIS really makes your life simple.
Interviewer	Then we are already at the last question, we are going really quick. So why would you or would you not recommend PRAIS to a friend or colleague?
Interviewer	I guess assuming well of course PRAIS is quite a niche it is a very specific application that you want to build with PRAIS but assuming a friend or colleague worked with a similar setting, why which are which not recommend PRAIS?
Participant A	**Well I would recommend PRAIS because it again going to save time with developing and like I said before it also in my opinion makes the code look neater and more understand able at that.**
	Lets see, more reasons to recommend PRAIS…
	Yeah, I found it nice to work with. Typical event driven design at least for what you are writing as business logic which is very nice.
	Yeah that would really be all,
	it's better documented than LiveSwitch that also a bonus
	so yeah it's I think in a lot of aspects it's superior..
Interviewer	It's funny that everybody so far mentioned that the LiveSwitch documentation really sucks
Participant A	Yeah we had a lot of struggles with that, that was not fun
Interviewer	Yeah, I know the feeling it's just trial and error and you hope the docs are right but sometimes they are just lying to you
Participant A	Yeah exactly
Participant A	All right, that's great that was already last question. That's clear. I maybe have one more question I'm not sure if you were actually involved in any of this. Typically you have this on video call or on audio and there are sometimes multithreading issues there because the frame is kind of kept by PRAIS on a thread so you cannot really modify it further. Did you have any issues with that?
Participant A	We did at some point I think run into an issue where we would store the video frame in a variable somewhere and then only later find out that, well since it is a reference that it would actually have changed at some point because well it was a reference to the thing that PRAIS kept. I think it's the only issue that we ran into for that, or at least is the only one I know of. Yeah we fixed this by just making a copy of the video frame.
Interviewer	Uhuh
Participant A	But we needed to keep it in memory anyway, so it was probably not really an issue.
Interviewer	All right okay. This is a very specific thing I keep struggling with. On the one hand you don't want to give a copy and on the other hand you do. Do you think if it would help if PRAIS would already give you a tread safe frame that you can just do anything with?
Participant A	Well I would say at first it might be more intuitive to work with but on the other hand if the user is going to make a copy of it for storing say for passing input to an algorithm later they are probably going to store it in their own format something anyway in that case it may only be a waste.
Interviewer	Yes that is also my concern and it's quite an overhead on memory and performance to keep to copy every frame you get
Participant A	Yes

Interviewer	Okay
Participant A	Yeah I would in this case especially since it's real-time focused maybe leave those concerns to the actual people building the application. That way they are in control of how much overhead they introduce.
Interviewer	**Yeah I guess maybe we need some more docs about how this works conceptually. If you know that the frame is not really safe to work with then you can copy it that would be nice maybe a good addition.**
Participant A	**Yes**
Interviewer	Any other remarks/comments/questions?
Participant A	Oh yes, out of curiosity… *A question irrelevant to the interview followed. Thereafter, the interview ended.*

Interviewer	Okay then we can start the interview in English. I have a question before I go into the actual list of questions, that's more about how you used PRAIS. So you each of you had a role in the team I guess. In what regard did you use PRAIS?
Participant B	I worked a bit on the bot side of the project which used PRAIS mainly but also quite some on the authentication part which was mainly the website. So I worked a bit with PRAIS but I didn't do like the main programming of the peers and the algorithms.
Interviewer	And that you do anything with LiveSwitch?
Participant B	I read some documentation but I didn't really do any programming with LiveSwitch.
Interviewer	Okay so you would say you have a pretty good understanding of PRAIS, what it looks like?
Participant B	Uhuh
Interviewer	But not so much about LiveSwitch, that's fine. There is one question that asks to compare them but I guess we can maybe skip it or maybe you can provide some insight.
Participant B	Uhuh
Interviewer	All right okay that's clear.
Interviewer	First question, overall how was your experience with PRAIS?
Participant B	**I enjoyed working with PRAIS.**
	Yes I will try not continue with the next question about what I liked.
	It's quite easy to use because of the documentation is clear.
	So of course you had to adjust the bit how does everything work in this setting in this context but
	as a library I understood it quite fast.
Interviewer	Okay very nice. That sounds positive. Yeah you already jumped into the second question, is there anything else that you would like to add. Other things that you think that are nice about PRAIS?
Participant B	**I also like that there was already an example present, the echo bot.**
	In the first week we could try to run that and then extend our idea from that. Otherwise I think it would be more difficult to know how to start and which functions you would need and
	by starting with such an example it was way easier to extend and understand how PRAIS was intended.
Interviewer	**Uhuh. Okay so the examples were really helpful?**
Participant B	**Yes. (belongs to above)**
Interviewer	Nice, okay that's good to know. Anything else maybe? Otherwise we can go to next question.
Participant B	I think that's it, yeah. Uhuh.
Interviewer	Okay then of course I would also like to know which things did you like the least about PRAIS and why?
Participant B	Maybe that it was like a NuGet package I'm not sure if there are like other ways you usually use to make libraries. But we had quite some dependency problemsm abd different versions so like with the acceptance test that was sometimes a bit of a hassle. When starting a new project then the dependencies were not correct but yeah I don't know if that is really PRAIS.

Interviewer	**So yeah NuGet package, you followed the installation instructions I guess. Any issues with that?**
Participant B	**That worked fine.**
	Maybe that was also more Visual Studio complaining rather than the NuGet package itself.
Interviewer	Yes some of your colleagues also mentioned this Visual Studio dependencies management is horrible. It sometimes always breaks randomly, you never know why.
Participant B	It seems like every member of the team has the same thing installed but that some way there are like three different types of dependencies in the background you don't know, or not really aware of how it worked.
Interviewer	Yes, I know the feeling, it's not fun.
Interviewer	Okay, so maybe anything else that you say I did not really like this about PRAIS?
Participant B	**No, not really. I was quite happy with how it works overall.**
Interviewer	All right, that sounds good. Thanks for that.
Interviewer	Yeah the next question, if you could change something about PRAIS, what would it be and why?
Participant B	I thought about you get already some message in the, like the box if so algorithm or peer joined, but if something goes wrong, I'm not sure if that's also an error which PRAIS tells. Or if the datastream is too slow, maybe PRAIS could do something with that because that would help with debugging.
Interviewer	**So, more logging?**
Participant B	**Uhuh, yes. (belongs to above)**
Interviewer	All right, yeah. At this point its not really nicely integrated with the LiveSwitch logging.
	So you think improving like the things PRAIS tells you about on what's going on, that would help you in debugging what's going wrong?
Participant B	**Uhuh.**
Interviewer	Okay, that makes sense. Maybe anything else that you say I would change this in PRAIS?
Participant B	No, not really.
	I think it's quite, the main functionalities are there and there is also not much more that you would want. You have like peers and algorithm and different streams. That kind of what you need. So I think that sufficient
Interviewer	Okay, so you feel that it is already complete?
Participant B	Uhuh, yes.
Participant B	**Like after that you had like a question is there is something missing… yeah a library could always have like more functionality and we looked into the authentication and adding a database and I think those are functionalities that a company would also use. So some interface for that would also be nice.**
	But yeah, if the PRAIS library doesn't cover everything you just take another library which does that stuff.
Interviewer	Yeah we of course want to offer everything, well not everything. But all the necessary stuff,
	okay so you mentioned authentication would be something to help by default like an authentication server you made.

Participant B	**Uhuh.**
Interviewer	Okay yeah I agree with that something we definitely want as well. At this point its not really secure with all the secrets that are stored locally.
Interviewer	Okay, so we already kind of covered the fifth question. Is there anything else that you would say I would also add this, because it seems useful? You mentioned logging already.
Participant B	No I think that's it.
Interviewer	All right. Then we go to the sixth question. What do you think about PRAIS's documentation?
Participant B	**(Documentation) I think it's good.**
	I also liked the conceptual figure. You get quickly and overview of what kind of functions there are in PRAIS.
	And like the documentation itself I only looked at that when I was using some specific methods and when I needed some information but looking through all of that is kind of difficult I usually find myself you don't know what you're looking for when you're just starting out so that figure in the example really help me to grasp what PRAIS was intended for.
Interviewer	Okay nice. Yes that's good to hear I put some effort into this figure thinking it would indeed have this purpose so it's nice to hear that it actually helps you understand the concepts better. So do you think the figure is enough or should also have some more explanation next to it?
Participant B	I think it's enough, you can assume that software developer that they are comfortable with reading such figures.
Interviewer	Uhuh
Participant B	So much documentation around it is probably not really needed.
Interviewer	All right, okay. Yeah that makes sense. I really try to balance the amount of text with the amount of actual information you're giving. If it becomes too much then there is no point, people don't look at it anymore.
Participant B	**Yeah it's not really a large model so I don't think you need additional explanation.**
Interviewer	All right, thanks for that.
Interviewer	So the seventh question is tricky I guess because you didn't really work with LiveSwitch on code level. Maybe based on some things you picked up within the team, what do you think regarding PRAIS versus LiveSwitch?
Participant B	**There was a lot of difficulty and also myself understanding LiveSwitch because of the limited documentation so I'm really sure that PRAIS saves a lot of time because the documentation was a lot clearer**
	and the main functionalities were already implemented for us. That was definitely the advantage of PRAIS.
Interviewer	That's good to hear, that's exactly what we want of course. That you just start, it works, tada, everybody happy.
Participant B	**And I also think you can work with PRAIS without understanding LiveSwitch in depth.**
	Which is also a good thing because they depend on each other but if you should be comfortable with both libraries completely then I would be a bit more hesitant to use PRAIS as is.
	But it was way clearer than LiveSwitch, so that's nice.

