Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Bachelorarbeit (Computer Science)

Methoden zur Bearbeitung von parallelen Aussagen in der automatischen Spracherkennung

Autoren	Peter Unger Yannik Roth
Hauptbetreuung	Prof. Dr. Mark Cieliebak
Industriepartner	Spinning Bytes AG
Externe Betreuung	Flurin Gishamer
Datum	30.06.2020

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Anleitung:
Kopieren Sie den untenstehenden Inhalt in Ihre Bachelorarbeiten. Dieser ist zu Beginn der Dokumentation nach dem Titelblatt mit **Namen und Datum** einzufügen.

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)
Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	**Name Studierende:**
Winterthur, 30.06.20	Peter Unger
Stachen, 30.06.20	Yannik Roth

Abstract

The creation of transcripts from interviews, surveys, group discussions and other conversation types is an essential task in various areas of the business world. Nowadays there are software solutions to automate this process. Such software systems are based on the automatic identification and conversion of statements from an audio signal to create transcriptions. Current software systems fail however, when identifying statements that overlap in time in the audio signal. This leads to an incomplete transcription of the conversation. Valuable information may be lost in the process, and in such cases, the result must be corrected manually to ensure the completeness of the transcript. The present work is about figuring out how to deal with the problem of incomplete transcriptions due to failed identification of parallel statements in the automatic speech recognition process. For this purpose, approaches for the identification and separation of parallel statements in an audio signal are investigated. In this thesis it turns out that no solution could be proposed for the automatic identification of parallel statements. However, the separation of individual statements from a mixed audio signal with application of neural networks is a promising approach. With the use of the proposed method, single statements could be successfully separated from a mixed audio signal. In addition, the editor of the transcription system Interscriber is being further developed to deal with parallel statements. We implement functionalities in the application to process parallel statements in the transcript. Statements can be marked as parallel and then adjusted. The goal is to give a user the possibility to quickly and easily correct incomplete transcriptions with the Interscriber application. The results show that the transcription service Interscriber, with the new feature implementations, gives a user the possibility to successfully capture parallel statements. However, the integration of the proposed method for the automatic separation of single statements from a mixed audio signal into the Interscriber application is still ahead.

Zusammenfassung

Die Erstellung von Abschriften aus Gesprächen in Form von Interviews, Umfragen, Gruppendiskussionen und weiteren Gesprächstypen ist in unterschiedlichsten Bereichen der Geschäftswelt eine unerlässliche Aufgabe. Heutzutage gibt es Softwarelösungen, um diesen Prozess zu automatisieren. Solche Softwaresysteme basieren auf der automatischen Identifizierung und Umwandlung von Aussagen aus einem Audiosignal, um Abschriften zu erzeugen. Derzeitige Softwaresysteme scheitern jedoch bei der Identifizierung von Aussagen, welche sich im Audiosignal zeitlich überlagern. Dies führt zu einer unvollständigen Transkription des Gesprächs. Wertvolle Informationen können dabei verloren gehen und das Gesprächsprotokoll muss in solchen Fällen nachkorrigiert werden, um die Vollständigkeit gewährleisten zu können. In der vorliegenden Arbeit geht es darum, herauszufinden, wie man dem Problem von unvollständigen Transkriptionen aufgrund gescheiterter Identifizierung von parallelen Aussagen in der automatischen Spracherkennung Abhilfe schaffen kann. Dazu werden Ansätze für die Identifizierung und Separierung von parallelen Aussagen in einem Audiosignal untersucht. In dieser Arbeit stellt sich heraus, dass keine Lösung für das Erkennen von parallelen Aussagen vorgeschlagen werden konnte. Das Separieren von einzelnen Aussagen aus einem gemischten Audiosignal ist jedoch mit der Anwendung von neuralen Netzwerken ein vielversprechender Lösungsansatz. Mit diesen Verfahren konnten einzelne Aussagen erfolgreich aus einem gemischten Signal separiert werden. Zusätzlich wird der Editor des Transkriptionssystem Interscriber für den Umgang mit parallelen Aussagen weiterentwickelt. Wir implementieren Funktionalitäten in der Anwendung, um parallele Aussagen im Transkript zu verarbeiten. Aussagen können als parallel markiert und anschliessend angepasst werden. Das Ziel besteht darin, einem Nutzer die Möglichkeit zu geben, unvollständige Transkriptionen mit dem Interscriber schnell und einfach zu korrigieren und zu vervollständigen. Die Resultate zeigen auf, dass der Transkriptionsdienst Interscriber mit den Erweiterungen einem Nutzer die Möglichkeit gibt, parallele Aussagen erfolgreich zu erfassen. Das automatische Separieren von Aussagen setzt jedoch noch die Integration von dem evaluierten Separierungsverfahren in die Interscriber Anwendung voraus.

Vorwort

Besonderer Dank geht an:

- Prof. Dr. Mark Cieliebak, für die Betreuung und Unterstützung bei dieser Arbeit. Seine offene Kommunikation und Denkanstösse während unserer wöchentlichen Meetings haben uns hervorragend in der Bewältigung der Problemstellungen weitergeholfen.
- Der Spinning Bytes AG, insbesondere Flurin Gishamer und Philippe Schläpfer, für die Unterstützung bei der Integration neuer Features in die Interscriber Applikation, sowie fachliche Diskussionen betreffend gewählter Architekturentscheidungen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Transkribierung von gesprochener Sprache in eine geeignete Textform ist in unterschiedlichsten Prozessen und Abläufen bei zwischenmenschlichen Interaktionen im Geschäftsumfeld unerlässlich. So sind zum Beispiel Journalisten, Anwälte und Ärzte, aber auch Forscher bei der Durchführung ihrer Arbeit darauf angewiesen, Konversationen dokumentarisch festhalten zu können. Traditionell wird diese Aufgabe an einen professionellen Dienstleister, dem Transkriptionisten, für eine manuelle Abschrift der Konversation ausgelagert. Diese Art der Transkribierung bringt eine sehr hohe Genauigkeit mit sich, ist jedoch auch immer mit einem grossen Arbeits- und Zeitaufwand verbunden. Automatische Transkriptionsdienste schaffen bezüglich der oben genannten Nachteile Abhilfe. Solche Dienste basieren auf die Datenverarbeitung von Gesprächen in Form von Audiosignalen. Ein Gespräch wird mit einem Mikrofon in ein digitales Format gespeichert und später durch eine Software verarbeitet, welche versucht, einzelne Aussagen im Audiosignal in Textform umzuwandeln und zu einem digitalem Transkript zusammenzuführen. Dafür kommen sogenannte Speech-to-Text Systeme zum Einsatz, welche in der Regel sowohl einzelne Sprecher identifizieren, als auch die Aussagen jeder Sprecher in Textform umwandeln. Reden mehrere Sprecher zur selben Zeit können diese Systeme jedoch kein klares Signal mehr verarbeiten, da die Signale, also die einzelnen Aussagen, zeitlich überlagert sind. Dies führt bei einem automatischen Transkribierungsprozess zu unvollständigen Transkriptionen, da aufgrund der Informationen aus dem überlagerten Signal keine eindeutige Aussage mehr erkannt werden kann. Wenn keine eindeutige Aussage mehr erkannt werden kann, wird während den überlagerten Aussagen auch kein Text transkribiert und es entsteht eine Lücke, die dann manuell von Hand gefüllt werden muss. Das Ziel eines

automatischen Transkriptionsdienstes sollte es aber sein, den Anteil von manueller Eingabe zu minimieren und genau hier setzt diese Arbeit an.

1.1 Die Interscriber Anwendung

Die Applikation namens “Interscriber”, welche von der Spinning Bytes AG entwickelt wird, ist ein Dienst für die automatische Transkription von Gesprächen. Sie nutzt dafür innovative KI-Systeme für die Umwandlung von Aussagen in gesprochener Sprache zu Text und zur Unterscheidung von Sprechern. Zu diesem Zeitpunkt wird dazu das Google Speech-to-Text KI-System eingesetzt. Zudem wird bereits an einem eigenen Speech-to-Text KI-System für die automatische Transkription von Gesprächen gearbeitet.

Weiterhin bietet Interscriber einen massgeschneiderten Editor, mit dem sich das automatisch erzeugte Transkript nachbearbeiten lässt. Damit können Fehler nach dem Transkriptionsprozess auf schnelle Weise korrigiert, und somit eine hohe Genauigkeit der Transkription gewährleistet werden.

In der nächsten Abbildung wird der Transkript-Editor der Interscriber Applikation dargestellt. Es wird linear von oben nach unten dargestellt, in welcher Reihenfolge welcher Sprecher was gesagt hat.

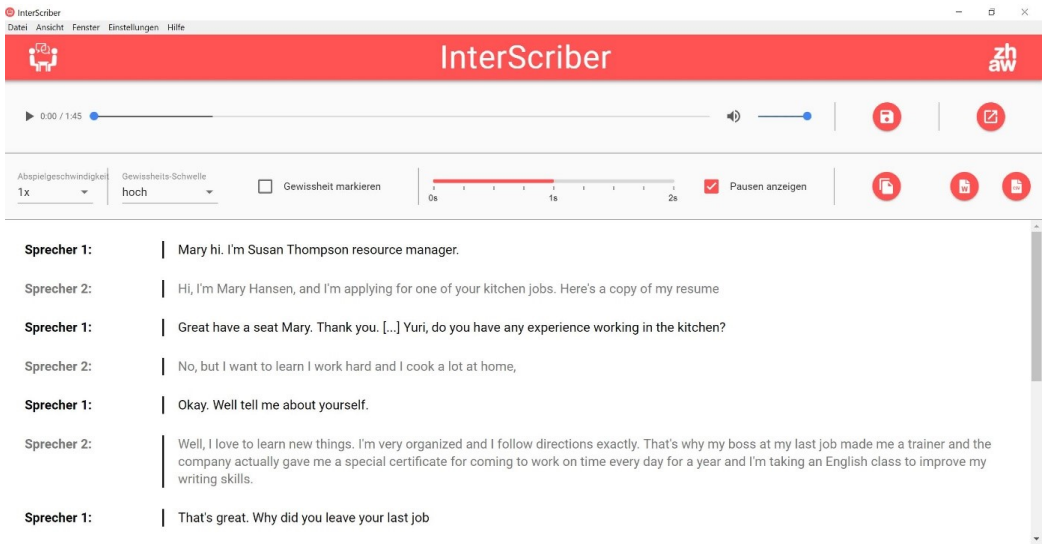

Abbildung 1.1: *Benutzeroberfläche des Transkript-Editors der Interscriber Anwendung.*

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, die Interscriber Applikation für die Erkennung und Bearbeitung von gleichzeitigen Aussagen unterschiedlicher Sprecher, sowie das Separieren von einzelnen Aussagen in einem zeitlich überlagerten Audiosignal, weiterzuentwickeln. Dabei liegt der Fokus auf eine nutzerfreundliche Darstellung und Bearbeitungsmöglichkeit von parallelen Aussagen im Transcript-Editor, um eine flüssige Arbeitsweise mit dem System zu gewährleisten.

1.3 Teilziele

Aus der Zielsetzung ergeben sich drei Teilziele, welche im Fokus dieser Arbeit liegen und in den folgenden Unterkapiteln jeweils näher erläutert werden.

- Erkennung von parallelen Aussagen aus dem Audiosignal
- Separierung einzelner Aussagen aus dem Audiosignal
- Darstellung paralleler Aussagen im Transkript-Editor

1.3.1 Erkennung von parallelen Aussagen

Es soll nach Ansätzen gesucht werden, die parallele Aussagen in einer Audiodatei erkennen können. Hierbei geht es nur darum, eine parallele Aussage zu erkennen und nicht darum, was genau gesprochen wird. Dies ist notwendig, da das für Interscriber verwendete Speech-to-Text System von Google nicht zwischen Hintergrundlärm, Pausen und parallelen Aussagen unterscheiden kann. Es liefert in diesen Fällen einfach nichts zurück. Dies führt natürlich zu einem Verlust von Informationen im Transkript. Sobald aber bekannt ist, dass in einem Abschnitt eine parallele Aussage vorliegt, kann man mit der Separierung einzelner Aussagen fortfahren, was auch das nächste Teilziel in dieser Arbeit ist.

1.3.2 Separierung einzelner Aussagen

Diese Arbeit soll die Grundlagen und Anwendbarkeit von Algorithmen für speaker-separation untersuchen. Dazu sollen zunächst vorige Arbeiten zu dieser Problematik aufgezeigt und anschliessend ein vielversprechendes Verfahren näher betrachtet werden. Je nach Resultat der Analyse kann ein Lösungsansatz für die automatische Separierung von zwei (oder mehr) gleichzeitigen Aussagen aus dem Audiosignal vorgeschlagen werden, mit welcher der Interscriber nach einer erneuten Transkription der jeweiligen Signale, beide Aussagen automatisch ermitteln kann. In Kapitel 2.3 werden die Grundlagen zu der Problemstellung der speaker-separation erläutert.

1.3.3 Implementierung paralleler Aussagen in Interscriber

Um den leichten Umgang mit dem Interscriber weiterhin gewährleisten zu können, soll eine intuitive Darstellungsform für die Ansicht und die Bearbeitung von parallelen Aussagen innerhalb des Transkripts evaluiert und umgesetzt werden. Dabei soll sich die gewählte Darstellungsform an das Designkonzept der bestehenden Applikation halten. Im Kapitel 3.1 wird dazu die bestehende Oberfläche der Applikation untersucht und verschiedene Entwürfe für die Darstellung paralleler Aussagen präsentiert. Im Kapitel 4.1, sowie 4.2, wird die Umsetzung der Anpassungen am Editor behandelt.

1.4 Überblick der Arbeit

Grundlagen

In diesem Kapitel werden die für diese Arbeit relevanten Grundkonzepte vorgestellt und erläutert. Es behandelt die Grundlagen von Speech-to-Text Systemen, sowie eine genauere Beschreibung der Problemstellungen bei der opverlapped speech Detektion und speaker-separation. Weiterhin wird die Softwarearchitektur, welche für Interscriber gewählt wurde, theoretisch beschrieben.

Konzept

Dieses Kapitel behandelt die Findung eines geeigneten Ansatzes für das Erkennen und Auswerten von parallelen Aussagen. Zudem werden Konzepte für die Darstellung paralleler Aussagen im Transcript-Editor vorgestellt und ausgewertet.

Umsetzung

Dieses Kapitel beschreibt die Umsetzung der Arbeiten, welche sich aus der Konzeptionsphase ergaben. Es erläutert die Implementierung der Features zur Anzeige und Bearbeitung von parallelen Aussagen im Editor, sowie die Erweiterung der Datenbankstruktur und Businesslogik der Interscriber Applikation. Weiterhin wird die Auswertung der evaluierten Ansätze für die Detektion und Separierung von parallelen Aussagen beschrieben.

Ergebnisse

In diesem Kapitel werden die erreichten Ergebnisse aus dieser Arbeit präsentiert und erläutert. Es behandelt folgende Ergebnisse:

- Erstellung und Editierung paralleler Aussagen im Editor (Kapitel 5.1) präsentiert den neuen Transcript-Editor, welcher parallele Aussagen darstellen und bearbeiten kann.
- Automatische Erkennung paralleler Aussagen (Kapitel 5.2) erläutert das Ergebnis für das Erkennen paralleler Aussagen aus dem Audiosignal der Transkription.
- Separieren einzelner Aussagen (Kapitel 5.3) erläutert die Lösung für das Separieren einzelner Aussagen aus einem gemischten Audiosignal.

Fazit

Dieses Kapitel diskutiert die erreichten Ziele und stellt sie der ursprünglichen Zielsetzung gegenüber. Weiterhin wird der taktische und strategische Ausblick für die Zukunft der Interscriber Anwendung in Bezug auf die neuen Features erläutert.

Kapitel 2

Grundlagen

Dieses Kapitel bietet die Grundlagen für die Methodiken bei der Transkription von Gesprächen, welche in dieser Arbeit behandelt werden. Hierbei wird bereits Bezug auf die Interscriber Anwendung genommen, um den Überblick über die Funktionsweise des Interscribers zu veranschaulichen.

