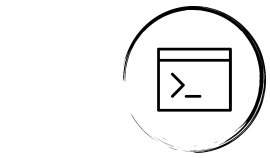

Prototype: `Speech2EPD`

Adriaan Kisjes (4279093)
Catherine de Weever (4255143)
Finn Hamers (5608279)
Floris Heijmans (5636981)
Iman Hashemi (5707617)
Lawko van der Weiden (5546540)
Mark Jan van Lieburg (5744520)
Mark Heinsbroek (5586771)
Mathijs Henquet (3995097)
Ruben Verboon (4227980)

Utrecht University
Department of Information and Computing Sciences

January 2019

Abstract

Care providers spend up to 40% of their time on administration. At the same time, speech-to-text technologies and other data processing techniques that could be combined to perform automated reporting, have seen large developments over the last couple of years. The client requested a proof-of-concept for a highly modular system that automatically generates reports from sessions between care providers and patients. This system had to be able to combine multiple types of input (audio, video and domotics) to form the reports.

The system that has been developed to meet this request will be described in detail within this report. The backend (server-side) of the system has a micro service architecture, which uses Docker, to ensure high modularity. The transcription of audio is performed by the Google Cloud Speech-to-Text service. This transcription, in combination with data received from audio and domotics, is used to act as input for the Micro Analysers, each of which runs in a separate Docker container. Data generated by the Micro Analysers is then used as input for a regression model that determines a "SOEP-label" for each sentence. Finally, a report is then created using these "SOEP-labels", after which it is sent to the client-side. The user will then view and optionally edit the report before approving it. Although a proof-of-concept is possible at this stage in time, multiple technologies this system relies on, such as speech-to-text conversion, still need to develop further for it to have any practical potential as a reliable report generator.

Contents

Part I

Result

Chapter 1

Product requirements and design problems

1.1 Product

`Care2Report` is a project which aims to reduce the time spend on administration by care providers. It aims to do this by automating the administrative process using captured audio, video and measurements by domotics of an interaction between a care provider and a patient. These different types of data are then processed and combined into a single report. The use of multimodal analysis in medical administration automation has not been done before. The goal of this project is to create a program, that demonstrates a basic multimodal analysis process and to design a generic architecture that can be used for follow-up projects.

This project will start the development of `Speech2EPD`, a service that automatically generates administrative documents based on the three modalities stated previously. The specific goal is to work in Dutch language medical environments and to be connected to the Dutch medical administrative database called EPD (Elektronisch Patiënten Dossier). To demonstrate how `Speech2EPD` will be used in practice, a detailed example of a general practitioner using it is described here:

Firstly, microphones and cameras are placed at strategic locations in the room where the consult will take place (e.g. the table where the main conversation takes place or the examination table). Where possible, medical measurement devices (e.g. blood pressure monitor, heartbeat monitor or thermometer) will be replaced with 'smart' equivalents that have connection capabilities. Every camera, microphone, and medical measurement device will be, preferably wirelessly, connected to a hub device (e.g. tablet or smartphone). This hub device will send the recorded data to a server that performs the analysis and generates a draft EPD submission. After the consult, the general practitioner will read this draft and perform edits where needed. The resulting EPD submission will then be uploaded to the specific EPD system the care provider uses. The final analysis architecture and hub device should be sufficiently generic so that it can be used in other medical domains such as in-home care or medical specialists in hospitals.

1.2 Requirements for the final product

The most important requirements for the final product of `Speech2EPD` are the following:

1. The system reduces the time the care provider has to spent on administration.
2. The system records clinically relevant actions that happen during a patient conversation.
3. The system analyses medical interactions and uses these to generate a report.
4. The care provider has to be able to verify the draft medical report, so the care provider can remove mistakes that where made in the generation.
5. The care provider should have minimal interactions with the system, so that the care provider can focus on the conversation with the patient.

Since this is the first project to work on this product and the main priority is a proof-of-concept and a future-proof analysis architecture, these requirements are primarily used to ensure that current choices do not inhibit these features in the future.

1.3 Requirements for the demo

There are two important requirements for the analysis architecture: The system has to be loosely coupled; components should be easily replaceable without major rewrites of the main code base. Of equal importance is the communication between the different analysis processes; results of one modality should be used to improve the analysis of other modalities. These two constraints are complicated to combine because communication between all analysis processes intuitively results in strong coupling. The dependencies of analysis results between analysers prove to be an additional problem. If text analysis needs an intermediary result from video analysis to perform its task, while videos analysis is waiting for a result of the text analysis step to generate the intermediary result, the analysis process can never complete. This is due to a dependency cycle between these processes.

The primary requirement for the demo analysis is to show that the architecture can handle multimodal analysis in an elegant way. Therefore, it is important that the demo analysis is multimodal or has some multimodal analysis steps; results of one analysis modality (e.g. text, video or measurement results) are used to enhance the analysis of another modality. Secondly, the analysis processes should collectively form a, to a certain extend, complete pipeline. It should use raw input data (video, audio and medical measurements) to produce some kind of usable summary. Finally, the analysis processes should, as much as possible, be a useful part of the final product so that the next team can simply extend the current analysis processes instead of completely replacing some or all of them.

Chapter 2

Description of product

2.1 Global description

The complete system is one that takes multiple forms of raw data as input (from audio, video and domotics) and uploads an EPD submission as its final output. Recorded data is sent from the client-side of the system to the server-side of the system. The server-side processes this data and returns a provisional EPD submission – which adheres to the SOEP-method – to the client-side. Now, the user has the option to view and edit the submission, before finally uploading it to the preferred EPD server (which is a stub for now). Both the client- and server-side of the system will be explained in detail within the next sections.

2.2 Frontend

The frontend consists of the Graphical User Interface (GUI) and the logic, ranging from button control to server communication, that runs behind the scenes on the device running the application.

2.2.1 Description of the GUI

In this section, the different pages of the GUI will be described. Screenshots of the application can be found in Appendix A. The GUI is split into three pages: the start page, the report page and the settings page.

Home page

The home page (see Figure A.1) is divided into two columns. The first column displays options with which the care provider can configure the recording. The second column shows the recording controls.

The top section of the recording options consists of a session selection drop-down and a new session button.

The bottom section consists of five options. The first option is the name of the client, in this drop-down list a client can be selected. The second and third options show a list of connected audio recording devices. The system default device is selected in the first of the two drop-down lists. At least one device has to be selected to make a recording. The fourth option shows a list of the connected video recording devices. If one is selected, video will be recorded in addition to the audio recording(s). The last option is disabled for now, but could be enabled in the future when Bluetooth connections are supported.

The recording controls consist of two buttons, to start and stop recording, and a label, displaying the duration of the current recording. Only one of the two buttons is enabled at a time, depending on the state of the program. Recording is only possible when a session has been selected.

Report page

The report page (see Figures A.2 and A.3) is also divided into two columns. A drop-down list with existing sessions can be found in the top right corner of this page. Reports belonging to the selected session will be shown.

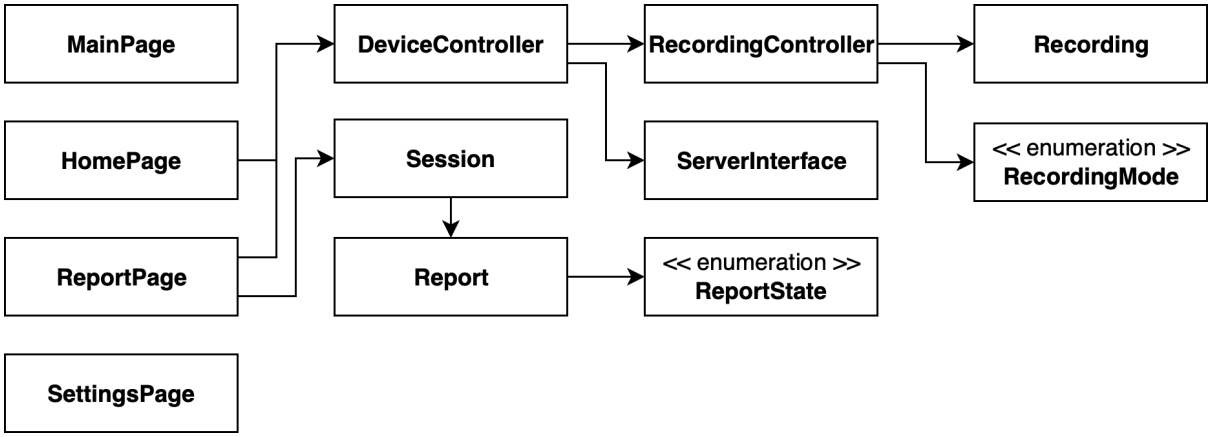

Figure 2.1: Class Diagram of the Frontend

The left column shows a list of reports belonging to the selected session. A report can be selected, after which the right column will display the contents of the chosen report. This includes the name of the report, the name of the client, the initial ailment or reason for the client to visit the care provider, the audio and optional video file(s) belonging to the report, and – after generation – the generated, editable summary. This right column also has two buttons: one to save the report and one to generate a summary for the report.

Settings page

At the moment, the settings page has two settings. The first is the name of the care provider, which is added to every report. The second is the colour scheme. The application can be set to either dark or light mode.

2.2.2 The logic behind the GUI

The GUI is made up of four classes, each of which inherit the Page class: the MainPage, the HomePage, the ReportPage and the SettingsPage. The MainPage divides the application into two columns: a navigation bar on the left of the screen and the content of one of the other pages in a Frame on the right. These Page subclasses have fields with custom class values. The important custom classes are the DeviceController, RecordingController, Recording, ServerInterface, Session and Report. A class diagram can be seen in Figure 2.1.

Device controller

When one of the controls on the home or report page is used, and the application needs to do more than just update the GUI, one of the methods of the device controller is invoked. The device controller saves reports to and reads them from a file. It also creates new reports and invokes methods from the server interface when communication with the server is required to generate a report. Using a recording controller, it also initialises, starts and stops both recording and interaction.

Recording controller

The recording controller has a recording mode, which can be Initialising, Recording or Stopped. It contains all the necessary fields to create and save three recordings (two audio recordings and one video recording), as well as the setup, start, stop and save functions for all of the recordings. Which recordings will be created is based on the user input on the home page while recording.

Recording

A recording has a media encoding format, based on the given audio or video encoding format. In case of an audio encoding format, the media encoding format audio gets a channel count of one (1) to make the recording compatible with the Google Cloud Speech-to-Text service.

Server interface

The server interface has a frontend endpoint client with a gRPC channel (for more information about gRPC, see Section 2.4). The functions of the server interface use the channel to communicate with the server and are invoked by the recording controller.

Session

A session has lists of reports, audio files, video files and Bluetooth data files that belong to that session. When a recording is stopped, the files that are saved are added to the corresponding list. When a report is added to the session, the files that can be added to the report are the ones that are part of the session.

Report

A report has a GUID that it gets from the server, the server ID. The server ID is assigned by the server to connect files to a report. It also contains a report name, status, client name, initial ailment, summary, array of file names, array of transcriptions and a list of words that have medical alternatives.

The array of file names can be filled with the following indexes: 0 - audio doctor (or only audio file), 1 - audio client (optional), 2 - video (optional), 3 - Bluetooth data (not yet implemented). The array of transcriptions can be filled with the transcriptions of the two audio files.

The status of a report can be one of three options: empty, filled or generated.

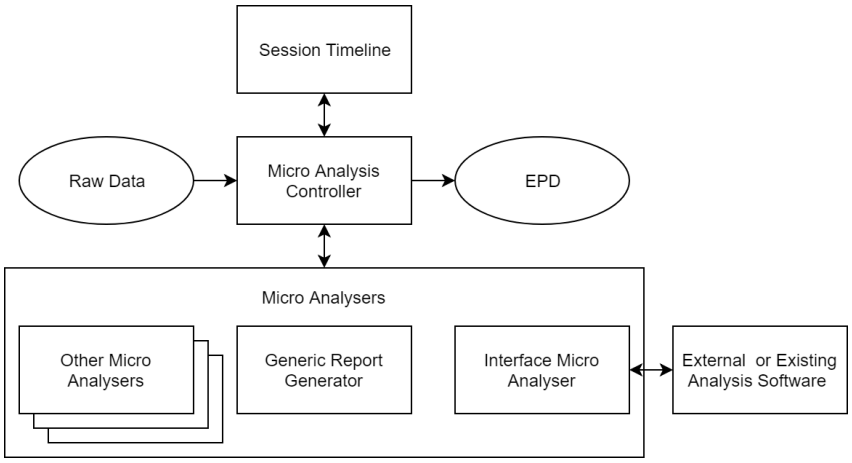

Figure 2.2: Micro Analyser Architecture

2.3 Micro Analyser Architecture

To solve the interdependency complications between analysers, while maintaining a loosely coupled system, the large single-modal analysers (e.g. video analyser, text analyser, and domotics analyser) were split into smaller Micro Analysers. Each Micro Analyser has a predefined input set and predefined output set. After it receives all the defined input data it performs the analysis and on completion, outputs its results. A Micro Analyser is not allowed to require intermediary input data or produce intermediary output data. These constraints ensure a more structured analysis process and allow for an automated approach to detect dependency cycles and to optimise the analysis.

The flow of information between the Micro Analysers is represented by a graph where the nodes are the Micro Analysers and the edges are the data dependencies. The Micro Analyser Controller uses a combination of existing graph algorithms to detect if there are cycles, if there are enough analysers to complete the analysis, and which analysers do not need to be executed to achieve the analysis results.

2.3.1 Micro Analysers

In Figure 2.3 ten examples of Micro Analysers are shown with descriptive name and in- and output types (note that despite similarities these are not the Micro Analysers programmed for the analysis demo). In Figure 2.4 the dependency graph of these ten analysers is shown. These two images perfectly illustrate the benefits of the Micro Analyser Architecture. There is no programmatically distinguishable difference

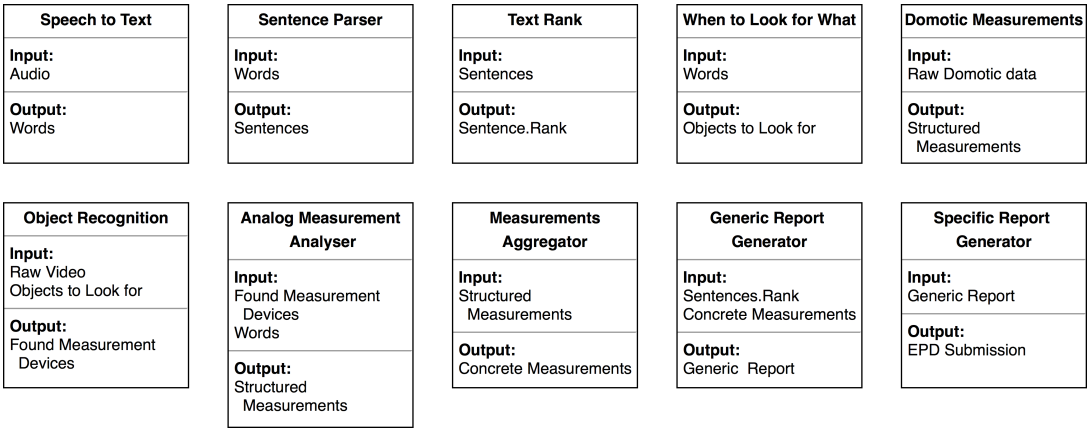

Figure 2.3: Examples of Micro Analysers

between single modal and multi-modal analysers. It is deliberately simple to replace an analyser without changing the rest of the system, with the only constraint being that in- and output have to remain the same. Lastly, it is visually and programmatically trivial to detect cycles and incomplete dependencies, and thus easy to check if the analysis pipeline is executable.

2.3.2 Micro Analyser Controller

The class that ensures that all constraints are satisfied and that controls the analysis process is the Micro Analyser Controller or MAC. The MAC is largely independent from the global server controller or communication layer, it is therefore easy to implement the Micro Analyser Architecture on a different server architecture. The Micro Analysers communicate with the MAC via an API, this API provides three main functionalities: registering which Micro Analysers are available, requesting input data and submitting output data. When the analysis begins, the Micro Analysers first register themselves by announcing what input and output they need or provide. With this information the MAC generates a graph of all analysers and checks if there no cycles exist, removes redundant analysers (analysers whose input is unavailable or whose output is not used) and finally checks if the analysis process can be completed by checking if there is a complete chain of Micro Analysers from the global analysis input and output. If all tests pass, the analysis process can begin. The MAC first checks which Micro Analysers can start with only the initial measurements as input. These analysers are then given a signal that they can start execution. When the first Micro Analysers are done, the MAC checks which Micro Analysers are ready to execute next. This process continues until all analysers are executed and the final analysis result is produced.

The dependency graph that is used in the final version of the system can be viewed in Appendix B.

2.3.3 Session timeline

The final part of the Micro Analyser Architecture is the database and data structure where the input data and intermediary results are saved for later reference. Since input data and intermediary results can be all possible data types, the database should allow all data types. To provide some structure to the database, all saved objects are given three extra properties: time range, semantic data type descriptor and a unique key. In Appendix C an example of how objects are sorted on time range and data type is illustrated. These objects form the Session Timeline or STL.

2.4 Communication layer

A technical requirement for the system was a loose coupling between the various analysers (as described in section 2.3.2) and the Micro Analyser Controller and other parts of the core system. Apart from a good principle in general, the proximal cause was that some desired analysers are more comfortably written in python, an industry standard for language processing related tasks. In addition, there was a desire to maintain the possibility of running these analysers on different processes or even different machines, as some can get potentially resource intensive.

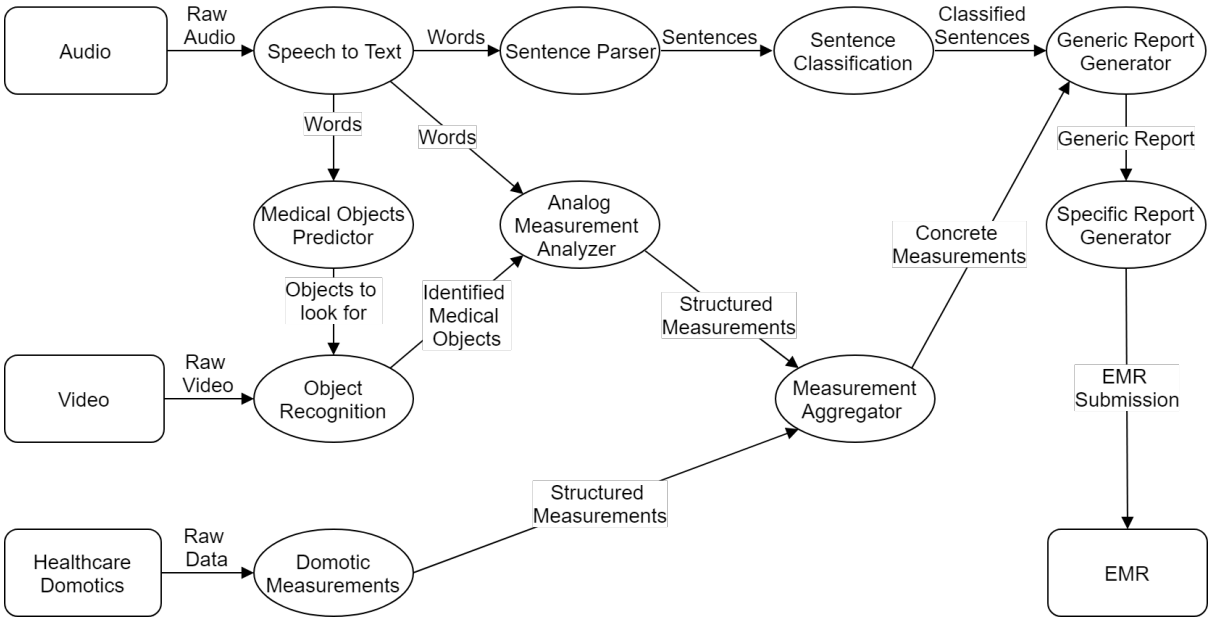

Figure 2.4: Dependency graph example; rounded rectangles represent global input and output, ellipses represent Micro Analysers.