Interviewer	Yeah that makes sense. Well of course PRAIS kind of hides LiveSwitch, if you still have to know LiveSwitch that still doesn't make sense at all. One of your colleagues did mention that if something goes wrong that it helps to know what LiveSwitch is doing.
	But you already mentioned that the logging would already help a lot there.
Participant B	**Yeah**
Interviewer	So you can at least see if the mistake is on your end or something within PRAIS.
Participant B	Yes especially like eventhandling things are more LiveSwitch related and also when working with the authentication we had like a token and for that we also needed to dive a bit deeper into LiveSwitch. But when we started doing that we were already pretty comfortable with PRAIS so that was not really an issue.
Interviewer	All right, okay. That sounds nice, maybe anything else about PRAIS versus LiveSwitch? I guess you already mentioned a lot.
Participant B	I think that's it.
Interviewer	All right we are going really first we are already at the last question. So why would you or would you not recommend PRAIS to a friend or colleague?
Participant B	**I think I would recommend it to a programmer friend if they would use a similar project since I found it quite easy to use but its quite specific for media streaming, that's not really something that you use in every project.**
	So, I'm not sure if I would recommend it.
	I liked using PRAIS so I would use it again if that would be possible.
Interviewer	**Okay so I assuming that somebody says hey I'm working on a streaming web app/application something then you would say okay consider using PRAIS?**
Participant B	**Yeah, uhuh. (above)**
Interviewer	All right, okay. That's good to know. Thanks for that. Do I have any other questions…. Do you maybe have any all the questions left?
Participant B	No not really.
Interviewer	All right, then I think that already wraps up the interview.

Interviewer	Do you have any questions about the interview itself?
Participant C	No, the structure is clear the questionnaire is pretty clear I think you can start
Interviewer	Nice nice. I would like to start with the question that is not on the list it's more for my understanding, in which way you used PRAIS, I can imagine you each took a role within the team, bots, front-end, so how did you use PRAIS?
Participant C	I did more of an overall job, so I used LiveSwitch on the front-end a lot and did review a lot of code on the backend that uses PRAIS so I've seen both being used but haven't written much code myself using it
Interviewer	Okay so you have a pretty good idea of what the interfaces are like.
Participant C	Yes I have a very good idea, I've also read the documentation multiple times. I have an idea of all the interfaces I just haven't struggled with a problem and come up with a solution because I've reviewed the solutions, not so much tried to create solutions. Which we mostly did in LiveSwitch not so much in C#.
Interviewer	All right all right, nice to hear that the documentation was actually read probably more than I did myself
Interviewer	Alright let's go to the first question, overall how was your experience with PRAIS?
Participant C	**I think PRAIS is pretty clear,**
	you tried to keep the amount of classes low and I think that really shows in the clarity of our code. Because if you look at our code it doesn't seem that hard, it seems to be very easy to do things.
	Like a bot joins, a function gets called, you do new datastream, go. And
	it just seems to function in a more streamlined way than you would maybe expect on a technical level. That was my first impression.
Interviewer	**Alright so you would say the simplicity is nice. That it's very easy to understand**
Participant C	**Yes (belongs to above),**
	I could explain why all these classes exist to someone who doesn't know about video streaming. And I think that's the biggest strength of having like a low amount of classes that are very clearly for one goal.
	And even within the classes there aren't that many functions. There are enough functions to do what you want enough methods but they are not too many. So for instance for LiveSwitch you have some classes that have way too many overwriters and functions and methods I think in LiveSwitch there are 10 ways to do somethings.
Interviewer	Okay, yeah that's nice to hear. The main goal is indeed to go for simplicity to make it super easy to understand. One of your colleagues also said nicely I think with PRAIS you can immediately focus on the business logic instead of having to think about all the technical stuff behind it. That's what we want
Participant C	**I do think that the one thing that we did experience was, its not really the fault of PRAIS but it is remarkable that you still needed to know LiveSwitch in a way. At the very basic level you can just do things with PRAIS, but you very quickly came into like oh why is this error thrown, because the log entries are still LiveSwitch. Why did this happen, is it my fault or LiveSwitch's fault? Then you still had to go to LiveSwitch documentation to find out what happened and then the abstraction kinda got in the way because you couldn't diagnose if it was your part, PRAIS, or LiveSwitch.**

Interviewer	Okay, I think that makes sense. We still have some way to go in terms of the logging. **Because all the LiveSwitch logging is kind of not really taken into the abstraction layer. So I think we can really improve on that.**
Participant C	**Yeah**
Interviewer	Do you also think this is because on the front-end you had to work with LiveSwitch or was it purely like C# stuff that went wrong?
Participant C	**yeah I actually think that if you understand LiveSwitch, you using the front-end was sometimes easier to diagnose problems than through PRAIS. Because when you did it through PRAIS you were under the impression that you did it right and most of the time you did do it right, but some very specific use case wasn't the main focus of the method in PRAIS and then it bugs LiveSwitch or something.**
	Or when a bot joined and its IP was wrong, and PRAIS doesn't care about that part but LiveSwitch does and then you are like how does an IP get parsed? Where does that even happen? I don't know. That never happened in Javascript because you could research where it was.
Interviewer	Yeah okay
Participant C	**So you could find where the problem was because you called LiveSwitch. So that's the only downside of abstraction layers. But I don't think that a PRAIS thing specifically.**
Interviewer	So basically it's nice until it goes wrong. Because then you are like okay where does it go wrong?
Participant C	Yes
Interviewer	That's true, that's a trade-off I guess
Participant C	Yeah
Interviewer	Okay, then the next question more specific so which things did you like the best about PRAIS and why?
Participant C	**The clarity in explaining everything. So we had a team where some people haven't ever worked with C at all or C# and we also had people that were experienced. And someone could write some code and show it to someone else and they understood what it should do. Like the business logic part I think that's very powerful.**
Interviewer	Nice nice
Participant C	Very important
Interviewer	That's nice to hear. Anything else that you think is good about PRAIS and why?
Participant C	**I think it's a good testament that a group with some programmers who have never programmed before could actually build a web streaming thing in less than 10, maybe only five weeks.**
	And that
	our progress was really nice. I think that's, yeah, a lot of that has to do with PRAIS.
	because I think the major setbacks we had was testing which is a university thing. And I think all libraries have that, where you have sealed classes and you have to make all the mocks and stuff yeah that is just hard. There is no way to get around that. And then we had LiveSwitch on the front-end being annoying a lot, LiveSwitch documentation held us back a lot.

	So, the parts on the server where that didn't happen, because it just worked, it just easily was readable, solutions were clear and it worked. I think that saved us a lot of time.
	And I don't know if everybody noticed that but compared to LiveSwitch on the front-end we spent more time getting LiveSwitch to work even though the app was already there than building things with PRAIS I think
Interviewer	Very nice, okay. That's exactly what we want.
Interviewer	Mainly going into this mocking and testing, X mentioned this in particular saying yeah we had to add all of these interfaces to make it testable. What is your opinion on this? Do you think it would help if PRAIS would already include these kind of interfaces?
Participant C	**Yes, so in a sense returning interface types, or allowing interface types to be sent would be useful**
	because I think. The problem is not that we should create interfaces because we should for everything in the code. But the problem was that we had three classes I think translate our versions of everything into the PRAIS versions of everything.
Interviewer	Yeah I saw those
Participant C	That class shouldn't be necessary
Interviewer	Yes
Participant C	**It should just accept something that implements the interface instead of requiring to send a peer for instance. You should require it to send a peer interface, not an object instance. That would've saved us, I think some time because now we have classes that just take our peer and convert it to some fake version of the PRAIS peer or mocked version of the PRAIS peer that would not have been necessary if we could just send the fake one directly.**
Interviewer	Yes if you have the interface you can implement it fakely.
Participant C	It would have exceptions but we know what's going to be called and what's not. But I don't think it's PRAIS' fault that we spent a lot of time on testing. That's really the other end.
	But some interfaces would've been nice.
Interviewer	Okay, thanks
Interviewer	Let's look at the next question, so which things did you like the least about PRAIS and why?
Participant C	**.net framework. Very clearly .net framework. We have all vowed to never use Visual Studio ever again, ever.**
	Because Visual Studio just does such a poor job of actually managing itself. We have had literal message boxes which said only catastrophic failure. What happened? We don't know. And the dependencies, just dependency failure everywhere, that have nothing to do with the entire project, just dependency. People had to install VM's just to get Visual Studio there. So we had two people on Macs, they have been working in low memory VM's the entire time
	because it was only Windows. I think that does set back mostly the algorithm side. When you make algorithms, most servers run Linux.
Interviewer	Uhuh.
Participant C	**So, for a hospital, deploying a C# app on Windows may be a stretch. So, I would really consider moving to .net core. If possible.**

Interviewer	Yeah we actually designed PRAIS with like vendor abstraction so we have LiveSwitch that you use we also have ICELink which is also FrozenMountain stuff. There is an LiveSwitch version that is .net standard, but that doesn't really support all the features with video and audio so that's why we did not do it yet but I can imagine that in the future we want to do this so we can run on Linux for example.
Participant C	And then you also get the freedom of choosing your own C# method framework things around .net core. Because .net core can run as a separate thing while
	.net framework is very integrated with Windows and the entire tooling system. And they just set us back a lot, to have to use Visual Studio sometimes. That was the thing we like the least I guess.
Interviewer	Okay, anything else that you say this is not so nice about PRAIS?
Participant C	**That you had to know some things about LiveSwitch, but I already mentioned that before, just for the clarity to included here.**
	Are there more things… No no other things.
	I think the .net core, the .net framework it kind of captures all the problems we had. If you just switch from .net framework it would solve nearly all the problems. Because most things where just due to that.
	We had dependencies that were unclear or nuget packages, I don't think it's the best method of packaging a thing.
	I'm also very used to everything that's not C#. So C# was not my native choice, but I think their packaging system just caused us, you have seen this in the demo, it caused us a lot of random failure. Never occurred on any other PC ever and then you do it and you start up a VM and one VM is fine and the other VM has a problem. It's incredibly annoying.
Interviewer	Yes you don't want to know how much time I spend on getting all the dependencies right. It's horrible
Participant C	Yes, so it's not a PRAIS problem but it's a PRAIS dependency problem
Interviewer	Uhuh
Participant C	I think that's the major setback. It's that you don't control the server so you can't make your own full SDK.
	But it would've been nicer if there was just a PRAIS server for which you have a SDK in every language to talk with.
	But that's what LiveSwitch kind of does, poorly on C# apparently. I don't know how you should solve that maybe allow creating bots in node, so if you make a Javascript API for the website also allow it to run on nodeJS so you can make bots in nodeJS. Maybe that's one of the easier options to create other platforms.
Interviewer	Yeah it is really a future thing to consider. It's a good point because .net framework is kind of outdated.
Participant C	**I think I also answered question four, what would you change. I would probably go with Javascript..**
Interviewer	Javascript we are working on
Participant C	Or python
Interviewer	Okay anything else you would change in PRAIS?
Participant C	**Maybe add more interfaces but we already talked about it.**
Participant C	**Add the authentication server into PRAIS, make it like a normal part of how it functions**