2.1 Speech-to-Text Transkribierung

Speech-to-Text beschreibt den Vorgang, verbale Aussagen in Text umzuwandeln. Im Kontext von automatischen Speech-to-Text Systemen handelt es sich um Software, welche aus einem gegebenenen Audiosignal Aussagen erkennt, und diese in Text umwandelt. Der Interscriber nutzt für den automatischen Transkriptionsprozess das Speech-to-Text System von Google [2], welches über eine Schnittstelle (API) angesprochen werden kann. Dieses System basiert auf der Anwendung von sogenannten deep-learning Methoden, um das Audiosignal auszuwerten und daraus die Textrepräsentation der Aussage zu erhalten. Nach Brownlee, "Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks."[1]

Das Speech-to-Text System von Google löst bereits zwei Aufgaben, welche für eine erfolgreiche Transkribtion notwendig sind. Die automatic speech recognition und die speaker diarization. In folgenden Unterkapiteln werden diese zwei Aufgaben näher beschrieben. Im Abbild 2.1 wird die Kommunikation des Interscribers mit dem Speech-to-Text System vom

Google in vereinfachter Form schematisch dargestellt:

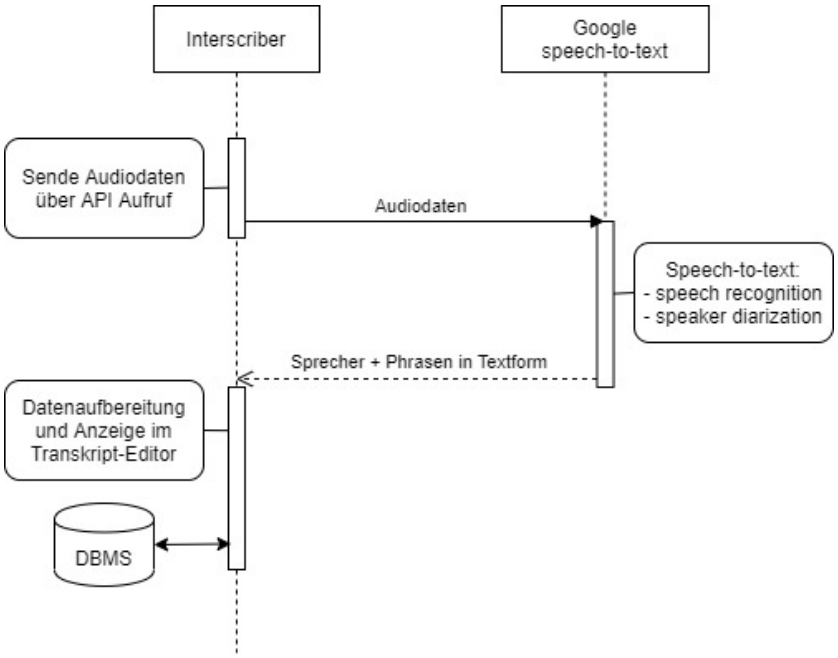

Abbildung 2.1: *Interscriber Kommunikation mit Googles Speech-to-Text System für die automatische Transkribierung von Gesprächen.*

2.1.1 Automatic speech recognition

Die Aufgabe der automatic speech recognition ist wie folgt definiert: "Performed by a functional unit, the perception and analysis of information that (a) is represented by human voice and (b) can be recognized by (i) an uttered word of a predetermined language, particularly a natural language, (ii) a phoneme of a predetermined language, or (iii) the vocal features of the speaker"[4]. Es geht also darum, aufgrund von gesprochener Sprache zu erkennen, welche Worte gesagt werden.

2.1.2 Speaker diarization

Speaker diarization ist die Aufgabe, aus einem Audiosignal die Identität des Sprechers von einer erkannten Aussage zu bestimmen. Sie klärt die Frage "who spoke when?"[18] Dabei wird das Audiosignal in einzelne Zeitabschnitte segmentiert, welche die Identität des jeweiligen Sprechers zugeordnet

bekommen. Diese Information wird benötigt um Aussagen im Transkript einem Sprecher zuordnen zu können, und damit das Gesprächsprotokoll aufzubauen.

2.2 Overlapped speech detection

Overlapped Speech Detection beschreibt den Vorgang, bei dem auf einem Audiosignal identifiziert wird, an welchen Stellen mehrere Personen gleichzeitig sprechen. Es geht nicht darum, wer parallel zueinander spricht, sondern nur darum, ob überhaupt parallel gesprochen wird.

Wenn jeder der Beteiligten ein eigenes Mikrofon hat und jeder Sprecher eine eigene Spur in der Aufnahme besitzt, ist das Problem relativ trivial. Es können einfach die Spuren der einzelnen Sprecher auf Überlagerungen mit den anderen Sprechern untersucht werden. In unserem Fall geht es aber darum, aufgrund des Audiosignals eines einzigen Mikrofons zu bestimmen, ob gerade mehrere Personen gleichzeitig sprechen oder nicht. Die Audiosignale sind alle überlagert in einem einzigen Signal zusammengefasst. Die Aufgabe ist es nun, auf Grund dieses einen Audiosignals zu erkennen, ob mehrere Sprecher gleichzeitig gesprochen haben.

2.3 Parallel-speaker Separierung

Die Disziplin der speech-separation beschreibt den Prozess, individuelle Sprachsignale aus einem gemischten Sprachsignal zu extrahieren. Das menschliche Gehör meistert diese Aufgabe mühelos. In einem Raum mit mehreren gleichzeitigen Unterhaltungen ist es uns möglich, sich auf ein Gespräch zu konzentrieren und Hintergrundgeräusche auszublenden. Diese Problemstellung ist auch unter dem Pseudonym "cocktail party problem"bekannt. [6], [8] Für Signalverarbeitungssyteme zur automatischen Trennung von Aussagen aus einem gemischten Audiosignal ist dies jedoch eine grosse Herausforderung. Im Falle der Interscriber Anwendung handelt es sich um eine spezifische Form der Signalverarbeitung, da nur ein Audiosignal für die Transkription verwendet wird. Diesen Fall bezeichnet man als "single-channel speech separation". In den letzten Jahren konnte für diese Problemstellung mit Sytemen basierend auf deep-learning Methoden[1] grosse Erfolge erzielt werden. [5], [9]–[11], [47]

2.4 Webapplikationen basierend auf MVC Design Pattern

Webapplikationen sind Webseiten, welche keine statischen Inhalte präsentiert, sondern die Inhalte dynamisch durch die Applikationslogik erzeugt werden. [3] Dadurch können Webseiten genutzt werden, um nutzerspezifische Inhalte anzuzeigen, ohne dafür eine statische Seite vorbereiten zu müssen. Zudem muss die Webseite bei einer Änderung an den Applikationsdaten nicht neu geladen werden, da der Inhalt durch die Applikationslogik präsentiert wird.

Diese Eigenschaft von Webapplikationen ist vor allem für den Transkript-Editor von Interscriber relevant, da nicht nach jeder Korrektur des Benutzers die Seite neu geladen werden muss, sondern der Inhalt dynamisch angepasst werden kann.

Das für Interscriber verwendete Model-View-Controller [19] Design Pattern (auch bekannt als MVC) beschreibt ein nützliches Architekturkonzept, um Softwaresysteme mit einer Benutzeroberfläche umzusetzen. Die Idee hierbei ist, die Funktionalitäten für die Oberfläche von dem zugrundeliegenden Datenmodell zu entkoppeln. Die Präsentationsschicht (View) ist für die Darstellung der Applikationsdaten für den Benutzer verantwortlich. Die Datenschicht (Model) beinhaltet sowohl die Daten, welche dem Anwender letztlich präsentiert werden, als auch die Applikationslogik, um die Daten zu modifizieren. Die Schnittstelle zwischen der Daten- und Präsentationsschicht ist dabei eine Steuerkomponente (Controller), welche Nutzerinteraktionen von der Präsentationsschicht an die Datenschicht weitergibt und Änderungen im Datenmodell wiederum der Präsentationsschicht mitteilt. Dies hat den Vorteil, dass die entkoppelten Schichten in ihrer Logik nicht abhängig voneinander sind und beliebig angepasst werden können. Der Controller ist für die Kommunikation zwischen diesen beiden Schichten verantwortlich und hält die Datendarstellung synchron zum aktuellen Datenmodell.

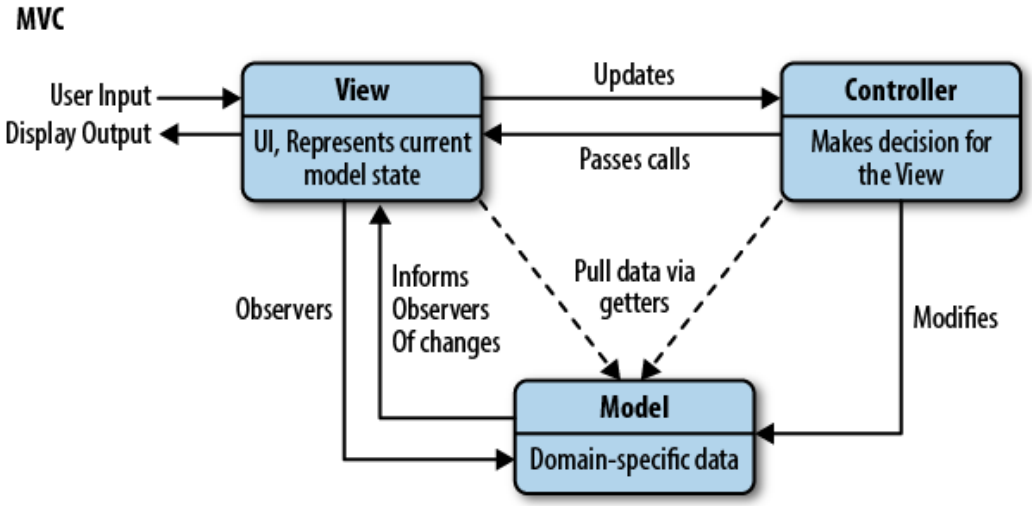

Abbildung 2.2: *Schematische Darstellung des MVC design pattern.[45]*

Kapitel 3

Konzept

In diesem Kapitel werden die Anforderungen an das System untersucht und Ansätze für die Umsetzung der neuen Features in die Interscriber Anwendung evaluiert. Zudem werden Methodiken sowohl für die automatische Identifizierung, als auch für die automatische Separierung von parallelen Aussagen in dem Audiosignal evaluiert.

3.1 Parallele Aussagen im Interscriber implementieren

3.1.1 Analyse der Interscriber Frontend Anwendung

Das Benutzeroberfläche der Interscriber Anwendung wurde, wie schon im letzten Kapitel erwähnt, als Webapplikation umgesetzt.

Für die Frontendlogik wird das Framework Vue.js [20] eingesetzt, welches ein MVC Ansatz nutzt um die Darstellung der Daten auf der Webseite zu steuern. Vue ist ein JavaScript Framework für die Erstellung von HTML basierten Benutzeroberflächen mit dynamischen Inhalten. Treten Mutationen zum Beispiel durch Hinzufügen, Aktualisieren oder Entfernen von einzelnen Werten in dem zugrundeliegenden Datenmodell auf, muss die Benutzeroberfläche auf diese Mutationen reagieren, um in Synchronisation mit den aktuellen Daten der Applikation zu sein.

Dabei werden in Vue.js sogenannte “Reusable Software Components”

eingesetzt. “A software component is a piece of code or an executable that encapsulates a set of related functions and can be included as it is in another software or application. A software component can be something as small as a function, or as large as an entire package or application executable.”[31] Vue stellt dabei die Brücke von den Applikationsdaten zu der präsentierten Benutzeroberfläche dar.

Der Interscriber nutzt zudem due Komponenten Bibliothek “Vue Material” [34]. Diese beinhaltet häufig genutzte Elemente auf einer Webseite, welche einem vorgegebenen Styleguide folgen.

Zudem werden die Anwendungsdaten der Frontendapplikation in einer zentralen Komponente, dem Vuex Store [33], verwaltet. Vuex ist eine state management library speziell für Anwendungen, welche mit dem Vue Framework umgesetzt werden. Dabei verkörpert Vuex ein spezielles design pattern um die Applikationsdaten vor unerwünschten Mutationen abzusichern.

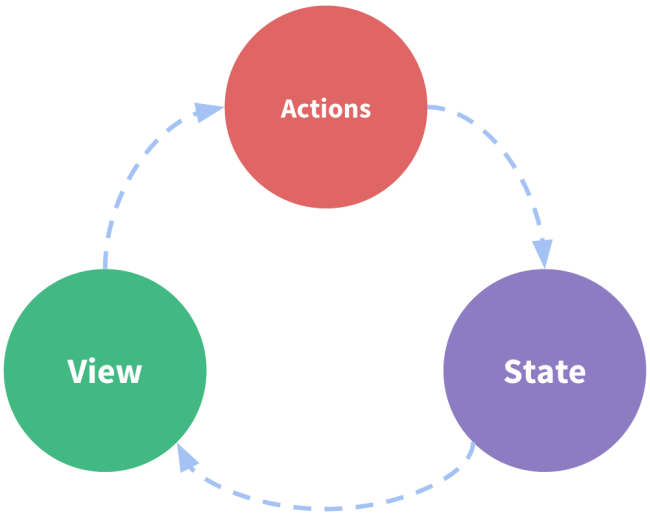

Abbildung 3.1: *Vuex Repräsentation des “one-way data flow“ Konzepts*

Die Implementation neuer Funktionalitäten soll sich an diesen Gegebenheiten orientieren.

- Konzepte von Vue für das komponentenbasierte Entwickeln von neuen Funktionalitäten nutzen

- Vue Material Komponenten verwenden für eine konsistenten Applikationsdesign
- Vuex als einheitlichen Datastore für den Umgang mit Applikationsdaten nutzen

3.1.2 Parallele Aussagen im Transkript-Editor

ParallelUtterance Komponente

Zunächst muss eine Vue Komponente entwickelt werden, in welcher die Logik für die Darstellung von parallelen Aussagen im Transkript Editor vorhanden ist. Dabei soll die Darstellung der parallelen Aussage so gewählt werden, dass sie das aktuelle Design des Editors komplimentiert und dabei intuitiv als parallele Aussage identifizierbar ist. Die grundsätzliche Idee ist, dass diese Komponente eine Erweiterung von einer gewöhnlichen Komponente zur Darstellung von einer Aussage im Editor ist, um so die bereits vorhandenen Funktionalitäten einer Aussagen-Komponente wiederzuverwenden. Eine einzelne Aussage im Editor zeigt den Sprechernamen an, gefolgt von dem Inhalt der Aussage.

Die Komponente für parallele Aussagen soll so erweitert werden, dass jeder Sprecher gefolgt von der Aussage des Sprechers - ähnlich zu einer einzelnen Aussage - dargestellt wird. Sie soll also eine gruppierte Liste an Aussagen darstellen. Jede Aussage ist dabei bündig zum Sprechernamen ausgerichtet. Um nun die gruppierten Aussagen von einzelnen untereinander stehenden Aussagen unterscheiden zu können, sollen um die einzelnen Sprecher eine Klammer dargestellt werden, um die Zugehörigkeit zur parallelen Aussage auszudrücken.

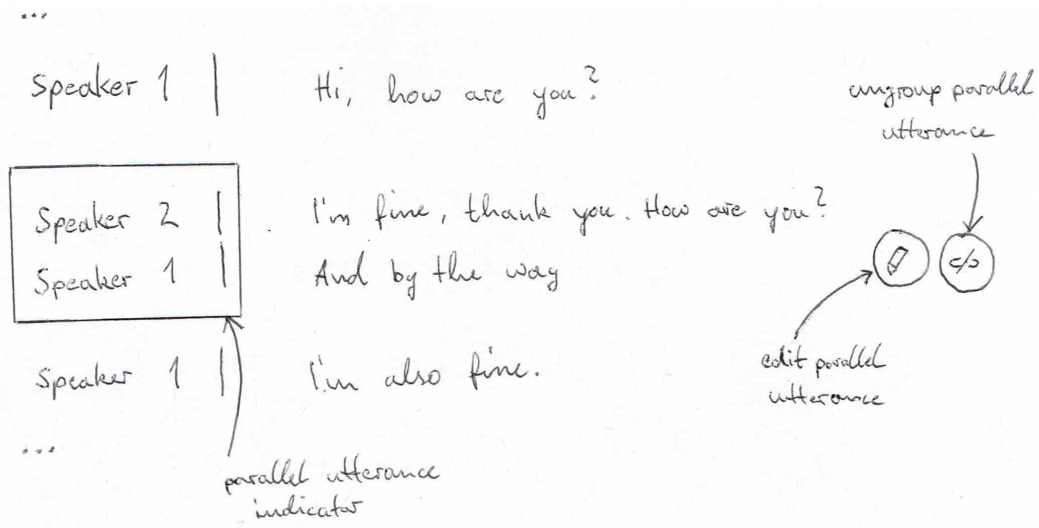

Abbildung 3.2: *Konzeptzeichnung für die Darstellung paralleler Aussagen im Transkript-Editor*

3.1.3 Bearbeitung von parallelen Aussagen

ParallelUtteranceDialog Komponente

Parallele Aussagen müssen zudem editierbar sein. Dies beinhaltet die Funktionalitäten zum:

- Hinzufügen einer Aussage zu der parallelen Aussage
- Editieren des Inhalts einzelner Aussagen in einer parallelen Aussage
- Entfernen einzelner Aussagen aus einer parallelen Aussage
- Anpassen der Zeitstempel einzelner Phrasen in einer Aussage

Diese Funktionalitäten müssen schnell und einfach für eine parallele Aussage durchführbar sein, um einen nutzerfreundlichen Umgang mit parallelen Aussagen im Transkript-Editor zu gewährleisten. Ausgehend von der Anzeige einer parallelen Aussage im Editor soll ein Bearbeitungsfenster dafür umgesetzt werden, welche all diese Funktionalitäten beinhaltet. Dazu wird ein Button zum Erreichen des Bearbeitungsfensters für eine parallele Aussage

bei der Hoveraktion über eine parallele Aussage angezeigt, mit welchen dann das Bearbeitungsfenster aufgerufen werden kann.