2.4.1 Technologies

A good means of communication between Micro Analysers and the Micro Analyser Controller was needed. This problem is usually called inter process communication or RPC (which stands for Remote Procedure Call). The decision was made to use an off-the-shelf solution with a proven track record and bindings for many different languages. This solution was gRPC, a collection of tools used and developed by Google.

The gRPC project is closely related to, and built upon, another Google project named protocol buffers. Protocol buffers (also protobuf) is a language-neutral, platform-neutral, extensible mechanism for serialising structured data. Such a structured data unit is called a message and can be compared to a struct in C#. These messages are defined in the '.proto' files written in the proto definition language which serves as an interface description language (IDL). By itself, protobuf allows for production from these messages definitions: bindings for data structures in a variety of languages including C# and python; together with tools for encoding an decoding them from a raw binary representation. The proto definition language also allows the specification of services which can be compared to C# interfaces. By itself, protobuf does not compile these service definitions, which is where the gRPC protobuf plugin comes in. This plugin produces various language bindings for these interfaces.

gRPC is then the concrete transport protocol for these services. Under the hood, gRPC uses http2 requests to send protocol buffers from one process to another. Apart from a simple request-response flow, a gRPC call can also send or receive a stream of messages.

2.4.2 Implementation

A project named Protocol exists in the system, which is depended upon by most other projects. In this project the message types and services are specified using proto definition files stored in the 'Spec/' folder. For example, the timeline events are specified by a protobuf definition as shown in Listing 2.1. Apart from message definitions, the abstract services implemented in the system are also defined. For example, analysers talk to a service called the analysis endpoint as displayed in Listing 2.2. From these protobuf definition files python and C# bindings are generated by bash or bat scripts.

The generated C# code can be described as follows. Each message declaration gets a class implementing get and set methods for the defined fields. These classes are 'partial' so there is a possibility of extending their behaviour, which is done in some cases. Two classes are generated for each service: an abstract '*Base' class, which can be inherited to implement a server of this service; and a '*Client' class

[1]For more information on the protobuf file format, see `https://developers.google.com/protocol-buffers/docs/proto3`

```
message TimelineEvent {
    repeated Interval range = 1;
    string id = 2;
    map<string, string> metadata = 3;
    oneof payload {
        string audio_file = 16;
        Sentence sentence = 17;
        string word = 18;
        string summary = 19;
        /* ... */
    }
}
```

Listing 2.1: Protobuf message example

```
service AnalysisEndpoint {
    rpc Register(Manifest)
        returns(stream RegisterResponse);
    rpc GetTimelineEvents(GetTimelineEventsRequest)
        returns(GetTimelineEventsResponse);
    rpc SetTimelineEvents(SetTimelineEventsRequest)
        returns(Response);
    rpc SetMetadata(SetMetadataRequest)
        returns(Response);
    rpc InstanceDone(InstanceDoneRequest)
        returns(Response);
}
```

Listing 2.2: Protobuf service example

Table 2.1: Example definitions for messages and services in the protobuf 3 fileformat[1] as used in `Speech2EPD`

which can be used to talk to such a service. The system contains two services, which are described in the following subsections.

2.4.3 Analysis endpoint

The analysis endpoint (see Listing 2.2) is the subsystem of the server controller the analysers communicate with. The server controller maintains one instance of this endpoint and exposes it as a gRPC server on the local network where the analysers are also present on. Since it is assumed that this server runs on a local network, there are no strong security requirements.

Each analyser connects to the analysis endpoint (i.e. creates a analysis endpoint client) on startup and calls the register method. This method informs the server controller what kind of analyser instances are served by this program (see section 2.3.1). The response to the register method is a long running stream of register responses. Whenever the the server controller wants to signal to an analyser that it should execute analysis, it sends such a register response. This register response contains the instance which is to be run and a unique identifier session id for this analysis request. Thanks to this session id, the analyser can call the other analysis endpoint methods, supplying the session id each time.

2.4.4 Frontend endpoint

The frontend endpoint is the subsystem of the server controller that clients connect to. The server controller exposes one instance of this endpoint. Since this endpoint is exposed to the greater internet, there are stricter security requirements.

When a client connects to the frontend endpoint it can request the creation of a new report, upload raw data to the report (such as audio and video), request report generation and finally remove the report again.

From a technical standpoint, how file uploads are handled if of principle interest. The reason that file uploads are nontrivial is that gRPC has an upper bound in the order of a few megabytes on the size of message that can be sent. To accommodate large uploads, data transfer has to be chunked into smaller messages. A file is abstracted as a blob which represents a bunch of bytes only identified by an hash of those bytes. To then upload a file, the frontend indicates the role of the data (e.g. 'audio-recording') and then specifies the hash of the blob together with the total size. After receiving an 'ok' from the server – which reserves a file to receive the bytes on – the client can then send chunks of the blob to the server. A chunk identifies the hash of the blob it belongs to and the offset (in bytes) of the data sent to the server. After receiving all the chunks, the server verifies the received data using the hash it received.

2.5 Server controller

The server controller is the master controller of the Speech2EPD project and is responsible for orchestrating the backend and frontend and manages the lifecycle of reports. While doing this, it has to serve as an intermediary between the Micro Analyser Controller or MAC (see Section 2.3), the analysers and

frontend clients. The analysers and frontend clients connect to the server controller using analysis endpoints (see Section 2.4.3) and frontend endpoint (see Section 2.4.4) respectively. In contrast, MACs are created and managed by the server controller.

2.5.1 The lifetime of a report

It is assumed that the client (frontend) already has all necessary files, such as audio recordings, video recordings and readouts from domotics, present. To create a report, a client first connects to the Frontend endpoint on a public facing server managed by the server controller. More information on this connection can be obtained in section 2.4.4. The client then signals that it wants to create a new report, to which the server responds by generating a fresh report id. This report id is included in all subsequent requests. The client then uploads all files as described in section 2.4.4, specifying the role of each file (e.g 'audio recording') these roles used in the analysis path configured in the server controller.

The analysis path encodes a list of input and output timeline events of the micro analysis process (see section 2.3) and a way to generate the initial timeline events (which can refer to the uploaded file blob roles). In this case, you might imagine an analysis path starting from 'audio-recording' and resulting in 'summary' where the timeline is populated by the location of the uploaded file blobs.

After all files are uploaded, the client signals that it wants the report to be generated. This triggers the creation of a new MAC and initialises its timeline with the initial timeline events as specified in the analysis path. The server controller then mediates communication between the connected analysers and the created MAC. For this it demultiplexes the requests it gets from the analysers to end up at the proper MAC. After the MAC is done, the server controller extracts all output events and returns them to the client. Finally, the MAC is discarded.

2.5.2 Managing the analysers

Implicit in the above description is the connection between the server controller and the analysers. Indeed, to ensure analysis can happen, the server controller has to ensure that enough analysers are available. Therefore, the server controller has a list of analysers that need to be connected before analysis can start. When analysers connect to the server controller, a mapping of all connected instances – together with a handler representing the analyser – is maintained. Whenever this list matches the required analyser list, the 'analysers connected' signal is triggered and analysis can be performed. This mapping also allows the server controller to start analysis of a specific analyser, on behalf of the MAC.

2.6 Classification

2.6.1 SOEP

A medical report consists of multiple sections. These sections contain information about the medical session and have to adhere to a certain protocol. The medical reporting protocol differs for each medical discipline. Because of this, it is impossible to create a single medical report that is suited for all medical disciplines. This system focuses on generating a report for general practitioners, but methods are designed generically to ensure extensibility towards other disciplines.

Medical practitioners in the Netherlands use the SOEP (Subjective Objective Evaluation Plan) protocol for medical reporting. SOEP assessments have to be made from recorded data of a medical session. This data can be either text, video or other forms of data originating from domotics devices.

2.6.2 Multinomial logistic regression

A key stage in this process is the classification of sentences. This classification is done by means of Multinomial Logistic Regression (MLR). MLR is a classification method that calculates the probability of an observed data point being of class x, where there can be multiple possible classes. Then, the class with the highest probability is chosen as the final class label for that dependent variable. The probability is calculated by a linear predictor function that uses features of the data. These features represent the independent variables.

The following linear predictor function describes the probability that an observation i has outcome class i, given the independent variables $x_{M,i}$ and their corresponding coefficients.

$$f(k,i) = \beta_{0,k} + \beta_{1,k}x_{1,i} + ... + \beta_{M,k}x_{M,i}$$

To solve this classification problem, one linear predictor function is made for each possible class. The linear predictor function uses features/independent variables and corresponding weights/coefficients for these features. The features are calculated using the data that has to be classified. Training data is used to calculate the weights of the features. The training data consists of medical sessions, formatted as sentences and preprocessed by the text analysis pipeline. It also contains other forms of data, such as video and domotics data. Each sentence is manually given a class label. The possible class labels are: Subjective, Objective, Evaluation, Plan and None.

Now, features are added. A feature is a value that represent a property of the data, of which it is presumed it has a significant influence on the class label of the data. Using the Multinomial Logistic Regression model, the weight/coefficient and the significance of each feature are calculated based on the training data and ground truth, the class labels that are manually given to the data. If a feature is insignificant, it is discarded in the linear predictor function.

After these coefficients are calculated and a collection of significant features is created, the linear predictor functions are used to calculate the class probabilities of test data.

2.6.3 Data

Data has been collected with the goal of training and then testing the resulting model. The data was recorded with people with some expertise in SOEP reporting.

This system has 18 recorded sessions to its disposal. These recordings are partitioned randomly to make two sets, a training and a test set. Most of the recordings are used for the training set, because the training stage is heavily dependent on the amount of recordings.

The quality of the data is questionable. This is due to the fact that no actual medical sessions of general practitioners could be used. Another potential problem, is that the sample size is somewhat lacking in size. The data has been recorded mostly by medical students and doctors, who have some knowledge about the structure of these medical sessions, but this knowledge remains incomplete. This is due to the fact that no actual general practitioners were solicited while creating the data. Despite this, a large part of the knowledge of the structure of medical sessions for general practitioners is present in the recordings. This knowledge translates to the MLR model that is created. This model is used to classify medical sessions.

2.7 Text analysis

After the transcription has been created, the result ends up in the text analysis pipeline. This pipeline consists of different analysers. Their task is to produce meta data from the words and sentences, to either improve the results of the features for classification or improve the analysis for other analysers. All of the analysers in the pipeline are written in C# or Python. The analysis processes that are part of the text analysis pipeline described in the following subsections.

2.7.1 TextRank algorithm

One of the methods used to create a summary is the TextRank algorithm. This algorithm calculates the similarity between two sentences, to determine which sentences are the most important. Similarity is calculated by checking how many words are used in both the sentences. The calculation could deviate from the actual similarity between sentences if stop words are not taken into account. With regards to the scope of this project, stop words are the most used words in the Dutch language (such as ‘de‘/‘the‘ or ‘en‘/‘and‘). Stop words are ignored when calculating the similarity between sentences.

2.7.2 Gazetteer

The gazetteer analyser is written in C#. This analyser does not use any special libraries, but instead it uses a database. The input of the gazetteer is a list of words of each sentence. The gazetteer first checks if the word next to a numerical word is a unit, by querying a database with said word. Then, it computes what measurement the unit represents by looking into a database, which can be one of the

following: height, temperature, weight, heart rate, distance, duration and volume. Lastly, the associated values are written to the corresponding fields.

2.7.3 Part-of-Speech tagger

This analyser was built using the python library called Frog. Words are used as input for this analyser. This analyser assigns parts of speech to each word. The parts of speech are the following: adjective, adverb, punctuation, determiner, noun, names and unknown, interjection, numerator, conjunction, pronoun, preposition, and verb. It also computes the confidence score between 0 and 1 in how certain it was about its tag label. After the parts and confidence score are computed, the associated value is written to the corresponding field, by using the event id and field name.

2.7.4 Chunker

This analyser is also built using the python library called Frog. A list of words from each sentence is used as input for this analyser. This analyser applies shallow parsing on the sentence and classifies the chunks with their grammatical function. There are three grammatical functions: adverbial phrase, verb phrase and noun phrase. After the functions are computed, the associated value is written to the corresponding field by using the event id and field name.

2.7.5 Lemmatizer

This analyser is built using the python library called Frog as well. Words are used as input for this analyser. This analyser assigns the canonical form of a word to each word. It also uses the results of the POS-tagger analyser to compute the canonical form of a word. After these forms are computed, the associated value is written to the corresponding field by using the event id and field name.

2.7.6 Speech Act

Speech Act represents the intention of a sentence. This analyser determines whether a sentence is a question, expressive or an assertion. The analyser receives sentences and POS-tags of the words in these sentences as input. The speech act of each sentence is then determined using a random forest classification model. This model uses five features to determine the Speech Act: sentence length, number of nouns, sentence ends with a noun or adjective, sentence begins with a verb and, the amount of W words in the sentence (“wie”, “wat”, “waar” etc.). This model was created in R using training data provided in Squid’s Blog [GigaSquid Software, nd]. The model is originally trained on English sentences, but also has about 70% accuracy when classifying Dutch sentences.

2.7.7 Alternative medical terms

Another feature that was implemented is based on finding alternatives for medical terms, since healthcare is the domain of the text analysis in this project. In a conversation, words relating to a medical term may occur, without using the correct official medical term. Using Snomed CT [Snomed, nd], every noun that is part of the output summary receives metadata with medical terms added to the Timeline Event. In the future, these medical terms could be shown in the GUI for the care provider to choose from.

Chapter 3

Architecture and design

3.1 Functional architecture

The functional architecture describes the different modules and the functional flow of the system. Not everything within this architecture has currently been implemented in the system. The architecture shows how other modules can functionally be added to the system.

The system aims to provide a prototype with which we can record audio, video and care domotics devices in order to automatically generate a report to be submitted into an EPD. The user will interact with a local device, both to collect data and to approve the draft report generated by the system. The analysis and the report generation will be executed on the server. This system goes through five stages:

1. **Retrieving data**
   In this stage, all raw data will be retrieved and collected. This process starts at the local device. The user will make known to the application what client it is dealing with. The user hits a record button, which commences the process of recording the session and collecting data. This data consists of:
   - Audio data
   - Video data
   - Care domotics data
2. **Analysing data**
   During the analysis stage, the raw data will be analysed by the Micro Analysers as described previously. This analysis will generate some values for each sentence, that can be used for generating a report.
3. **Generating report**
   In this stage, the draft report will be generated using the analysed data. The report generator can use analysed data, care provider knowledge and medical guidelines to decide on what data should be included in the report. Furthermore, the report generator can use these to structure the report.
4. **Verifying report**
   In this stage, the user will verify the report and approve it. After the report generator has created the draft report, the server will send the draft report back to the local device. The user can read and edit this report freely. Further, the user can add extra documents that need to be included in the EPD report. After the user has edited the report, the user can approve the report. When the report has been approved, it will be sent back to the server.
5. **Sending EPD report**
   During this final stage, the server will connect with an existing EPD system to send the approved report. This is the final stage of the system. After this stage, the server will wipe all the raw data and analysed data.

The following components are not yet implemented in the current system:

- Care provider knowledge

- Medical guidelines knowledge
- EPD connection

3.1.1 RE4SA

To create the functional architecture, a model called RE4SA (Requirements Engineering for Software Architecture) was used. This model lets one describe a system using jobs, epics and user stories. These descriptors are in respective order of abstraction, seeing as a job consists of epics and an epic consists of user stories. The epics and user stories roughly map one-on-one to modules and features. The jobs, epics and user stories that describe the system are to be found in Appendix.

3.2 System architecture

The system architecture has been divided into two parts: the local device and the server. All components in the architecture have been described in detail in section 2, but here follows a short description of the design of the architecture on the next page.

3.2.1 Local device

The local device consists of the application and the the sensors that are connected with it. The application includes the following modules: *Device Controller*, *Local device interface* and *Device UI*. The sensors include the microphone, the camera and the care domotics. These sensors collect data and send it to the *Device Controller*. The *Device Controller* is connected to all other modules of the local device. It is connected with the local device interface, which can send and receive data to and from the server. It is also connected with the UI, so it can control the data that the UI receives from the system and control the data the user provides the system through the UI.

3.2.2 Server

The server consists of a *Server Controller*, *Server Interface*, *EPD interface*, and one or multiple *Micro Analysis Controllers*. The server controller handles the communication between the different parts of the server. The server interface handles the communication with the local devices. The EPD interface handles communication with the specific EPD service . The MAC handles the analysis process by controlling the different Micro Analysers. The *Session Timeline* is used to store data during the analysis process.

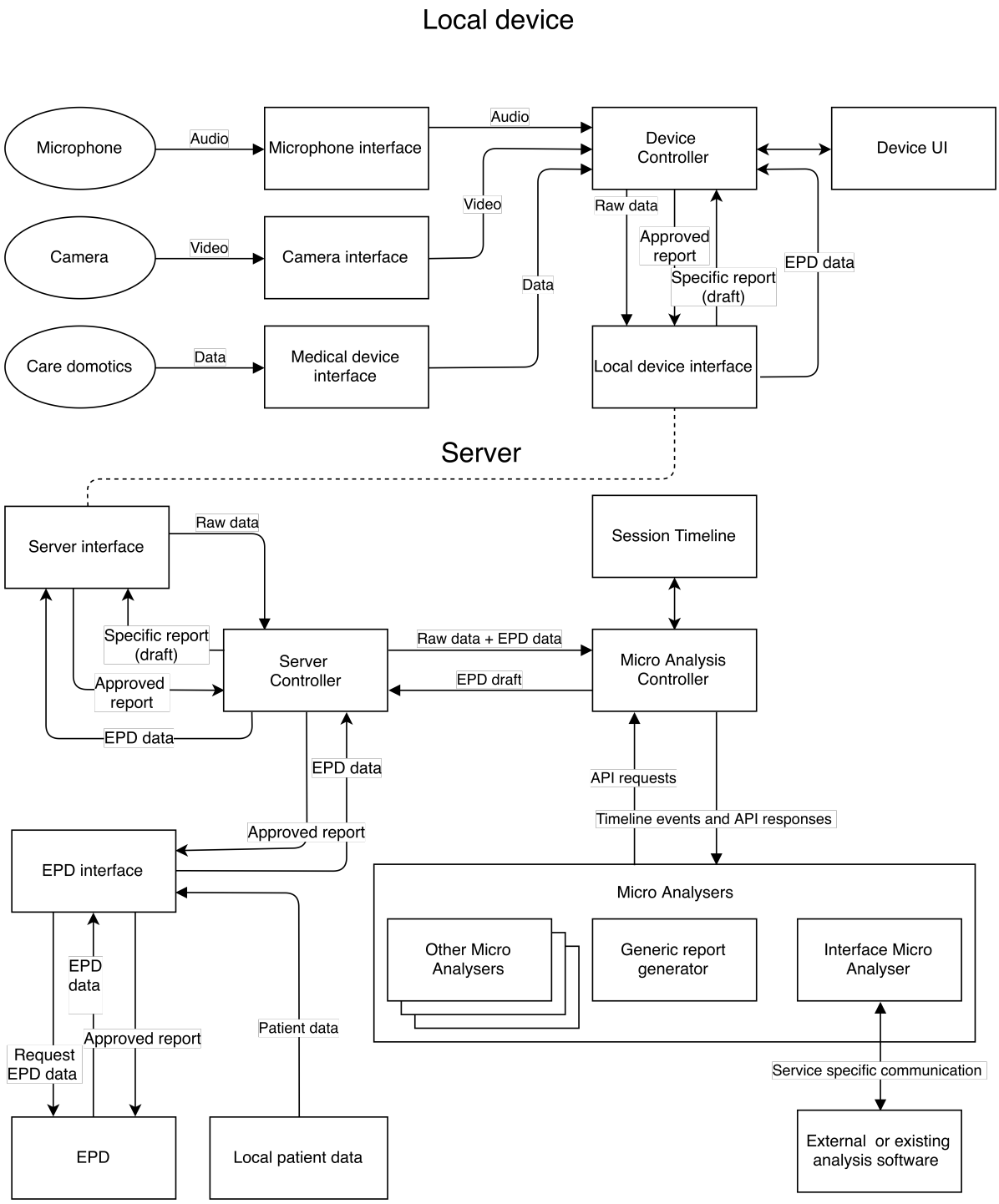

Figure 3.1: System Architecture; The EPD interface has not yet been implemented.