	so you never. Also abstract away the LiveSwitch words, so we had client ID for the auth server because you have some client registered to the authentication system then you have a client ID for LiveSwitch but you also have a device ID you have a peer ID and they all have some meaning within PRAIS. PRAIS uses them in a specific way so the device ID for bot is apparently always the machine name. That something we didn't know and that's a choice. The moment you start going out of PRAIS, you need that info. That was something I would change I would make sure that you never need to. That principle of a device ID is never a thing because you don't make the token. You never do, so I think I would include that part. But you already want that yourselves as well. That would be something to change.
Interviewer	Yes I looked at your server's code and it looks really nice its well structured. Especially considering the future, I already talked to Zoran in the team like hey can we just replace our own authentication server with this please. Out of curiosity, this is more like a future work for ourselves. I guess you made most of the authentication server? So now you generate LiveSwitch tokens, as I mentioned we also have ICELink which is basically PRAIS using a different implementation. There we have different tokens that we generate ourselves. How easy would it be to change the authentication server such that it can generate different tokens depending on the API call.
Participant C	We now kind of struggled with the fact that LiveSwitch has some very specific parameters that it needs. For instance device ID. And I think that if you want to use it for multiple implementations, which would be very nice. You should either make some scheme that you know which is gonna generate, so it's not random anymore, it's some contract between the SDK and you. That you will know when you use LiveSwitch the device ID will always be something, something specific. Generated from some generic thing that both implementations could use as a parameter. Because otherwise you are going to have a lot of parameters. We have a standard contract, that's going to be in the software design document it's the API specification for the thing. I would make sure that that works for both of them and reduce the amount of parameters as much as possible. So if it is possible to make it the same for all the clients or something, do that. And never ask for it and just do that for LiveSwitch or guess it for a client or create it predictably or make it a toggle in the management panel, that's not included its some todo line somewhere. But if you create a management panel for companies to add their apps to the system, make them fill it in there for instance. So they can pre-provide it, that would help a lot. So what you could do is, if you toggle it to LiveSwitch in the panel, it would ask you: your new bot, what should we call it? And use that name on both the bot and the server for instance, then you already eliminate the device name.
Interviewer	Yes, yeah exactly
Participant C	Do some sort of thing with that
Participant C	Maybe the best would be if you could just provide a client ID a very specific unique device identifier because then you can have multiple devices and an authentication method, so either the ID token or the secret based thing. Then you would have only four parameters and we would know which role you get, would know which environment thing you get, you would know the user ID. So

	thats all possible if you have some prior information sharing and we did not have that so it was a lot of parameters.
Interviewer	All right okay
Interviewer	In my project I don't think I will be able to add all of this, I'm almost done, I will be done in October and there is also still holiday coming up. But I think it's really worth adding the server to PRAIS as an internal part because now having the secret locally is really not done.
Interviewer	That wraps up the fourth question. Then the fifth, is there anything that you think is still missing in PRAIS? And why?
Participant C	I think I'm less qualified to answer what features are missing because I didn't try to solve a problem with PRAIS so I didn't have the problem like oh I can't close a data channel for example so I can't give you those I think maybe X is the most qualified to give you like this method is missing and this method is missing.
Participant C	**(missing things:) But in general it's PRAIS logging I think. Just have PRAIS log useful messages because LiveSwitch does not.**
Interviewer	That's indeed something on the roadmap. Good point. Indeed the function calls that are missing are already mentioned by your colleagues so that's taking care of.
Participant C	Yeah
Interviewer	Anything else that comes to mind?
Participant C	Not really, no.
Interviewer	All right let's look at the six question, what do you think about PRAIS's documentation? You read it a few times
Participant C	**Very clear, nicely auto generated. That's always a plus because that also tells me that you made proper documentation in your code. Because it can auto generate this. I think it's clear,**
	maybe the only part that I would improve is the general concepts. So even though there is a diagram and it's in English so there is a data channel class what would probably create a data channel surprise. It may be a bit hard for some developers to understand. I think we did most of our understanding through just creating some random demos. And I think that's a good way to learn but not everybody wants to learn that way. So they want to know what can I do with a media source. Not even what is a media source but what could I theoretically do with it? What kind of things are media sources? And I think that mostly I think we also struggled with that on the front-end, we thought it would be really easy to send video, it wasn't. In that sense like what is a media source, is that dependent on LiveSwitch? For instance it says the type of media source, media source type, okay camera screen and custom, but what could a custom media source be? Is it anything that sense just frames? So maybe go into detail on what you can expect more. That would be an improvement.
Interviewer	I don't really explain like each of the classes in more detail right. I should probably add that.
Participant C	**(Documentation:) You explain what it does, but not what I could do with it. That would be the only improvement you could make.**
	(Documentation:) But I think it's more than sufficient now and it's amazing compared to what LiveSwitch did.
	So yes I think that's already pretty great and also the examples helped a lot.

	(Documentation:) It also shows how easy it is to create a thing.
Interviewer	Okay thanks for that. Thats some nice feedback and easy to add
Interviewer	Okay let's see the seventh question is tricky because well let's not really focus at how much time you compared to LiveSwitch did save you time how does that compare?
Participant C	I think if I look at the LiveSwitch documentation I've read it for Javascript but I'm also reading it in C now just to see.
	(Time saving:) Uhm, infinite probably.
	The LiveSwitch documentation is really poor, it also really assumes that you know why you are doing things. So it assumes that they know what a video stream is on like a bit level. Which codec do I want, I don't know I don't care. I just want some video stream okay. It goes into detail on enabling acoustic echo cancellation, I don't really care about that I want the library to fix that right. Whats an SFU connection? And then I have to read it, it says it stands for selective forwarding unit, still don't know anything. What's a selective forwarding unit? I think that even reading the documentation for LiveSwitch took most of the documentation reading time and I think that shows that, it's even though we haven't used LiveSwitch that's a major improvement because I don't have to select one of the 10 codecs to do it. And also it may be a downfall if the codec doesn't work, so I do then expect PRAIS to find out what works. That's the challenge but I think that if it does that then you are saving a lot of time.
Interviewer	All right. Okay that's good to hear. Let's see let's go to the last question. Why would you or would you not recommend PRAIS to a friend or colleague?
Participant C	Okay why would you, okay.
	For starters I would recommend PRAIS if someone was building an AI platform.
	(Recommendation:) The only reason I wouldn't yet is because it doesn't feel complete.
	So if I build an app for ING and I go to the ING website for developer documentation I get 10 million pages with the most detailed thing and I get a developer panel and I can add my app there and I can go go go. I get a base project and
	here it seems that the product works but I can't download it yet, I can't create an account yet, I have no panel to register my app I just get the C# code from some out of band communication and I think that the only thing that's.
	And it's not within the scope of what you're doing I guess so it's not at all related to the project scope but after this project is finished. Philips should make a clear developer push developer marketing thing for this. I think then it feels more complete as a product not the project.
Interviewer	Yeah I agree of course we are in Philips research so we kind of do it on the spot and improvise but indeed if you want to offer it as a proper platform there is much more that we need to add. Let's see if that happens in the future, would be cool.
Participant C	**Yeah it's more of a very very long proof of concept technical demo, at least what it seems to me from the outside and I don't think that's bad, but before you sell it you do need some more things around it.**
Interviewer	You mean that SEP was a proof of concept?

Participant C	**No I think PRAIS itself is because you have the, you don't have some, yes some missing parts so I can't create my own secret I can't create my own conference at the moment because that is missing it's more of a technical demo even for us as users. Than it is a full platform yet.**
	And it's not for PRAIS specifically, it's not that hard to transition, I also had other projects where I could make the technical demo in an hour and the entire implementation to three months. So for PRAIS its not that much of the step, its adding a developer panel, its having you create your own conferences, link some bots in the panel and then you get all the secrets and you put them in. That's it. That would solve the entire thing. Right now it's too much of an insider thing to recommend I think. But again that's not within the scope of what you have to do I guess.
Interviewer	Yeah that's true, but it's also true that it's not complete yet. So its a good point. Okay that wraps up the last question. Do you have any other remarks comments questions?
Participant C	I guess not.

Interviewer	Okay we can switch to English just to keep this consistent with the others. Yeah there is one question I would like to ask that not really on the list it is for my understanding , how you used PRAIS?
Participant D	Have you already looked at the code that we supplied? Just as a question
Interviewer	A bit yes
Participant D	**As you may have noticed, we didn't really use PRAIS the same way anymore as you prescribed in the documentation mainly because it allowed us to be able to test PRAIS with unit testing**
	So we created some sort of helper class and we did dependency injection instead of extending algorithm base with the actual bot. Besides that, we used PRAIS well the main functionality that we used in PRAIS were event listeners. Going onPeerLeft onDataChannelOpened, onVideoReceived, onDataReceived, and of course we send some data strings over datachannels and we use the custom media stream I think those are the parts of PRAIS that we used
Interviewer	Alright so it was mostly for you on the C# end and yeah I looked at the code and I was indeed seeing this whole basically copy of the PRAIS interface that you added to make it testable
Participant D	Yes
Interviewer	Okay that makes sense
Interviewer	And you didn't really work on the front-end, it was mostly algorithms?
Participant D	What do you mean front-end?
Interviewer	The web apps
Participant D	No, we didn't really use PRAIS a lot on the web apps mostly I think its the LiveSwitch demo
Interviewer	Yeah, so what I mean is you developed on the algorithm side or also on the web apps?
Participant D	Oh, I developed on both
Interviewer	So you also have experience with LiveSwitch?
Participant D	A little bit, I didn't really look too much at the LiveSwitch specifics
Interviewer	Okay, good to know
Interviewer	I think that's clear let's go to the first official question so, overall, how was your experience with PRAIS?
Participant D	**Overall I think that I had a good experience with PRAIS.**
	PRAIS is really easy to use and I don't think we encountered any real bugs with PRAIS.
	Which was really nice also I think that the documentation of PRAIS is quite good especially when you try to compare it with LiveSwitch.
	Yeah, good documentation also helps that you added these XML comments really helps the IDE to highlight them and just allows for faster development.
	The two things that were a bit less when using PRAIS were I think the exceptions sometimes LiveSwitch throws an exception and PRAIS catches it but does not propagate it. This makes it somewhat uneasy to deal with exceptions and
	testing was a bit less easy.
	But I can elaborate on that later.