Das Bearbeitungsfenster soll als Dialogfenster über den normalen Editor erscheinen und zeigt den Inhalt der parallelen Aussage isoliert an. Hier ist es nicht mehr notwendig, dass diese wie bei der parallelen Aussage Komponente extra umklammert werden um die Gruppierung zu veranschaulichen, da die Aussagen bereits isoliert zum Rest des Transkripts im Bearbeitungsfenster identifizierbar sind. Die Darstellung gleicht einzelner Aussagen aus dem Transkript Editor.

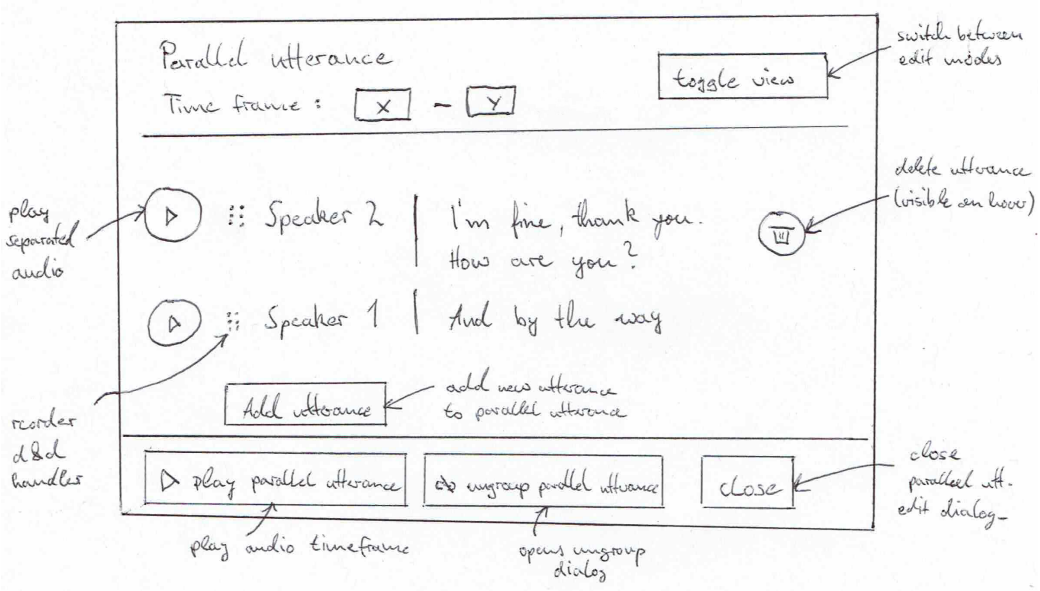

Abbildung 3.3: *Konzeptzeichnung für das Bearbeitungsfenster einer parallelen Aussage.*

Zusätzlich soll es einen zweiten Bearbeitungsmodus geben für die Anpassung der Timestamps einzelner Phrasen in der parallelen Aussage. Hier sind alle Aussagen und ihre Wörter in jeweils einem Zeitstrahl dargestellt. Die Wörter der Aussagen können an den jeweilig richtigen Ort im Zeitstrahl gezogen werden. Somit können die Timestamps der manuell im continuous Modus von Hand erfassten Worte korrigiert werden.

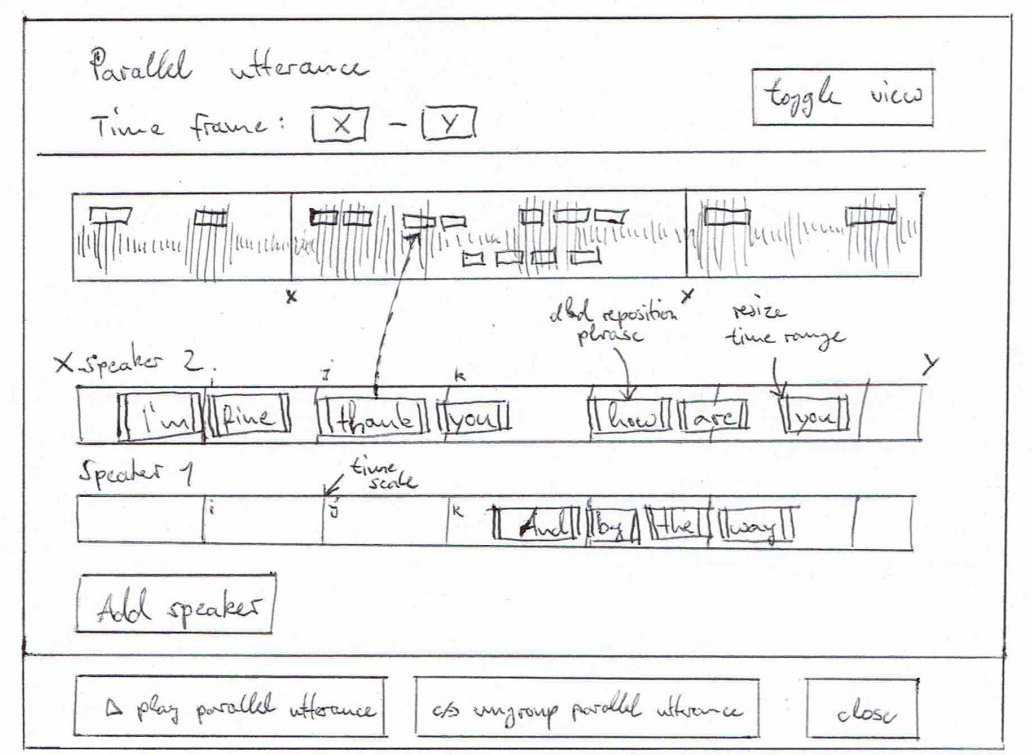

Abbildung 3.4: *Konzeptzeichnung für das Bearbeitungsfenster im "Timesegments"Bearbeitungsmodus.*

Das Auflösen einer parallelen Aussage muss zudem auch möglich sein. Dabei bleibt die Frage, wie einzelne Aussagen aus der parallelen Aussage wieder in das Transkript einfliessen. Ein einfacher Ansatz dafür ist, dass der Nutzer eine Aussage selektieren kann, die zu einer normale Aussage im Transkript umgewandelt wird. Dafür wird ein zusätzliches Dialogfenster vorgesehen, welches die Selektion einer Aussage erlaubt, welche als normale Aussage übernommen werden soll.

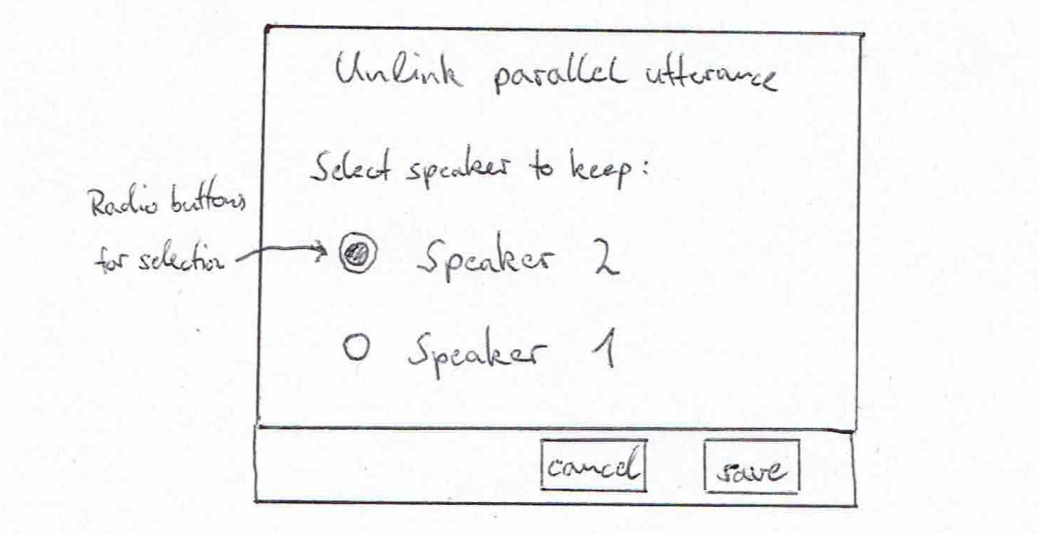

Abbildung 3.5: *Konzeptzeichnung für das Dialogfenster zur Auswahl der Aussage, welche beim Auflösen einer parallelen Aussage behalten werden soll.*

3.1.4 Analyse der Interscriber Backend Anwendung

Das Backend von Interscriber wurde mit Python für die Logik und Verarbeitung der Daten und PostgreSQL für die Speicherung der Daten umgesetzt. Die Bereitstellung der API-Schnittstellen zum Frontend geschieht über das Flask-Framework[28]. Die Kommunikation zwischen Datenbank und Applikation erfolgt über das SQLAlchemy-Framework[29].

3.1.5 Anpassungen an das Datenbankschema

Eine Aufgabe im Backend ist, die relationale Datenbank so anzupassen, damit sie mit parallelen Aussagen umgehen kann. Der für diesen Teil relevante Teil der bestehenden Datenbank ist in der folgenden simplen Abbildung dargestellt:

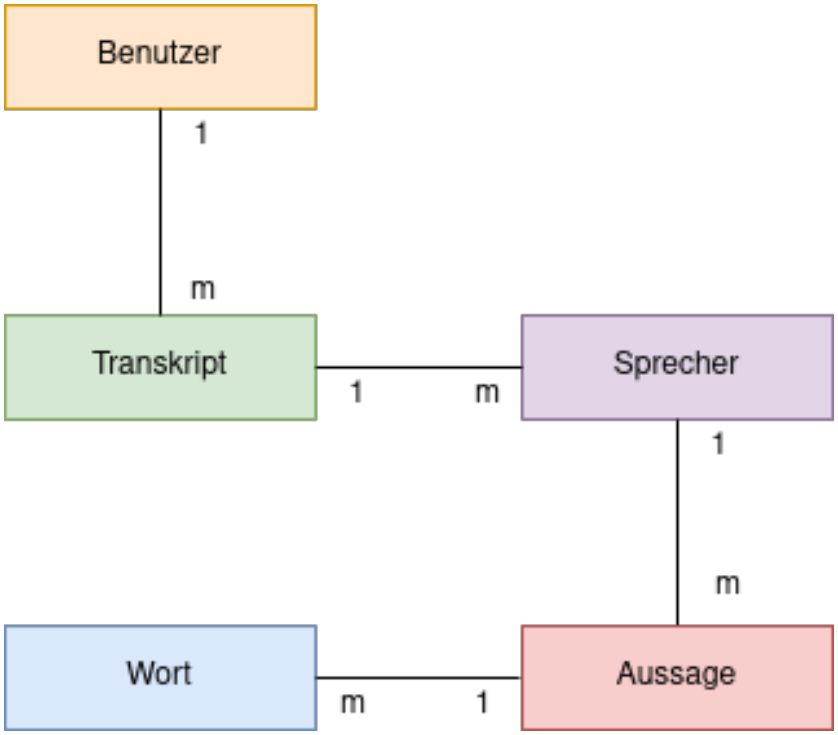

Abbildung 3.6: *Vereinfachtes Datenbankschema von Interscriber*

Ein Wort gehört zu einer Aussage, eine Aussage gehört zu einem Sprecher und ein Sprecher gehört zu einem Transkript. Um ein komplettes Transkript zu erhalten, wird in der Datenbank ein Join über Transkript, Sprecher, Aussage und Wort durchgeführt. Sortiert wird das Resultat nach den jeweiligen Zeitstempeln der einzelnen Worte. Die ganze bestehende Logik der Interscriber Applikation geht implizit davon aus, dass zwei unterschiedliche Worte eines Transkripts nie die gleichen Zeitstempel haben können. Ansonsten funktioniert diese Sortierung und die darauf basierende Logik nicht mehr. Der Ansatz, der verfolgt wird, um parallele Aussagen in dieser Datenbank abspeichern zu können, hat als Ziel, die restliche Logik der Applikation so wenig wie möglich zu beeinflussen. Dies erspart es, die komplette Applikation umschreiben zu müssen.

Die Idee hinter dem Ansatz soll es sein, parallele Aussagen rekursiv aneinander anzuhängen. Eine Aussage kann also neu wieder auf eine Aussage zeigen, was bedeutet, dass diese Aussagen parallel zueinander sind. Dafür muss kein Attribut geändert werden, lediglich ein paar neue hinzugefügt werden. Die Änderung würde dann ungefähr aussehen, wie in der folgenden Abbildung:

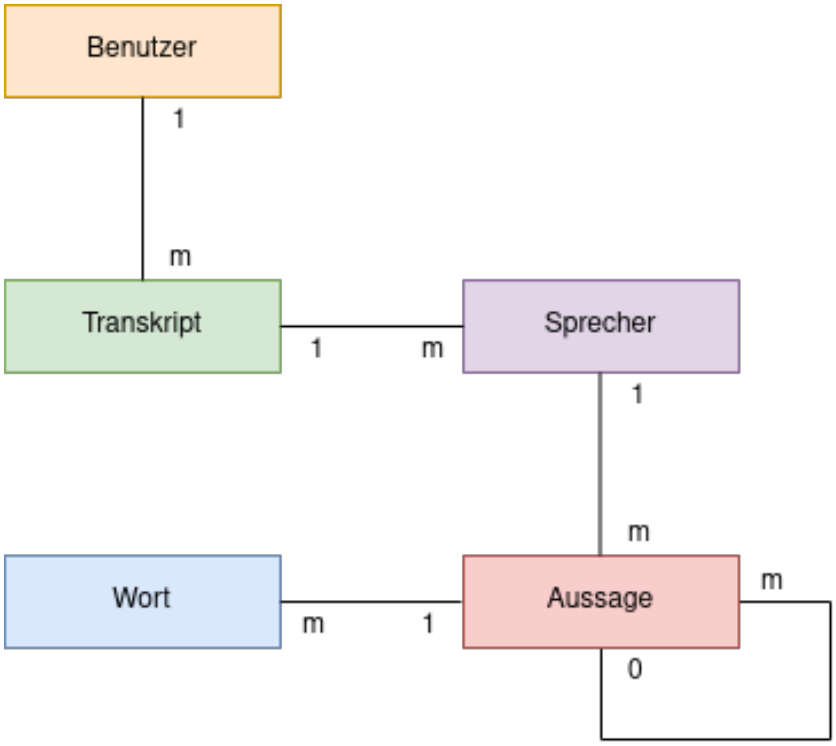

Abbildung 3.7: *Vereinfachtes angepasstes Datenbankschema von Interscriber*

3.1.6 Erweitern der Interscriber API

Damit parallele Aussagen im Backend verarbeitet werden können, sind zusätzlich zu den bestehenden Schnittstellen neue notwendig, die sich um die Verarbeitung der Daten für parallele Aussagen aus dem Frontend kümmern.

Es werden Schnittstellen zur:

- Erstellung
- Auflösung
- Umsortierung
- Hinzufügung
- Entfernung

von parallelen Aussagen benötigt.

Wenn eine parallele Aussage nicht erkannt wurde, muss sie natürlich erstellt werden können. Dazu kann eine normale Aussage in eine parallele Aussage umgewandelt werden.

Im Falle einer Irrtümlichen Erfassung eines parallelen Aussageblocks, muss dieser wieder aufgelöst werden können. Dazu muss bestimmt, welcher der vorhandenen parallelen Aussagen in eine normale Aussage konvertiert werden soll. Die restlichen Aussagen werden gelöscht.