Part II

Initial plan

Chapter 4

Specification and planning

4.1 Introduction

With the current rise of speech to text technologies a significant amount of time could be saved. These technologies can automatically transcribe conversations that previously would have had to be transcribed manually. For care providers (CPs) this could mean spending less time on administration, leaving more time for more important tasks. Using computers to analyse text also opens up a lot of other possibilities that can reduce the time a CP needs for his or her administration. The goal of this project is to reduce the time a CP spends on administration by automatically generating an electronic patient dossier (EPD) submission from a conversation with a patient. This goal will be achieved by both developing new analysis techniques and by using existing technologies.

Speech-to-text algorithms are not yet perfect. Some of the mistakes they make can be found and corrected but some can not. Since accuracy is very important for CPs this can cause a great deal of problems until better algorithms are developed. Another problem is that patient data is very confidential. Data recorded by speech to text algorithms is no exception. New laws have been and are being created to make sure that patient data is only accessible by the right people. It is important to think about these laws while working on this project. Breaking any of these laws will make the project unusable and it could easily result in being held liable for leaked data.

The client intends on this project being a prototype. Further improvements will be made on it in the future. This means that the aim is to build a proof concept and not a product for the end user. The result of this is that less time will be spent on the issues with speech to text while more time will be spent on generating actual EPD submissions.

4.2 Product vision

For care providers **that** have one-on-one patient interaction: **The** `Speech2EPD` **is an** automated medical reporting tool **that** automatically generates medical reports using non intrusive sensors. **Unlike** the care provider manually filling in the report, **our product** does this for them, saving time.

4.2.1 Plan and justification

The explanation behind the prioritisation choices that have been made and the outline of the general plan for implementing the system.

1. The three user stories: 'Recording of patient interaction in office', 'Transcribing of patient interaction' and 'Patient interaction summarisation' together form the minimal viable product. The initial focus of the project will be on these three items.

2. The second priority is streamlining and improving these core user stories as exemplified by the stories 'Detection of medical terms', 'Correcting patient interaction transcription', 'Relistening recording based on transcription', 'Review interaction summary' and 'Submitting summaries to an EPD'. These user stories are aimed at refining the minimal viable product into something care providers would be glad to use.

3. The product also need to be usable in a portable format: 'Patient interaction on the go'. A large amount of the infrastructure developed in items one and two will be developed with this step in mind so the main challenge will be in finding the right recording, transmitting and processing set-up. The priority will still be creating a single viable product and this item is secondary. After a viable product has been created this item will guarantee that the product can be used in other situations.

4. Next will be the incorporation of external medical devices: domotics. Implementing this will be reasonably difficult because the information about the standards these devices use is limited. The focus will be on interfacing one medical device with a widely used standard with our system.

5. The incorporation of video analysis is also important because it is an integral part of the multimodular system that is proposed by the `Speech2EPD`. However, implementing sophisticated computer vision is a relatively complicated task. The goal is to incorporate basic computer vision, showing the viability and utility of the module in the system.

6. The next item is interfacing with an existing EPD. This is relatively important, because it demonstrates the utility of the concept.

7. The final item concerns mood analysis. Multiple papers have been written on mood analysis both through tone of voice and through written word. It is possible to make assumptions about the mood a person is in by listening to their tone and speed at which they talk [Dasgupta, 2017]. The amount of emotions that can be predicted is however small. The conclusions would be limited to predicting if someone is happy, sad, angry or scared. To make a computer conclude any other emotions from human speech a learning algorithm would most likely be needed [Shami and Verhelst, 2007].

4.3 User stories

Recording of patient interaction in office

User story	As a care provider I can unobtrusively record patient interaction in my office.
Priority	High / 'Must have'
Description	A way to faithfully capture high quality conversation between care providers and patients needs to be developed. This will require the use of hardware that is portable, cost efficient and quality sufficient.
Completion criterion	A conversation between a patient and care provider can easily and accurately be recorded in a fixed (office) setting.

Recording of patient interaction on the go

User story	As a care provider I can obtrusively record patient interaction on the go.
Priority	Medium / 'Should have'
Description	In addition to the aforementioned hardware requirements the device also needs to be portable.
Completion criterion	A conversation between a patient and care provider can easily and accurately be recorded in a portable setting (on the go).

Transcribing of patient interaction

User story	As a care provider I can view the transcription of a patient interaction.
Priority	High / 'Must have'
Description	The interaction between the patient and the care provider needs to be transcribed. For this research needs to be done into what kind of speech to text technology is available for the Dutch language. It is also important to recognise who said what for later analysis.
Completion criterion	A recorded conversation between a patient and care provider needs to be accurately transcribed with annotations describing which person said what.

Correcting patient transcription

User story	As a care provider I can correct the transcription of a patient interaction.
Priority	Medium / 'Should have'
Description	The care provider should be able to correct possible mistakes in the automatically transcribed text.
Completion criterion	A care provider should be able to review the transcription and correct parts of it.

Re-listening to a recording based on transcription

User story	As a care provider I can correct the transcription of a patient interaction by listening to a time matched recording.
Priority	Medium / 'Should have'
Description	A care provider should be able to re-listen to parts of the recording using the transcription. When the provider detects a mistake they can easily find the location of the mistake and correct it.
Completion criterion	A care provider should be able to listen to parts of the recording, matched to locations in the transcription.

Patient interaction summarisation

User story	As a care provider I can obtain a summarised patient interaction.
Priority	High / 'Must have'
Description	The transcription of a medical interaction needs to be analysed using natural language processing. For each treatise that we wish to support there should be a way to recognise this treatise and report the relevant information.
Completion criterion	Relevant treatises need to be extractable and reportable from a transcript.

Detection of medical terms

User story	As a care provider usage of medical terms and jargon should be highlighted in the transcript.
Priority	Medium / 'Should have'
Description	We can possibly use systems like farmacotherapeutischkompas.nl, thuisarts.nl and NHG richtlijnen to detect and contextualise the usage of medical terms.
Completion criterion	The system needs to be able to detect all relevant medical terms.

Review interaction summary

User story	As a care provider I can review and correct the summarised patient interaction.
Priority	High / 'Must have'
Description	The above analysis should also be used to highlight possibly important features (i.e. keywords and codewords) of the transcription so that care providers can easily scan the transcript for important features. Furthermore each item in the summary needs to be linked to a piece of text in the transcription so that a care provider can review the summary for correctness.
Completion criterion	A care provider should be able to see which parts of the summary are derived from which parts of the transcription and should see which parts of the transcription contain certain keywords related to the treatises.

Obtain mood analysis

User story	As a care provider I can view a patient's mood from the recorded interaction.
Priority	Low / 'Could have'
Description	Given a transcript/recording, mood analysis should be performed.
Completion criterion	An EPD submission includes an accurate assumption of what the patients mood was.

Submitting summaries to an EPD

User story	As a care provider I can easily submit summarised reports to my EPD system.
Priority	Medium / 'Should have'
Description	An algorithm should be used to create a summary of the EPD, the summary includes the most important information and is shorter than the original EPD
Completion criterion	The created summary includes the important points of the EPD and is at least 50% shorter than the original EPD.

Incorporation of domotica

User story	As a care provider results of common tests (such as blood pressure) appear automatically in the interaction summary.
Priority	Medium / 'Should have'
Description	The system automatically includes the recorded data that is sent to the collection device. The sent data includes enough information for an algorithm to place the recorded date at the proper location in the EPD.
Completion criterion	A device can send data that will be automatically included inside the EPD.

Video recording

User story	As a care provider I can unobtrusively record a patient interaction through video.
Priority	Medium / 'Should have'
Description	Both the care provider and the patient are inconvenienced as little as possible by the video recorder.
Completion criterion	A care provider can record their interactions with patients without either of them being inconvenienced.

Video analysis

User story	As a care provider the video recording of the patient interaction needs to be analysed to extract medical features.
Priority	Low / 'Could have'
Description	Certain actions performed by either the patient or the care provider need to be recognised through a video recording. The system will then automatically include the recognised actions into the EPD.
Completion criterion	Video can be processed so certain actions can be recognised and added into the EPD.

Privacy

User story	As a care provider the system should never jeopardise the confidential relationship between me and my patient. As a software development/exploitation company the system should be compliant with privacy directives so that we do not face liabilities. As a patient I want my data to be safely handled.
Priority	High / 'Must have'
Description	All patient data is processed and saved in accordance with the relevant laws.
Completion criterion	All patient data is processed and saved in accordance with the relevant laws.

Maintenance and administration

User story	As the system administrator I want to easily manage and monitor all connected devices and sensors.
Priority	Low / 'Could have'
Description	A system administrator needs to be able to see which devices are connected. If there are problems connecting a device the administrator needs to be able to see what the problem is.
Completion criterion	An administrator can see which devices are connected correctly and which devices have problems and why this is the case.

Customisability

User story	As a care provider I want to be able to specify my own template when generating EPD entries.
Priority	Low / 'Could have'
Description	Although letting care providers specify their own template is perhaps outside of the scope of this project, the ability to have variable templates is important for future proofing the system.
Completion criterion	A care provider can choose their own template and use this to display the EPD.

Voice commands/interaction

User story	As a care provider I want to give voice commands to the system to influence the content of the report.
Priority	Medium / 'Should have'
Description	The care provider should be able to use pre-configured commands and keywords to directly put notes in the generated report. The commands can also covertly signal some information to be put into the medical report.
Completion criterion	There needs to be an option to configure commands which trigger special behaviour by the transcription and summarisation modules.

4.4 Project milestones

4.4.1 Minimal viable product milestone

The first goal will be to create a minimal viable product to serve as a skeleton upon which later development will take place. As part of this minimal viable product development will be done on an initial implementation of the most important interaction modality: speech. The milestone is complete when a rudimentary infrastructure is in place for recording, transcribing and summarising patient interaction. This milestone will be the main focus from the start.

4.4.2 Architectural design milestone

For the future viability of the project creating a good architectural design is of vital importance. While working on the minimal viable product, work will also be done, in collaboration with the customer, on the architectural design of the system. From the architectural design it should be clear how all parts of the system fit together and how possible future extensions of the project fit in.

4.4.3 Refinement milestone

The minimum viable product will be improved upon by giving end users more control of the summarisation. For example: end users can revise and correct parts of the transcript and summary; end users can re-listen to parts of the recording; end users can use voice commands and keywords to directly influence the report that will be generated. As part of these refinements the system should also be able to interface with at least one EPD system.

In this milestone the summarisation algorithm will be expanded to make more concise and informed reports. The algorithm needs to know what information is relevant and what information can be left out. To do this rules will be implemented that tell the program what words are and what words are not relevant. This is very static and needs to be improved or adjusted by hand which can be a great deal of work. A second option is to use AI techniques to determine what information is important. This can be done by recognising specific medical terms or other terms that are very rare. If a sentence contains words like 'blood pressure' or the name of a medicine then there is a high chance that the sentence is important. If a sentence however contains the word 'hello' then it is likely part of a simple greeting and not very important.

4.4.4 Domotics milestone

External devices can be added onto the system, these devices will record and transmit data that is put into the generated report automatically. The initial implementation can be some kind of appendix with the extra information. But ideally the information is included with the other information relevant to the subject. It can be risky to let an AI write new text so the goal is to search the report for information about the same subject and add to it. This way information concerning a certain subject is grouped together.

4.4.5 Completed project

Depending on the amount of time left improvements can be made to the data integration of both speech to text and external devices, more external devices can also be added. Since speech to text is already

being developed by other, much larger, companies it is better to focus on the areas that will improve this project the most. For this project it will most likely be the integration of the speech to text data and external device data.

4.5 Project models

4.5.1 Class diagram

Multiple things need to be interchangeable, this is important because the project is a prototype and so it can be more easily improved upon in the future. The best way to make this possible is to create interfaces that connect with the replaceable items in the project. A class diagram will clearly show which items can be replaced and which can't. Class diagrams will also improve the ability for the members of the project to communicate immensely. If changes need to be made to the projects class layout, which is inevitable, a class diagram will make this a lot easier.

4.5.2 Database model

For now the project is only going to need a database for referencing medial terms but it is very likely that the database will need to be expanded in the future. To make sure the database stays properly managed a model can be a very good tool to have.

4.5.3 Mathematical models

There are no plans to include mathematical models within the project. Machine learning has been discussed and has very little chance of being added to the project but if it is added it will need at least some mathematical models.

4.6 Extra research

4.6.1 Privacy legislation in health care.

The rules that this project has to abide by are the 'Algemene verordening gegevensbescherming' (AVG) as well as some mandatory assessments and extra measures [Autoriteit Persoonsgegevens, 2018].

Algemene verordening gegevensbescherming (AVG)

The right to data portability One (CP) is allowed to take personal data with them and transfer them to another CP. Data portability applies to data that is indirectly provided by the use of a service or device. Examples are data generated by a pacemaker or blood pressure monitor. The law does not apply to data that is not directly or indirectly provided by the use of a service or device. Examples are conclusions, diagnoses, suspicions and treatment plans.

The right to oblivion One (client) can ask the care provider to destroy their patient data. The law Wgbo states that one has to keep one's medical record for a minimum of 15 years. The right to oblivion does not include EPDs as a whole but one (client) can ask to have parts of their EPD deleted.

The right to insight One (client) has the right to look into one's personal data that one (CP) has processed.

The right to rectification and addition One (client) has the right to ask an organisation to rectify, to supplement or to shield one's personal data. One can ask for rectification if one's personal data are one of the following:

- Factually incorrect.
- Incomplete or irrelevant for the purpose for which they were collected.
- In another way used in violation with the law.

The right to restriction of processing One has the right to restrict the use of their data in certain situations.

The right with regard to automated decision-making and profiling One has the right to a human look at decisions about them.

The right to object to data processing One has the right to ask an organisation to not use one's personal data any longer.

The right to clear information about what one does with their personal data One (CP) is obliged to clearly inform new and existing clients what one does with their personal data and why. A privacy declaration is the most convenient way to meet the requirement of this law.

The right to accountability (new to AVG) One must be able to show that the correct technical and organisational measures to protect the data of one's clients are met as well as that the data processing meets the requirements of the AVG. The requirements are the following: no more personal data is processed than is necessary for the purpose of the processing of the data, access to personal data is limited for staff members and personal data is kept no longer than necessary.

Data Protection Impact Assessment (DPIA)

This is a tool to estimate the privacy risks beforehand so one can take measures to reduce the risks. A DPIA is only mandatory when the data processing has a probability to cause a high privacy risk for the one's whose data is processed. To determine this, the AVG states the following, an organisation has to carry out a DPIA when the following occurs: Personal aspects are being systematically and extensively evaluated based on automatic processing, including profiling, based on which policy decisions are made. Particular personal data or criminal data are processed on a large scale. Or if one follows people in a public accessible area on a large scale, with camera surveillance for example. The Autoriteit Persoonsgegevens has a list with all the types of processing where one has to carry out a DPIA. If the type of processing one wants to carry out is not on the list, one can make use of the 9 criteria that the European privacy supervisors have set up. If the type of processing meet two or more of these criteria then it is mandatory to carry out a DPIA.

Official for data protection (FG)

A FG is someone in the organisation that has the role to check if one abides to the AVG. According to the law a FG is mandatory in the three following situations: Government agencies and public organisations such as the government, healthcare institutions and educational institutions. Organisations that follow individuals or that present their activities on a large scale, such as profiling people to produce risks estimations. Or organisations that process particular personal data, such as someones health, race, political views, beliefs or past criminal records.

Register of processing activities

This register contains information about the personal data one processes. Organisations with more than 250 employees are mandated to keep up with this register. Organisations that have less than that are not obliged to keep up with this register unless the following situations are applicable: The processing of personal data is not incidental. One processes data that possesses a high risk for the rights and liberties of the owner of said data. Or one processes particular personal data, such as health, political views, beliefs or criminal past records.

Extra measures

It may be necessary to take extra measures. Extra measures one can take include the following:

NEN 7510 This is an important standard for information security in the health care sector.

Logging This means that each employee has to keep track of when they viewed someone's patient file and whose file they viewed.

Data Protection Policy In this policy one arranges which employee has access to which data. This can be mandatory in some cases: for hospitals, municipalities, social media companies and commercial information agencies.

Law Cliënten Rechten This law is applied in addition to the AVG law. This law creates conditions for the secure electronic exchange of medical data in the health care sector. The Law Cliënten Rechten states that one (CP) has to ask for one's client's permission to make their data available via an electronic exchange system [Zorgvisie, 2017]. Furthermore the client has the right to access their file for free and the right to have an electronic copy of their file. One (CP) has to ensure that the electronic exchange system that they use records who has made data available and who has seen it.

4.6.2 Electronic medical records

Every care provider is required to maintain medical records for patients. These records have to contain at least the following information [KNMG, 2018]:

- Information about medical treatments
- Important data for the continuity of the treatment
- Personal Information
- Written declarations

According to V&VN, the style of reporting has to meet the following criteria [V&VN, 2011]:

- Thoughtful, correct, verifiable, systematic and complete.
- Concise.
- Unambiguous. There has to be no doubt about the meaning of words or sentences and they have to be written in a way that only one interpretation is possible.
- Objectively. There has to be a clear distinction between objective observations and interpretations. Evaluations have to be substantiated.
- Concrete.
- Clear.

These medical records are stored in so called EPD (Electronic Patient Record) systems. Many care providers use different EPD systems. The EPD system used depends on the preferences and needs of a care provider. The healthcare sector contains many disciplines. Each discipline has its own needs regarding the system. Some examples of the different systems categories are JGZ (child protection), WIS (district nursing), HIS (general practitioners) and HAPIS (general practitioner centers). For each of these categories there are multiple software suppliers that deliver EPD systems. This results in many different systems that all work slightly differently. There is however an upside to this: many of these systems use the same guidelines or standards since this is a requirement of being part of the National EPD system. These standards are explained in the next chapter.

Standards

Most medical records are stored in the system of a patient's general practitioner. This is usually the starting point of health care. There are various protocols that are used for communication between the general practitioner and other care providers. The general practitioner usually gets an overview of the treatment a patient received by other care providers. Some examples of these guidelines are HA-HAP (general practitioner communication with a general practitioner centre) [NHG, 2016] and HASP (general practitioner communication with a specialist) [Richtlijnendatabase, 2017].

Communication between various providers often happens using HL7. HL7 is a worldwide standard for safe communication between EPDs [HL7, nd]. The HL7 standard is universal for all different healthcare disciplines. HL7 is only used for communication, the actual data stored inside EPDs can still differ. That is why Nictiz (an e-health specialist centre) has developed certain standards. The Nictiz standard differ

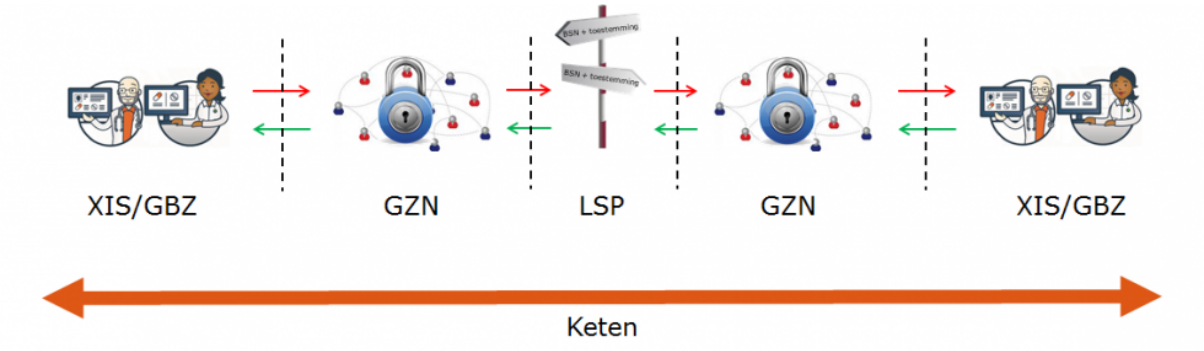

Figure 4.1: AORTA Infrastructure

for each of the different system categories. This means that not all systems can obtain information from each other, but this is not necessary. A child protection system does for instance not need information from a district nursing system. The benefit of the Nictiz standards is that for example two different child protection systems use the same type of data. This is mandatory to be able to have a National EPD system.