Interviewer	Okay so you have some cases where for example I modified PRAIS a bit for the datachannels to throw exceptions and I guess there are other examples where you have looked into the logs to see that there was something going wrong while you did not see that on the PRAIS level?
Participant D	Yes, sometimes LiveSwitch prints these errors in the console but in the actual code runtime you cannot catch any exception or very easily detect that something went wrong and then deal with it
Interviewer	Yeah okay because it's all internal and some kind of DLL that super deep
Participant D	Yes
Interviewer	Okay good to know
Interviewer	Yeah I guess we will cover the specifics in the remaining questions. So, anything else on the first question?
Participant D	No I think that was it for the first question
Interviewer	Okay
Interviewer	So, which things that you like the best about PRAIS? And why?
Participant D	**I think that what I like the best about PRAIS was the simplicity.**
	There was not a lot of when you look at LiveSwitch you have like lots of functions and some functions do similar things but you're not sure how they differ and that kind of stuff when you look at PRAIS you just have a couple of functions they are really clear in what they're supposed to do and it's really easy to use.
	Just you are looking for something, something easy like send datastring. There you have some function send it synchronously send it asynchronously just really straightforward to use. I think that was what I liked most about PRAIS
Interviewer	All right, so it's mostly simplicity
Participant D	Yes
Interviewer	Okay, nice nice
Participant D	**Like if you want to create simple applications with web RTC then this is really the way to go if you ask me**
Interviewer	And what do you think about more complex applications?
Participant D	**Well because PRAIS contains a lot of sealed classes and not a lot of interface or at least not a lot of interfaces that are public within the library it doesn't really allow to modify PRAIS so when you use PRAIS you should really stick with the PRAIS functionality mostly. So, if you have to do stuff that is a bit different than what PRAIS does then, I think it easier to just use LiveSwitch instead.**
Interviewer	All right, so yeah the sealed classes was intentional indeed to the basic idea behind it is that if you extend a class and I change PRAIS. Then your application may break
Participant D	Once it changes yeah
Interviewer	That's why we seal it to basically make sure okay that extending shouldn't be needed.
Interviewer	Okay, so it's mostly you're limited to the PRAIS interface of course and then in case you need something else you okay yeah.
Participant D	Yes
Interviewer	Anything else about what do you like which things that you like the best
Participant D	**Well simplicity and the documentation. I think those are really the things that were the best**
Interviewer	All right, great, thanks for that
Interviewer	Then we also of course would like to know which things that you like the least about PRAIS? And also why?

Participant D	I think I already mentioned them so it's twofold.
	On the one hand there is the exceptions, they are sometimes catched by PRAIS but not propagated I would really like to see them propagated in some way either just propagate the actual exception or throw a different one something like that.
	And also of course document these exceptions with the XML documentation and on the website.
Interviewer	Okay
Participant D	So that's for the exceptions. I think that's what I like the least. Secondly,
Participant D	**It's not really easy to test,**
	I didn't really realize yet that there is a good reason to have sealed classes which I agree with but perhaps it is a good idea to create interfaces for the sealed classes to allow the user to or the developer to substitutes those sealed classes for another one. So for example, when you want to test whether sending a message or whether a message is sent or something like that when you receive a message you can create a modified version of the data channel and use that one instead. Like I think it's called the strategy pattern, I'm not sure
Interviewer	Yeah it's a long time ago that I learned about these patterns. But, okay, so in general you are saying that having the ability to extend the classes would make testing much easier but also making your own custom data channel these kind of things?
Participant D	Not really extend, extending classes may be useful at some point but it's not really necessary. I think there is also no scenario in which you really want to change the behavior of the class you either want to replace all of it or you want to extend it and then you can also something else you can create some sort of wrapper before or after calling the function on the data channel you can do some other stuff, so extending is not really necessary, I think. So they can still be sealed, that's fine. I think it's actually better when you think about it.
	But having interfaces for these classes and somehow being able to substitute them that would be useful.
Interviewer	Okay, so you define the interface and that can change of course but you can put different objects basically behind it
Participant D	Yes
Interviewer	All right, yeah
Interviewer	Yeah I so in your code I saw that you have like an interface for every peer, media stream, all the PRAIS concepts you had an interface. Okay, good point, thanks for the feedback. Very useful to get it
Participant D	It's really just mostly for testing that we did that. For the functionality it's not really necessary. But could still be useful sometimes I suppose
Interviewer	This is exactly why I think this interview is nice to do because I never, while I test PRAIS itself but I never test applications that use PRAIS, so it's good to get this feedback like hey, you basically had to make all the interfaces yourselves while it could be part of PRAIS of course.
Participant D	Yes that would be nice
Interviewer	All right,
Participant D	Also, because we created those interfaces and of course we want to keep the work that we do minimal but maximal functionality so we only created interfaces for the parts of PRAIS that we actually used. And not the other parts. Which has somewhat limited our use of PRAIS indirectly

Interviewer	Yes that makes sense of course, you are not going to make an interface that you will not use
Participant D	Uhu
Interviewer	I guess that if we offer this from PRAIS then you are free to use whatever you like. That should be fine.
Interviewer	All right, any other remarks regarding things you did not like?
Participant D	No, no other remarks
Interviewer	Okay, great. Thanks for the feedback, very insightful useful
Interviewer	Then the next question, so if you could change something about PRAIS, what would that be? And also why?
Participant D	**So one thing that I would change I already mentioned is the exceptions.**
	I don't think I need to repeat that.
Interviewer	No
Participant D	**One thing, something that I would change or something that I would add are the interfaces.**
	Which I'm also not going to repeat further
Interviewer	Okay
Participant D	**Something else that I'm not really sure about but I had some doubts about it. To get the PRAIS functionality you have to extend algorithm base. I'm not really sure if that's like the best way to do it or whether there is any design choice behind this.**
	Maybe you can explain is there a reason you chose to extend algorithm base or
Interviewer	Well, initially the idea was that okay you have an algorithm and all you have to do is make that algorithm extend algorithm base and then tada you have a lot of functionality. But, actually like two weeks ago I was also working on a bot myself and I did not really have this extension requirement and I think I'm just going to make it non-abstract so you can also just instantiate the class having to extend it.
Participant D	Okay
Interviewer	Because now we are basically limiting the user while the user should be free in extension or just instantiate it and use it.
Participant D	Hmhm
Interviewer	I guess that answers your question?
Participant D	Yeah, mostly it does. This also relates a bit to the interfaces as discussed earlier. But having to extend algorithm base means that you're stuck with the actual implementation that requires network activity.
	So if you extend algorithm base it's not testable.
	That was our main concern with that I think. So, I think that if I had to change this then I would create an interface that contains the PRAIS functionality such as open data channel and also has some way to register event listeners, for example on peer connected etc.
Participant D	And then have some concrete implementation that is provided in PRAIS so it's more like dependency injection that you do. You can test, yet it just allows you to test your functionality or algorithm essentially
Interviewer	Yeah yeah because you can have the interface in your test and you only provide the functionality for the functions that you need.
Participant D	Yes, but like you can have the actual implementation PRAIS to run the algorithm in production scenario but when you want to test you may want to use something with

	the same interface but with different functionality so that for example you don't require network activity and that would be useful.
Interviewer	All right, okay, so also I guess again testing would be much easier if you don't have this class but an interface that you can use yourself.
Participant D	Yes
Interviewer	Okay
Participant D	So it's like, when you construct the algorithm you pass some object that contains the PRAIS functionality and then the algorithm expects something that extends the interface.
Interviewer	I guess this nicely aligns with the other feedback on okay having interfaces for every PRAIS concept essentially would really help in the testing especially
Participant D	**Yes, overall I'm really happy with PRAIS**
	so don't get me wrong
Interviewer	No no no
Participant D	**But, the testing is a bit more difficult**
Interviewer	This is the part of PRAIS that they never had to do myself so it's nice to see that there are some limitations and that we can probably make it easier by having these interfaces.
Interviewer	All right,
Participant D	Yeah it's nothing structurally wrong with PRAIS that can't be changed. That would be at the cost of other things I think.
Interviewer	Yeah it's of course always design and trade-offs. This was the, you are the first group that uses PRAIS. It was like a first usability test to see if there are any bugs or issues and I am already very happy that there are no major bugs only like little usability things that we can improve. So that's actually very nice result I think.
Participant D	Uhuh
Interviewer	Okay, so that was the question. I guess the next question is very similar but this is more about missing stuff. You already address some things like interfaces, exception throwing. I guess there is, maybe there is something else? That you feel like is missing?
Participant D	Yes, there is one thing that we didn't really need in our scenario but I can imagine that it may be useful in some other cases. Imagine having an algorithm that is in the conference with a load of peers and it sometimes does require to send or receive data or video with some of these peers. But not all of the time, then it would be a bit of a waste to have data channel and a media stream open with every peer all the time. And I think that in PRAIS at the moment there is no real functionality to close datachannels and media streams. You have told a bit about the limitations in closing datachannels. But maybe for media streams it's really useful to be able to explicitly close these as well. Then you can say, hey I need to send data to this peer now just a little bit, I'm opening a media stream and when you're done you close it again. So that when these or when you want to receive something at some point you're not receiving all these bunches of data and you can yeah lower the load a bit.
Interviewer	That's a good point, actually media streams are quite resource intensive especially encoding audio video is really CPU intensive. There is a function that can close a media stream so I think that's already there but indeed datachannels is not present. Luckily datachannels are quite cheap
Participant D	Less resource intensive