Die Reihenfolge der Aussagen in einem parallelen Aussageblock soll geändert werden können. Dazu braucht es aus dem Frontend die Daten über die neue Abfolge.

Wenn eine parallele Aussage in einem parallelen Aussageblock fehlt, soll sie diesem Block hinzugefügt werden können. Dazu werden vom Frontend die Daten zum Sprecher und der gesagten Worte benötigt.

Sollte eine parallele Aussage in einem parallelen Aussageblock nicht mehr gewünscht sein, soll diese entfernt und gelöscht werden.

3.2 Parallele Aussagen erkennen

Es muss ein System evaluiert werden, dass zuverlässig zwischen einem einzelnen Sprecher und mehreren gleichzeitigen Sprechern unterscheiden kann.

Ein Ansatz dazu liefern Tsai und Liao. Sie erreichen mit einem stochastischen classifier eine Detektionsrate von bemerkenswerten 98.9% für parallele Aussagen.[7]

Classified	Actual	
	Single-speaker Speech	Overlapping Speech
Single-speaker Speech	99.8%	0.2%
Overlapping Speech	1.1%	98.9%

Abbildung 3.8: *Confusion matrix des Resultats von Tsai und Liao[7]*

Ihr System wurde allerdings nur mit Mandarin und einem eher kleinen, selbst erstellten Datensätzen trainiert und getestet. Es muss also getestet

werden, ob die Resultate auch mit englischer oder deutscher Sprache und auf anderen Datensätzen erreichbar sind. Ebenfalls wurde von Tsai und Liao nicht getestet, wie das System auf Hintergrundgeräusche reagiert.

3.3 Parallele Aussagen separieren

Die Literaturrecherche hat eine Vielzahl an Ansätzen für die Separierung von einzelnen Aussagen in einem überlagerten Audiosignal ergeben. Traditionelle Verfahren basieren auf probabilistische Modellierungsansätzen, nicht-negative Matrixfaktorisierung oder Geräuschquellen Lokalisierung. In der folgenden Auflistung sind einige dieser Ansätze aufgeführt:

- ManyEars [12]
- FASST [13]
- HARK [14]
- openBliSSART [15]

Bei weiterer Recherche haben sich jedoch Ansätze, welche auf der Anwendung von deep neural networks basieren, als am vielversprechendsten herausgestellt. In der Abbildung 3.9 ist ein Scoreplot mit den erreichten Resultaten einige dieser Ansätze aufgeführt. Der Scoreplot wurde aus der Seite “PapersWithCode” [38] entnommen.

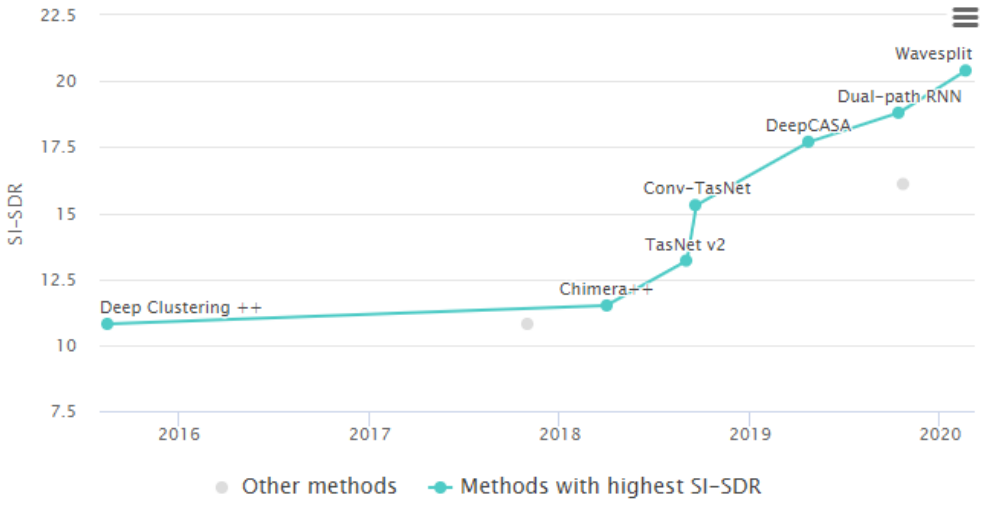

Abbildung 3.9: *Scoreplot dnn basierter speaker separation Systeme angewendet auf den wsj0-2mix [37] Datensatz. [38]*

Einige dieser Ansätze sind hier nochmals aufgeführt:

- Conv-TasNet [39]
- Wavesplit [40]
- Dual-path RNN [41]
- DeepCASA [42]

Ebenso sind wir in der Evaluierungsphase auf ein sehr interessantes Projekt gestossen, das speech-separation Toolkit "Asteroid" [16]. Dieses Projekt wurde für Forscher und Wissenschafler entwickelt und stellt eine Vielzahl an deep-learning basierten Ansätzen, sowie eine gute Auswahl an Datensätzen für diese Probemstellung bereit. Die einzelnen Schritte für das erfogreiche Trainieren eines Netzwerks werden zudem von diesem Tool unterstützt:

- Datensatz bereitstellen
- Audio-Mixturen erzeugen
- Zusammenfassen der Datensatz-Informationen in eine Textdatei
- Trainingsphase eines DNN

- Trainiertes DNN evaluieren

Ziel ist es, mithilfe des "Asteroid" Toolkits ein DNN zu trainieren und anschliessend zu evaluieren, mit welchem erfolgreich parallele Aussagen aus einer Audio-Mixtur separiert werden können. Die Ergebnisse sollen von dem Google speech-to-text System, welches im Interscriber eingesetzt wird, erfolgreich transkribiert werden.

3.3.1 Auswahl des Datensatzes

Das "Asteroid" Tookit unterstützt die Nutzung einiger Datensätze, in Teilen auch für andere Probemstellungen. In der Tabelle 3.1 werden die für die speech-separation relevanten Datensätze aufgeführt.

	wsj0-mix	WHAM	WHAMR	Librimix	SMS-WSJ	Kinect-WSJ
Nr. sources	2 or 3	2	2	2 or 3	2	2
Noise		✓	✓	✓	✓	✓
Reverb			✓		✓	✓
Nr. channels	1	1	1	1	6	4
Sampling rate	16k	16k	16k	16k	16k	16k
Hours	30	30	30	210	85	30
Release year	2015	2019	2019	2020	2019	2019

Tabelle 3.1: *Verfügbare Datensätze für die Aufgabe der speaker-separation, welche in Asteroid bereits unterstützt werden. [16]*

Die meissten Arbeiten verwenden den "wsj0-mix" [37] Datensatz für das Trainieren und Evaluieren der vorgeschlagenen DNN-Architekturen. Dieser ist jedoch nicht frei zugänglich. Die WHAM [43] und WHAMR [44] Datensätze bilden eine Erweiterung auf Grundlage eines bestehenden Datensatzes. Mixturen aus dem gewählten Datensatz werden mit "Noise" und "Reverb" Charakteristiken augmentiert. Daher sind diese Datensätze im Einzelnen nicht brauchbar. Der "LibriMix" [35] Datensatz ist frei zugänglich, bietet Datensätze für 2 und 3 parallele Sprecher, und inkludiert wahlweise Mixturen mit "Noise" Charakteristiken. Er bietet zudem um ein Vielfaches an Testdaten. LibriMix wurde so konzipiert, dass es versucht die Schwächen aus dem "wsj0-mix" auszubessern. Für die Umsetzung der speech-separation soll

augrund der genannten Vorteile der LibriMix Datensatz verwendet werden, welcher direkt über das Asteroid Toolkit geladen werden kann.

3.3.2 Auswahl der DNN-Architektur

Eine vielversprechende DNN-Architektur muss noch gewählt werden. Aus der Abbildung 3.9 geht hervor, dass die Wavesplit Architektur die besten Trainingsergebnisse erzielt hat. Jedoch ist bisher noch keine Implementierung der Architektur zur öffentlichen Verfügung gestellt worden. Für das Dual-Path-RNN hingegen wurde eine Implementierung veröffentlicht. Eine Implementierung dieser Architektur ist zudem auch in dem Asteroid Toolkit vorhanden. Es verspricht gute Resultate und hat eine verhältnismässig kleine Grösse, sodass es einfacher zu trainieren ist. Weitere Architekturen, zum Beispiel das “ConvTasnet”, “Deep Clustering” und “Chimera++” sind ebenfalls in Asteroid verfügbar. Diese Implementierungen erzielen jedoch weniger gute Ergebnisse. Daher fällt die Wahl der DNN-Architektur für die speech-separation auf die DPRNN Architektur.

Kapitel 4

Umsetzung

In diesem Kapitel geht es darum, wie die im vorhergehenden Kapitel beschriebenen Konzepte im Zusammenhang mit der bestehenden Interscriber Anwendung umgesetzt worden sind.

4.1 Darstellung paralleler Aussagen im Editor

Zunächst wird der Transkript-Editor erweitert, um die manuelle Erstellung und Darstellung paralleler Aussagen zu ermöglichen. Aus der Konzeption geht hervor, dass dafür eine neue Vue Komponente implementiert werden muss, welche eine Liste an Aussagen Komponenten zusammenfasst, und diese in einem angepassten Darstellungsformat anzeigt. Die Vue [20] Komponente "ParallelUtterance" wird dazu in der Applikation angelegt. Nachfolgend wird der Begriff "ParallelUtterance" verwendet um auf diese Komponente zu verweisen.

4.1.1 Ausgabe der Komponente für parallele Aussagen im Editor

Aktuell werden in der Editor Komponente alle Aussagen von dem geladenen Transkript über den Vuex store [33] geladen und mittels "dynamic list rendering"[22], [24] als "UtteranceEditable" Komponenten

in die HTML Struktur eingefügt. Die “UtteranceEditable” Komponente soll wiederverwendet werden, um die darin enthaltenen Funktionalitäten in der Liste der Aussagen innerhalb der “ParallelUtterance” Komponente übernehmen zu können.

Das dynamische Einfügen (respektive Rendern) der Aussagen in den Editor macht dabei keine Fallunterscheidung zwischen unterschiedlichen Aussage-Typen, sie erzeugt pro Aussage aus dem Vuex store genau eine “UtteranceEditable” Komponente. Der folgende Codeabschnitt (Listing 4.1) zeigt in vereinfachter Form die Logik für das dynamische Rendern der Aussagen.

Listing 4.1: *Vereinfachter Code für das dynamische Rendern der einzelnen Aussagen in der Editor Komponente*

```
<div
  v-for="(utterance, index) in getUtterances"
  :key="utterance.id"
>
  <div ref="segment">
    <div class="speaker">
      {{ getSpeakerName(utterance.speaker.id) }}
    </div>
    <UtteranceEditable
      class="utterance"
      :utteranceId="utterance.id"
    />
  </div>
```

Wir führen in Vorbereitung zur Fallunterscheidung eine neue Komponente namens “SimpleUtterance” ein, welche lediglich die “UtteranceEditable” Komponente konzeptionell erweitert. Sie führt keine zusätzliche Logik ein. Dadurch ist es möglich, das dynamische Rendern der Aussagen im Editor mit einer Fallunterscheidung zu erweitern.

Die Vue Direktive “v-if” [21] ist dafür ein geeigneter Ansatz. Wird diese Direktive auf eine Komponente angewendet, so wird die Komponente nur von Vue erzeugt wenn die angegebene Bedingung erfüllt wird. Als Bedingung wird überprüft, um welchen Aussagen-Typ es sich bei der aktuellen Aussage im dynamischen Renderprozess handelt. Dafür wird die Datenstruktur einer Aussage im Vuex store mit der zusätzlichen Eigenschaft des Aussagen-Typs erweitert.

Nun ist es möglich, je nach hinterlegtem Typ in der Datenstruktur einer

Aussage, dafür eine andere Vue Komponente im dynamischen Renderprozess auszugeben. Der folgende Codeabschnitt (Listing 4.2) zeigt in vereinfachter Form die Umsetzung für den dynamischen Renderprozess mit mehreren Komponenten, basierend auf einer einfachen Fallunterscheidung.

Listing 4.2: *Vereinfachter Code für das dynamische Rendern einzelner Aussagen mit Fallunterscheidung für die Auswahl der korrekten Komponente je nach Typ der Aussage*

```
<div
  v-for="(utterance, index) in getUtterances"
  :key="utterance.id"
>
  <SimpleUtterance
    v-if="utterance.type === 'single'"
    :utteranceId="utterance.id"
  />
  <ParallelUtterance
    v-else-if="utterance.type === 'parallel'"
    v-bind:utteranceId="'utterance.id"
  />
</div>
```

Parallele Aussagen, welche einer Aussage zugeordnet sind, werden im Vuex store mit dem Typ “pu-meta“ versehen. Dadurch werden diese Aussagen nicht direkt von der Editor Komponente erzeugt, da sie in der Fallunterscheidung keiner Komponente zugeordnet wurden. Dies ist das gewünschte Resultat. In der ParallelUtterance werden nun über das gleiche “dynamic list rendering” Verfahren einzelne Aussagen innerhalb der ParallelUtterance erzeugt. Dazu werden vom Vuex store alle Aussagen vom Typ “pu-meta” geladen, welche die ID der parallelen Aussage als “master-utterance” hinterlegt haben und die Aussagen werden als “UtteranceEditable” plus den Namen des Sprechers in das HTML eingefügt. CSS-Regeln wurden nun in der ParallelUtterance hinzugefügt, um die Liste der einzelnen Aussagen in der Komponente visuell erkenntlich zu machen.

4.1.2 Manuelles Unwandeln einer Aussage in eine parallele Aussage und vice versa

Das Umwandeln einer Aussage in eine parallele Aussage wurde über eine Option im Kontextmenü der Aussage implementiert, welches über einen

Rechtsklick auf die Aussage erreichbar ist. Dabei wird dem Backend über einen API-Aufruf mitgeteilt, welche Aussage nun als parallele Aussage persistiert werden soll. Nachdem die Umwandlung von der Businesslogik abgehandelt wurde, wird der Transkript-Editor die Aussagen neu rendern und die umgewandelte Aussage wird erfolgreich als parallele Aussage dargestellt. Dabei wird automatisch das Bearbeitungsfenster für die neu erstellte parallele Aussage angezeigt, um diese anzupassen und weitere Aussagen in die parallele Aussage aufzunehmen. Wird das Dialogfenster geschlossen ohne weitere Aussagen hinzugefügt zu haben, würde die parallele Aussage nur aus einer Aussage bestehen. Da dies kein gewünschtes Resultat ergibt und auch keinen Sinn macht, wird die parallele Aussage wieder automatisch in eine einzelne Aussage umgewandelt.

Eine parallele Aussage soll zudem auch manuell wieder zu einer normalen Aussage umgewandelt werden können. In der Konzeption haben wir dafür eine Methodik gewählt, bei der beim Umwandlungsschritt zurück in eine normale Aussage genau eine Aussage aus der parallelen Aussage gewählt werden kann, welche als normale Aussage umgewandelt wird. Die restlichen Aussagen aus der parallelen Aussage werden verworfen. Dazu wird eine Vue Komponente Implementiert, welche als Dialog dargestellt wird und die Sprecher der jeweiligen Aussagen anzeigt. Die Auswahl wird dabei mit "Radio buttons" umgesetzt. Diese erlauben die Selektion von genau einer Option aus der gesamten Auswahl. Abbildung 4.1 zeigt das Resultat der Implementierung dieses Dialogs.