The national EPD system from VZVZ (Vereniging van Zorgaanbieders voor Zorgcommunicatie)

Background

The VZVZ [VZVZ, ndc] is responsible for the electronic exchange of medical information through a national sharing point. The ministry of Health, Welfare and Sports (VWS) was responsible for it before the VZVZ took over this role. The original idea of the architecture, called AORTA, originates from 2002. Patients are often treated by multiple different care providers. So it is essential that medical information can be shared between care providers. The original idea was to have a centralised database where the information would be stored (the national EPD). The ministry of VWS and Nictiz were responsible for the implementation of this system. In 2011 the Senate of the Dutch parliament voted against the implementation of a centralised EPD system. This resulted in the founding of the VZVZ. The AORTA infrastructure that is used now is decentralised meaning that information is not stored in a central database but in the systems of care providers that obtained the data. The way this information can be shared will be explained in the next chapters. An important difference with the original idea is that a patient's information can only be shared after the patient has given explicit consent to this.

Structure

A global overview of the AORTA infrastructure is shown in Figure 4.1. The different parts are explained in more detail below.

LSP (Landelijk Schakelpunt) The LSP is a reference index based on social security numbers. Care providers can use it to look into medical information about patients obtained by other care providers. The medical information is not stored in a central database but in the EPD system of the care provider that obtained the information. The LSP acts as a reference to the location of this data. [VZVZ, ndb]

GZN (Goed Beheerd Zorgnetwerk) To connect a system it should be able to securely transfer data to another GBZ (Goed Beheerd Zorgsysteem). This secured connection is provided by GZNs. A GZN is a network service provider that conforms to the standards and has been tested by the VZVZ. [VZVZ, nda]

GBZ (Goed Beheerd Zorgsysteem) Care providers use different systems to store patient information. If a care provider wants their system to be connected to the national EPD system it should comply with certain quality and information standards. Qualified systems that can be connected to the LSP are called GBZs. Systems should adhere to the Nictiz standards (XIS qualification) and be tested and qualified by the VZVZ. [VZVZ, ndd]

Structure of a care provider-patient conversation and report

There is a difference in reports based on a first meeting with a patient (the intake) and the subsequent meetings.

In general, (first) meetings/care provider-patient conversations have the following structure [NVMO, 1990]:

- **Small talk**
  The meeting starts off with some small talk about weather, traffic, etc.
- **Exploration phase**
  After the small talk the care provider will move on to the exploration phase by asking a question like 'What can I do for you?' or 'How can I help you?'. In this phase the patient will describe the reason for coming to the care provider. The care provider will then ask specifying questions and rephrase/summarise things the patient has stated to clarify and to get a complete description of the complaint.
  Example:
  For a complaint about pain the care provider will ask questions in these six dimensions:
  - Type of pain.
  - Severity of the pain.
  - Localisation of the pain.
  - Chronology of the pain (When did it start?, how did it develop?, etc).
  - Situation when the pain started.
  - Influence on the pain (Is there something that makes the pain worse or better?, was something tried to do anything against the pain so far?, etc).
- **Context information**
  The care provider will ask questions to get general information about the patient's health, lifestyle (alcohol, other substance use), working situation, home situation, family (could include medical information as well), medical history and other medication.
- **Further specification of the complaint and physical examination**
  After the exploration phase and the collection of context information the care provider will have a rough idea of what the diagnosis could be. They will then start asking more specific questions about the complaint. If the care provider deems it necessary, this part also includes doing tests and physical examination.
- **Diagnosis and further actions to be taken**
  The care provider will tell the patient what the diagnosis is and what the course of action is going to be.

The conversation might have a different structure if the situation calls for that. However, a care provider will probably still have this structure, or a structure similar to this, in mind during the course of the conversation.
Another way to describe the structure of a care provider-patient conversation is in terms of the SOEP-system [NHG, 2013]:

- **Subjective**
  Complaints, feelings and symptoms that made the patient go to the care provider.
- **Objective**
  Symptoms and other information that the care provider establishes. This includes data retrieved from medical examinations.
- **Evaluation**
  Evaluation and interpretation of the symptoms, if possible, including one or multiple diagnoses.
- **Plan**
  The course of action to be taken going forward. This could include further examination, redirection to another care provider, medication, etc.

Report content based on the care provider-patient conversation In general, if the care provider talks about a certain topic, it is probably relevant and will probably end up in the report. Not only is it important to include what symptoms the patient has, but also what symptoms the patient does not have. Furthermore, it is important to include that a care provider has asked certain things or has explored certain paths. These all lead back to the goals of reporting: verifiability, accountability and ability to reconstruct the course of events.

The conversation might also include other people, like a patient's partner or parents. The care provider will probably also ask them questions, which are all to be included in the report. The information must be traceable which means that it must be included if certain information about the patient does not come from the patient themselves.

The structure of a report will be very similar to the structure of a conversation. The SOEP-system is a widely used way to structure reports. Reports of first meetings (intake) are usually more extensive than subsequent reports. The care provider might use abbreviations or symbols to shorten/summarise certain information.

Non-verbal content The medical report can also contain information that has not been discussed with the patient. For example, if the patient was drunk, had a bad smell, has bad teeth. These things would also end up in the report. Furthermore, if the care provider notices a significant difference to other conversations this can also end up in the report.

Potential difficulties There are several things in a care provider-patient conversation that are worth noting:

- The conversation will likely have small talk at the beginning. This small talk is irrelevant and should probably not end up in the report.
- When a care provider asks a simple question to the patient, it is possible that a patient gives a slightly irrelevant or an overly detailed answer. In most of these cases, the essence of the answer can probably be shortened. For example, the care provider might ask the patient about how their sleep has been lately, to which the patient might give a very long answer detailing exactly per day how their sleep was that day. Such answers will probably be summarised in the report as 'varying sleep pattern'.

The general principle to reporting is that if the care provider talks about something or asks a question about something, it is probably relevant enough to be included into the report. The conciseness of the report does not come from leaving out information, but from shortening and summarising the answers/complaints given by the patients. It is more desirable to have included irrelevant information into the report that the care provider can later remove, than to have missed and not included relevant information. Structuring of the information included in the report can be done using the SOEP-system. It is important to ensure that the care provider has full control over the final report.

Conclusions

The different disciplines in health care that can be chosen for the system can be divided into three categories: general, specific and special care.

A discipline that falls under general care is general practice. This discipline is broader than others, so if this discipline is chosen there is more variability in the system. A general practitioner has a standard place where one-on-one conversations with patients are being held: their office. This means that there is no (extra) background noise for which extra measures would have to be taken (implementing/using special software, using expensive microphone). Choosing this discipline would also ensure that a large group of care providers would be able to use the system. A final benefit is that the EPDs of general practitioner adhere to the Nictiz standard, making it is very clear what information should be included in the report.

Two disciplines that fall under specific care are dermatology and otology. The downside of choosing a specific discipline is that only a specific domain will be targeted. This means that the system would be highly specialised with little variability making it less useful for other disciplines.

Two disciplines that fall under special care are home care and psychology. It would be difficult to generate a meaningful report from a conversation between a psychologist and a patient. Complex software has to be implemented to achieve this. For example, for documenting how the person is feeling, mood analysis has to be used.

Home care has widely different standards than the other disciplines, which makes adjusting it to a different discipline difficult. The conversations in home care do not take place in a fixed place so there will potentially be background noise. This means that if home care is chosen it will be hard to develop a system that can be used in other disciplines and measures will have to be taken (implement/use special software, buy an expensive microphone) to deal with the background noise.

Taking into account the pros and cons of each of these disciplines the decision was made to focus on the general practitioner (HIS). There are several important reasons for this: it is a common discipline in health care, the discipline adheres to the Nictiz standard and the conversations take place in a fixed space so background noise will be consistent.

4.7 Relation with lectures

4.7.1 Aspects of the lectures that can be used

So far the lectures have covered the following topics: Agile & Scrum, risks, planning, GitLab, Continuous integration and communication with the client. Future lectures include: testing, build management and scrum with discipline. The lectures spend most time teaching about difficulties surrounding the project as opposed to the actual technical difficulties encountered during the project.

The first item from the lectures that is actively being used during the project is related to scrum. In the lectures multiple ways were taught to handle a project: waterfall was discussed briefly but the main focus was on scrum. The reason for this is that scrum has shown an increased rate at which software projects were completed successfully. The project team will make a shippable project every two weeks and hold a stand up meeting every day at exactly 12:50 in accordance with the principles of scrum.

Two scrum roles will be used within the project team: the product owner and the scrum master. In this project the role of scrum master has been divided between two people so that more people can improve their management skills.

Another aspect of scrum that will be used is the project backlog, this is significantly easier to use than a requirement document. After creating user stories, a product backlog can be created. By utilising MoSCoW and planning poker an importance level can be assigned to each item. This combined with the shippable project that is created every two weeks will help ensure that the product complies with the client's wishes.

Two other useful aspects from the lectures are risk management and communication with the client. Approximately 57% of projects fail because of bad communication. This means that it is vital to have an open line of communication between both team members themselves as well as between team members and the client. Proper risk management is also vital to the completion of a project, getting ahead of problems and solving them ahead of time will make everything proceed more swiftly.

4.7.2 Difficulties

Although the lectures focus on the non technical part of the project, difficulties can still arise on the technical side. Choices will need to be made during the project and it is not possible to make the correct one every single time. With the lack of experience of the group members it is likely difficulties will occur here. The different kinds of choices will require knowledge that could be missing in the project group, this is an opportunity to learn but can also mean that time is wasted on bad choices.

On the non-technical side there are also multiple difficulties and challenges that are not discussed in the lectures. The biggest challenge that might lead to difficulties is the project team itself. Working so close together with a team for such a long time is not something that many people in the project group have experience with. Difficulties and conflicts that might arise in this dimension are not discussed in the lectures. In a team with ten different people there is likely going to be conflicts and it will be up to the project team to solve these. Conflicts could both improve the project, since different options will be discussed, or decrease the quality of the end product, since time could be wasted on arguing over matters that do not impact the final product.

The lectures only give a global view of scrum and its aspects, this means that not every subject is covered. Things like structuring the various meetings or the various management roles are not discussed in depth. This might cause difficulties since different team members might make different assumptions about how things should be done or what a certain role should include.

Chapter 5

Architecture, design and technical choices

Introduction

Every system has an architecture, which serves as a kind of blueprint for the system and the developing project. This architecture lays out some of the tasks that need to be executed by the software developers.

To begin to create a new system from scratch there are choices to be made about the hardware on which it has to run, possible existing software that it needs to support, and what programming languages will be used. These are all closely intertwined, and each option has its pros and cons.

This part includes research into different choices that need to be made for the `Speech2EPD` system. After each research section an advisory conclusion is drawn. It also includes system specifications consisting of a high-level system description and code standard.

5.1 Development platform

5.1.1 Android native app development

Pros

1. Android is the most used operating system on mobile phones
2. Cheap one-time fee to distribute an app on the Play Store
3. Cheap phone and tablet options
4. Java is very similar to C#, which all of the developers are familiar with
5. Development can be done on any computer running Windows, Mac or Linux

Cons

1. Large differences in hardware between devices running Android
2. Large spread in Android versions on the devices in use
3. Different app versions needed for different screen sizes
4. Only one way of distributing licensed applications

5.1.2 iOS native app development

Pros

1. Multiple ways of distributing the application
2. Modern programming language (Swift) that creates fast applications

3. Hardware is almost all the same, no large differences between models
4. One version of the app can work on all models of iPhone and iPad in use
5. Xcode has good simulators, bug-fixing and testing possibilities
6. Apple products have automated software updates, which means that most users use either the same version of iOS or one version older.
7. iOS is the most used operating system on tablets

Cons

1. Xcode only runs on Mac OS (virtual machines could be a solution)
2. Many of the developers do not know Swift
3. The price of both the developer (enterprise) program and the hardware is quite steep
4. Swift might not work well with existing technologies because of its short existence so far

5.1.3 Universal Windows Platform development

Pros

1. Apps written with the UWP API can easily be distributed to desktop computers and hand-held devices running Windows 10
2. Cheap hardware options
3. Multiple choices for distribution
4. All developers are very familiar with C#

Cons

1. Only one model phone runs Windows 10 Mobile
2. Tablets (or 2-in-1s) rarely run on Windows 10 compared to iOS or Android
3. Development can only be done on a Windows computer (or with a virtual machine)

5.1.4 Phone and tablet options

Since the three main possibilities for mobile operating systems are Android, iOS and Windows; only devices running these OSs are considered. All modern mobile phones and tablets have multiple cameras (front and back) and microphones. The quality of the camera could influence the performance of the action recognition software and should be evaluated carefully.

The phones and tablets listed below are all quite new models to make sure they give the best quality of raw data possible. The choices for phones and tablets running iOS are easy as they are just the three newest models that are currently being sold by Apple themselves (the iPhone X was discontinued because of the neglectable differences with the newer iPhone XS).

The choices for phones and tablets running Android are a bit more difficult. In the case of the phones, the two most popular Samsung devices were chosen as well as a different popular model from the up-and-coming brand OnePlus. In the case of the tablets, the three best selling tablets of 2018 were chosen. This includes two Samsung Galaxy Tab models as well as a popular Huawei model.

The choice for phones running Windows is quite straight-forward since there is only one model running Windows 10: the HP Elite x3. In the case of tablets, models from three different brands have been chosen: ASUS, Lenovo and Microsoft.

The stores of the different platforms all have a store fee of approximately 30%, making this a non-issue when choosing between platforms.

When making a final choice the difference between tablets and phones should be carefully considered, as the most used operating system on phones is Android, while on tablets it is iOS. In general, if you want to develop an application that can run on tablets the safest bet is to develop it for iOS as only a relatively small amount of people have a tablet that runs Android. If the choice is to develop for phones, both Android and iOS can be good choices, since the market share for Windows is significantly lower.

5.1.5 Conclusion

If the product has to be used by a large amount of care providers, as well as on-the-go, the best option would be to write an app for either Android or iOS, or both. The most used operating system on mobile phones is Android, while the most used operating system on tablets is iOS.

However, since this is supposed to be a prototype and proof-of-concept, these things are not as important to consider. With regards to quick development – without having to learn a new language – Windows would be the best bet as everyone in the development team is familiar with C# and most have a laptop running Windows.

If the prototype is good enough it can also be implemented on the desktop computers of care providers with minimal changes because of the UWP API. Because of this, the application will be useful even when it does not yet run on most mobile devices.

Because of the above mentioned reasons the application will be developed using the Universal Windows Program API.

5.2 Speech recognition

There are two main options for speech recognition: Cloud-based speech recognition and offline speech recognition APIs/services.

5.2.1 Cloud-based

Cloud-based speech recognition services like Google's Cloud speech-to-text have the following architecture: speech is recorded locally and sent to Google's servers. There, the server will convert the audio file into a text file using its highly advanced speech to text algorithms (which also use neural networks and machine learning technology).

Other (top of the industry) Cloud-based speech recognition services such as Amazon Transcribe and Bing Speech are not being considered because they do not support the Dutch language.

5.2.2 Offline

There are also a few options for offline speech recognition.

Examples are:

- CMU sphinx: an open source speech recognition API
- Google's offline (Android) speech recognition
- Windows' UWP offline speech recognition

This software can be run locally on a device. This has the obvious advantage of always being in full control of the data.

5.2.3 Punctuation

One important feature of the speech recognition API is the ability to detect interpunction. This is crucial for text analysis. Google Cloud speech recognition service [Google, nd] and CMU Sphinx [CMUSPhinx, nd] both have the capability to detect and include punctuation in speech transcription.

5.2.4 Privacy

There is one main problem concerning the use of a Cloud-based speech recognition service: privacy. As mentioned before, recorded audio would have to be sent to the server of the speech recognition supplier to be transcribed. Audio data from conversations between care providers and patients is highly sensitive. It is not clear whether using a Cloud-based speech recognition service would be compliant with the strict privacy guidelines concerning this data.

However, the to be designed system only has to be a proof-of-concept. The goal is to prove that the concept of `Speech2EPD` has potential so that more research and development can be done towards the system. The product is not yet going to be used in a real situation so the system does not yet have to adhere to all strict privacy standards. They are however being considered carefully during the design

of the product. In this case it is our opinion that using the best possible technology is more important than adhering to the strict privacy guidelines. This is because we think that it will vastly improve the quality of the proof-of-concept, which will improve the chances of further research and development. We will, however, decouple the speech recognition functionality as much as possible and document the functionality of it extensively so that it can be replaced easily when privacy issues do have to be taken into account.

5.2.5 Conclusion

Our preferred Cloud-based speech recognition service is Google Cloud's speech recognition. It is one of the few Cloud-based speech recognition services that supports the Dutch language (Amazon and Bing do not), it is regarded as one of the best speech recognition services out there (although we could not find a comprehensive and independent comparison between the different services) and it is also incredibly well-documented and relatively easy to implement.

5.3 Text analysis

To realise a satisfactory textual analysis, multiple linguistic tools will have to be evaluated. In the coming part a number of tools will be researched with the goal of defining the pros and cons of each tool. At the end a preference towards a tool (or tools) will be expressed, with regards to the requirements of the program.

5.3.1 Requirements

The to be used tool or tools will have to meet some requirements. These requirements are defined as follows:

1. Multilingual (priority for Dutch and English)
    (a) This allows the option of adding other languages besides Dutch in the future
2. Compatibility with medical terms (possibly supplied by the development team)
    (a) Option: train own model using datasets of conversations
3. Preprocess the raw text, for example by splitting it into sentences.
4. Ranking system (possibly with custom rules)
5. Possibility to run analysis server side with regards to performance
    (a) Optional client side
6. Cloud services

5.3.2 Summarisation algorithms

The two most used types of summarisation are extraction based and abstraction based summarisation. In this chapter we will discuss how these approaches work and what the advantages and disadvantages of each are.

Extraction based

The most conventional and most used way for automatic text summarisation is extraction based summarisation. The core idea is to statistically select the most important parts of the input text and combining them to build the summary.

A proven algorithm that does this is TextRank. It selects important sentences by looking at the similarities between all sentences and selecting the ones that have the most similarities with other sentences. It then combines those sentences, in the order they were in the input text, to create the summary. Significant disadvantages of this extraction based approach are that the produced summaries can be bloated if the source sentences were bloated and that references such as 'she' and 'that' do not make sense if the previous sentence is not part of the final summary [Bhargava et al., 2016]. It might be possible to solve some of this by extensive postprocessing.

Abstraction based

The most promising and most cutting edge approach to summarisation is abstraction based. Instead of using existing parts of the source text it extracts the meaning, abstracts it, and builds new concise sentences to form the summary [Khan and Salim, 2014].

Most abstract based approaches rely on either large semantic data sets or large rule sets to make natural language 'understandable' and workable for a computer. Unfortunately, because the field is very new and it is resource intensive to create these databases, few Dutch versions exist. This is especially the case for the most useful ones such as an Abstract Meaning Graph [Banarescu et al., 2013].

Possible preprocessing

Most summarisation algorithms are made for written text. Which causes them to be dependent on text specific metadata like sentence separation, paragraph separation and chapter headings. It also means that they do not account for speech artifacts such as 'uhms' and long grammatically incorrect sentences with lots of stumbling and repeated words. To account for this it might be necessary to perform extensive preprocessing on the raw text.

5.3.3 Natural language processing tools

WordNet

WordNet itself is a semantic lexicon of the English language, which groups English words in so-called synsets (sets of synonyms). It also contains short definitions and several semantic relations between sets [Fellbaum, 2005]. This tool by itself will be of no use to us, since it is defined solely for the English language. There are however some tools which have been derived from this lexicon with the goal of adding more languages (including Dutch) to it. The Dutch version is called the Open Dutch WordNet and contains 117914 synsets [Postma et al., 2016].