Interviewer	They are like always open in the back you don't really see that but that's what's happening and yeah I still think it should be would be nice to have a ways to also close datachannels explicitly but at this point it's just cheaper to not have it. But it's a good point, I think for media streams it's already taken care of.
Participant D	Okay, yeah we didn't need it in our project so I didn't really look into it. But if it's present already, then that's great.
Interviewer	Thanks thanks
Interviewer	So, anything else that you think is missing?
Participant D	No I think there is not really much missing for the rest.
Interviewer	Okay, perfect
Participant D	Not that I encountered at least
Interviewer	Hmhmm
Interviewer	Okay, then the sixth question you already also briefly talked about it. So what do you think about PRAIS' documentation?
Participant D	**Yeah, I think the documentation is really nice.**
	So, it's well-documented functions have their descriptions about what they do and what they return.
	They have XML comments which is useful in the IDE.
	Only the thrown exceptions that you added at some point they aren't documented or not completely I think but I think that's the only thing that I would change.
	For the rest, it's just really good
Interviewer	Okay, good to know, I put quite some effort in this so it's nice to see that it's actually being used. Very nice. I guess I should indeed do another sweep through it and see if all the exception docs are up-to-date because I changed some things there.
Interviewer	Yeah, I guess that covers the documentation question.
Participant D	Yeah I think so too
Interviewer	Then a more difficult question, how much time, as you already said you did not really work too much with LiveSwitch, but comparing PRAIS versus LiveSwitch how much time do you think you saved by using PRAIS instead of using LiveSwitch.
Participant D	Okay, I think that you also have an interview scheduled with Bart and Bart should be able to better answer this question than I. He worked a bit more with LiveSwitch in the web apps, yeah,
	I looked a little bit into the LiveSwitch documentation and it was really horrible. So, yeah, like LiveSwitch has lots of functions, lots of functionality but looking at the documentation you have no clue what it does. So, I think that this would save me quite a bit of time. Maybe about 30 hours in this project I guess.
	If I had to implement all this from scratch but but this estimate could be quite off because I'm not sure how much time it takes to get that part of the PRAIS functionality that we used from using LiveSwitch.
Interviewer	Uhuhh
Interviewer	Yeah I guess estimating this is super difficult especially in terms of hours but indeed the general impression is that LiveSwitch is much harder to know what it's doing. I also have the same experience actually that looking at the docs you have functions

	and you have to guess what they do. You mostly learn it from their code examples. Okay, and you basically feel the same that
Participant D	**It did save me time, I'm sure about that. How much, I don't know exactly but it did save me time.**
Interviewer	That's good to know, that also the goal of PRAIS. To all of this difficult networking and streaming stuff is hidden.
Participant D	I can imagine that I'm not really sure about this but I can imagine that when you use LiveSwitch you have to take care of the encodings yourself do that somehow and when you use PRAIS it's just done for you
Interviewer	Yeah, there's a lot of, yes encoding is luckily done by LiveSwitch itself but there is some certain connection setup, message exchange you have to do. Especially to bring it all together into the PRAIS interface with peers and streams in these different types of connections that's where the most work is.
Interviewer	And also, PRAIS was designed to, just for your info, not really relevant, but now we use LiveSwitch under the hood but we also have ICELink which is also by FrozenMountain.
Participant D	Yes, I noticed that in the documentation you have this implementor class that you can more or less switch
Interviewer	Yes, so
Participant D	Or will be able to switch
Interviewer	Exactly that's what I was working on for the last few weeks I added ICELink as well so now you can basically use LiveSwitch and ICELink because we have this Javascript API, thats basically what you did in the browser but then also nicely abstracted and easy to use. But that uses ICELink, so now that it all integrates nicely and PRAIS as all becomes available in the browser and for algorithms.
Participant D	That's nice
Interviewer	Yes,
Interviewer	Let's go to the last question, yeah, so, you probably entered this in the questionnaire but why would you or would you not recommend PRAIS to your friend or colleague?
Participant D	**Okay, so if my friend or colleague has to do relatively simple things with LiveSwitch with not too many peers I think or just send receive some data processes it nothing really fancy then I would really recommend PRAIS because it's very simple easy to use.**
	When you need to do things that PRAIS cannot do at this moment then I think I would recommend not to use PRAIS but use something that's easier modifiable. Yeah.
Participant D	So it depends a bit on the goal of the friend or colleague.
Interviewer	So mostly, it's again like you said complexity wise PRAIS seems to supply functionality for relatively simple use cases
Participant D	Simple use cases but it does cover most use cases. But there is always this, it's also for example when you try to create visualizations. It's really easy to you just use Excel for things and that works really easy for most things. But, when you want to modify it a little bit it becomes very difficult really quickly. So, then you're better off using either no some python library seaborne something I that I think it's kind of the same with PRAIS,
	when you want to do basic things, PRAIS is great and it covers most scenarios which is really nice.

	So mostly I would just recommend PRAIS, but when you have to modify the behavior or do something different then probably not.
Interviewer	Okay, make sense.
Interviewer	I guess you also of course with SEP you were kind of forced to use it but this is all assuming you have a similar future scenario or you need streaming of audio video data?
Participant D	Yes
Interviewer	Okay, I think that's clear. Maybe any other final remarks? Where done with the questions.
Participant D	I don't have any further remarks regarding usability study.
Interviewer	Okay, then thank you very much.
Participant D	You're welcome

Interviewer	So let's switch to English and just start I guess
Participant E	All right
Interviewer	So I have a question to start of that not on the list actually. This is more for my understanding so I would like to know you used PRAIS but in what way? What part of PRAIS did you use in your development? Because I guess each of you had a separate role in the team?
Participant E	I personally did not work with PRAIS that much. PRAIS was mainly used at the bots side of things. Mainly when those were being set up for the first time, I think that was mainly I think X, K, and C who worked on those parts. I worked with elements of PRAIS for a bit. When trying to do the video uploading although that was more a web app side so I had to deal with LiveSwitch which I don't ever want to deal with LiveSwitch shenanigans again.
Interviewer	Okay, so you kind of had to connect to PRAIS when you were working on the web apps
Participant E	Yes, I had to connect to PRAIS I had to make sure that it was sent over correctly. Using a media stream
Interviewer	All right, okay. So no direct development with PRAIS, more indirect?
Participant E	No, I did not directly use PRAIS. I used the PRAIS LiveSwitch dependencies of PRAIS
Interviewer	Okay
Participant E	To make the functionality happen
Interviewer	Okay great, that's clear
Interviewer	Yeah, so I guess for most of the questions it means you can draw from all those dependencies yeah how you had to use those or the documentation maybe you read those
Interviewer	So the first question is how was your overall experience with PRAIS?
Participant E	Again,
	I didn't work with it that much but generally pretty positive. I noted that the functions that it's supposed to do, it does them very well.
	I think audio quality is generally very accurate, the video quality, there is pretty much no deterioration in that as far as I can tell, images are clear images and crisp.
	It's not that hard to use.
	Pretty much all the functions you want from it are already there.
Participant E	**There is authentication to a degree using the secrets, it's not the most secure thing in the world. But it's definitely better than nothing.**
Interviewer	Yes we are indeed working on that. The authentication server that you made is the first step to make it all more secure. Okay nice to hear that there is also some experience on the audio video quality side. That's good to hear. Any other comments on the overall experience?
Participant E	How do you mean just trying to work with it?
Interviewer	Yes
Participant E	Not really. The interfaces,
	we had to create some interfaces to perform some class construction so we can not like have all our classes depend on PRAIS which makes them pretty much impossible to unit test.
Interviewer	Yes

Participant E	**So we had to write some interfaces for several things so we could use more classes and generally make testing possible at all.**
Interviewer	Yes, X mentioned indeed that for testing purposes you had to create a lot of interfaces.
Participant E	Yeah
Interviewer	All right
Participant E	I personally was not the one creating the interfaces I think that was X again. But I did have to use them a bunch because I wrote a lot of test cases that involved those kinds of those interfaces.
Interviewer	Ah okay
Participant E	I was involved mainly in testing during the later part of the project.
Interviewer	All right so you have some experience with the C# interface
Participant E	Yes
Interviewer	All right okay nice
Interviewer	So let's go to the next question which things that you like the best about PRAIS and why?
Participant E	Okay I will first give an unserious answer which is
	what I like the best about PRAIS is not having to use LiveSwitch.
Interviewer	Hahaha all right. I get that
Participant E	Now for more serious answer. I think what I like best is it's always hard to define what you like best.
	I would say that as an overall experience it was fairly easy to set up and install. Especially for someone who has no experience in both C# and typescript which are the two languages that we used.
	So that was very steep learning curve for me.
Interviewer	I can relate to that. Okay nice
Participant E	So it was especially nice to have
	the thing that really helped me understand how PRAIS works are the examples that were on the PRAIS website.
Interviewer	Okay nice
Participant E	**I played around with those a bit. I got those to work pretty quickly.**
Participant E	**Except for the Microsoft Azure one.**
Interviewer	That one is more of an issue because you need the Azure key
Participant E	Yes and I don't have an Azure key
Interviewer	Yes that's the biggest disadvantage but it's good to hear that the examples actually help.
Participant E	**Yes, it might be good to have maybe one or two more examples that are a bit more elaborate.**
	I think it would be echo bot right
Interviewer	Yes that's a super basic example
Participant E	**Yes it's a really basic example. It shows the idea behind PRAIS very well but I think leaves a bit too much to the imagination to really give you an idea of like this is what it is really capable of.**
Interviewer	Yes, I think we can add some more complex examples with some more streaming maybe two other algorithms as well.
Participant E	Yes maybe one or two more complex examples…