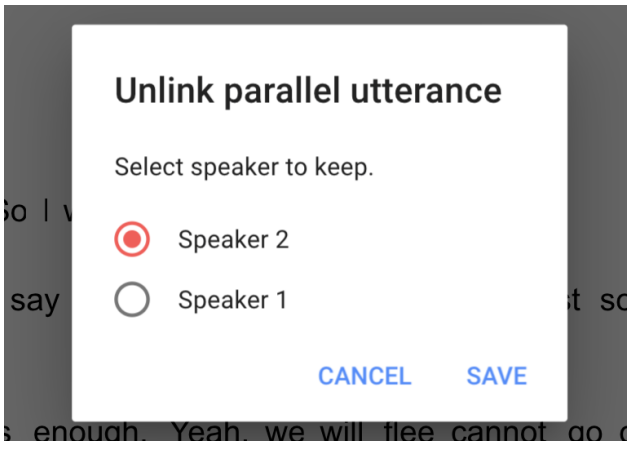

Abbildung 4.1: *Implementierte Darstellung für den "Unlink" Dialog, welcher die Auswahl des zu behaltenen Sprechers, und damit verbunden seine Aussage, ermöglicht. Wird beim Umwandeln einer parallelen Aussage in eine normale Aussage angezeigt.*

4.1.3 Darstellung der Aktionsknöpfe für die Bearbeitung und Auflösung einer parallelen Aussage

Im nächsten Schritt müssen die Bearbeitungsoptionen für eine parallele Aussage implementiert werden. Um schnell die Bearbeitungsoptionen zu einer parallelen Aussage erreichen zu können, werden Aktionsknöpfe hinzugefügt, welche angezeigt werden, wenn man mit der Maus über eine parallele Aussage fährt (auch als Hover bezeichnet). Diese Art von Aktionsknöpfen werden als "floating action buttons" (oder kurz FABs) bezeichnet [46] und werden nach Designvorlage verwendet um die Hauptaktion einer Seite abzubilden. In unserem Fall zeigen wir die FABs nur bei einen Hover an, um die Möglichkeiten zum Bearbeiten und Umwandeln einer parallelen Aussage abzubilden. Es wird ein FAB für das Aufrufen des Bearbeitungsfenster, sowie ein FAB für das Aufrufen des "Unlink"-Dialogs, programmiert. Abbildung 5.1 zeigt die Darstellung der FABs bei einem Hover über eine parallele Aussage im Editor.

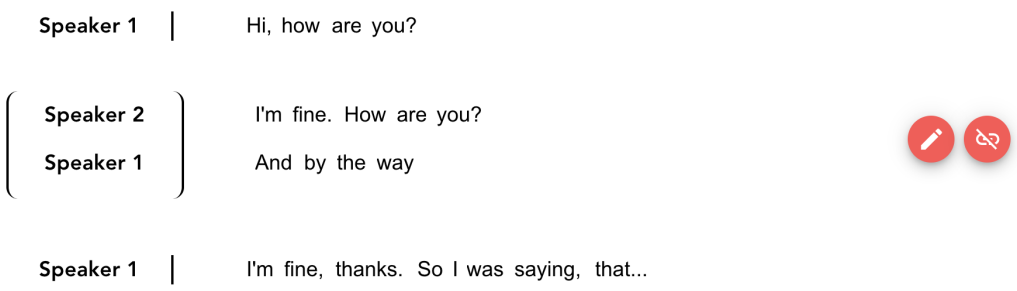

Abbildung 4.2: *Implementierte Darstellung einer parallelen Aussage im Transkript-Editor, welche mit der Maus gehovert wird. Dabei werden die "floating action buttons" zum Bearbeiten und Auflösen dieser parallelen Aussage eingeblendet.*

4.2 Editieren von parallelen Aussagen im Editor

Aussagen im Transkript können nun zu parallelen Aussagen umgewandelt werden. Diese sind jedoch noch nicht anpassbar. In der Konzeptionsphase haben wir dazu einen Editierfenster vorgesehen, welches nun implementiert

werden soll. Die Komponente “EditParallelUtteranceDialog” wird hierfür erzeugt und soll in Form eines Dialogs im Editor dargestellt werden. Der Dialog wird über den Bearbeitungs-FAB in der ParallelUtterance aufgerufen.

Diese Komponente nutzt das gleiche Vorgehen wie die ParallelUtterance um die zugehörigen Aussagen aus dem Vuex store zu laden und sie als “UtteranceEditable” Komponenten mittels “dynamic list rendering” in den Bearbeitungsdialog zu laden. Es werden jedoch andere CSS-Regeln eingesetzt, damit die gewünschte Darstellung der Aussagen im Bearbeitungsdialog erreicht wird.

Aus der Konzeption geht hervor, dass sowohl der Inhalt, als auch die Zeitstempel einzelner Aussagen editierbar sein müssen, und dafür eine Aufteilung der Darstellung in Form von verschiedenen Bearbeitungsmodi vorgesehen ist. Jedoch sollen einige Funktionalitäten unabhängig vom Bearbeitungsmodus zur Verfügung stehen. Zwei Vue Komponenten werden für die Realisierung der Bearbeitungsmodi angelegt. Einmal für den “continious” Modus und einmal für den “timesegments” Modus. In der weiteren Abhandlung werden diese Bezeichnungen verwendet, um auf den jeweiligen Bearbeitungsmodus zu referenzieren. Zunächst werden allerdings die Bearbeitungsmodi-unabhängigen Funktionalitäten hinzugefügt.

4.2.1 Bearbeitungsmodi-unabhängige Funktionalitäten

Im EditParallelUtteranceDialog soll der gesamte Zeitstempel für den Abschnitt im Audiosignal des Transkripts angezeigt werden. Dieser wird bestimmt, indem über alle Aussagen in der ParallelUtterance iteriert und der minimale und maximale Zeitpunkt über alle Zeitstempel bestimmt wird. Hierzu eignen sich sogenannte “computed properties” [23] von Vue.

Computed properties

Computed properties stellen in Vue Funktionen dar, bei welchen der Rückgabewert der Funktion als Eigenschaft (Property) der Vue Komponente gesetzt wird. Diese Property kann dann innerhalb der Komponente an beliebiger Stelle wiederverwendet werden. Dies könnte man auch mit einer einfachen Methode der Komponente realisieren, jedoch gibt es zwischen diesen Varianten einen entscheidenden Unterschied. Methoden werden pro

Aufruf neu evaluiert und die Verarbeitungszeit pro Aktualisierung der Darstellung nimmt zu. Wenn der Rückgabewert zudem kein primitiver Datentyp wie “String” oder “Number” ist, wird pro Methodenaufruf eine neue Referenz zu dem Rückgabewert erzeugt. [25] Das erneute Rendern einer Vue Komponente wird angestossen, sobald sich eine verwendete Property ändert, um damit die Darstellung der Komponente synchron zum Datenmodell zu halten. Dazu überprüft Vue auf die Gleichheit des neuen Wertes zu der aktuellen Property. Bei komplexen Datentypen wird auf die Gleichheit der Referenz überprüft, welche in diesem Fall ungleich ist. [26] Dadurch wird der Renderprzess erneut angestossen, obwohl der tatächliche Inhalt der Datenstruktur sich eventuell gar nicht geändert hat. Computed properties “merken” sich den Rückgabewert (wird auch als “memoization” [27] bezeichnet) und werden erst neu evaluiert, wenn sich eine in der Funktion verwendete Variable ändert. Dadurch wird die Referenz des Rückgabewerts über Renderprozesse hinaus beibehalten und unnötige Berchnungen oder erneutes Rendern anderer Komponenten kann verhindert werden. Das Listing 4.3 zeigt den Code für die Implementierung der computed property.

Listing 4.3: *Berechnung des Zeitabschnitts einer parallelen Aussage basierend auf den enthaltenen Aussagen. Wird als "computed property" in die Vue Komponente implementiert.*

```
computed: {
  // ... ,
  timeFrame: function () {
    let tFrame = {
      from: null,
      to: null
    };
    if (this.parallelUtterances) {
      this.parallelUtterances.forEach(utterance => {
        const utteranceStartTime = utterance.phrases[0].
           startTime
        const utteranceEndTime = utterance.phrases[utterance.
           phrases.length - 1].endTime
        if (tFrame.from === null || tFrame.from >
           utteranceStartTime) {
          frame.from = utteranceStartTime
        }
        if (tFrame.to === null || tFrame.to <
           utteranceEndTime) {
          frame.to = utteranceEndTime
        }
      });
    }
    return frame;
  },
  // ...
}
```

Audioaufnahme im Zeitfenster der parallelen Aussage abspielen

Diese Information über den Zeitabschnitt der parallelen Aussage ist ebenso notwendig, um damit die Audioaufnahme im richtigen Zeitabschnitt abspielen zu können. Diese Aktion wird über den Abspielbutton unten links im Dialog ausgeführt. Dabei wird der globale Audio-Event-Bus [55] angesprochen, dass dieser die Aufzeichnung für diesen Zeitbereich abspielen soll. Der Audio-Event-Bus ist als Vue Komponente bereits im Interscriber integriert.

Auflösen der parallelen Aussage

Weiterhin wird neben dem Abspielbutton ein Button für das Auflösen der parallelen Aussage eingefügt, da diese Aktion auch direkt im Bearbeitungsfenster möglich sein soll. Das Ausführen dieser Aktion ruft den bereits implementierten "Unlink" Dialog auf (zu sehen in Abbildung 4.1), welcher das Auflösen der parallelen Aussage direkt im Bearbeitungsfenster ermöglicht.

Erstellen neuer Aussagen

Das Erstellen einer neuen Aussage innerhalb der parallelen Aussage soll zudem auch zu jedem Zeitpunkt im Bearbeitungsfenster möglich sein. Ein Knopf zum Auslösen dieser Funktionalität wird unter der Liste der bereits vorhandenen Aussagen eingefügt, und ist sichtbar, sofern nicht bereits alle Sprecher aus dem Transkript in der parallelen Aussage vorkommen. Wird dieser geklickt, öffnet sich eine Eingabemaske für die benötigten Attribute einer neuen Aussage. Zum Einen ist das die Auswahl des Sprechers, und zum Anderen ein Eingabefeld für den Inhalt der neuen Aussage. Dabei werden Sprecher, die bereits in der parallelen Aussage vorkommen, aus der Auswahl herausgefiltert. Wird diese Eingabemaske bestätigt, bekommt das Backend des Interscribers die Daten für die neue Aussage zugeschickt, welche daraufhin als Typ "pu-meta" in der Datenbank hinterlegt wird. Der Transkript-Editor läd die neuen Transkript-Daten in den Vuex store ein und die neue Aussage wird in die ParallelUtterance geladen. Abbildung 4.3 zeigt die implementierte Eingabemaske für neue Aussagen.

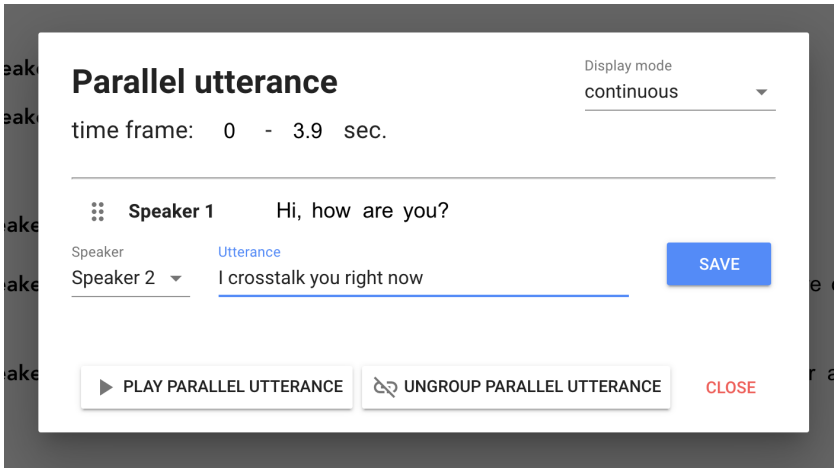

Abbildung 4.3: *Zeigt die Option zum Hinzufügen einer Aussage im Editorfenster einer parallelen Aussage. Das Editorfenster ist hierbei auf den Bearbeitungsmodus "continuous" eingestellt.*

Auswahlbox für den Bearbeitungsmodus

Nun wird noch eine Auswahlbox implementiert, mit welchem zwischen dem "continuous" und dem "timesegments" Modus gewechselt werden kann. Die Auswahlbox wird mit einer Vue Property über die "v-model" [21] Direktive verknüpft, und die Komponenten für die jeweiligen Bearbeitungsmodi werden mit einem conditional rendering über die "v-if" Direktive je nach aktiver Option in der Auswahlbox dargestellt.

4.2.2 Bearbeitungsmodus continuous

Das Grundgerüst des Editordialogs für parallele Aussagen ist erstellt und die Vorbereitungen für das Darstellen der Bearbeitungsmodi ist abgeschlossen. Der Bearbeitungsmodus "continuous" wird nun entwickelt. In diesem Modus soll es möglich sein, Aussagen inhaltlich zu bearbeiten und die Reihenfolge der Darstellung der Aussagen anzupassen. Zudem sollen einzelne Aussagen über einen Aktionsbutton aus der parallelen Aussage gelöscht werden können.

Bearbeitung einzelner Aussagen

Die Funktionalitäten für die inhaltliche Bearbeitung einer Aussage sind bereits in der "UtteranceEditable" Komponente implementiert. Daher wird "UtteranceEditable" Komponente an dieser Stelle wiederverwendet. Folgende Features sind in dieser Komponente inbegriffen:

- Einzelne Phrasen in der Aussage können inhaltlich editiert werden
- Neue Phrasen können der Aussage hinzugefügt werden
- Bestehende Phrasen können aus der Aussage entfernt werden
- Über das Kontextmenü einer Phrase kann zwischen Gross- und Kleinschreibung gewechselt werden
- Ebenfalls über das Kontextmenü können Phrasen aus der Audioaufnahme abgespielt werden
- Beim Hovern über eine Phrase wird dessen Zeitstempel in einem Tooltip angezeigt

Umsortierung einzelner Aussagen

Für die Umsortierung der Aussagen verwenden wir eine Vue Komponente aus der community namens "vue-draggable" [56]. Diese Komponente erlaubt es, die Reihenfolge von Items in einer Liste über einen drag & drop Mechanismus umzusortieren. Um diese Komponente auf die Liste der Aussagen anzuwenden, müssen die einzelnen "UtteranceEditable" Komponenten von der "vue-draggable" Komponente umschlossen werden. Mit den Standardeinstellungen wird allerdings die komplette Aussage als drag & drop Handler konfiguriert. Dieses Verhalten passen wir an, indem wir eine extra Schaltfläche als drag & drop Handler angeben und diese links neben der Aussage platzieren. Wird nun eine Aussage neu sortiert, wird die resultierende Anordnung dem Backend mitgeteilt, und die neue Sortierung wird in der Datenbank persistiert. Im Listing 4.4 wird der Code für die Implementierung der Sortierfunktion mittels der "vue-draggable" Komponente in vereinfachter Form aufgezeigt.

Listing 4.4: *Vereinfachter Code welcher die Anwendung der "vue-draggable" Komponente für die Liste der Aussagen im Bearbeitungsmodus "continuous" demonstriert.*

```
<draggable
  v-model="parallelUtterances"
  @start="drag=true"
  @end="drag=false"
  handle=".handle-drag"
  v-bind="dragOptions"
  class="drag-wrapper"
  v-bind:class="{ isdragging: drag }"
>
  <div class="utterance-wrapper"
    v-for="parallelUtterance in this.parallelUtterances"
    v-bind:key="parallelUtterance.id"
  >
    <div class="speaker">
      <md-button class="handle-drag">
        <md-icon>drag_indicator</md-icon>
      </md-button>
      <div class="speaker-name">
        {{parallelUtterance.speakerName}}
      </div>
    </div>
    <div class="utterance">
      <UtteranceEditable
        :utteranceId="parallelUtterance.id"
        :key="'editable' + parallelUtterance.id"
      />
    </div>
  </div>
</draggable>
```

Löschen einzelner Aussagen

Nun fehlt noch die Funktionalität zum Löschen einer Aussage aus der ParallelUtterance. Hierzu wird das gleiche Konzept wie schon bei der Anzeige der Aktionsbuttons mittels FABs auf einer parallelen Aussage im Editor angewendet. Hovert man über eine Aussage, so wird ein FAB angezeigt, mit dem man die Löschaktion für die Aussage ausführen kann. Dieser ist am rechten Rand der Aussage positioniert. Wird die Löschaktion ausgeführt, wird das Backend wieder über die gewünschte Änderung benachrichtigt und die Aussage wird aus dem Transkript entfernt.

4.2.3 Bearbeitungsmodus timesegments

Der Modus “timesegments” wird nun implementiert. Dieser wird als “TimeframeEditor” Komponente im Interscriber angelegt. Dieser Bearbeitungsmodus soll es ermöglichen, die Zeitstempel einzelner Phrasen in den Aussagen anzupassen. Die Aussagen werden ebenfalls über “dynamic list rendering” erzeugt, jedoch verwenden wir nun keine “UtteranceEditable” Komponente mehr, da die Funktionalitäten dieser Komponente an dieser Stelle nicht gebraucht werden. Stattdessen wird eine neue Komponente angelegt, welche die benötigten Features implementiert. Wir nennen diese Komponente “PhrasesTimeline”. Sie fungiert im übertragenen Sinne als Zeitstrahl, und soll einzelne Phrasen relativ zu ihren Zeitstempeln positioniert darstellen.