ConceptNet

ConceptNet is used to create word embeddings – representations of word meanings as vectors, similar to word2vec and GloVe (remember from DAR). Although the performance of ConceptNet seems to outperform these other libraries [Speer and Havasi, 2012]. A benefit of using ConceptNet would be that it is compliant with other lexicons, such as the aforementioned WordNet. The requirement of capacity of the Dutch language is also met by this tool. Using this tool in programming is done by making an HTTP request to their API and parsing the resulting JSON response.

NLTK

(Python)

NLTK is not so much a lexicon itself, but rather an interface to a bunch of different corpora and lexical resources (including WordNet). It contains a full set of text processing libraries (classification, tokenization, stemming, tagging, parsing, and semantic reasoning) and also has compatibility with the Dutch language [Bird et al., 2009].

spaCy

(Cython = written in Python, translated to C for increased performance)

spaCy is similar to NLTK as they are both used for natural language processing. spaCy, however, is built from the ground up using neural networks and based on a different language (Cython) which allows it to have a better performance than NLTK. Similar to NLTK, spaCy also has compatibility with the Dutch language [spaCy, nd].

The focus on neural networks and deep learning, which seems to be inherent in this tool could be beneficial as there might be a need for the option of introducing Dutch medical terms to a lexicon.

Conclusion

The tools that have been described in this section are useful in a sense that words can be assigned a certain meaning. This meaning can then be used to improve analysis by designating keywords and finding synonyms for these words. To conclude, these tools are a supplement to the textual analysis as

a whole. NLTK and spaCy seem like they comprise the most features and tools compared to the former two options. Both of these tools are free, so there are no differences in pros and cons in that respect. Technology-wise it seems that spaCy outperforms NLTK due to its higher performance and focus on neural networks.

5.3.4 Medical lexicon

Since the `Speech2EPD` project will be used in the field of medicine, the text analysis will have to incorporate medical terms. This will require some sort of lexicon or database containing medical terms in Dutch (and optionally English as well), which can serve as a weighting factor that increases the weighting of medical terms.

Commercial medical databases

When doing research on medical databases or APIs, the two options that came up were commercially available databases and APIs. The first of these is a product called Base24 [Base24, 2014], but after inquiring about the actual use and costs of using this piece of software there has been concluded that this product will not be useful to the project.

The second and final commercial piece of software we came across is Medipim [Medipim, nd]. This product is comparable to Base24, but differs in the sense that it also serves as a database for cosmetic data. Medipim is not useful within the scope of this project for the same reasons as Base24.

Webcrawler

After concluding that commercially available Dutch databases/APIs were not going to be of use, research has been done towards another option: using a webcrawler to obtain data from databases or large pieces of text.

Public medical databases There are some options for public databases containing Dutch medical data, such as: apotheek.nl [KNMP, 2018], geneesmiddeleninformatiebank.nl [CBG, 2018] and gipdatabank.nl [Nederland, nd]. However, all of these databases are protected by copyright so using these options by means of a webcrawler will not be considered.

Wikipedia Wikipedia offers a large amount of data which could be of great use in weighting medical terms. It is legal to download the entire Dutch part of Wikipedia, after which a tree of pages which are categorised as either pharmaceutical or medical can be built. It is important to realise when considering Wikipedia as a source of data that the data does not have to be perfect (seeing as Wikipedia is open source and possibly incorrect in some places), since the medical data will only be used to increase weighting of medical terms in a conversation between a care provider and a patient.

Conclusion

To conclude, the best bet for a medical database or lexicon would be using Wikipedia. This is due to the other options either being too expensive, not legally available for use or even not useful in regards to the system requirements. The fact that the data does not have to be perfect, since it will be used for statistic textual analysis, means that Wikipedia can be used as a valid source.

5.3.5 Conclusion

The current text analysis strategy is to start with a basic statistical algorithm, such as text rank as the backbone of the program. This will ensure that there is a working prototype from the beginning, since the basic algorithm is easy to implement. Later on in the development process it is easy to expand text rank, making it more powerful in this specific context. This can be achieved by giving the program access to medical and semantic databases. To ensure the program generates usable output it will be fine-tuned with domain specific rules. These rules can include giving the sentences the doctor said more weight or forcing select keywords to always be a part of the final summary.

The reason for not choosing an abstract approach, is that we do not possess the necessary training data, time or expertise. We also believe that the enhanced statistical approach will be sufficient for a

working demo. The program will be designed in such a way that it will be easy to replace our text analysis with a more powerful one after this project.

5.4 Video analysis

To test and ensure that our server architecture can handle video analysis and its results, existing video analysis software will be used or minimally improved. This will be interface with our controller and the results will be used for the EPD and for the improvement of the text analysis. Selfmade custom video analysis will not be developed because it is not possible within the current time frame. Research into what video analysis software can be used will be done in the first sprint. The development of the server and the local software will be done with video analysis in mind from the very start.

5.5 Server side architecture

5.5.1 Microservice architecture

A system requirement from the client is that the product must be highly modular. This means that the possibility of having various kinds of recording devices should be maintained, since parts of our project might be developed further or replaced entirely in later development. For this reason a so-called microservice architecture will be used. Wikipedia describes a microservice architecture as follows:

> A microservice is a software development technique that structures an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop, and test and more resilient to architecture erosion. It parallelises development by enabling small autonomous teams to develop, deploy and scale their respective services independently.

This is highly beneficial for the program since, as mentioned above, it will allow for easier independent and parallelised development. This is useful as the voice-to-report epic can be effectively broken into independent technical modules within the system. To be more specific, there are three stories: a voice-to-audio story focusing on hardware and audio capture; an audio-to-text story focusing on transcription (text to speech) technologies; a text-to-report story focusing on extracting semantic meaning and generating reports from the transcript. In a microservice architecture these stories can be developed as largely independent services with standard generic protocols serving as cross boundary communication.

A good technical match for the microservice architecture are containers, see Figure 5.1. They are operating system level lightweight visualisations of environments in which services can run. They further decouple the services and offer maximal flexibility in deployment: services can be deployed side-by-side or on different hosts. There is also a choice between running them locally or in the cloud.

5.5.2 Containers and orchestration

A container engine is a tool for specifying and running containers. For this there are two main contenders: docker and RKT. Of these two, docker is more well-known and tested but RKT claims to have a superior security architecture.

In conjunction with a container engine a so-called container orchestrator can be used. The orchestrator manages nodes (physical machines) and spins up containers using a container engine. Docker itself has a container orchestrator named Docker Swarm, but the product leader in this space is Google's Kubernates. Kubernates works with Docker as its container engine by default but it can be configured to work with its own oci-o engine.

There is currently a standardisation effort to standardise container specification formats and container runtimes under the name Open Container Initiative (OCI). As part of this initiative the docker toolchain has been decomposed into dockerd, containerd, containerd-shim and runC.

Containerd and runC have been donated to a neutral organisation. runC is an OCI compliant runner of container images. Containerd is a daemon managing the running containers and calls runC to run the images. Dockerd manages container repositories, user authentication, etc. and remains part of Docker proper. When using kubernates it is possible to replace the containerd part of the infrastructure with

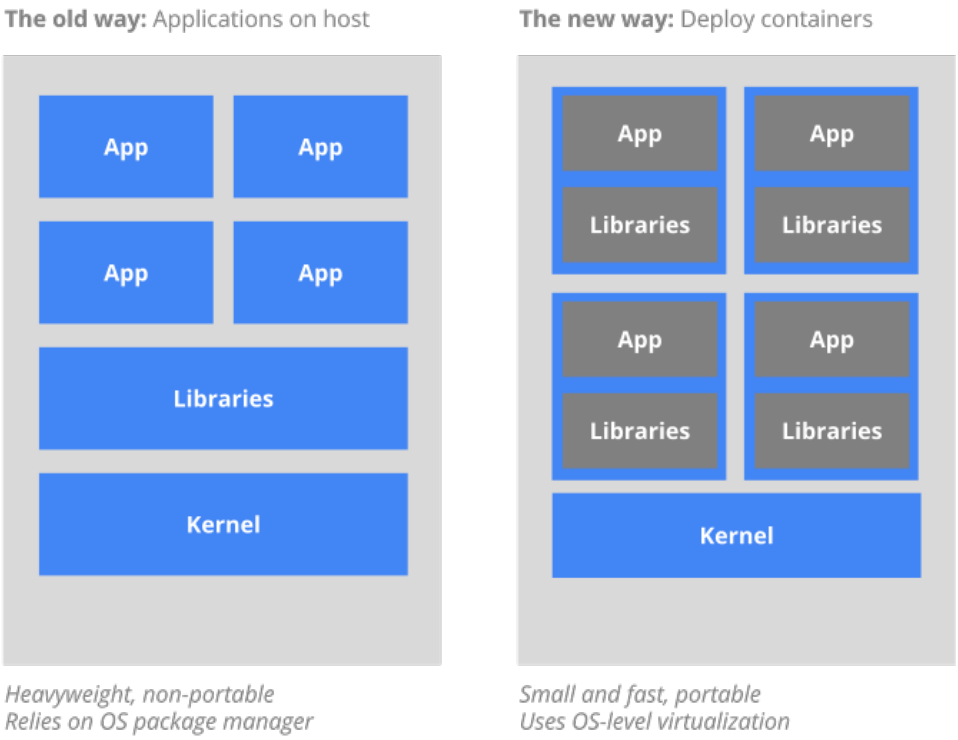

Figure 5.1: Usual vs container based architecture (Image courtesy of `Kubernetes.io`)

their own OCI runtime implementation named oci-o which also uses runC to actually run the containers. This might be a future direction Kubernates is developing towards.

For now using the combination of Docker and possibly Kubernates is recommended. Docker is the most mature and complete container engine and its formats are standardised by OCI, so switching away later will be relatively hassle free. Kubernates will be more useful once there are many components and the management of the system becomes more important. For now Kubernates documentation is written with Docker in mind, so this is the easiest route to take while retaining the possibility to change the container engine at a later point.

5.5.3 Service languages

With the microservice architecture it will be easy to write different parts of our product in the best programming language for the job. For this product there are multiple attractive programming languages.

Python

The first is Python. Python is attractive because it is the language of choice for machine learning. Many libraries are written with Python interfaces and usage of Python is the industry standard. A downside is that the dynamic typing makes it harder to write large applications. Python is most suited for writing 'glue' code between different parts of the system and libraries. Python is less suited for writing big subsystems. There are docker images available to serve as a basis for the microservices.

C# via .NET Core

The second is C#. All of the developers already have experience with C#. With .NET Core the language fits into the Linux/docker based microservice architecture. Microsoft produced images are available in the Docker repository. The positive open source/Linux moves Microsoft has been making, makes C# even more attractive than its competitor Java. From a technical perspective, C# is also a more ergonomic language.

5.6 Domotics

One important aspect of the Speech2EPD is the use of domotics to reduce the administrative tasks of the care provider. This can be done by connecting a medical device with the system and reading the results of the examination of the medical device and documenting it automatically. This is not an easy task. First of all, a way has to be found to interface with various medical devices. Once this has been done, the data that the medical device is outputting can be transmit to our own server. From there, a data aggregator will combine the input with other data from the Speech2EPD to compose a report.

5.6.1 Interfacing

Interfacing with the medical devices is the most difficult part. According to [Lantronix, 2010], almost every medical device has a serial or communications port that can be used to send data to another device. Some medical devices also have network connectivity built-in. However, the physical connection and the protocols and data standards may differ. Two promising electronic data exchange standards for medical data have been found: HL7 and IEEE 11073. However, it is still unclear how widespread and viable these standards are.

Devices[1] that claim to be able to interface with most medical devices and provide IoT (Internet of Things) capability at the same time have also be found. Using these would make the interfacing significantly easier, but more research to the viability of these products has to be done.

5.6.2 Connectivity

If a device that does both interfacing and connectivity cannot be used, both aspects will have to be developed. Once the data has been transmitted by the medical device, the data needs to reach the system. The proposed way of doing that is by using IoT connectivity. For this a microcontroller with Wi-Fi capabilities will be used. There are several ways to connect the microcontroller to a network. If the system is used by a general practitioner, the microcontroller can connect to the Wi-Fi in the building. If the system is used by a home care assistant, there are two options. First, if the tablet that is used by the home care assistant has access to a mobile network, the device can be connected to the tablet's hotspot of this network. If the tablet does not have access to a mobile network, the data can be saved and transmitted once the device has a Wi-Fi signal (for example when the home care assistant returns to the practice). For this to be possible the assumption has been made that the data transmitted by these medical devices is of a very small size.

5.6.3 Proof-of-concept

It is not yet clear if we can build (or buy) a device that can easily interface with a plethora of medical devices. Because of this, the proposal is to focus on one device. A device will be chosen that uses a standard that is widely used (if there is one overwhelming winner, an attempt will be made to choose this one). It will also be a priority that the medical device that is used is widely used in the field that will be chosen to base the demo on. By doing this it is guaranteed that the system is easily extendable and relevant in the demo.

5.7 System specifications

5.7.1 System overview

This section lays out a general overview of the architecture and design of the system. The current design can be divided into two relatively large components: the *local device* and the *server*. These two separate systems are connected to each other and will exchange data. The most important exchange being the sending of raw data from the local device to the server for analysis.

Local device

The following describes the system architecture (see Figure 5.2) for the local device. This device will handle all the external inputs and preferably perform as little calculations and analysis as possible.

[1] https://nl.mouser.com/new/lantronix/lantronix-sgx-5150-gateways/

Instead, the server will do most of the analysing, which will be described further on.

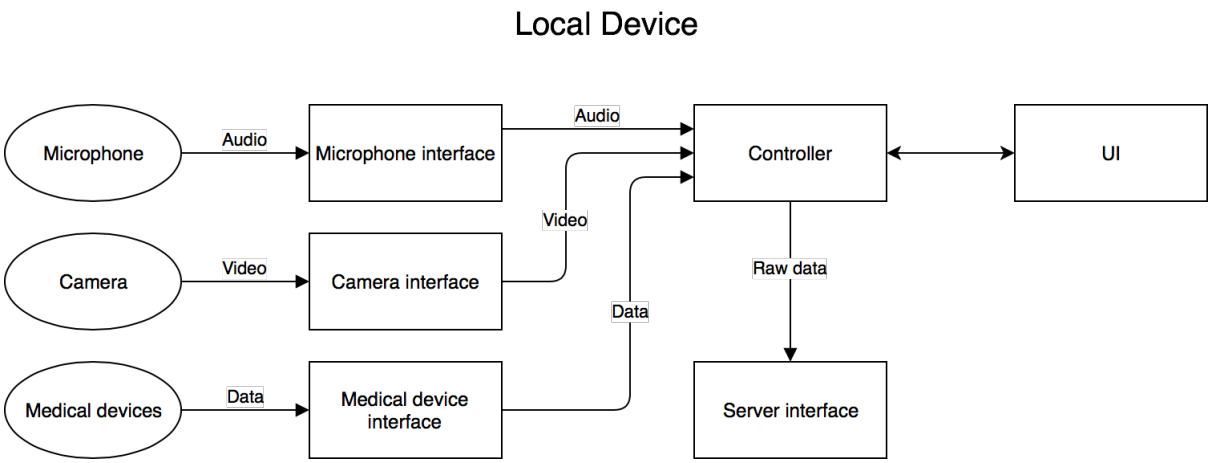

Figure 5.2: Overview of the system's local device architecture

Controller

The controller is arguably the most important component of any system. The controller of the local device in this project it mostly serves two purposes. First and foremost, it receives the data from the interfaces (possibly after some analysis to decrease the amount of data) and sends this raw data to the server interface, which in turn will then send the data to the server by a predefined protocol. Second, it communicates with the UI, which is how the device communicates to the user.

Interfaces

The use of interfaces allows for interchangeable external devices. This allows for, in a later stage, deviating from earlier plans and using different devices that end up being more closely linked to the desires and requirements of the final product.

Server interface This interface represents the connection of the local device to a server. This communication will use a standardised protocol to improve flexibility. The controller manages the data that will be sent to this interface.

External device interfaces These interfaces represent the connections of the local device to external devices. These interfaces are all connected to the controller, which will handle the data flow of the input data.

External devices

External devices are used to collect data in the form of audio, video or a format related to a specific medical device. These devices come in the form of microphones, cameras or medical devices. The specific device used can be changed due to the use of interfaces, which should handle the input data and translate this into a standardised format.

UI

The UI offers the user a means of communicating with the system. Details regarding the implementation of the UI will be laid out in a later stage.

Server

In general the system has been designed (see Figure 5.3) in such a way that every component or microservice can be exchanged for something else (and hopefully better) later on. That is why all components communicate with the system via interfaces and all data streams go through the controller.

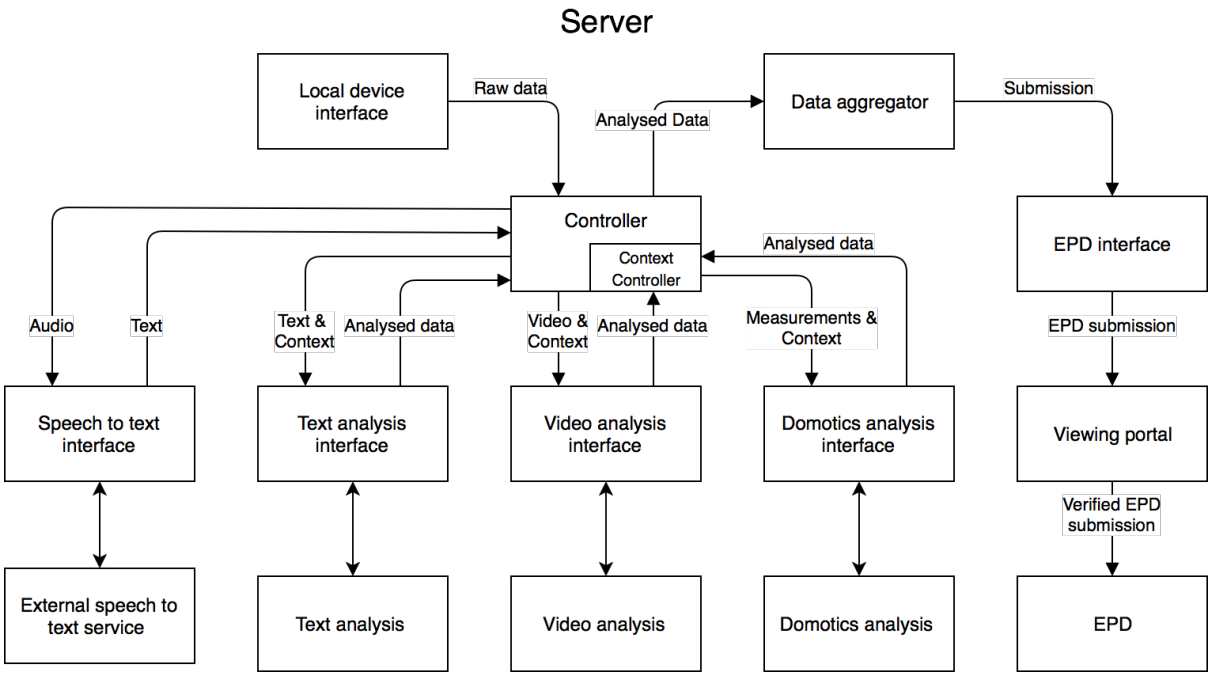

Figure 5.3: Overview of the system's server architecture

Controller

The controller controls the data flow of the server, it gets the raw data from the local device interface and divides it between the different analysers, collects the data and passes it to the data aggregator.

Context (sub)controller The context controller handles the communication between the different analysers, it does this by extracting relevant information from analysed data and passing this to the next analyser. This ensures that some analysers can use context from other means of analysing to improve their own analysis.

Speech-to-text

The speech-to-text interface handles the speech to text. In the first version of the system this will communicate with the Google speech-to-text services. But its communication with the controller will be standardised and documented so that it is trivial to replace the speech-to-text service simply by changing the interface.

Analysis interfaces

Our analysis interfaces will handle the communication between the controller and specific analysis microservices. This makes it easy to choose for a different analysis approach later on. The communication between the controller and an interface will be standardised and documented. To ensure that results from one analysis microservice can be used in other analysis microservices, the input for these services will include generic context data.