	I really like the fact that there were actually like examples that are like well explained.
Interviewer	Okay
Participant E	**There is a at least a simple example that as well explained. It helped me understand it better.**
Interviewer	Great sounds good. Thanks for the feedback
Interviewer	Any other thing that you like best? Otherwise we can go to the next question.
Participant E	Not really anything else coming to mind at this point.
Interviewer	Alright let's see which things that you like the least about PRAIS and why?
Participant E	Things I like the least… I haven't worked with it very extensively. But, I did hear from other people that sometimes PRAIS expects very very particular input in some cases. I don't quite recall what exactly the issues they ran into were. But it was some issue where something should have been more general should've accepted a more general input but instead it required something very very specific which made it again difficult to abstract from that.
Interviewer	All right, okay I'm also not sure.
Participant E	I don't quite recall what that was.
Interviewer	If it ever comes to mind. But I guess somebody else will probably mention it
Participant E	Again, that may have been a LiveSwitch thing. I don't know.
Interviewer	Yeah, I'm not sure
Interviewer	Except for that, any other things that you think this was not nice PRAIS could be better?
Participant E	About PRAIS.
	Not really, there weren't really any things that jumped out to me is like this is bad. This needs to be improved or changed by like significant amount
Interviewer	That's nice to hear, okay
Interviewer	Very nice feedback of course to hear. Especially you guys are the first group that's actually using PRAIS to its fullest. So overall I think it's quite successful considering that you finished all the requirements which I'm very impressed by
Participant E	Honestly I'm pretty impressed surprised by that as well. It was generally a nice surprise do not have any or basically any stress when it came to the project until like two weeks ago when we had to basically finish the code. There were a few things that still needed to be done that weren't and they needed to be done rather quickly or the acceptance test.
Interviewer	That's normal I guess in any project.
Participant E	That was pretty much the only time we had to like really rush to get things done.
Interviewer	Okay, sounds still like a pretty decent planning timewise.
Participant E	I suppose that may be caused by people being at home and having nothing better to do.
Interviewer	That's also a bit weird yes in this case working from home with remote collaboration.
Interviewer	Okay so let's see If you could change something about PRAIS, what would it be and why?
Participant E	**Well I briefly mentioned this changing the way secrets are handled.**
	I don't know if you can just use any phrase as a secret. You just write password 123 and that counts as a secret. I don't know if that's particularly true
Interviewer	Yes you could

Participant E	**But support for a more advanced way of authentication would be a good idea.**
Interviewer	Uhuh
Participant E	There was anything that I told of but I slipped my mind just then. There was something, if I remember I will bring it up.
Interviewer	Okay sure
Interviewer	So authentication again that's indeed something we are looking into more advanced ways after authentication I guess you already implemented that yourselves so that's nice. Good point.
Interviewer	What do you think is still missing in PRAIS? I guess that is pretty similar question to what you already mentioned regarding authentication. Is there anything else that you would say this would be nice to add?
Participant E	Let me think. I honestly have no idea. I think I'm going to blame this one on my like of experience with software design. I wouldn't know like what functions would be like very helpful to a programmer or. It difficult for me to come up with something that I have very little experience with
Interviewer	That's perfectly fine, you already mentioned that you didn't work too much with PRAIS so I guess this is a tough one. I mean it doesn't need to be on the functional level, it could be anything. I don't know. But if you at this point can't come up with anything that's fine.
Participant E	Yeah it's probably an implementation thing but if someone else with a video feed would connect to this conference I assume their video feed would also be shown here.
Interviewer	Yep
Participant E	Yeah okay
Interviewer	Yeah so indeed authentication really needs to move to the server and that's what we are trying to do you can only join these kind of conferences if you have the tokens that you got from the server. And now anyone with this URL can join this session. So, I hope nobody will join.
Participant E	Oh wait, I think I remember like a thing that I thought of to add maybe.
	I believe at some point we ran into a thing where we wanted to reliably open and close datachannels but there was no method for that.
Interviewer	X mentioned that closing datachannels is indeed not provided functionality. Media streams can be closed, datachannels not really. That's a limitation in the current implementation indeed. As a trade-off datachannels are quite cheap to keep open so we don't really mind if they stay open. And adding the closing functionality would make each individual data channel more expensive to keep open.
Participant E	That is true, but on the other hand they would also not be open for as long. Because you can just close them when you don't need them instead of having to close them automatically.
Interviewer	True,
Participant E	I suppose for like short bursts of usage, such functionality would be preferable but for a very long period of time you would want a l lighter, a lightweight version.
Interviewer	Okay
Participant E	I haven't done the. I honestly have nothing to base this of but I think it would be nice feature to maybe have it is an optional toggleable datachannels or something.

Interviewer	Uhuh, that's a good feedback. I think it's design wise also weird that media streams can be closed but datachannels cannot. So good that you mention this. I will think about adding it. Thank you.
Interviewer	If there is nothing else regarding potential additions. Then the next question is about the documentation. So what do you think about PRAIS' documentation?
Participant E	**(the documentation) It's better than LiveSwitch, I can tell you that much.**
Interviewer	Haha I heard the same from many other people. So that's nice
Participant E	That I say this mainly because LiveSwitch documentation is really really bad. It has approximately 80% of functions in there which is not enough and those functions have no descriptions which is bad. And it doesn't even say that some functions are deprecated while they really really are. I asked a friend of mine, he is also studying an IT study. I asked him like what do you think would be the difference between setMediaSource and setCustomMediaSource? Like I don't know, probably nothing. No, setMediaSource has been deprecated for like two years. But it doesn't say that anywhere.
Interviewer	Oh okay, interesting. That's LiveSwitch.
Participant E	Yeah
Interviewer	I know, the documentation is not really up-to-date. There are some functions that exist but are not mentioned in the docs.
Participant E	Yes, which is incredibly annoying.
Participant E	**But, no, in general the documentation for PRAIS is quite clear.**
	I think everything is present, I haven't found any functions that do exist but aren't in the documentation.
Interviewer	Uhuh, okay great
Participant E	**Most of them are self-explanatory. Something like media source.stop(), I just pulled it open right now the documentation**
Interviewer	Aah okay
Participant E	**Yeah, most things are pretty much everything is like very well specified. Every method has a description of every class has a description. It says how to use the methods, says what a method returns. With even a description of the type that it returns which is something you don't see all the time. Especially helpful if you have a lot of custom classes custom records custom datatypes.**
Interviewer	Uhuh yeah.
Participant E	**I suppose the only thing that I would add would be something like maybe another example like I said before that really shows you more how to use a specific methods because that's not always clear from just the documentation**
	but that goes for pretty much any API or program language.
Interviewer	Uhuh
Participant E	How to use a method is not always clear from just looking at what the method like the how you call the method.
Interviewer	Yes, you need to more context in which scenario would you use the method
Participant E	Yes, that's why I usually go to the Internet for like pretty much every method if it's not exactly clear how to use it
Interviewer	Yeah, okay. So more examples would definitely help
Participant E	That is something I would like to see personally. It helps you understand and it really eases up on troubleshooting. If you have like an example that you know that works.

Interviewer	Uhuh, okay, clear thank you.
Interviewer	Anything else from the docs? Otherwise we can go on
Participant E	**No, (the documentation) they are very complete as far as I can tell. That's all**
Interviewer	Okay, thank you
Interviewer	Yes, so, I guess you use LiveSwitch on the front-end and PRAIS not too much on the C# side. Still, could you try to estimate whether or not you saved time using PRAIS and how much?
Participant E	I can probably estimate a bit. LiveSwitch has the functionality behind LiveSwitch is actually pretty decent although I'm constantly bashing the documentation. LiveSwitch is kind of confusing to work with. Because there are duplicate functions that you don't know about or you do know about but you don't know that they don't work.
	So, using PRAIS is definitely preferable.
	I think in hours just setting up the bots I think that took about a week. That took one sprint to set up the bots, it was like two people so let's say that took 40 hours this time. I think it would've taken at least another week to set up the basic framework for the bots such that they can connect to the conference. I think it would've taken at least another week. There is so much more research to do, so much more trial and error.
Interviewer	Okay, here. I guess that estimating is super difficult because you of course never saw LiveSwitch in C#. But you can say that PRAIS did save you time?
Participant E	Yeah, although maybe it's not like a full week because as far as I'm able to tell the LiveSwitch C# API is actually better than the typescript API.
Interviewer	I think so too, yes
Participant E	Because there are functions that are possible in C# but not possible in typescript.
Interviewer	True
Participant E	Which is frustrating. So, I think it would take approximately like 20 hours more so it would be 1 ½ weeks something. 1 ½ weeks by two people
Interviewer	Uhuh, okay
Interviewer	**Yeah, sounds good, I guess the main point here was to check if you think PRAIS actually saves you time.**
Participant E	**I definitely think so**
Interviewer	It is really hard to say how much.
Interviewer	All right, then let's go to the last question. So why would you or would you not recommend PRAIS to a friend or colleague?
Participant E	Well PRAIS has a, it's a very niche set of people that would use PRAIS. So like the chances of that occurring would be like not very high.
Interviewer	Uhuh
Participant E	Because you would have some building exactly a real-time communication application that makes use of code that you can't easily share.
Interviewer	Yes true, it's a very specific application. So let's assume you have a friend or colleague that says say I'm going to build something with streaming, would you recommend or would you not recommend PRAIS?
Participant E	**Well the first thing is I would say please don't use LiveSwitch.**
	The second thing I would say is hey I worked with PRAIS this one time it was pretty good you should try looking into that.
Interviewer	All right, so definitely like a recommendation to look at it.

Participant E	**Seven or 8/10, like I wouldn't be able to actually convince them to use PRAIS if they have already chosen something else. Because I don't have experience with any other real-time communications API.**
Interviewer	Yeah yeah
Participant E	So have they already decided on something else, like hey sure. But
	if they just mention like hey I'm trying to build some application how would you suggest I approaches this? I would definitely at least mention PRAIS that they should look into it
Interviewer	Okay, clear. That pretty much wraps it up I guess I don't think I have any other questions. Maybe you have any other remarks? Do you want to add anything?
Participant E	Not that much. I just want to say that, well, as a quick thank you for all the clear support that you provided to us throughout the project. User stories were generally pretty clear, you are open to feedback and you responded very quickly to a lot of things. So, thanks for that. That definitely made our project a bit easier.