Darstellung einzelner Phrasen in einem Zeitstrahl

Zunächst wird die Darstellung einzelner Phrasen in der PhrasesTimeline Komponente realisiert. Dazu werden die einzelnen Phrasen wieder per “dynamic list rendering” in der PhrasesTimeline erzeugt. Mittels CSS-Regeln werden die einzelnen Phrasen absolut positioniert. Die Position und Länge der einzelnen Phrasen wird ebenfalls über CSS-Regeln gesteuert. Da die resultierenden Werte in Pixel angegeben werden, muss die Position aufgrund des Zeitstempels und der gesamten Pixelbreite der PhraseTimeline berechnet werden. Der Zeitstempel ist in der Datenstruktur einer Phrase vorhanden, die Pixelbreite bestimmen wir aus dem gerenderten HTML Element der PhraseTimeline Komponente. Die Berechnung der Position und Breite erfolgt dabei für jede Phrase einzeln. Zusätzlich wird mittels eines Canvas-Elements in der PhrasesTimeline Komponente eine Zeitskala gezeichnet. Die Darstellung der einzelnen Phrasen ist wie vorgesehen implementiert, jedoch fehlen noch jegliche Bearbeitungsoptionen.

Abbildung 4.4: *Darstellung einer Phrase im "timesegments" Modus im Bearbeitungsfenster einer Aussage. Die Phrase ist relativ zu ihrem Zeitstempel positioniert. Die Zeitskala zeigt die Einheiten in 0.2 Sekunden Schritten an.*

Zoomfunktion für einen Zeitstrahl

Zuerst muss eine Property in der "PhrasesTimeline" Komponente hinzugefügt werden, um den Wert für den Zoomfaktor in der Komponente verwenden zu können. Daraufhin wird die Zoomfunktion implementiert, indem eine Event-Handler Funktion auf das "Scroll" Event in einer "PhraseTimeline" Komponente registriert wird. Wenn nun mit der Maus in der "PhrasesTimeline" Komponente scrollt, wird die Event-Handler Funktion ausgeführt. Diese Setzt einen neuen Wert für die Zoomfaktor Property. Damit die Phrasen tatsächlich vom Zoomfaktor beeinflusst werden, muss der Zoomfaktor noch in die Berechnung der Position und Breite jeder Phrase einfliessen.

Zoomfaktor und Skrollposition synchronisieren

Als Nächstes sollen die Einstellungen des Zoomfaktors und der Skrollposition eines Zeitstrahls mit den restlichen Zeitstrahlen sychronisiert werden, damit die gewählte Ansicht für alle "PhrasesTimeline" Komponententen konsistent bleibt. Dazu werden Methoden in der "PhrasesTimeline" Komponente implementiert, mit welchen die entsprechenden Properties gesetzt werden. Diese Methoden sind dabei auch ausserhalb dieser Komponente aufrufbar. Die Idee besteht darin, dass beim Setzen einer der Properties ein Event an die darüberliegende Komponente, dem "TimeframeEditor",

geschickt wird. Dieses Event überliefert den neuen Wert der Property an den TimeframeEditor. Dieser wiederum ruft in allen “PhrasesTimeline” Komponenten die Methode zum Setzen der Property auf. Da die “PhrasesTimeline“ Komponenten dynamisch erzeugt wurden, müssen diese in Vue über die “$ref” Direktive angesprochen werden.

Editieren des Zeitstempels einer Phrase

Zum Editieren der Zeitstempel einzelner Phrasen werden drag-and-drop Mechanismen implementiert. Das Phrasenelement kann nun mit gehaltener Maustaste verschoben werden. Dabei wird die Mausbewegung in X Richtung gemessen und auf die Position der gehaltenen Phrase angewendet. Die drag-and-drop Handler für das Ändern der Länge einer Phrase werden an den Rändern der Phrase angesetzt.

Abbildung 4.5: *Darstellung des Dragindikators für das Verschieben einer Phrase auf der “PhrasesTimeline” Komponente.*

Abbildung 4.6: *Darstellung des Dragindikators für das Reskalieren einer Phrase auf der “PhrasesTimeline” Komponente.*

Darstellung der Waveform der Audioaufnahme

Die Darstellung der Waveform für die Audioaufnahme des Transkripts soll einen Überblick über den zeitlichen Abschnitt von der Phrase im Transkript geben. Als Ansatz wählen wir die Nutzung der Web-Audio-API [57], welche in den gängigen Webbrowsern standartisiert zur Verfügung steht. Ein Audio Kontext wird dazu erstellt und ein ArrayBuffer wird aus den Audiodaten erzeugt. Daraufhin wird versucht, die Funktion “decodeAudioData” der Audio Kontext Instanz zu nutzen, um den benötigten AudioBuffer für die Weiterverarbeitung zu erstellen. Es hat sich herausgestellt, dass die Funktion

“decodeAudioData” nicht mit WAVE [58] formatierten Daten umgehen kann, die Audioaufnahme wird jedoch nur in diesem Format bereitgestellt. Ein alternativer Ansatz zum Vearbeiten des Audiosignals mit JavaScript für die Darstellung der Waveform, konnte im Rahmen dieser Arbeit nicht erarbeitet werden. Dieser Teil der Implementierung konnte nicht erfüllt werden. Der Bearbeitungsmodus ist dadurch allerdings nicht in seinem geplanten Funktionsunfang eingeschränkt.

4.3 Erweitern der Schnittstellen im Backend

Damit das Backend die neue Funktionalität bereitstellen kann, sind neue Schnittstellen zum Frontend erstellt worden. Diese sind:

- Erstellen - Eine Aussage wird als “parallel parent” markiert. An solch eine Aussage können Aussagen mit Typ “parallel child” angehängt werden.
- Auflösen – Einer Aussage wird die “parallel master” Markierung entfernt. Alle angehängten Aussagen werden gelöscht.
- Hinzufügen – Einer Aussage wird mit den Daten aus dem Frontend erstellt, als “parallel child” markiert und an eine Aussage angehängt, die als “parallel parent” markiert ist. Der Rang der neuen Aussage entspricht der neuen Anzahl Kinder des “parallel parents”. Sie wird also ans Ende der Liste gestellt.
- Entfernen – Eine Aussage die als “parallel child” markiert ist, wird gelöscht und vom entsprechenden “parallel parent” abgehängt. Die Ränge der verbliebenen Kinder wird angepasst.
- Anpassen der Reihenfolge – Die Reihenfolge der parallelen Aussagen untereinander wird angepasst.

4.4 Anpassungen an dem Datenbankschema

Um diese neuen Eigenschaften, die in der Businesslogik erstellt wurden, auch in der Datenbank zu modellieren, wurde die Datenbanktabelle für die Aussagen mit folgenden Attributen erweitert:

- Aussagetyp, welcher entweder "nicht-parallel", "parallel-parent" oder "parallel-child" ist.
- Eine rekursive Beziehung, welche bei Aussagen, die den Typ "parallel-parent" haben, auf die entsprechenden "parallel-child" Aussagen zeigt.
- Paralleler Rang, welcher die Reihenfolge der Aussage innerhalb eines parallelen Aussageblocks angibt.

Wichtig zu beachten ist, dass der "parallel-parent" sozusagen die Hauptaussage ist, und die "parallel-childs" sind die dazu parallelen Aussagen. Alle "parallel-childs" eines parallelen Aussageblocks zeigen dabei auf den selben "parallel-parent". Somit können alle "parallel-childs" auf einmal abgefragt werden. Dies ist auch der Grund für das eingeführte Rang Attribut. Ohne dieses könnte die Reihenfolge der Aussagen nicht mehr abgebildet werden.

Die Alternative war, die zueinander parallelen Aussagen in einer Art verketteten Liste abzuspeichern. Dies hätte das Rang Attribut überflüssig gemacht, jedoch wurde damit der Datenzugriff langsamer und die Umsortierungsfunktion komplizierter. Anstatt einfach die Rang Attribute anzupassen, müssten dann alle Zeiger umgehängt werden.

Durch die Verwendung von SQLAlchemy als Framework zur Kommunikation zur Datenbank, kann für die Erweiterung der Datenbank auf das Tool Alembic[30] zurückgegriffen werden, was eine Datenbankmigration vereinfacht. Alembic erkennt automatisch Änderungen am Datenmodell und erstellt eine passende Migration für die Datenbank.

In der folgenden Abbildung wird die Anpassung noch einmal graphisch gezeigt:

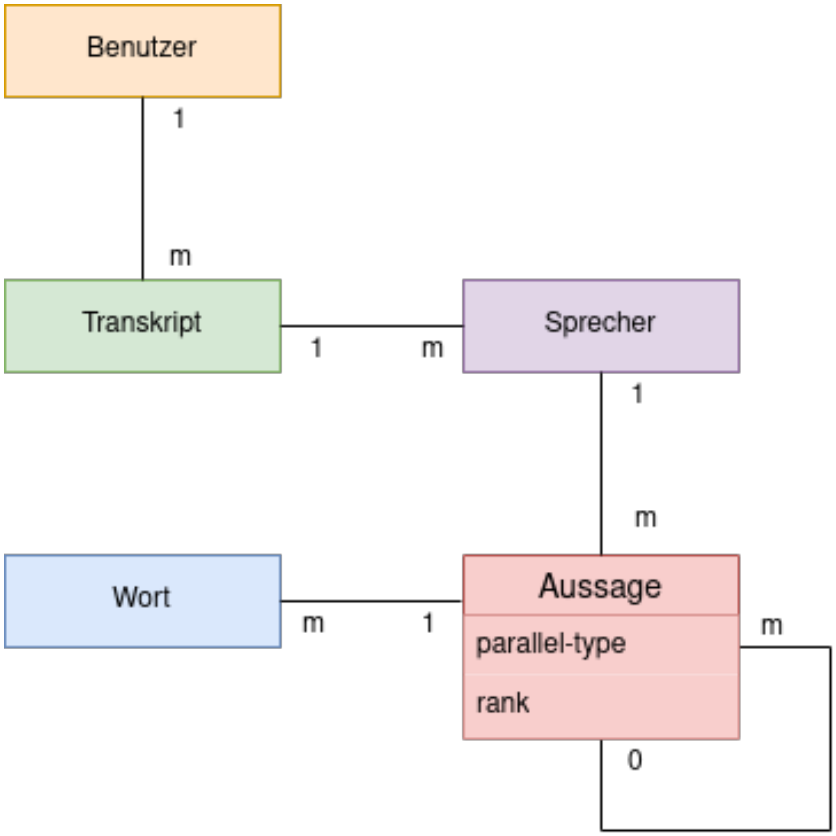

Abbildung 4.7: *Erweitertes vereinfachtes Datenbankschema von Interscriber*

4.5 Speech-separation mit Asteroid

In der Konzeptionsphase wurde die Nutzung des Toolkits “Asteroid” [16] für die speeker-separation vorgeschlagen. Weiterhin fiel die Wahl der Netzwerk-Architektur auf das “Dual-Path-RNN” [41] in Kombination mit dem “LibriMix” [35] Datensatz.

Die Schritte der Umsetzung orientieren sich an die vorbereiteten Etappen in Asteroid. Ein zentrales Shell-Script wird hierfür von dem Toolkit mitgeliefert, welches bereits mit sinnvollen Voreinstellungen befüllt ist. Mit dem Argument “–stage” lässt sich die Ausführung einzelner Etappen steuern. Da somit die Durchführung der einzelnen Steps trivial sind, werden keine näheren Erläuterungen zu der Codeausführung aufgeführt. Bevor die Schritte ausgeführt werden können, muss vorher noch die Vorlage für das Trainieren

des DPRNN-Netzwerks angepasst werden. In der Vorlage wird der “WHAM“ [43] Datensatz genutzt. Der Datensatz wird auf LibriMix umgestellt.

Datensatz vorbereiten

In der ersten Phase wird der LibriMix Datensatz heruntergeladen. Für den vollen Datensatz benötigt man zirka 600 Gigabyte an Festplattenspeicher. Er beinhaltet 2 Varianten, welche sich in der Menge der Daten unterscheiden: einen 100 Stunden Trainings-Datensatz und einen 360 Stunden Trainings-Datensatz. Die verwendeten Daten für die Erstellung der Mixturen kommen ursprünglich aus dem LibriSpeech [36] Datensatz. Diese reinen Audiodaten aus dem LibriSpeech Datensatz werden zu einer Audiomixtur zusammengefügt. Für eine erweiterte Problemstellung wird zusätzlich noch eine Version mit zusätzlichen Störgeräuschen (Noise) erzeugt. Speziell bei LibriMix in Asteroid ist, dass das Herunterladen des Datensatzes, die Erstellung der Mixturen und das Erzeugen der Metainformationen zusammen in einem Script abgehandelt wird. Dadurch kann nach Abschluss dieser Phase mit dem Training begonnen werden.

Trainieren des DPRNN-Netzwerks

In diesen Experimenten beschränken wir uns vorerst auf einen spezifischen Teil des Datensatzes.

- Datensatz mit 100 Stunden Trainingsdaten
- 2 Sprecher in der Mixtur
- 2 Sekunden Segmente
- Mixturen ohne Störgeräusche

Version 1: 16k clean

Zunächst wird ein naiver Ansatz getestet, um ein Gefühl für die Stolpersteine in der Trainingsphase zu bekommen. Wir setzen die Parameter für die erste Version des Netzwerks sehr hoch. Folgende Parameter sind dabei interessant:

- Sample Rate: 16000
- RNN repeats: 6
- Batch size: 1

Die restlichen Parameter sind auf den Vorgegebenen Werten geblieben. In diesem Training ist aufgefallen, dass eine Batch size von 1 zu einem unstabilen Trainingsfortschritt führt. Zudem ist die Trainingsdauer sehr hoch; eine Epoche benötigt 3 Stunden. (Bei 100 Stunden Trainingsdaten)

Version 2: small 16k clean

Da das Modell im vorigen Schritt bereits 9 von 11 Gigabyte Grafikspeicher benötigte, haben wir uns entschieden die Anzahl der RNN Blöcke innerhalb der DPRNN Architektur zu halbieren. Damit konnte diese Version mit einer Batch size von 2 trainiert werden. Die Performance des Trainings hat damit spürbar zugenommen, und es konnten in weniger Zeit bessere SI-SDR [50] Werte erreicht werden. Wir verwenden für dieses Modell folgende Parameter:

- Sample Rate: 16000
- RNN repeats: 3
- Batch size: 2

Version 3: small 16k noisy

In dieser Version haben wir statt der Parameter, den Datensatz auf Testdaten mit Störgeräuschen umgestellt. Dieses Modell verwendet die gleichen Hyperparameter wie Version 2. Wie bereits erwartet, macht das Training nicht mehr so schnell Fortschritte. Die Lernrate ist in etwa gleich gut zu der aus Version 1 beobachteten Lernrate.

Version 4: 8k clean

Für diese Version wird eine sampling rate von 8000 Hertz für die Mixturen verwendet. In den Literaturverweisen werden ausschliesslich diese sample rate

verwendet, daher wollen wir die Performance in unserem Setup überprüfen. Wir verwenden für dieses Modell folgende Parameter:

- Sample Rate: 8000
- RNN repeats: 6
- Batch size: 2

Version 5: small 8k clean

Als letztes wird eine Version mit 3 RNN Blöcken und 8000 Hertz trainiert. In diesem Experiment wollen wir beobachten, wie schnell das Training vorankommt. Parameter:

- Sample Rate: 8000
- RNN repeats: 3
- Batch size: 3

Die Resultate der trainierten Versionen sind im Kapitel 5.3 aufgeführt.

Kapitel 5

Ergebnisse

5.1 Erstellung und Editierung paralleler Aussagen

Durch die Implementierung von parallelen Aussagen in den Transkript-Editor der Interscriber Anwendung, können nun im Transkript-Editor parallele Aussagen verwaltet werden. Zusammengehörige parallele Aussagen sind durch eine Klammer um die beteiligten Sprecher gekennzeichnet, siehe Abbildung 5.1. Es ist in dieser Ansicht nicht erkennbar, wie sich die beiden parallelen Aussagen zeitlich zueinander verhalten.