Text analysis

Text analysis receives raw text from the controller combined with possible context data from other previously analysed data. It combines these two pieces of data and analyses it. In the current planning it will perform minimal preprocessing and generate metadata from the raw text to return it to the controller. The metadata will initially include a per word relevance value, but can be extended to include semantic data and usable context for other analysis microservices. The system will be designed in such a way that later on, if the text analysis is exchanged for a more advanced abstraction based approach, it will be able to handle an abstract meaning representation graph as output.

Video analysis

Similarly to the text analysis, the video analysis microservice receives raw (video) data plus context data, combines the two, analyses this combination, and then returns this analysis to the controller.

Domotics analysis

Similarly to the text analysis, the domotics analysis microservice receives raw (domotics) data together with context data, combines the two, analyses this combination, and then returns this analysis to the controller.

Data aggregator

The data aggregator combines the preprocessed data from the different analysis microservices to create the submission. It uses the meta data from text analysis to create the summary and adds data from domotics and video. It extracts the data for the EPD specific fields from the preprocessed data and adds this data to the submission.

EPD interface

This interface receives a submission containing all the necessary data for an EPD from the data aggregator. This data will then be neatly translated into a certain, standardised EPD format and sent to the viewing portal, where users will have a chance to verify the resulting submission.

Viewing portal

This portal allows for users (medical professionals) to manually verify or, where needed, correct the EPD submission. After this final step of verification, the now verified EPD submission will be passed on and saved to the actual EPD of a patient.

5.7.2 Programming language

For the project multiple programming languages will be used. For the GUI of the application C# will be the main language. The server side will consist of multiple microservices which can each use their own language, one that will most likely be used is Python.

5.7.3 Code standard

To ensure uniformity in the code even though it is being written by multiple developers a code standard has to be established. Because of the decision to develop an application for Windows 10 using the UWP, one of the programming languages used is C#. There are a great deal of existing C# style guides, this code standard overlaps with multiple of those.

Naming guidelines

Pascal case The first letter in the identifier and the first letter of each subsequent concatenated word are capitalised.

Camel case The first letter of an identifier is lowercase and the first letter of each subsequent concatenated word is capitalised.

Case sensitivity

1. Do not create two namespaces with names that differ only by case.
2. Do not create a function with parameter names that differ only by case.
3. Do not create a namespace with type names that differ only by case.
4. Do not create a type with property names that differ only by case.
5. Do not create a type with method names that differ only by case.

Abbreviations

1. Do not use abbreviations or contractions as parts of identifier names.
2. Where appropriate, use well-known acronyms to replace lengthy phrase names.
3. When using acronyms, use Pascal case or Camel case for acronyms more than two characters long.
4. Do not use abbreviations in identifiers or parameter names.

Word choice

Avoid using class names that are duplications of commonly used .NET Framework namespaces. In addition, avoid using identifiers that conflict with keywords in commonly used languages.

Type name confusion

Different programming languages use different terms to identify the fundamental managed types. Use names that describe a type's meaning rather than names that describe the type. In the rare case that a parameter has no semantic meaning beyond its type, use a generic name.

Do not create language-specific method names. In the extremely rare case that it is necessary to create a uniquely named method for each fundamental data type, use a universal type name.

Class naming guidelines

1. Use a noun or noun phrase to name a class.
2. Use Pascal case.
3. Do not use a type prefix on a class name.
4. Do not use the underscore character.
5. Where appropriate, use a compound word to name a derived class. The second part of the derived class's name should be the name of the base class.

Interface naming guidelines

1. Use a noun, noun phrase or an adjective that describes its behaviour to name an interface.
2. Use Pascal case.
3. Prefix interface names with the letter `I`.
4. Use similar names when you define a class/interface pair where the class is a standard implementation of the interface. The names should differ only by the letter `I` prefix on the interface name.
5. Do not use the underscore character.

Attribute naming guidelines

Always add the suffix `Attribute` to custom attribute classes.

Enumeration type naming guidelines

1. Use Pascal case for `Enum` types and value names.
2. Do not use an `Enum` suffix on `Enum` type names.

Static field naming guidelines

1. Use nouns, noun phrases or abbreviations of nouns to name static fields.
2. Use Pascal case.

Parameter naming guidelines

1. Use descriptive parameter names.
2. Use Camel case.
3. Use names that describe a parameter's meaning rather than names that describe a parameter's type.

Method naming guidelines

1. Use verbs or verb phrases to name methods.
2. Use Pascal case.

Property naming guidelines

1. Use a noun or noun phrase to name properties.
2. Use Pascal case.

Event naming guidelines

1. Use an `EventHandler` suffix on event handler names.
2. Specify two parameters named *`sender`* (of type object) and *`e`*.
3. Name an event argument class with the `EventArgs` suffix.
4. Name events with a verb.
5. Do not use Before or After prefixes. Instead make use of the tenses in the English language.

Error raising & handling guidelines

1. All code paths that result in an exception should provide a method to check for success without throwing an exception.
2. End `Exception` class names with the `Exception` suffix.
3. Use the three common constructors when creating exception classes in C#.
4. Create grammatically correct error messages with punctuation. Each sentence in the description string of an exception should end in a period.
5. Create meaningful message text for exceptions, targeted at the developer.

Style & comment guidelines

1. Indent four spaces at a time.
2. Use the indentation suggested by the .NET environment when creating C# code.
3. Always place an open curly brace ({) on a new line.
4. Comments should be placed on a separate line, not at the end of a line of code.
5. Begin comment text with an uppercase letter.
6. End comment text with a period.
7. Insert one space between the comment delimiter and the comment text.
8. Use comments to explain why something is done, not what it is doing, unless it is very complex.

Chapter 6

Risk analysis

6.1 Introduction

This document contains a risk analysis for the `Speech2EPD` project by Kièli Ltd. It gives an overview of the stakeholders and their interests in the first section. The second section establishes both the functional and non-functional constraints.

In the third section risks are specified per category. Risks are then shown on an impact probability matrix to visualise their impact and their probability. The most important risks (those with high probability and/or high impact) are then expanded upon with a risk plan.

Finally the fifth section states the relation between the risk analysis and the lectures given as part of the software project of Utrecht University.

6.2 Stakeholders

The stakeholders for this project are:

- The client
- The development team
- The project bureau
- The patient
- The care provider
- Health institutions
- The government
- Taxpayers
- Future developers
- Utrecht University

A full overview of the stakeholders and their interests can be seen in tables E.1 and E.2 in the Appendix.

6.3 Constraints

6.3.1 Functional constraints

The text to speech algorithm needs to be able to transform audio to text with enough accuracy. After transformation, processing can be done on the text to further increase the accuracy. The final result should be as close to 100% accurate as possible.

6.3.2 Non-functional constraints

The program needs to conform to the Dutch privacy laws concerning patient data. Every piece of data that is going to be processed by the system has to be secure.

Protection needs to be provided for the data after it has been recorded, any processing done on the recordings and the data stored on the server.

6.4 Risks

In the following section, a number of risks will be specified in the following categories: communication, stakeholders, data, domain-specific knowledge and technology. These risks will be portrayed on an impact probability matrix. Finally, some risks that are defined as high in both impact and probability will receive a plan of action towards dealing with said risk.

6.4.1 Risks per category

Communication

- Multiple team members are working on the same task, leading to overlapping work.
- A team member does not finish a crucial part on time and does not let the rest of the team know.
- The team's vision of the product does not align with the clients vision.
- Team members get into a quarrel, either with each other or with the client.
- The client does not communicate well.
- Too much time spent on stakeholder engagement, leaving insufficient development time.
- Inadequate preparation by team management for meetings, increasing communication time.

Stakeholders

- The client loses interest in the project and stops being involved.
- A team member with specific knowledge gets sick for a long time.
- A team member drops out.
- The team runs out of time before having a viable product.
- A team member loses interest in the project, negatively impacting their contribution.
- The team does not possess the knowledge to complete the project.
- Additional, initially unplanned activities have to be done.
- Working with sensitive data means that patients need to agree to being recorded, patients might not agree.
- A company might release a similar product, making ours uninteresting to our client.
- Product increases or does not decrease administration time for medical professionals.
- Product costs more money than it saves.
- Medical professionals do not like using the product.

Data

- The data used to develop and test the product with differs a lot from actual doctor-patient interactions.
- Recorded data is too noisy to convert to an accurate transcription.

Domain-specific knowledge

- The system's transcribed EPD entries do not interface well with actual standards in use.
- The system's text analysis is not capable of dealing with medical terms and other business standards.
- New domain-specific knowledge leading to previous work becoming deprecated.

Technology

- Speech to text is not accurate enough.
- Speech to EPD process takes too long performance-wise.
- Architecture lacks flexibility, is not right for the product, or is unfeasible.
- Subpar text to EPD resulting in poor quality EPD entry.
- Technical component is not right for the product, standard, or a best practice.
- The product has poor multimodality.
- The product has security flaws.
- The product is not extensible.
- The product is not user friendly.

6.4.2 Impact probability matrix

The impact probability matrix shows the probability and impact of each of the risks listed above. Each value is on a scale of −10 to 10, where −10 represents a `Low` probability/impact and 10 represents a `High` probability/impact.

The impact probability matrix can be seen in Figure 6.1, each of the risks is labeled at their respective places on the matrix, where the center of the label represents their values.

Figure 6.1: Impact Probability Matrix

6.4.3 Risk plan: most to least important

Inaccurate transcription As of the year of this project, 2018, speech-to-text services still make many mistakes. These mistakes will need to be dealt with as best as possible to make the program usable. The amount of mistakes the speech to text makes will be minimised by:

1. Using a custom database with Dutch medical terms;
2. Processing the audio of the recordings and;
3. Processing the text files generated by the speech to text program.

Poor multimodality For the client, the multimodality of the product is one of the most important features. Each of the different components should work together fluently. If this feat is not realised, the concept is not proven.

Poor text analysis It is key for the product to succeed, that the right information from an interaction is taken and used to create an entry in the EPD. The risk of having EPD entries of low quality will be minimised by using text analysis technologies and by improving the analysis through each iteration.

Architecture non-extensible The architecture of the product should be extensible, especially since its main purpose is to be a proof-of-concept. Avoidance of a non-extensible architecture will be achieved by firstly using proven methods within the field of architecture and secondly by continuously consulting with our client, who is an expert in the field of software architecture.

The product has one or more poor technical components The product should not use any components that are not up to par. For example, components with known security flaws, components of low quality, components that lack standard interfaces, etc. This will be avoided by using proven technologies when developing the product.

Product vision of client and team not aligned The product the team develops should be (close to) what the client is expecting it to be. The risk of this not happening, will be avoided by having frequent communication with the client and by showing the client prototypes every two weeks, on which they can give feedback. Thereby, the team knows what direction to pursue.

Poor performance An EPD entry can not be generated instantly and time will be needed to process the streams of raw data into an EPD entry. This time needs to be as short as possible, so that medical professionals can access the resulting EPD entry as quickly as possible.

It should also be possible for EPDs to be improved after changes have been made to the EPD generator. To realise this when dealing with a large amount of EPDs the generation speeds needs to be quick enough. Avoidance of long generation times will be realised by working with proven technologies (e.g. Google's Cloud speech-to-text).

Security flaws Medical data is classified as most sensitive by privacy laws, which means that this data should be treated with utmost discreteness. Avoidance of the chance of this happening can be realised by working with proven security technologies. Security is not a priority, due to the fact that we are developing a proof-of-concept. However, it should not be neglected.

Sick team member If a team member becomes sick for a longer period of time, the story they were working on might not be finished in time. This risk will be minimised by using version control software and having a code standard so that anyone's work can be continued by other developers within the team.

Unplanned activities come up Unplanned and important activities might arise, preventing the team of getting its work done within a sprint or at all. This risk will be minimised by making sure the planning within a sprint is not too tight. If there is no extra time left, it will be minimised by using time that would have otherwise been put into less important stories.

6.5 Relation with lectures

During lectures, it has been mentioned that correctly assessing risk is difficult. Estimates were made based on our current knowledge, but we fully realise that these might not be correct. We have, however, ascertained the risks that we feel require the most attention and have come up with several approaches of dealing with them. Where these approaches are in the form of either avoidance, minimisation, contingency plans, or tolerance.

One of the important lessons from the lectures was that we should not only plan for technical risks. Therefore, we have tried to estimate the most important non-technical risks within our matrix as well. Ascertainment and assessment of the risks were done by having discussions within the team. Risks were adjusted upon receiving an update on the product vision from our client.

The prime difficulty that is within our control, and for which we will need additional knowledge, is the text analysis. There is a lot of freedom on how to write this part of the program. Therefore, knowledge on how to make the right decisions regarding this component will be crucial. The speech to text component will be done by using Google's speech-2-text software, which leads to little to no control over the quality of the transcripts of the conversations. The architecture is another important risk, but the knowledge in this field seems up to par within our team.

Chapter 7

Test plan

7.1 Top-level overview

The system has a complex architecture, which is mainly due to the fact that multiple microservices need to be implemented. This calls for a testing focus on the architecture of our system. However, this does not mean that other elements will be disregarded since parts of the system are complex in their own right. This complexity calls for straightforward functional and non-functional testing strategies. To streamline the entire testing process, continuous integration as well as tooling and test automation will be used.

7.1.1 Quality aspects

To further specify the requirements of our system some quality aspects will be defined in this section.

ISO 25010

This norm describes quality aspects of software in a generic sense. It separates the models *Product quality* and *Quality in use*, with a total of 31 sub-characteristics divided between the two. The ISO 25010-norm will be followed during both development and testing. [ISO/IEC JTC 1/SC 7, 2011]

System-specific aspects

While most aspects are covered by the ISO 25010-norm, there are some aspects that should be prioritised in our project. These are: a flexible architecture, maintainability and the possibility to exchange parts of the system.

7.2 QA strategy

The QA strategy describes the functional and non-functional testing as well as the automation of testing and the use of continuous integration.

7.2.1 Functional testing

Functional testing consists of the strategies of unit, integration and system testing as well as the system-level testability. The different means of testing will be performed respectively in chronological order.

Unit testing

Two possible strategies will be compared with regards to unit testing: *code first, test second* and *test-driven development*. As the names indicate, the difference is in when the tests are implemented.

Code first, test second The most intuitive unit testing strategy that comes to mind is to first write the code that needs to be implemented and only then write tests for it. The intuitive nature of this approach means that it is easy to implement this strategy for developers within the team who might be inexperienced in the field of testing.

Unfortunately, this strategy requires more time to be invested in refactoring and rewriting code when it turns out that testing is difficult with the implemented code. Discipline and good design choices could, in theory, remedy this. If this strategy is chosen, a requirement would be that the *test second* part of this strategy would have to be carried out during the same sprint. [Clifton, 2014]

Test-driven development This strategy is based on the idea of writing tests before the actual code. Firstly, dependencies on other objects with no implementation will be implemented with empty interfaces and missing properties will be implemented with stubs. Using these stubs and interfaces tests can be written more easily with the use of *IntelliSense*. The stubs and interfaces also allow for the code to compile and for all names to be synchronised.

At this stage the written tests will be run and they should all fail since the referred code has not been implemented yet. Finally, the actual code that needs to be implemented can be written. Running the same tests again should now result in all of them succeeding.

This strategy requires the developer to think of what the code should do **before** writing the actual code. This instinctively leads to better design choices and cleaner code. A downside to using this strategy is that developers within the team who might be inexperienced in the field of testing would need to invest a significant amount of time to become familiar with this method of coding. Thus, while this strategy would likely lead to better code, it would also cost a considerable amount of time. [Clifton, 2014]

Conclusion To conclude, *test-driven development* would result in better code at the cost of more time while *code first, test second* would result in less clean code but would be easier to implement. However, if the time cost of refactoring and rewriting code are taken into account, then *test-driven development* comes out ahead. Therefore there has been chosen for a *test-driven development* strategy.

Integration testing

During this stage combinations of tested units will be tested. This is done to expose faults within the integration of these units. Integration testing is particularly important in this project on account of the many microservices that are contained within the architecture. [Software Testing Fundamentals, ndc]

Strategies

- **Big Bang**: All components are integrated together in **one** system and then tested. This approach is unable to determine the location of a problem and will therefore not be considered.
- **Incremental**: Instead of integrating every component together at once, each level of dependency is iteratively integrated.
  - **Top Down**: Start testing in the *super-component* that combines every component, with the use of stubs. Then go down level-by-level, repeating the process of using stubs, until each component has been tested in combination with its dependencies or other related components.
  - **Bottom Up**: The exact opposite of Top Down. Start by testing the leaf nodes (when viewing the system as a tree) and iteratively move up level-by-level.

Conclusion It quickly became clear that the Big Bang strategy was not going to be of much use. An incremental approach, however, does seem suitable. This leaves the choice between *Top Down* or *Bottom Up*. Of these two, the *Bottom Up* strategy seems the more intuitive choice, yet the combination of *Test-driven development* and the *Top Down* strategy is more natural. Therefore, the *Top Down* strategy will be used for integration testing.

System testing

System testing will be done after integration testing has been successful. This means that this stage of testing will be performed on a complete and integrated system, with the goal of evaluating the system's compatibility with the requirements. The use of system testing with regards to our project is limited by the fact that the final product is a prototype/proof-of-concept. [Guru99, ndc]

Acceptance testing

Acceptance testing is the fourth and final stage of testing. This stage of testing tests whether the system is acceptable for delivery with regards to the system's compliance to the **business** requirements.

This will be implemented by having two-weekly meetings with the client in which we will demo the current product. The client will then be able to give feedback and if necessary adjust the course of the project. [Software Testing Fundamentals, nda]

Testability

System-level testability will be ensured by following good design choices, which should lead to strong cohesion, loose coupling and encapsulation. Another means of ensuring system-level testability is to guarantee that the code is understandable (mostly by use of documentation) and suitable, which means that the system follows its expected behaviour. All of these will improve testability by decreasing the amount of tests needed. Testability is also improved by the unit testing strategy of Test-driven development as this strategy enforces good design at a low level. [Rodriguez et al., 2014]

7.2.2 Other QA instruments

Since the .NET stack is being used primarily in this project, the many tools that the Visual Studio IDE provides can be utilised. These consist of linting, testing and debugging tools, each of which are extremely useful towards ensuring a high code quality.

7.2.3 Continuous integration

Continuous integration (CI) is a practice where developers of a team intermittently merge their code changes into a shared repository, in the case of this project hosted on a GitLab server. The aim of continuous integration is to prevent integration problems. [Codeship, nd]

The CI strategy of this project is that developers branch from the development branch when working on code edits. When this edit is complete (feature implemented, bug fixed, etc.) the developer creates a merge request to the development branch. If the merge request is approved the branch is merged and deleted. At the end of every sprint the development branch is merged into the master branch for the minimum viable product increment.

One element of CI is Continuous Testing. The GitLab built-in CI/CD functionality can be used for automated testing, with our private runners and the university's shared runner that is available. [GitLab, nd]

The private runners can be configured on Docker. A big advantage of using Docker to run tests is that images can be defined to be used per job, for example the `microsoft/dotnet` image can be used for building and testing the front-end, and a `python` image can be used for building and testing Python code that is part of the back-end.

7.2.4 Tooling and test automation

Test automation is the use of software tools to control the execution of tests and to compare the actual behaviour with the preferred behaviour of a program.

Visual Studio Unit Tests can be used for the front-end. For parts of the back-end code that are written in Python, testing tools like `nose` can be used. Nose includes the `nosetests` script that can run easy to write python tests and has options to generate a coverage report.

Adding other tools if needed is also possible. All tools used, however, should be compatible with GitLab CI, which is used to automate the running of tests.

7.2.5 Non-functional testing

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or security. Since this project's goal is to build a prototype, some parameters will not be tested thoroughly for the reason that these parameters will not be important to the scope of this project. [Guru99, ndb] There will, however, be focus on the following parameters:

- **Usability testing**
  Measures ease-of-use of the system.

- **Documentation testing**
  Determines whether our documentation matches the functionality of the system.
- **Performance testing**
  Regards the performance of the system as a whole.
- **Recovery testing**
  Determines whether the system can deal with being down.