K NuGet generation

To define what should be included in the PRAIS C# API NuGet package, we use a so-called *nuspec* [30] file (see Figure K.1). In addition to package content, the *nuspec* file also defines metadata such as package name, version, and dependencies. Noteworthy is that the package also includes an *Install.ps1* file (see Figure K.2) that configures some settings upon NuGet install. To build the NuGet package, we run *nuget pack PRAIS.nuspec* from a command prompt in the folder where the *nuspec* is located.

```
<?xml version="1.0" encoding="utf-8"?>
<package >
  <metadata>
    <id>$id$</id>
    <version>$version$</version>
    <title>$title$</title>
    <authors>Robin Mennens</authors>
    <owners>Philips Research</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <license type="expression">MIT</license>
    <projectUrl>https://healthrtc.org/docs/index.html</projectUrl>
    <description>PRAIS C# SDK NuGet Package</description>
    <releaseNotes>First version.</releaseNotes>
    <copyright>Copyright 2020</copyright>
    <tags>PRAIS, C#, SDK</tags>
    <dependencies>
      <group targetFramework=".NETFramework4.7.2" >
      </group>
    </dependencies>
  </metadata>
  <files>
    <file src="install.ps1" target="Tools"/>
    <file src="bin\Release\*.dll" target="lib\net472"/>
    <file src="bin\Release\lib\win_x64\*.dll" target="content\lib\
       win_x64" />
    <file src="bin\Release\lib\win_x86\*.dll" target="content\lib\
       win_x86" />
  </files>
</package>
```

Figure K.1: Source code for PRAIS.nuspec

```
# Sets the 'Copy to output directory' property of the dlls in the
    lib folder to 'Copy If newer'

param($installPath, $toolsPath, $package, $project)

function MarkFileASCopyToOutputDirectory($file)
{
    $file.Properties.Item("CopyToOutputDirectory").Value = 2
}

MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x64").ProjectItems.Item("
    libaudioprocessingfm.dll"))
MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x64").ProjectItems.Item("libopenh264fm.
    dll"))
MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x64").ProjectItems.Item("libopusfm.dll"))
MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x64").ProjectItems.Item("libvpxfm.dll"))
MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x64").ProjectItems.Item("libyuvfm.dll"))
MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x86").ProjectItems.Item("
    libaudioprocessingfm.dll"))
MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x86").ProjectItems.Item("libopenh264fm.
    dll"))
MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x86").ProjectItems.Item("libopusfm.dll"))
MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x86").ProjectItems.Item("libvpxfm.dll"))
MarkFileASCopyToOutputDirectory($project.ProjectItems.Item("lib").
    ProjectItems.Item("win_x86").ProjectItems.Item("libyuvfm.dll"))
```

Figure K.2: Source code for Install.ps1

L Project management

Figures L.1 and L.2 show a Gantt chart that illustrates the planning of this project. Table L.1 lists all risks that we identified during this project.

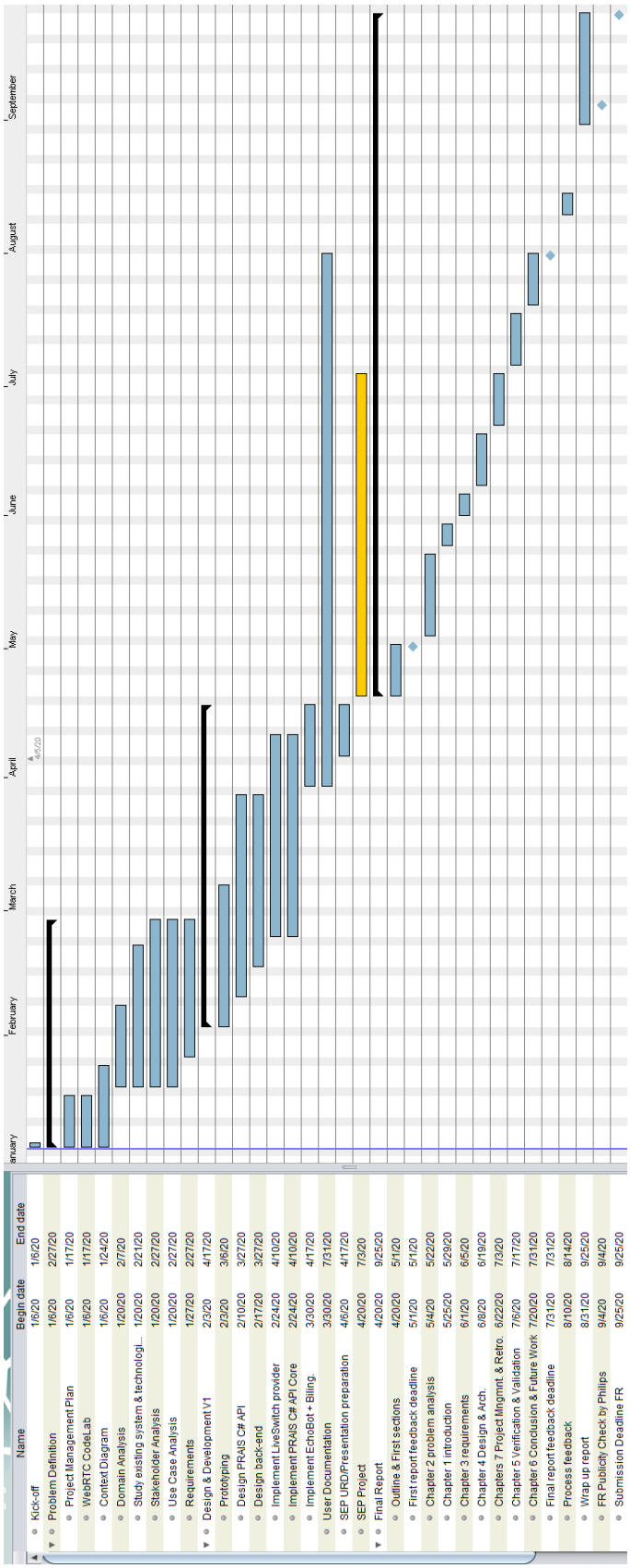


Figure L.1: The first part of the Gantt chart that shows the project planning. The second part is illustrated in Figure L.2

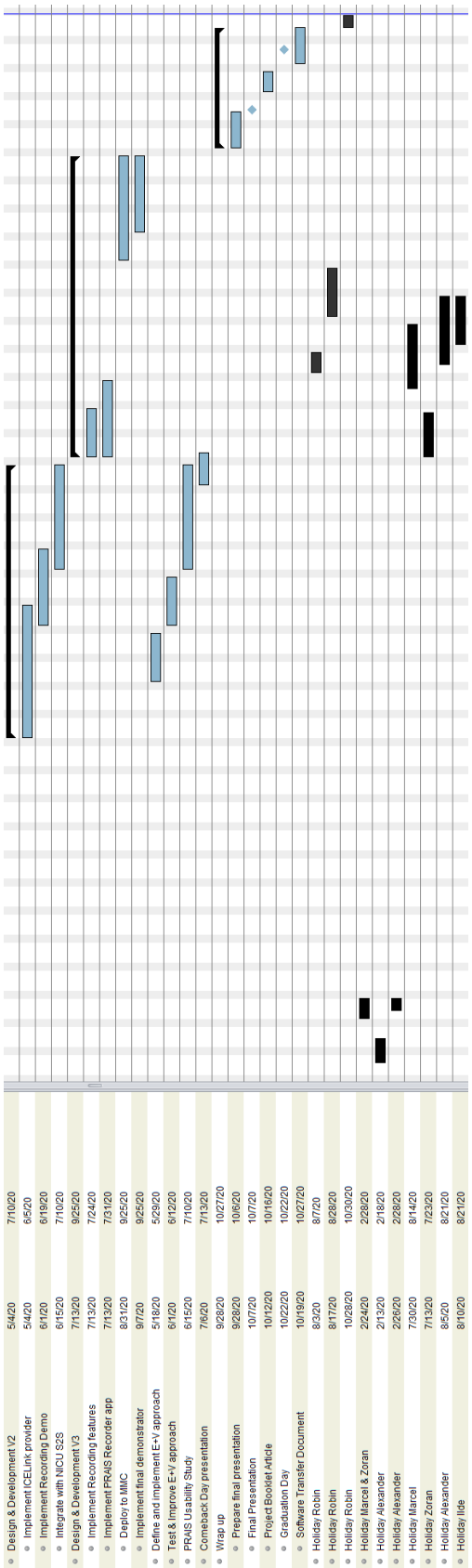


Figure L.2: The second part of the Gantt chart that shows the project planning. The first part is illustrated in Figure L.1

ID	Status	Description	L	I	P	Mitigation action (reduce likelihood)	Contingency action (reduce impact)
18	Open	Recording accurate timestamps for each video frame turns out to be more difficult than expected. So, there is a risk that we cannot fulfill this requirement. Consequently, MMC may not benefit from the recording functionality after all.	4	4	16	There are two aspects to consider here. Firstly, we can reduce the likelihood of having unaccurate timestamps. This means that we should pick the best solution regarding the recording of timestamps per frame. Secondly, by talking to MMC and discussing the issues we have, we may come to a common understanding and hopefully also a solution that works for both of us.	Depending on which technical solution we pick, the impact may be large or small. In general, we aim to pick the solution that is the easiest to implement but also has the most value. In the worst case, this risk leads to an unsuccessful collaboration between us and MMC.
6	Open	I will start developing the system with mainly our own team as stakeholders. While there is a lot of experience in the team, this may mean that the final solution is very team tailored and does not really suit other stakeholders.	3	4	12	Involving other stakeholders as early as possible will help mitigate the risk. Robin will talk to Marcel/Zoran about potential candidates and plan accordingly. Also, by first implementing core functionality that is definitely required, we make sure we have a strong basis to which specific functionality can be added.	Robin should not be afraid to throw things away in case they turn out to be unfitting. The awareness of the risk will help with this.