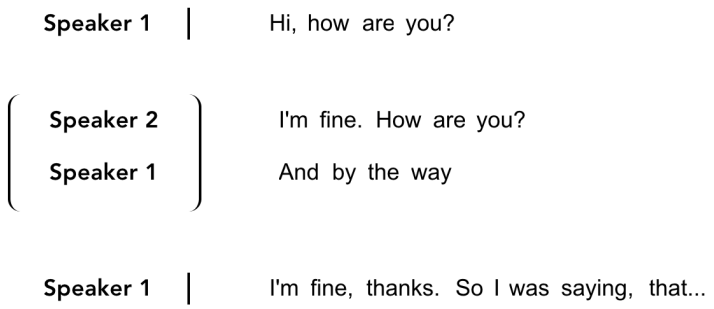

Abbildung 5.1: *Zeigt das Ergebnis der Implementierung paralleler Aussagen im Transkripteditor.*

Zur Bearbeitung eines parallelen Aussageblocks wurden zwei verschiedene

Modi erarbeitet. Der continous Modus ist der einfachere der zwei Modi. Er ermöglicht es, parallele Aussageblöcke zu bearbeiten. Es können Aussagen hinzugefügt, entfernt und verändert werden. Korrekturen an den Wörtern der Aussage kann direkt im Dialog vorgenommen werden. In der Abbilding 5.2 wird der Dialog für den continuous Modus gezeigt.

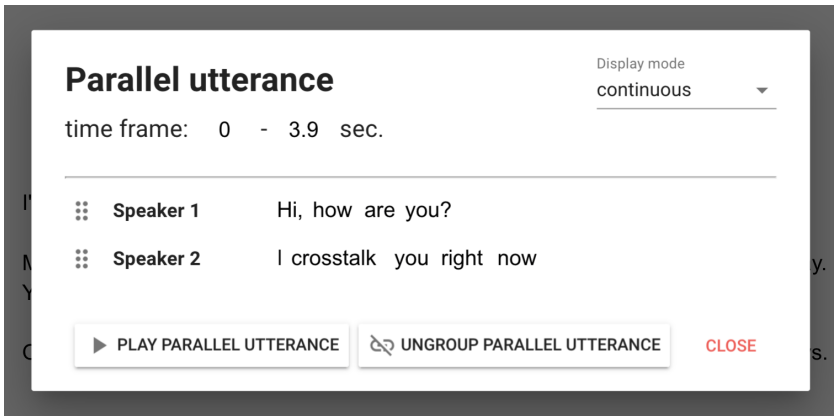

Abbildung 5.2: *Zeigt das Ergebnis der Implementierung des Editorfensters zum Bearbeiten von parallelen Aussagen im "continuous"Modus.*

Der zweite Modus ist der timesegments Modus. Dieser ermöglicht es, die Zeitstempel der Wörter anzupassen. Dies ist deshalb relevant, da von Hand erfasst Aussagen nur geschätzte Zeitstempel besitzen. Um nun diese Zeitstempel zu korrigieren, kann die Länge eines Wortes, wie auch die Position im Vergleich zu den Worten der dazu parallelen Aussagen angepasst werden. In der durch Abbildung 5.3 gezeigten Situation die Worte "Hi" von Speaker 1 und "you" von Speaker 2 praktisch gleichzeitig geäussert worden.

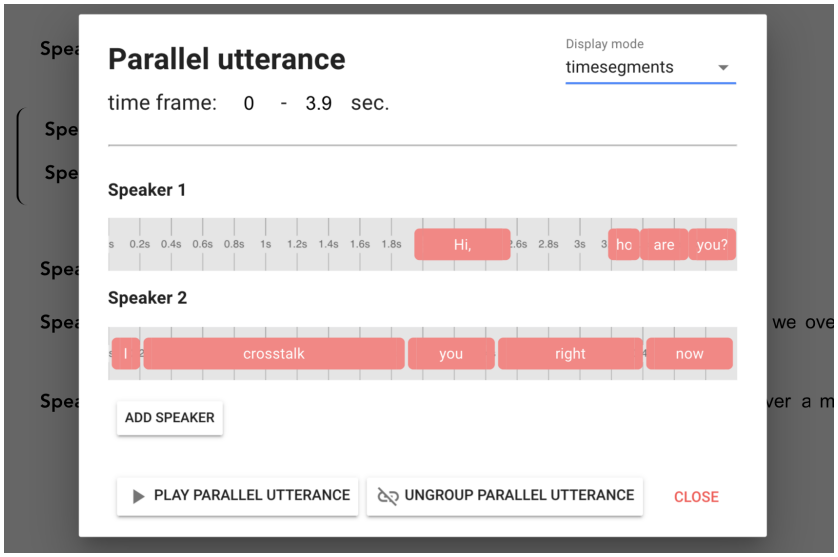

Abbildung 5.3: *Zeigt das Ergebnis der Implementierung des Editorfensters zum Bearbeiten von parallelen Aussagen im "timesegments"Modus.*

5.2 Automatische Erkennung paralleler Aussagen

Das Ergebnis für die automatische Erkennung von parallelen Aussagen ist ernüchternd. Auf Grund mangelnder Zeit wurde das Problem bis auf eine Literaturrecherche nicht behandelt.

5.3 Separieren einzelner Aussagen

In der Durchführung der speaker-separation Aufgabe wurden fünf verschiedene Versionen eines neuralen Netzwerks basierend auf der DPRRN [47] Architektur trainiert. Die verwendeten Parameter können im Kapitel 4 nachgeschlagen werden. Die Resultate für die Performancemessungen von den Authoren [16], [47] konnten mit der gewählten Trainingszeit nicht erreicht werden, jedoch ergaben die Auswertungen der Tests den Umständen entsprechend gute Resultate. Dabei ist zu beachten, dass wir den "LibriMix" [35] Datensatz anstelle des "wsj0-2" [37] Datensatz für unsere Durchführung

gewählt haben. Ein direkter Vergleich ist somit ohnehin nicht möglich. Zudem ist die Trainingsphase des Netzwerks ein zeitintensiver Prozess, daher wurden die verschiedenen Versionen des Netzwerks nicht einheitlich lang trainiert. Abbildung 5.4 zeigt die erreichten Trainings-Epochen über die dabei aufgewendete Zeit.

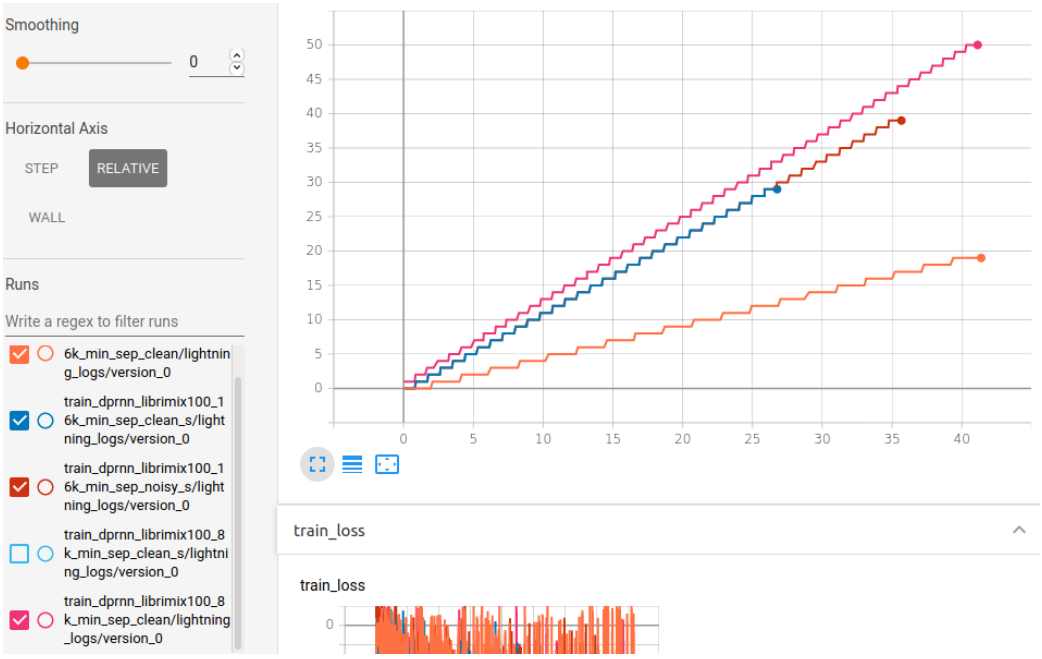

Abbildung 5.4: *Trainingszeit der trainierten Netzwerke für speaker-separation.*

Für die Validierung der Resultate aus den trainierten Netzwerken wurden pro Version zehn verschiedene Audio Mixturen verwendet, welche im Trainingsdatensatz nicht vorkommen, dass heisst diese Mixturen sind den Netzwerken völlig unbekannt. Das verwendete Framework “Asteroid” [16] verwendet für die Validierung des Netzwerks das Python package “pb_bss_eval”[49] und beinhaltet verschiedene “expectation-maximization” Algorithmen (EM-Algorithmen) für speech-separation. Die Grundlagen für die Implementierten EM-Algorithmen stammen aus dem Paper ”Tight integration of spatial and spectral features for BSS with Deep Clustering embeddings”[48]. Die Tabelle 5.1 fürt die Resultate der Validierung für jedes trainierte Netzwerk auf. Folgende Werte wurden dabei gemessen:

- **SI-SDR**: scale-invariant signal-to-distortion ratio [50]

- **SDR**: signal-to-distortion ratio [51]
- **SIR**: signal-to-interference ratio [51]
- **SAR**: signal-to-artifacts ratio [51]
- **STOI**: short-time objective intelligibility [52]

	16k clean	Small 16k clean	Small 16k noisy	8k clean	Small 8k clean
SI-SDR	8.5806	10.7284	7.6824	**12.5477**	11.4050
SDR	9.0982	11.2499	8.3713	**13.1520**	12.0121
SIR	15.2600	18.6709	18.9057	**21.1224**	19.0308
SAR	11.0146	12.6146	9.2193	**14.2447**	13.3665
STOI	0.8509	0.8849	0.8124	**0.9031**	0.8921

Tabelle 5.1: *Resultate der Validierung der in dieser Arbeit trainierten DPRNN Netzwerke.*

In der gegebenenen Trainingszeit konnten die besten Ergebnisse mit der "8k clean" Version des Netzwerks erreicht werden, also mit einer Abtastrate von 8 Kiloherz und keine zusätzlichen Störgeräusche für das verwendete Audiosignal. Dabei ist uns aufgefallen, dass die Trainingsrate relativ stark von der gewählten Batchsize abhängig ist. Die Batchsize bestimmt, wie viele Datenpaare gleichzeitig für eine Trainingsphase des Netzwerks genutzt werden. Um so höher die Batchsize gewählt wird, desto stabiler wird der Verlauf des Gradienten, welcher die Grundlage für das Aktualisieren der Parameter im Netzwerk darstellt, und damit einen stabileren "Lernprozess" ermöglicht [53]. In der Durchführung wurde eine einzelne "GForce RTX 2080 Ti" [54] Grafikkarte verwendet, welche 11 Gigabyte an Speicherkapazität besitzt. Die "16k clean" Version des Netzwerks hat bei einer Batchsize von 1 bereits 9 Gigabyte Speicher benötigt. Somit war es nicht möglich eine höhere Batchsize für diese Version zu wählen. Der Trainingsfortschritt ist viel weniger konsistent im Vergleich zu den Trainingsphasen der anderen Versionen mit höherer Batchsize. Wir verwenden in der Versionsbezeichnung einiger Netzwerke das Synonym "Small", welches aussagt, dass 3 RNN Blöcke in der DPRNN Architektur (statt in der vorgeschlagenen Architektur mit 6 RNN Blöcken) verwendet wurden. Durch die kleinere Architektur konnte die Batchsize erhöht und damit die Performance der Trainingszeit angehoben werden. Dies muss bei der Interpretation der Ergebnisse berücksichtigt

werden. Stünden mehr Resourcen und Zeit für das Trainieren der Netzwerke zur Verfügung, können unter Umständen andere Ergebnisse für die Wahl der besten DPRNN Version herauskommen. Die Abbildung 5.5 veranschaulicht die Validierungsresultate für alle Versionen nach jeder Epoche in der Trainingsphase.

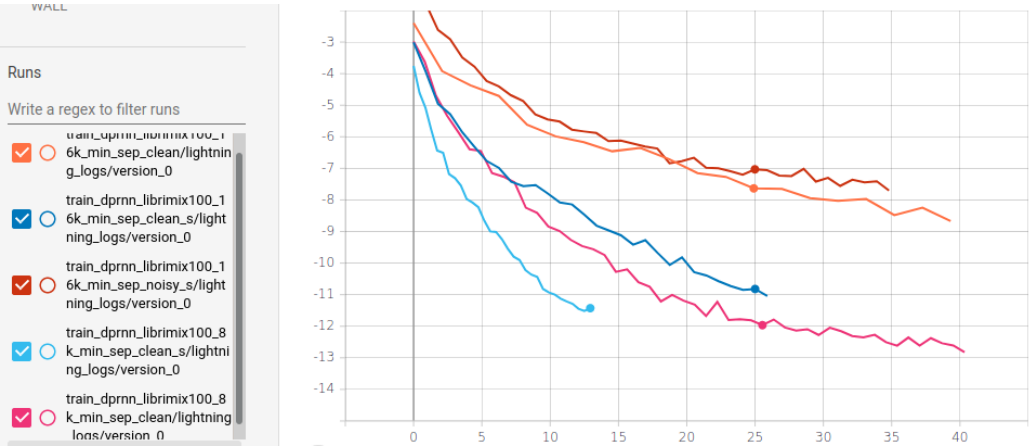

Abbildung 5.5: *Validierungsergebnisse der trainierten Netzwerke für speaker-separation.*

Kapitel 6

Fazit

6.1 Interpretation der erreichten Ziele

Unsere Implementation von parallelen Aussagen im Interscriber zeigt eine Möglichkeit, wie diese in einer Transkriptionstool umgesetzt werden können. Da ein automatisches Transkript nie komplett ohne Fehler sein wird, ist es wichtig, dass es komfortable und praktische Lösungen gibt, wie diese Fehler korrigiert werden können. Diese haben wir mit unserer Lösung geschaffen.

Durch die Evaluation von Methoden zur Separation einzelner Aussagen in einem überlagerten Audiosignal wird gezeigt, wie daraus die einzelnen Aussagen herausgefiltert werden können. Dies bietet zukünftig die Möglichkeit, parallele Aussagen einzeln zu transkribieren.

6.2 Review zur Erfüllung der Zielsetzung

Das Ziel der Darstellung paralleler Aussagen im Transkript-Editor wurde erreicht. Parallele Aussagen können von einem Interscriber Benutzer von Hand erfasst, geändert und danach auch sinnvoll dargestellt werden. Durch den timesegment Modus ist es zudem möglich, dass die Benutzer die Zeitstempel der Worte innerhalb einer parallelen Aussage korrigieren können.

Das Ziel der Separierung einzelner Aussagen wurde zumindest als Proof of Conecpt erreicht. Es kann ein überlagertes Audiosignal mit mehreren Sprechern so separiert werden, dass für jeden Sprecher ein eigenes

Audiosignal entsteht. Die Integration dieses Systems im Interscriber fehlt jedoch.

Das Ziel zur Erkennung von paralleler Sprache ist definitiv nicht erreicht worden. Dies bedeutet für den Interscriber im Moment, dass die Verarbeitung von parallelen Aussagen nicht komplett automatisiert ablaufen kann.

6.3 Taktischer Ausblick

Dadurch, dass das Ziel der automatischen Erkennung von parallelen Aussagen nicht erreicht worden ist, kann als Zwischenlösung die Arbeit, die eigentlich der Algorithmus erledigen sollte, auf den Benutzer abgewälzt werden. Denkbar wäre hier die Möglichkeit, dass der Benutzer von Hand angibt, in welchen Abschnitten sich parallele Aussagen befinden, und diese dann automatisch separiert und einzeln transkribiert werden. Hier muss jedoch daran gedacht werden, dass die Sprecher der einzeln transkribierten Aussagen noch irgendwie auf die Sprecher des restlichen Transkripts gemappt werden müssen. Das von Interscriber momentan verwendete Speech-to-Text System von Google ist dazu im Moment noch nicht fähig.

Ein weiteres Ziel ist sicherlich, den geschriebenen Code zu refaktorisieren, um ihn aufzuräumen und übersichtlicher zu machen. Dies wird den zukünftigen Entwicklern das Leben erleichtern, da ein übersichtlicher Code einfacher zu warten ist. Das ist vor allem deshalb relevant, da das Softwareprodukt dieser Bachelor-Arbeit nicht einfach nur ein Proof of Concept ist, sondern Teile davon auch produktiv eingesetzt werden sollen.

Ebenfalls sollte überlegt werden, ob es eine bessere und genauere Möglichkeit gibt, die Zeitstempel der von Hand erfassten Worte zu bestimmen.