7.3 Management aspect

7.3.1 QA Milestones

There is no milestone specifically for quality assurance. Instead, every milestone contains quality assurance aspects. As the conclusion of the section QA strategy states, there has been chosen for Test-driven development. This means that tests have to be written for every piece of code before the code itself has been finalised. Every separate part of the project will be done in the same order: create the needed interfaces and stubs, then write tests for these new items where needed and finally fill in the actual code.

This plan both rewards careful planning of the system architecture and lessens the overhead. The time put into the system architecture will be doubly rewarded and there will be fewer surprises that could potentially cost a lot of time.

The project GANTT can be viewed in Figure D.1 below.

7.3.2 Effort/person-month

On average it is expected that 1.2 PM will be used on quality assurance. Every major aspect of the system will have a dedicated group of members working on it, each member of the group will also be responsible for writing the needed tests. The verification of their code will be done by another group member and the tests will also be verified to make sure the work has been done properly. One member has been chosen for the system testing. The average member will likely spend less time than 1.2 PM on testing while the person responsible for system testing will likely spend much more time on it.

Part III

Reflection

Chapter 8

Planning

8.1 Sprint progression

8.1.1 Sprint 1

Initial sprint goal

Make a complete list of requirements with accompanying user stories and create the product vision.

Progress made

This sprint wasn't an official sprint yet and mainly focused on understanding what the client expected from us and making the initial design. Most time was spent on research into subjects related to the end result. The client only had a rough idea about the end result and left much of its implementation up to the imagination of the project team. Sprint one was used for crafting a clearer vision of the end result of the project, doing research into the field of medicine and team building.

The first version of the milestones were crafted together with the client, the accompanying user stories were written afterwards. Multiple decisions that were necessary for starting up were also made. Google Cloud's Speech-to-Text was chosen as the speech to text algorithm this project would use. C# was chosen as the primary programming language with as end goal a developed application for windows tablets.

Topics as privacy, patient rights, different kinds of analysis on human actions and security were also researched. During the start of the project anything could be included into the scope of this project so decisions needed to be made quickly to make the scope smaller. Privacy was abandoned since using the best speech to choice engine had a higher priority and Google Cloud's Speech-To-Text makes no promises of privacy. Because this will become an issue in the future the Google Cloud's Speech-To-Text will be easily replaceable for another speech to text that will have privacy guarantees. This project was going to focus on a prototype for a speech to EPD system so anything that involves a real world application will most likely be implemented much later.

Sprint evaluation

As the initial sprint goal suggests, very little code was written in this sprint. This was overall a very productive sprint for which the sprint goal was realistic and completed in its totality. An initial, but not too significant, start was made on work of the next sprint.

The client approved of the milestones but they mentioned that the milestones should be done in parallel where possible and not in a linear fashion. The milestones were changed accordingly and now multiple milestones could be worked on at the same time.

8.1.2 Sprint 2

Initial sprint goal

Create a skeleton application based on the architecture created with the client. The application will contain little to no actual functionality, but adding this in future sprints should be easy. The architec-

ture should be as extensible as possible for the next team and the speech-to-text algorithm should be replaceable.

Progress made

The end goal of finishing the initial skeleton application was not completed in time. Access to the Google speech to text algorithm was delayed and progress on the system architecture was slow. The frontend skeleton application had been completed, but the backend was not finished which meant that the initial connection was not possible.

Sprint evaluation

In retrospect, the client meeting concerning the architecture was planned too late which made it hard to achieve significant progress on individual analysers. Any work without the proper architecture was likely to be mostly a waste of effort until a proper plan was made. This, combined with the fact that little was known about the Google speech-to-text output quality, slowed progress down considerably.

The client left most of the architectural design up to the project team which meant that the architecture could have been finished earlier. With the potential risk of having to do work twice, the decision to postpone the implementation of the design was still correct. Together with the lack of access to the speech-to-text algorithm this meant that much work on the backend was pushed back. Instead of this, more research was done into possible inclusions for the project and more time was spent on preparing for the client meeting about the architecture.

8.1.3 Sprint 3

Initial sprint goal

The initial pipeline from client device to generated EPD report needs to be complete. The first domotics device will be added to the frontend and the first sketches of the eventual EPD will be made.

Progress made

The analysers on the backend will work together with Docker to enable analysers to be written in different programming languages. A new architecture has been thought up called the medical consultation time line, which will be refined into the session time line (STL) in future sprints. This new architecture required the team to refactor a large part of the code but made future additions to the system much easier.

A little progress has been made on the domotics device, but due to the architectural rework and lack of the actual domotics device little progress in this area was made. Some time was spent on researching the potential implementation of video analysis in future sprints. While one subteam was implementing the new architecture another subteam was drafting the first functional design for the program.

Sprint evaluation

The ultimate sprint goal, a complete pipeline from frontend to backend, has been completed. With the new architectural design much will have to change, potentially breaking the working pipeline. Because of these drastic changes, communication within the project team was lacking. Implementing the new architecture will take a significant amount of time but the result will be worth it. Instead of making a fixed chain of analysers, an analyser controller will be used. This controller will guarantee that analysers are executed in the correct and most optimal order according to their dependencies.

In the previous sprint a great deal of progress was pushed back because of the incomplete architecture. Because of this the choice was quickly made to reduce the time spent on other subjects and to first complete the architecture. This resulted in little time being spent on the domotics device and only superficial research being done into the layout of an EPD report. The client promised to provide a domotics device and some EPD examples, so having spent little time on these matters during sprint 3 will have little effect on the eventual result.

8.1.4 Sprint 4

Initial sprint goal

Finish the first version of the report generator and include data of one domotics device. The STL needs to be visualised and the backend should be finished.

Progress made

The first version of the report generator has been finished, so in this aspect the sprint was successful. There are however some things that are still lacking and some other things that have been sacrificed to make progress on the initial report generator. The connection between the different backend components is one of them. Even though the report could be generated, this was through a hard coded and temporary connection.

A very good visualisation of the STL has been drafted and the client was happy with it. Good progress was made for the backend but because the team lacked a server to run everything on some parts were impossible to finish and most code was not tested.

In this sprint the decision was made to abandon the addition of a domotics device for the foreseeable future and potentially not include it in the project at all. The combination of limited resources and the client having difficulties procuring such a device for the team made this an easy decision.

Sprint evaluation

In previous sprints groups of team members were working together, but to make communication smoother subteams were created. These teams had a minimum of two people in them and each subteam member was equally responsible for completing the tasks assigned to the subteam. This also meant that tasks were assigned to subteams and it will then be their responsibility to make subteam members start and finish tasks in time. Subteams had some trouble starting up, but they were forced to hold a weekly meeting and appoint a subteam leader which made the process a lot smoother.

On request of the client, one subteam drafted a functional design according to the RE4SA principle of the entire system. Even though this functional design looked to have the potential to make progress easier, it had very little effect. The resulting functional design was useful, but because nobody in the team had any experience with this subject, progress was difficult. A functional design is necessary and time should be spent on it, but the RE4SA principle was not the correct choice for this project.

8.1.5 Sprint 5

Initial sprint goal

The pipeline, starting with the recording of a conversation and ending with a displayed report, has been completed and is testable.

Progress made

The sprint goal was way too large and work was not even close to being done at the end of the sprint. Two subteams were not able to finish their assigned work in time resulting in other subteams being delayed because of this. Both of these subteams underestimated their assignments which resulted in them being unable to finish everything, even though they did work for the allotted time.

The initial goal of a system that is complete enough to demo and test has been reduced to a system that works from a command line and is largely untested. This system did include some text processing, which resulted in a summary that could be printed in the command line. This improvised use of the command line made it possible to show the progress to the client, who was more than satisfied. All of the work that has not been completed is absolutely necessary for the end result, so all of it is pushed to the next sprint. Testing over the complete system has even been postponed until sprint 7 in the new planning. This will be made possible by severely cutting the time spent on video analysis.

The connection between the different analysers in the backend as well as the connection between backend and frontend are both still incomplete. The analysers themselves also still need to be formatted properly into the new, and constantly updating, system architecture.

Sprint evaluation

After the sprint meeting all team members agreed on the plan for sprint 5 but non of the subteams managed to finish their tasks. Because sprint 7 and 8 were filled with only optional tasks, one horrible sprint will not result in a failed end product. Two subteams were expected to finish their work early, after which the subteam members would switch subteams. These teams not finishing their work meant that there was no way the other teams would finish either.

Despite the bad overall result the progress made within subteams was good. The subteams have become much more efficient at working together and this will only improve in the future. Because each subteam was pushed to their limits, the maximum amount of work each team can do in a sprint has become much clearer. This will make assigning work to each subteam much easier in the future.

8.1.6 Sprint 6

Initial sprint goal

Rudimentary object recognition is possible and included into the report. The first version of a SOEP report should be able to be generated and a demo to show this needs to be finished through a GUI. This means that the connection both within the backend and between the front- and backend is complete.

Progress made

The sprint goal has not been reached, this time however, due to external factors. Almost all of the separate parts were finished, but the connection between them was incomplete at best. Because of the temporary command line solution that was used for the demo during the previous sprint, lots of things could still be tested together, but a decent demo was impossible.

A final decision has been made concerning the STL and after finishing this version no more significant changes will be made to it. The choice has been made to ensure that this version of the architecture will be completed and no further overhauls to it will be allowed. Any overhaul after this point will negatively affect the end result. Improvements on the current architecture are however always necessary.

Sprint evaluation

The subteams made as much progress as they could under their circumstances. Clear decisions have to be made about time management from this point on though. These two sprints have made new ideas very risky, so the focus until the system is complete will be on refining and testing current features. This will most likely take the entire duration of both sprint 7 and 8, leaving only sprint 9 in which a significant amount of time should be reserved to write the end report.

8.1.7 Sprint 7

Initial sprint goal

The system should be able to generate a first version of a SOEP report and a demo to show this needs to be finished through the GUI. This means that the connection both in the backend and between the front- and backend is complete.

Progress made

The connection has finally been competed resulting in the first proper demo. The STL was finished at the start of the sprint and refined where necessary according to the requirements of the other subteams. This resulted in all connections being made and the complete of the entire pipeline. Additionally, preparations were made for the coming two weeks of vacation, so team members could still continue to work on the project.

Sprint evaluation

Because of the significant progress that has been made during this sprint towards the sprint goal, a completely new item has been added for the first time in 3 sprints: a start has been made on logistical regression. After talking to an expert on logistical regression as well as consulting the client, this was chosen as the technique to implement for the SOEP classification. Logistical regression both requires

little test data and is able to reach significant results before the final version of the project needs to be done.

8.1.8 Sprint 8

Initial sprint goal

Implement the first version of the logistical regression model and refine current features.

Progress made

A very basic logistical regression model has been made, which includes 4 features. At least 10, but most likely more features, will be included in the future, which will improve the results significantly. This model currently works with a little over 500 sentences. This amount will double in sprint 9.

Sprint evaluation

A good amount of progress has been made and the completion of the first version of logistical regression in sprint 8 is impressive. The model is very basic however and still requires a significant amount of improvements. All of the remaining features necessary for the logistical regression model will be completed during sprint 9.

8.1.9 Sprint 9

Initial sprint goal

An advanced logistical regression model with significant results is completed. The resulting model is displayed in the GUI on the client side after an audio file has been sent to the server.

Progress made

This was the last sprint so it was not only important to complete this sprint goal but enough time needed to be left for completing the final product. With the classification being just about finished last sprint the only changes were adding new features to the model. This has been done and with these new features the model has improved significantly. The model is still far from perfect but with the limited resources available better results are going to be hard to reach.

The GUI on the client side was already there so the only changes were to change the GUI to be able to display the new report type. This was also done successfully without trouble leaving enough time to incorporate the improvements suggested by the client.

The rest of the sprint has been used to write documentation and comments for the existing code. A lot of time has also been spent on completing the testing for the final product. The integration testing has also finally been completed enabling the tests for the compatibility of different analysers.

Sprint evaluation

Overall a very successful sprint, the planning of this sprint has left a lot of free time for the team members to improve upon existing code. Not only was the sprint goal easily reached leaving a lot of time but the free time was spent well. With the amount of testing done on the current product extending the code will be a lot easier.

Chapter 9

Architecture, design and technical choices

The final architecture was reached through many iterations of design and implementation. These iterations will be laid out in the following paragraphs.

9.1 Frontend

There was no initial design for a frontend architecture, as the focus was very much on the backend architecture. When work on the frontend had started, it was decided to have a structure of models, controllers and pages. The final architecture of the frontend is described in Section 2.2.2. This architecture came together in multiple iterations, which are described in this section.

The very first version of the frontend merely consisted of pages without extra classes. All the necessary logic behind the controls in the pages was implemented in their own classes. This was done on purpose, because the controls added at that point in time were only added to suggest what would be possible. Once a better understanding was established between what the client wanted and what was possible, a controller (the device controller) was created to handle the logic behind multiple pages.

Throughout the development, the architecture was expanded as needed. A server interface and multiple models were added, as well as a recording controller. At this point in time, there was not yet a Session model and Reports were standalone. After consulting with the client, this was changed and Reports became part of a Session. Recording would only be possible when a Session was selected. By following this approach, the care provider would not have to scroll through an ever increasing list of files to add to a Report, but only files part of the Session the Report belongs to would be shown.

9.2 Backend

The initial design of the backend architecture was, as described in the initial plan, designed towards the necessity of interchangeable components of the system. This resulted in a design with some large, overarching analysers (e.g. Text analysis, Video Analysis, etc), that each communicated with a controller. These were supposed to serve as interfaces to analysing software, which could optionally be created by the team.

This original backend architecture was completely overhauled a few times. Each time, this was due to problems that arose during implementation of the design, which made us come to new realisations. The first of these overhauls was instigated by the problem of dependencies between these analysers. To combat this, an architecture consisting of multiple Micro Analysers without dependencies on each other was devised. This transition would change the original architecture to a Micro Analyser architecture, as described previously. This new architecture added to the flexibility of the architecture by the inherent nature of such a design.

This new design included a Micro Analyser Controller and a datastructure that was able to encompass all of the input data. Firstly, this controller improved the system by automating the ordering of all the Micro Analysers. The new datastructure (named Session Timeline or STL) allowed for saving each piece of data as a Timeline Event with an id and a start- and endtime. With standardisation and consistency in mind, a superclass was created in C# that could be inherited by Micro Analysers. This class would

provide the developer with all necessary functionality from the backend. During implementation, the backend team consistently found new ways of extending these ideas, which led to more functionality, but also had the consequence of the system becoming more complicated.

This complication instigated the next large change, after a team member noted that this complication would lead to difficulties in the departments of connection and ease-of-use for (future) developers. Another consequence of this complication was the difficulty of communication of the system with different languages than C#. Some of the functionality was therefore moved, so that developers would not have to deal with it. This led to, among other things, a more simplified Timeline Event which improved the connection between analysers and the system. As a means of improving the ease-of-use of the system for developers, an API to the backend was introduced. This consisted of a collection of functions that allowed for clear and intuitive interaction with the backend, such as getter and setter functions of the STL. Apart from a few small changes or improvements, this would remain as the final backend architecture for the rest of the duration of the project.

9.3 Connection

The initial plan was to base our backend on the micro service architecture. We chose gRPC as inter process communication standard and decided on the isolation of our analysers in docker containers during runtime. Of these two choices, the docker containers were ultimately not concretely implemented. The possibility to do this is, however, still present in the system. Since every analyser runs its own process, switching to a dockerized system is still a possibility for future iterations. The reason for not implementing the docker architecture concretely was two fold. Firstly, the architecture was still in flux for a large portion of development. Some early attempts were made to set up every analyser with a docker container, but because of architectural changes this had to be obsoleted. The second consideration is more practical; development ease and iteration speed suffers from a dockerized set-up. We still believe that keeping the possibility of loosely coupled dockerized infrastructure in mind during development was a worthwhile investment and hopefully improves extensibility moving forward.

There was some friction within the team about the role of inter process communication in our architecture. In the end, however, it was decided that our choice for this inter process communication solution should be leveraged to allow many different kinds of implementations of analysers on the backend; effectively enabling developers to write an analyser in whatever language is appropriate for the task. This meant that a the gRPC interface became an explicit part of our architecture, instead of an implementation detail. In the end, the choice of gRPC was satisfactory, since it offers a robust, battle tested, cross language solution. There were no significant difficulties with the solution or the python/C# bindings we used. The only rough edge was the need of writing bash/bat scripts to automate code generation, but this went quite swimmingly in the end.

9.4 Transcription

It was discovered quickly that Google's cloud speech-to-text does not provide speaker diarisation for the Dutch language. Another tool that does this was also not found. In a later stage, it was also discovered that the quality of Google's Cloud speech-to-text service was not as good as expected, despite being one of the best in the industry. These discoveries proved to be significant problems that needed to be solved.

To solve the lack of speaker diarisation problem, the way of recording a conversation was changed. Conversations were recorded using two different microphones: one to record the voice of the care provider and one to record the voice of the patient. Both of the recordings are sent to the transcription service separately, which ensures identification of distinct speakers. Unfortunately, due to the quality of the microphones used, the other person could still be heard on both recordings. However, this problem was eliminated with some preprocessing of the audio. As another means of improving the transcriptions, both parties spoke slowly and articulated very well during the recordings. This measure had to be taken due to the transcription still being of insufficient quality. By applying these solutions, the different speakers were distinguishable and a transcription of sufficient quality was obtained.

It should be noted that the means of obtaining this sufficient transcription is not desirable. Manual audio preprocessing, as well as using an unnatural way of speaking during recording sessions, were necessary to reach this result. It is highly probable that if high quality microphones, along with some automated preprocessing, were used, that the quality of the transcriptions would improve drastically.

As it stands, it can be stated that the quality of transcriptions is likely not high enough to be able to successfully run the system.

Realistically, it can be said that the technology to run the system successfully is not available. This statement was reached by concluding that a sufficient output of Google's speech-to-text service could only be reached by unnaturally modifying the input audio. This has been discussed with the client and the decision was made to ensure that the audio should be of excellent quality, even if this meant the use of unnaturally spoken language and manual audio preprocessing. The motive behind this decision was the scope of the project, considering it is a proof-of-concept. It is likely that the technology will significantly improve to the point where real conversations can be used in the near future.

The problem of insufficient transcriptions, which could effectively ruin the output of the entire system, has been solved quite successfully. However, some mistakes were made during the process of reaching this result, which could in hindsight have been avoided. The largest of these mistakes would be the untimely acquisition of test data. By the time the system was being tested with real data, a significant part of the system had already been built. At this time, it was discovered that the transcriptions that were obtained from the Google speech-to-text service were of insufficient quality. This caused a great deal of difficulties, since this problem had to be solved within a short time frame. Testing of the system with real data should have started much sooner.

9.5 Text analysis

To improve the results of the features, preprocessing steps were performed on the words and sentences. These steps either improve analysis or directly improve the resulting summaries. The original goal was to implement question recognition. After researching this it was found that speech act classification should be done. To do this other preprocessing steps had to be preformed like part-of-speech tagging (POS tagging).

For POS tagging and chunking the Apache OpenNLP library was used at first, but it turned out not to be compatible with the system because this library does not support .NET core. The possibility of connecting a program built in .NET framework with a program built in .NET core was then looked into, this is however not possible. By looking into other libraries, a java library of Apache OpenNLP was found. POS tagging and chunking were then implemented in Java. That meant that an analyser base for Java was needed for these analysers to be able to connect with the system. Building an analyser base in Java would however have been a significant amount of work. It was therefore decided the analysers would only be written in two programming languages: C# and Python. It was difficult to find a library that both supported the Dutch language and was written in either C# or Python. After some more research, one was found: Frog.[1] It was quite difficult to install Frog at first, but after overcoming these and applying some workarounds Frog was successfully installed. Finally, POS tagging, chunking and lemmatizer analysers were built in Python with the help of Frog library and the Python analyser base.

No existing libraries where found to perform speech act on Dutch sentences. Only one model was found for speech act classification for the English language. This model was in the programming language Clojure. The way this model was build was documented well and was under public license. The training data and method used in the article were used to build a train a model in R that could be used. There were various issues with the model. It is based on English language, it has only five features and more could not be added. It was decided to use this model anyway, because their was no priority and time to create and train our own model on the Dutch language. To improve the results this would have been useful.