19	Open	Arjan started working part-time. Since there are quite some dependencies on him doing certain activities, there is a risk that not everything can be done in time.	3	4	12	Communication is key. Arjan will be working one day per week, so it is vital to align in time what he can work on. where possible, I can also do some activities myself.	The Biggest dependency on Arjan is the deployment to MMC. So we should prioritize that. It is also essential to not create any other dependencies on him.
1	Open	Stakeholder absence: when stakeholders are not available at a time that I need them, issues may occur.	3	3	9	I will ask my stakeholders as soon as possible when they will be unavailable (due to holidays for example). If, after some time, informal meeting cannot take place due too busy schedules then I will schedule meetings in the agenda.	By planning any stakeholder interaction as soon as it is possible there should remain enough time, even when they are temporarily unavailable. When one of my supervisors is unavailable I always have other supervisors to talk to.
8	Open	The students do not really have baby footage or sensory input available. This may cause some issues when running Roel's apnea detection algorithm or Ralitsa's pose detection algorithm.	4	2	8	We should check in time whether we can distribute the data that Roel used to train his algorithm. Of course, this doesn't reflect how such an algorithm should be used, but it gives us something to work with.	We do not care too much about the algorithms themself, more about how they fit into a conference. So in that sense, if it turns out that we cannot get the algorithms to work, then we can also replace them with 'fake' algorithms that take the same inputs and produce the same outputs as the original algorithms.
15	Open	The algorithms developed by Ralitsa and Roel are not done yet when SEP starts. There is a risk that also during SEP they do not manage to finish the implementation of the algorithms.	2	4	8	This is mostly out of my hands because it is their responsibility to implement the algorithms. Nevertheless, in case of need, I could help out with the implementation.	The stub algorithm implementations, i.e., only the interface, should be enough to complete the flow of data. The disadvantage is that the algorithm output is random which reduces the wow factor of the demo.

16	Open	Participation in the PRAIS usability study is voluntary. Consequently, we rely on the willingness of the SEP students to actually participate. There is a risk that only a few of them take part in the usability study.	2	4	8	This is mostly out of my hands. All I can do is remind them often about participating and to approach them personally.	If time is an issue for the students that do not want to participate, then I can ask them to only fill in the questionnaire. That would give me some partial results. In my report, I shall discuss potential reasons why students do not want to participate.
17	Open	Acquiring a new ICELink license is taking much longer than expected. In parallel, I'm encountering some technical issues that have been solved in new ICELink versions. So, there is a risk that I cannot address these technical issues because we do not get the ICELink license in time.	2	4	8	I can keep asking about the license and its acquisition status. Furthermore, I shall express the need for having an updated license.	Most technical issues arise because browsers are using newer versions of certain protocols (DTLS). This creates an incompatibility between the ICELink version we have and the browsers. A temporary but inconvenient workaround is to install an older version of Chrome, for example. While this will work for us, it is definitely not desirable at MMC.
14	Mitigated, we are already in contact with Ilde	I'm developing functionality for the project at MMC, but there is no guarantee yet that the PhD student is going to use it before the end of my project. Hence, there is a risk that I cannot evaluate the implemented functionality with that student.	0	3	0	This is out of my hands because the PhD student is being hired by MMC. What may help, however, is an early demonstrator that shows them that we already have the functionality in place. This may push them to speed up the process of hiring the PhD student.	The main use case is to develop a recording algorithm, which should work in any setting. So, it is still possible to test his algorithm ourselves. Also, we may be able to do a test run at MMC even when the PhD student has not started yet.

5	Mitigated, we are in contact with MMC.	One of the goals of my project may be to run a pilot (at a hospital). At this point, it is not certain whether it is possible. And even if it would be possible, then there is also no knowledge about when it would be done. So, there is a risk that due to planning issues I cannot do a pilot during my project.	0	3	0	Early discussions with Marcel will help in identifying potential pilot opportunities.	If it turns out that a pilot cannot be done given the time constraints, then the alternative would be to do a demo at Phillips. So, ideally I work on a use case that can be done both as a pilot as well as a demo.
11	Mitigated, did not occur	Within Philips, there is going to be a migration from Gitlab to Github. Currently, all code resides in Gitlab, which means that at some point all code needs to be migrated. There is a risk that this introduces complications and issues.	0	3	0	Only use basic GitLab CI/CD functionality. We have some in place now, and it shouldn't be too hard to replace in case there is a move to Github.	There is always a local copy of the code on Robin's computer. In the worst case, we will have to transfer all the code manually. It is also important to figure out when exactly the migration will happen such that we can anticipate it. Finally, by only having simple CI/CD functionality we can reduce complexity, which should make the move to GitHub easier.
9	Mitigated, Roel decided to continue working for Philips because he wanted to learn python.	Roel Montree will not stay long after SEP starts. So if there are technical issues, it may be hard to solve them.	0	3	0	Robin will have a session with Roel where Roel explains the algorithm in detail. How to use it, the inputs/outputs, etc.	We do not care too much about the algorithm itself, more about how it fits into a conference. So in that sense, if it turns out that we cannot get the algorithm to work, then we can also replace it with a 'fake' C# algorithm that takes the same inputs and produces the same outputs as the Matlab algorithm.

10	Mitigated, we decided to not implement LiveSwitch in JavaScript. Instead, we adapted the default LiveSwitch demo. Also, we added ICELink to PRAIS	In the original project scope it was predicted that there wouldn't be a very high development effort on the client-side, i.e., the browser, because we already have the Javascript RTC API. At this point, however, it seems very likely that we will use LiveSwitch in the backend. Consequently, we will also have to update the research RTC API such that it can communicate with LiveSwitch. There is a risk that this requires quite some development effort.	0	4	0	It is essential to discuss and assess within the team how much effort it would be to update the Javascript RTC API. Furthermore, we can discuss in the team whether it is even needed to update the Javascript RTC API. It may, for example, be enough to simply use the frozen mountain demo. In case we do need to update the Javascript RTC API, it may be possible to get some help from Zoran/Arjan since they developed it and are familiar with it.	We could decide to leave the client-side out of scope and to simply assume that it's there. Still, we would require some way to connect browsers to algorithms. Overall, proper scoping and prioritization is key to making sure that all relevant parts will be implemented.
12	Mitigated, all required features were implemented on time.	The SEP students will have to work with the system, which means that when they start there should be a minimal viable version that works good enough for them to complete the project. So, there is a risk that the system is not ready yet.	0	4	0	First, defining and agreeing on what is included in the minimal viable system is essential. Robin shall do this by defining priorities of the requirements. Second, finishing the minimal viable system shall have the highest priority until the SEP project begins.	Assuming there is at least a minimal workable system, i.e., the minimal viable system is not complete but there is at least some functionality. Then, it is essential to come up with a project that is still doable and also has value to us. In the worst-case, when there is no working system at all, we may have to cancel the SEP project.

7	Mitigated, we decided to not use Matlab. Instead, Roel shall convert his algorithm to Python.	During the SEP project, we want the students to work with Roel's Apnea detection algorithm that was developed in MatLab. Since our Framework is developed in C#, there are some incompatibility issues. Arjan has written a wrapper for a Matlab algorithm before, so we know its possible. Still, it may take some time for the students to write a new wrapper.	0	2	0	When time allows, we could first try ourselves to write a simple wrapper for the algorithm. Thereby we would already have some knowledge on how to make it work.	We do not care too much about the algorithm itself, more about how it fits into a conference. So in that sense, if it turns out that we cannot get the algorithm to work, then we can also replace it with a 'fake' C# algorithm that takes the same inputs and produces the same outputs as the Matlab algorithm.
3	Mitigated, this isn't a risk to the project itself, more a constraint.	The Research Javascript RTC API deviates from the one that is developed for the reference architecture. Not only may the interfaces differ, but also the functionality can be different. For example, the Research Javascript RTC API may provide some functionality that is not provided by the unified communications API. This may lead to a scenario where my project provides an unrealistic view of what would be possible if one were to further develop the system after my project.	0	3	0	I can compare both API interfaces to see if there are any significant differences. Determining, however, if the technology behind the API behaves the same is more difficult. This would require some testing, which can be time-consuming.	I can discuss with my stakeholders (expectation management) the risks involved in using the Research Javascript RTC API. Awareness of the risks can help in accepting the potential issues that may arise. I can design the system in such a way that there is a low dependency on the Research Javascript RTC API such that it can be easily replaced by the unified communications API.

2	Mitigated, we will use the Research Javascript RTC API	The unified communications API is still in development and has not matured yet. This means that there may be bugs, that the API can still change, and that when functionality is missing it may take a long time before it can be added.	0	4	0	Close communication and collaboration with the team that is developing the API should help me in identifying issues well in advance. Also, by learning to work with the API early on I can identify issues in an early stage, which should give me enough time to point them out (and hopefully have them fixed).	I can design the system in such a way such that there is a low dependency on the unified communications API. By doing so it would be easier to completely replace the API if needed.
4	Mitigated, we will use the Research Javascript RTC API	The unified communications API is using Twilio, which brings along some costs. Currently, Phillips customers who use this API are covering these costs together. This means that if I wanted to use the API as well, there may also be some costs involved. As far as I know, there is no budget available for my project.	0	3	0	An alternative would be to work with the API that is present at Phillips research. The risks, however, involved in taking this approach are described in risk #3.	By discussing this risk early on with my company supervisor we can potentially find some budget. The alternative would be to use the Research Javascript RTC API. See risk #3, however.
13	Mitigated, SEP will be done online	With the coronavirus, it is uncertain whether the SEP project is going to take place.	0	4	0	This is out of my hands because it is organized by TU Eindhoven.	By monitoring the SEP project state, i.e., by being in close contact with Lou, I will know in time whether the project will continue. As long as there is uncertainty about its continuation, I shall discuss with my supervisors what the best alternative would be.

Table L.1: The risks that we identified during the project. The ID, L, I, and P columns represent Identifier, Likelihood, Impact, and Priority, respectively. The P column is color coded with a gradient from red to yellow that represent high to low priority respectively.

PO Box 513
5600 MB Eindhoven
The Netherlands
tue.nl

PDEng SOFTWARE TECHNOLOGY