6.4 Strategischer Ausblick

Ein grosser Punkt für eine zukünftige Arbeit ist ganz klar, ein System zu entwickeln, dass parallele Aussagen in einer Audioaufnahme erkennen kann. Dies scheint im Moment ein noch ungelöstes Problem zu sein. Eventuell kann dazu das System für die Separierung von parallelen Aussagen benutzt werden, um dann an Hand der separierten Audiosignale zu erkennen, ob eine parallele Aussage vorliegt.

Sobald ein System zu Erkennung von parallelen Aussagen existiert, sollte es möglich sein, ein praktisch vollautomatisches Transkriptionstool zu erstellen, welches auch mit parallelen Aussagen umgehen kann. Es müsste dann lediglich noch korrekturgelesen werden, sonst aber würde keine manuelle Arbeit mehr anfallen.

Ein weiterer Punkt ist die Weiterentwicklung des hauseigenen Speech-to-Text Systems, um von Google und anderen Anbietern unabhängiger zu werden. So müssen die Daten der Nutzer nicht in fremde Hände gegeben werden. Dies würde es ebenfalls erlauben, die Verarbeitung von parallelen Aussagen direkt in Speech-to-Text System zu integrieren.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Jason Brownlee. (Aug. 2019). What is deep learning?, Adresse: https://machinelearningmastery.com/what-is-deep-learning/ (besucht am 30. 06. 2020).

[2] Google. (2020). Speech-to-Text: Automatic Speech Recognition, Adresse: https://cloud.google.com/speech-to-text/ (besucht am 30. 06. 2020).

[3] Adobe. (2020). What are web applications and dynamic web pages, Adresse: https://helpx.adobe.com/dreamweaver/using/web-applications.html (besucht am 30. 06. 2020).

[4] M. H. Weik, "Automatic speech recognition", in *Computer Science and Communications Dictionary*. Boston, MA: Springer US, 2001, S. 88–88, ISBN: 978-1-4020-0613-5. DOI: 10.1007/1-4020-0613-6_1147. Adresse: https://doi.org/10.1007/1-4020-0613-6_1147.

[5] Y. Luo und N. Mesgarani, "Tasnet: Time-domain audio separation network for real-time, single-channel speech separation", in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, S. 696–700.

[6] D. Wang und J. Chen, "Supervised speech separation based on deep learning: An overview", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Jg. 26, Nr. 10, S. 1702–1726, 2018.

[7] W. Tsai und S. Liao, "Speaker identification in overlapping speech", *Journal of information Scienece and engineering*, Jg. 26, S. 1891–1903, 2010.

[8] E. C. Cherry, "Some experiments on the recognition of speech, with one and with two ears", *J. Acoust. Soc. Amer*, Jg. 25, pp. S. 975–979, 1953.

[9] P. Huang, M. Kim, M. Hasegawa-Johnson und P. Smaragdis, "Joint optimization of masks and deep recurrent neural networks for monaural source separation", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Jg. 23, Nr. 12, S. 2136–2147, 2015.

[10] X. Zhang und D. Wang, "A deep ensemble learning method for monaural speech separation", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Jg. 24, Nr. 5, S. 967–977, 2016.

[11] Y. Isik, J. L. Roux, Z. Chen, S. Watanabe und J. R. Hershey, "Single-channel multi-speaker separation using deep clustering", *Interspeech 2016*, Sep. 2016. DOI: 10.21437/interspeech.2016-1176. Adresse: http://dx.doi.org/10.21437/Interspeech.2016-1176.

[12] F. Grondin, D. Létourneau, F. Ferland, V. Rousseau und F. Michaud, "The manyears open framework", *Auton. Robots*, Jg. 34, Nr. 3, S. 217–232, Apr. 2013, ISSN: 0929-5593. DOI: 10.1007/s10514-012-9316-x. Adresse: https://doi.org/10.1007/s10514-012-9316-x.

[13] Y. Salaün, E. Vincent, N. Bertin, N. Souviraà-Labastie, X. Jaureguiberry, D. T. Tran und F. Bimbot, *The Flexible Audio Source Separation Toolbox Version 2.0*, ICASSP, Poster, Mai 2014. Adresse: https://hal.inria.fr/hal-00957412.

[14] K. Nakadai, H. Okuno, H. Nakajima, Y. Hasegawa und H. Tsujino, "An open source software system for robot audition hark and its evaluation", English, in *2008 8th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2008*, 2008 8th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2008 ; Conference date: 01-12-2008 Through 03-12-2008, 2008, S. 561–566, ISBN: 9781424428229. DOI: 10.1109/ICHR.2008.4756031.

[15] B. Schuller, A. Lehmann, F. Weninger, F. Eyben und G. Rigoll, "Blind enhancement of the rhythmic and harmonic sections by nmf: Does it help?", Jan. 2009, S. 361–364.

[16] M. Pariente, S. Cornell, J. Cosentino, S. Sivasankaran, E. Tzinis, J. Heitkaemper, M. Olvera, F.-R. Stöter, M. Hu, J. M. Martín-Doñas, D. Ditter, A. Frank, A. Deleforge und E. Vincent, "Asteroid: The PyTorch-based audio source separation toolkit for researchers", *arXiv preprint arXiv:2005.04132*, 2020.

[17] A. Leff und J. T. Rayfield, "Web-application development using the model/view/controller design pattern", in *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, 2001, S. 118–127.

[18] X. Anguera, S. Bozonnet, N. Evans, C. Fredouille, G. Friedland und O. Vinyals, "Speaker diarization: A review of recent research", *IEEE Transactions on Audio, Speech, and Language Processing*, Jg. 20, Nr. 2, S. 356–370, 2012.

[19] G. Krasner und S. Pope, "A cookbook for using the model-view-controller user-interface paradigm in smalltalk80", *Journal of Object-Oriented Programming*, Jg. SICS Publication, Nr. 26-49, S. 356–370, 1988.

[20] E. You. (2020). API — Vue.js, Adresse: `https://vuejs.org/` (besucht am 30. 06. 2020).

[21] ——, (2020). Vue.js - The Progressive JavaScript Framework, Adresse: `https://vuejs.org/v2/api/` (besucht am 30. 06. 2020).

[22] ——, (2020). List Rendering — Vue.js, Adresse: `https://vuejs.org/v2/guide/list` (besucht am 30. 06. 2020).

[23] ——, (2020). Computed Properties and Watchers — Vue.js, Adresse: `https://vuejs.org/v2/guide/computed` (besucht am 30. 06. 2020).

[24] Hassan Djirdeh, "An introduction to dynamic list rendering in Vue.js", *freeCodeCamp*, 1. Mai 2018. Adresse: `https://www.freecodecamp.org/news/an-introduction-to-dynamic-list-rendering-in-vue-js-a70eea3e321/` (besucht am 30. 06. 2020).

[25] Léna Faure, "Grasp "By Value" and "By Reference" in JavaScript", *Hackernoon*, 4. Juni 2017. Adresse: `https://hackernoon.com/grasp-by-value-and-by-reference-in-javascript-7ed75efa1293` (besucht am 30. 06. 2020).

[26] Joshua Clanton, "Object Equality in JavaScript", *A Drip of JavaScript*, 7. Apr. 2020. Adresse: `http://adripofjavascript.com/blog/drips/object-equality-in-javascript.html` (besucht am 30. 06. 2020).

[27] Chidume Nnamdi, "Understanding Memoization in JavaScript to Improve Performance", *Bits and Pieces*, 21. Nov. 2018. Adresse: `https://blog.bitsrc.io/understanding-memoization-in-javascript-to-improve-performance-2763ab107092` (besucht am 30. 06. 2020).

[28] A. Ronacher. (2020). Flask - web development one drop at a time, Adresse: `https://palletsprojects.com/p/flask/` (besucht am 10. 06. 2020).

[29] M. Bayer. (2020). SQLAlchemy - The Python SQL Toolkit and Object Relational Mapper, Adresse: `https://www.sqlalchemy.org/` (besucht am 15. 06. 2020).

[30] ——, (2020). Alembic, Adresse: https://alembic.sqlalchemy.org/en/latest/front.html#project-homepage (besucht am 15. 06. 2020).

[31] N. Omer, S. K. Jha und S. Kumar Khatri, "Maintaining reusable software components", in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, 2019, S. 1350–1352.

[32] ECMA. (2020). ECMAScript® 2021 Language Specification, Adresse: https://tc39.es/ecma262/ (besucht am 30. 06. 2020).

[33] Vuex. (2020). What is Vuex? — Vuex, Adresse: https://vuex.vuejs.org/ (besucht am 30. 06. 2020).

[34] Vue Material. (2020). Vue Material - Material Design for Vue.js, Adresse: https://vuematerial.io/ (besucht am 30. 06. 2020).

[35] J. Cosentino, M. Pariente, S. Cornell, A. Deleforge und E. Vincent, *Librimix: An open-source dataset for generalizable speech separation*, 2020. arXiv: 2005.11262 [eess.AS].

[36] V. Panayotov, G. Chen, D. Povey und S. Khudanpur, "Librispeech: An asr corpus based on public domain audio books", in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, S. 5206–5210.

[37] J. Hershey, Z. Chen, J. Le Roux und S. Watanabe, "Deep clustering: Discriminative embeddings for segmentation and separation", English, in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016 - Proceedings*, 41st IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016 ; Conference date: 20-03-2016 Through 25-03-2016, Bd. 2016-May, United States: Institute of Electrical und Electronics Engineers Inc., Mai 2016, S. 31–35. DOI: 10.1109/ICASSP.2016.7471631.

[38] PapersWithCode. (2020). wsj0-2mix Leaderboard — Papers With Code, Adresse: https://paperswithcode.com/sota/speech-separation-on-wsj0-2mix (besucht am 30. 06. 2020).

[39] Y. Luo und N. Mesgarani, "Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Jg. 27, Nr. 8, S. 1256–1266, Aug. 2019, ISSN: 2329-9304. DOI: 10.1109/taslp.2019.2915167. Adresse: http://dx.doi.org/10.1109/TASLP.2019.2915167.

[40] N. Zeghidour und D. Grangier, *Wavesplit: End-to-end speech separation by speaker clustering*, 2020. arXiv: 2002.08933 [eess.AS].

[41] Y. Luo, Z. Chen und T. Yoshioka, *Dual-path rnn: Efficient long sequence modeling for time-domain single-channel speech separation*, 2019. arXiv: 1910.06379 [eess.AS].

[42] Y. Liu und D. Wang, *Divide and conquer: A deep casa approach to talker-independent monaural speaker separation*, 2019. arXiv: 1904.11148 [cs.SD].

[43] G. Wichern, J. Antognini, M. Flynn, L. R. Zhu, E. McQuinn, D. Crow, E. Manilow und J. Le Roux, "Wham!: Extending speech separation to noisy environments", in *Proc. Interspeech*, Sep. 2019.

[44] M. Maciejewski, G. Wichern und J. Le Roux, "Whamr!: Noisy and reverberant single-channel speech separation", in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mai 2020.

[45] Addy Osmani. (2020). MVC for JavaScript Developers, Adresse: https://www.oreilly.com/library/view/learning-javascript-design/9781449334840/ch10s02.html (besucht am 30. 06. 2020).

[46] Material.io. (2020). Buttons: floating action button - Material Design, Adresse: https://material.io/components/buttons-floating-action-button (besucht am 30. 06. 2020).

[47] Y. Luo, Z. Chen und T. Yoshioka, "Dual-path rnn: Efficient long sequence modeling for time-domain single-channel speech separation", in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, S. 46–50.

[48] L. Drude und R. Haeb-Umbach, "Tight integration of spatial and spectral features for bss with deep clustering embeddings", in *Proc. Interspeech 2017*, 2017, S. 2650–2654. DOI: 10.21437/Interspeech.2017-187. Adresse: http://dx.doi.org/10.21437/Interspeech.2017-187.

[49] Department of Communications Engineering University of Paderborn - GitHub. (2020). fgnt/pb_bss: Collection of EM algorithms for blind source separation of audio signals, Adresse: https://github.com/fgnt/pb_bss (besucht am 30. 06. 2020).

[50] J. L. Roux, S. Wisdom, H. Erdogan und J. R. Hershey, "Sdr – half-baked or well done?", in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, S. 626–630.

[51] E. Vincent, R. Gribonval und C. Fevotte, “Performance measurement in blind audio source separation”, *IEEE Transactions on Audio, Speech, and Language Processing*, Jg. 14, Nr. 4, S. 1462–1469, 2006.

[52] C. H. Taal, R. C. Hendriks, R. Heusdens und J. Jensen, “An algorithm for intelligibility prediction of time–frequency weighted noisy speech”, *IEEE Transactions on Audio, Speech, and Language Processing*, Jg. 19, Nr. 7, S. 2125–2136, 2011.

[53] H. B. Curry, “The method of steepest descent for non-linear minimization problems”, *Quart. Appl. Math.*, Nr. 2, S. 258–261, 1944.

[54] Nvidia. (2020). Grafik neu erfunden: NVIDIA GeForce RTX 2080 Ti-Grafikkarte, Adresse: `https://www.nvidia.com/de-de/geforce/graphics-cards/rtx-2080-ti/` (besucht am 30. 06. 2020).

[55] N. Lotanna, “Using event bus in vue.js to pass data between components”, *LogRocket Blog*, 18. Sep. 2019. Adresse: `https://www.theguardian.com/world/2017/mar/12/netherlands-will-pay-the-price-for-blocking-turkish-visit-erdogan` (besucht am 30. 06. 2020).

[56] SortableJS - GitHub. (2020). SortableJS/Vue.Draggable: Vue drag-and-drop component based on Sortable.js, Adresse: `https://github.com/SortableJS/Vue.Draggable` (besucht am 30. 06. 2020).

[57] MDN contributors. (2020). Web Audio API - Web APIs — MDN, Adresse: `https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API` (besucht am 30. 06. 2020).

[58] Library of Congress Collections. (2012). WAVE Audio File Format, Adresse: `https://www.loc.gov/preservation/digital/formats/fdd/fdd000001.shtml` (besucht am 30. 06. 2020).

Anhang

School of Engineering

Automatische Transkription von Interviews [Speech Processing, Frontend]
BA20_ciel_05

BetreuerInnen: Mark Cieliebak, ciel
Fachgebiete: Software (SOW)

Studiengang: IT
Zuordnung: Institut für angewandte Informationstechnologie (InIT)

Gruppengrösse: 2

Kurzbeschreibung:
Interscriber ist ein innovatives KI-System, mit dem man Interviews automatisch transkribieren kann: Aus einer Audio-Aufnahme wird ein Text erzeugt, der zeigt wer wann was gesagt hat. Dafür werden automatische Systeme für Speech-to-Text und Speaker Diarization eingesetzt, z.B. Google API oder Deep Speech. Da die automatische Transkription normalerweise Fehler enthält, bietet Interscriber zusätzlich einen massgeschneiderten Editor, mit dem man die Texte nachbearbeiten und verbessern kann.

Aktuell existiert ein Minimum Viable Product (MVP) von Interscriber, mit dem man die wesentlichen Schritte einer Transkription auf Englisch und Deutsch durchführen kann. Das MVP wird zurzeit bei Pilot-Kunden eingesetzt und getestet.

Ziel der Arbeit

In dieser Arbeit soll Interscriber erweitert und optimiert werden. Dabei liegt ein Fokus auf User Experience und einer flüssigen Arbeitsweise im Editor, da dies entscheidend ist um effizient mit Interscriber arbeiten zu können. Darüberhinaus soll die Funktionalität erweitert werden, da viele User Stories bisher nur rudimentär oder gar nicht umgesetzt wurden. Dies sind z.B.:

* Integration von neuen Analyse-Methoden, z.B. für Erkennung von gleichzeitigen Sprechern

* Mobile App mit der man Audio-Files direkt vom Smartphone hochladen kann

* Payment-Modul für die Web-Applikation

* Erweiterung auf Italienisch und Französisch, um alle relevanten Sprachen der Schweiz abzudecken

Die zu entwickelnden Features können in Absprache mit dem Auftraggeber festgelegt werden.

Technologien:

* Frontend: Electron mit Vue.js

* Backend: REST in python mit flask, nginx und postgresql

* Speech-To-Text: Google API, IBM Watson, Mozilla

Falls Sie Interesse an diesem Thema haben, können wir gern einen Termin abmachen und die konkrete Aufgabenstellung besprechen. Email: ciel@zhaw.ch oder Telefon: 058 934 72 39

Die Arbeit ist vereinbart mit:
Yannik Roth (rothyan1)
Peter Unger (ungerpet)

Weiterführende Informationen:
https://www.dropbox.com/s/o73ijxjusyh2y3b/Screenshot%20Interscriber.jpg?dl=0