9.6 Video analysis

The video analysis features currently available in the system are not as extensive as initially planned. After researching action and object recognition technology, it was found that action recognition would not be possible to implement within the scope of this project due to insufficient technical resources. Furthermore, it was found that the impact of object recognition on the overall analysis process would be very limited. The reasons for this being: the currently available hardware for the project and the unavailability of data sets containing medical objects.

This led to the decision to drop action recognition and to only implement a proof-of-concept of object recognition in a way that can be extended and improved upon by future teams.

[1] `https://languagemachines.github.io/frog/`

A different method of recognising whether certain medical objects had been used during a conversation was however developed: object recognition using QR codes. For this, QR codes first have to be attached to medical objects. Next, the QR code detector analyser scans all QR codes within every frame of the video of the conversation. Finally, the QR code movement analyser uses the positions of the detected QR codes to determine whether they have moved. Moved objects are then classified as having been used.

Because of the downsizing of video analysis, research was done into photo analysis. This research was done near the end of the project, which led to the conclusion that there was not enough time left to implement this.

9.7 Classification

The final step in the pipeline is the classification of sentences. The initial idea was to create features that modify the probability of a specific class for a specific sentence. These features would output a value that was representative of the influence of that feature on the probability of a certain class. The values were manually chosen, so there was no way to confirm the actually validity of the rules.

It was discovered that this approach to classification would result in excessive uncertainty about the outcome and validity of it. The choice was therefore made to train a Multinomial Logistic Regression model that calculates coefficients for the features, instead of manually choosing these. This did, however, require a large amount of training data. As mentioned earlier, the acquisition of data was was not done on time. This caused a problem, since training data is essential when building a machine learning model. A significant amount of time and resources were used to attain and preprocess the data that was necessary.

It was initially attempted to develop the learning module in C# (using ML.NET) and in Python. The reason for this was that modules built in these languages could be easily integrated in the system. However, the decision was later made to use R, because it proved a more extensive platform for machine learning problems and easier to use.

In total, there were 18 sessions of data recorded. Unfortunately, there turned out to be some issues with the data that were noticed after the model had been created. The training data was recorded in two sessions, with three different medicine students/medical professionals. On account of this, there were some significant differences between the two batches. The most obvious one is that the conversations in the first batch of sessions, which consisted of eight sessions, were significantly shorter than the conversations in the second batch. Because there were more individual sessions in the second batch of recordings, this resulted in the model performing better on longer conversation. This led to the following conclusion: attributes like length of sessions should be uniformly represented in training data. Another issue that was noticed is that correct classification of O and E was significantly less frequent than the classification of S and P. This could be due to two factors:

- Because of the nature of a SOEP report and the structure of a session with a general practitioner, O and E are overall less frequent in a SOEP report than S and P are. This, in combination with the low amount of training data that was available, could result in a lacking representation of these classes in the model; and
- It could also be the case that the features that were implemented for the creation of the model were overly focused on attributes of the S and P classes. This could be fixed by adding features that focus on attributes of the less frequent classes.

Chapter 10

Risks

10.1 Speech-to-text

This project is dependent on the accuracy of speech-to-text, a field in which significant developments still need to be made. Because of this, a large amount of extra preprocessing needs to be done to generate a transcription that can be analysed.

10.1.1 Inaccuracy

The most time consuming issue during this project was the fact that speech-to-text is fairly inaccurate; there are ways to counteract this but the output will never be perfect. The complete process consists of multiple steps: the words need to be spoken out and recorded, the audio needs to be run through a speech-to-text processor and finally the outputted text needs to be analysed. This means that this process has three areas that could be improved: the words can be recorded as clearly as possible, the accuracy of the speech-to-text process itself can be improved and its output can be processed to improve the result. The actual speech-to-text process falls outside the scope of this project, leaving two possibilities: improve the recording quality and improve the transcription after it has been generated.

Since the speech-to-text transcription of the audio of a normal conversation was of insufficient quality, the speakers on each recording were told to pronounce the words as clearly as possible. In addition to this the speakers also had to be very close to the microphone so that everything they said was recorded properly. Editing the end result of the speech-to-text is also possible, but initial attempts were largely unsuccessful.

10.1.2 Splitting sentences

In the output of the speech-to-text a great deal of sentences were stuck together. Both when the same speaker said multiple sentences after each other and when the second speaker answered the first one. If the same speaker says two sentences in a row it was not too much of an problem but it does become an issue when the analysers need to know which sentence was said by which speaker. Two techniques were proposed to deal with this problem:

- Using two separate microphones, where each speaker is only to be heard on one of them, this enables sentences to be sorted by speaker with 100% accuracy. However, this still leaves the problem of sentences of one speaker being pasted together.
- A learning algorithm that learns how normal sentences end can make educated guesses at where to split a sentence when it is unusually long.

In the end, only the first of these techniques was used due to time constraints.

10.1.3 Speaker diarisation

As mentioned in the previous section, it is almost impossible to accurately split two speakers when recording with a single microphone. Even when there is a significant difference in the loudness and pitch of two voices, it is not 100% accurate. Using two microphones is therefore the best and easiest solution to this.

This solution also ensures that there is no problem determining who the patient is and who the care provider is. If this was not the case another analyser, which would have recognised patterns or other vocal properties, would have been needed for this. By making the care provider use the same microphone every session this problem is solved, in this case the patient will always be using the other microphone.

10.2 Architecture

It was clear from the beginning, that the architecture would be a very important aspect of this project. Multiple other teams will develop this project further, which means that the architecture needs to be thought out properly and clearly described. Because of this, a significant portion of time has been spent on making the architecture as future proof as possible.

10.2.1 Dependencies between different analysers

Multiple analysers within this project need output from other analysers. This means that analysers need to be executed in the correct order and, preferably, also in the most efficient order. Because of this, the Micro Analyser Architecture has been developed. This architecture forces each analyser to declare its input and output before the entire analysis process is started and it makes sure all dependencies are properly adhered to. The Micro Analyser Controller creates a graph for the whole process, which ensures that the analysers are executed in an efficient order that minimises the time analysers have to wait on each other.

10.2.2 Modularity of the architecture

It is very important that future teams can add new analysers to the project as easily as possible. It is also very likely some analysers will be replaced or removed. The speech-to-text service used during this project is not private so when the project is used in practice this will need to be replaced.

10.3 Problems within the team

For most of the members of the team, it was the first time working in such a large group on the same project for such a long consecutive time. Each member had to learn that there were nine other developers working on the same project and adjust themselves accordingly. Generally, there were no real problems within the team.

10.3.1 Team member is not able to finish their work

There could be many reasons for a team member not being able to finish their work. The solution used during this project is to create small subteams for each separate part of the project; the minimum size of such a team is two.

If a subteam member was in need of help, another subteam member would be asked for help before asking for help from someone outside of the subteam. Verification of work was also done within a subteam. This means that, when one person finishes a task, another person within their subteam checks whether the work has been done properly. If for whatever reason a subteam member is not able to finish their assigned task in time, they were to communicate this properly within the subteam, so that another subteam member could ensure that the task is still finished in time.

There is always the risk that a team member does not communicate properly or outright lies about their progress. Making team members attend the daily stand-up meetings as often as possible, as well as having subteam members verify the work of other subteam members, were measures that were thought of to prevent this. In one case, a subteam was reduced to one person, which led to some problems that would have been prevented by these measures. In most of the other cases these measures worked excellently and when something wasn't finished it was because the assigned task spanned multiple sprints.

If a deadline still was not met, it was up to the scrum master to deal with the fallout this causes. This could mean leaving certain features out of the project and, if needed, reprimanding team members who delivered poor work.

10.4 Problems with the client

Communication between the client and the project team was very important. Proper agreements needed to be made to ensure that the client would receive an end result that they are satisfied with. The project team needs to communicate clearly what they can and what they cannot accomplish within the given time frame.

10.4.1 Lack of communication with the client

The client can lose interest in the project or simply be unavailable for a certain amount of time. In these cases the project group still needs to be able to work. The solution we used was to make a roadmap spanning from the start to the end of the project. Even though this problem was never an issue, the client was still happy with the roadmap. They gained a better understanding of what the project team thought themselves capable of finishing within the entire time frame.

10.4.2 The client has an impossible request

One team member was designated as a liaison between the client and the project group, the product owner. If the client had a request this team member would be responsible for communicating this to the other team members. If a request was impossible to finish within the scope of the project the liaison would have to inform the client.

Within this project, this was only necessary once, with regards to video analysis. The client was under the impression that the project team could just use existing techniques and make significant progress this way. This was, however, not the case. Not only are the current public techniques not able to gain any significant results, these techniques are very difficult to implement. The product owner explained this properly to the client and they understood this request was impossible very quickly. Instead of this, the project team proposed another idea, to use QR codes to recognise objects and measure any movements of these QR codes. Even though this was far from perfect, the client understood this was the best this team could do within the limited time frame.

Chapter 11

Process

During the first ever meeting of the project team, only a few decisions where made regarding the process. As part of the project it was set that the team had to use SCRUM as a basic foundation of the process. It was also decided that there would be bi-weekly sprint review meetings with the client on Thursdays followed by the bi-weekly sprint planning meeting on Friday afternoon.

11.1 Meetings

Thursday and Friday were chosen because the complete team was available on Friday afternoon. A number of team members were not available on other weekdays. The day of the sprint planning meeting never changed during the project. It was, however, discussed to move the meeting to Monday to have it as a start of the week. Some members of the team preferred Monday because it was not possible to start working after the meeting. It was never moved to Monday because not all team members, as well as the supervisor, would not be able to attend.

The client review was planned on Thursday so there would be time to prepare for the next sprint, using the feedback received from the client. The sprint review has in some instances been moved to Wednesday, because of unavailability of the client on Thursday.

11.2 Management roles

As a requirement for the project, it was mandatory to appoint a management team consisting of a chairperson, scrum master and a product owner. These were not yet set in the first meeting; only interest in these roles was polled. This was done due to limited time in the first meeting.

During the first sprint planning meeting the structure was further defined. The management roles were decided: Ruben was appointed as Chair, Mathijs as product owner and Mark and Cathy would be scrum masters. The idea behind having two scrum masters was that they would switch every two sprints. By doing this both Mark and Cathy could experience the role. However, in practice the role was never switched and Mark – who started as the scrum master – would be scrum master for the entire project. The benefit of having only one person as scrum master, was that the knowledge would stay with one person; disregarding the need to exchange information every two sprints. In a future team it would have been useful to immediately decide one scrum master, as having two merely creates confusion.

11.3 Reflection

The team had a less productive start during the first few sprints. Working in a team of ten people on a project like this was a new experience for everyone. Some time for adjusting and getting accustomed was needed. To make sure the process would improve, there was reflection on the past sprint every sprint planning meeting. By doing this, positive and negative aspects of the process could be pointed out and decisions could be made accordingly. This was especially beneficial during the first few sprints. Reflection would, at first, take up at least half the meeting, but by doing it the productivity of the team was quickly improved.

11.4 Management team

At the beginning of sprint 4, the management team took a bigger role in facilitating the process. It was decided that they would have a meeting every Wednesday to evaluate the process and the individual functioning of the management team members. This would lead to more clarity about their functioning and would allow the management team to come up with suggestions for improving the next sprint in order to have more time for the planning in the sprint planning meeting. This decision assured that the process in the following sprints would run smoothly. On a number of occasions, the management team could resolve or prevent issues before the impact was noticeable.

11.5 Subteams

The first suggestion the management team made, was about improving the division of work by introducing subteams. During the reflection of sprint 3, it was noted that the division of work was unclear. This unclarity resulted in merge conflicts and small conflicts between team members due to the reason that multiple people where working on the same features.

Subteams assured that tasks were clearly distributed and that subteam members had to internally discuss issues. This prevented any of the previously mentioned problems in the upcoming sprints. Because of the variety of work, a lot of different subteams had to be created, which in turn led to members being part of multiple teams. This worked in the context of this project, but in another project it might be useful to propose a clearer task division, with the goal of decreasing the amount of subteams. Another idea is to create subteams consisting of pairs of members, as one of the most efficient subteams of the project consisted of a pair.

11.6 SCRUM

The use of SCRUM was not always accurate. The way the team worked from the fourth until the seventh sprint was more similar to the Waterfall approach. The team still used SCRUM methods like daily stand ups, the scrum board, bi-weekly demo's etc. However, at the end of those sprints the product was not shippable. This was caused by the scale and ambition set at the beginning of those sprints. It became evident that it was impossible to achieve the shippable product in merely two weeks. Luckily, this did not cause any major problems. It was, however, not a desirable use of SCRUM. In a future project, it would be a better practice to set realistic sprint goals. This would lead to proper SCRUM iterations where a shippable product is realised after each sprint.

11.7 General impression

After the implementation of subteams and the weekly management team meetings, no major decisions where made related to the improvement of the process. After these final changes, the process ran quite smooth and the team would work effectively in most sprints that followed. Reflecting on the process after every sprint, meant that problems would be tackled on time. This bi-weekly reflection would be useful in a future team.

Chapter 12

Communication

The basis of communication within the team was defined during the first meeting. Communication at the office would consist of going up to team members and talking to them and daily stand-up meetings for team-wide updates. It was decided that a WhatsApp groupchat would be created for operational, day-to-day use. Finally, a Slack page would be created for project-related communication. Examples of such communication would be important changes within a subteam.

12.1 Team meetings

Meetings are an essential element towards good communication within a team. As for our team, meetings can be divided in daily stand-up meetings and sprint meetings.

During the daily stand-up meetings each attending team member would first describe what they had done since the previous stand-up and then what they plan to do until the next stand-up. After every team member had a chance to speak, someone (usually the chair) would ask if anyone was in need of help. This usually concluded the stand-up, but in some occasions this moment of team members being in the same room with each other would be used for small announcements. This structure of stand-up would mostly remain the same for the duration of the project. Some adjustments were made along the way, such as the updating of the scrum board after or during the meeting.

The daily stand-up meetings were, at first, held at 12:45 'o clock. The reason for this being, that everyone would be free at this time due to the lunch break. This was changed to 12:55 later on in the project, because some team members were having difficulties making the meetings if courses would last to 12:45.

Sprint meetings were held bi-weekly and communication-wise, these meetings were used for announcements. Furthermore, these meetings were also the chance for team members to notify the team if there would be any prolonged absence during the upcoming sprint.

12.2 Subteams

Communication within subteams has consistently been described as excellent. This could be attributed to subteam members being able to sit in the same room together due to these teams being small, as well as the use of the slack page would allow for efficient communication.

At some point, it was decided that subteams should have meetings to make decisions and share information. In practice, however, this was rarely necessary, since communication within a subteam was already sufficient for these points.

12.3 Scrum board

The scrum board was an important medium for communication during the course of the project. This allowed everyone to maintain an overview of the project during a sprint. For example, if a team member would run out of tasks, this member could take up a new task by examining the scrum board.

There have been a few cases where team members did not actively update the scrum board. These occurrences were directly noticeable by the rest of the team and after delivering feedback during the following sprint meeting, no similar problems occurred. In the final stages of the project, the scrum

board proved less useful due to most team members working on the deliverables. The progress in this department was monitored by a spreadsheet containing To-Dos.

12.4 Communicate! game

At two occasions, the team had to attend an hour long workshop, where each team member would play the so-called Communicate! game. At the first occasion, the game was played once, whereupon the supervisor would teach to us some communication tips. Then each member would play the game again, usually resulting in a better score.

Long term results with regards to communication were not noticeable within the team, but individually, team members did learn a thing or two.

Chapter 13

Testing

In the beginning stages of our project, testing was not important; there was no code written to test yet. Instead, a lot of research and writing of documentation had been done. The first matter at hand with regards to testing, was writing a testplan. The testplan consisted of our progress – how we test the backend (server side) and the frontend (client side) – and the challenges that the team faced. As a means of managing the testing, the management team decided to have a subteam for testing. This subteam had monthly report meetings with testing supervisor Wishnu Prasetya. The subteam kept Wishnu up to date with the progress of testing in the team by sending short updates on progress and challenges for unit, integration and system testing. The subteam also assured tests were written for every important piece of code. Lastly, there were two kinds of testing that had to be done: functional and non-functional testing.

13.1 Functional testing

In the case of functional testing, the first step that was taken was writing unit tests for each finished Micro Analyser (both in C# and Python). It was decided that the team member who had written the analyser, had to write tests for it as well. By that approach, the original writer could discover the flaws in their code, fix them and ultimately learn from them. The analyser was verified when it not only met the code standard, but when tests were written for it and these had passed. At first, untested code would be merged with the working code. In some cases, this caused the previously working code to not work anymore. Finding and then fixing the bug was time consuming. Thus, the lessons-learned here are to not postpone writing tests for your code and to not merge untested code with working code.

The next step that was taken, was implementing integration testing . To realise this, all the written unit tests were tested in combination with each other. At first, it was decided to use the Top Down strategy, but during implementation of integration tests, it was found that the Bottom Up strategy would prove a better fit for our system. Integration testing was implemented by creating a small program that runs all of the unit tests combined.

Next, system testing was implemented. During the implementation of these tests, a bug in the connection between frontend and backend was encountered and consecutively solved.

Lastly, exploratory testing was implemented with the goal of testing the frontend. This was chosen with respect to encouraging creativity towards problem solving. There was no time to implement automated/scripted testing for frontend, yet the frontend is still built in an automated manner using Gitlab CI. A Unit Test Project for Universal Windows applications exists for testing the fronted. At first, there was another test project with the goal of testing the UI. However, this test project ended up being scrapped, since it was not working and would take too much time to fix compared to the added value. A lesson-learned is to prioritise frontend testing in an earlier stage, since this aspect of testing caused the most problems.

13.2 Non-functional testing

We decided not to write security tests, since the system is only a prototype. This meant reaching actual results had a priority over ensuring security in the system.

To test usability, we decided to let non-developers work with our system and give us feedback. That way we could have a different view on our system and thereby find areas of interest that need some more attention.

Chapter 14

Conclusion

Initially, the `Speech2EPD` project started off as a highly ambitious plan to aid in revolutionising the administrative process for care providers. This initial plan can be briefly outlined as follows:

Administration should be automated through a combination of speech recognition, text analysis and video analysis technologies. This inherent multimodality has a high priority for the final product. As the project is a proof-of-concept, flexibility in design is a high priority as well. These characteristics should emerge through design and implementation of the system. It should be straightforward to add, remove or alter analysis processes to the system.

Although seeming daunting at first, a realisation of the initial plan has been reached. This realisation was reached through multiple stages of development.

The first of these stages consisted of research and design. During this stage, it became evident that speech-to-text technology is far from perfect in the present-day. This discovery meant that practical applications are not yet possible. Yet in the scope of this project, we were able to preprocess the audio in such a manner that usable transcriptions were attained. A first draft of the system architecture was designed during this stage as well. This architecture proved that the requested multimodality was attainable. After this first clear stage, followed multiple iterations of implementation, design and refactoring/rewriting.

Video analysis proved a difficult aspect to implement meaningfully. The added value of action recognition in particular would have been quite useful when combined with the other disciplines in the system. This, however, turned out to be impossible to implement within the scope of the project.

While generating training/test data, it was discovered that much information that is necessary to generate a complete report is not explicitly mentioned in actual, real-life conversations. This means that this implicit data would still have to be manually added to reports or that this data would have to somehow be generated from the conversation (which would be incredibly difficult).

To conclude, due to the current state of some technologies on which the system depends, a practical application seems improbable. The implemented Micro Analyser architecture, in combination with the Session Timeline datastructure, did prove capable of handling this situation, as well as being future-proof as it allows for simple extension. This means that when technology improves, this can be added in a simple manner.

Chapter 15

Discussion

Since this is such an ambitious project, there were many innovations that were not realised. These innovations would either be planned at the start or be conceived during the span of the project. Here follow some of these ideas:

Automatic machine learning The fact that every Micro Analyser instance has a fixed input and output facilitates the creation of a standard machine learning module that tests for correlations between input and output for every Micro Analyser over big sets of previously analysed data. This of course depends on whether it is allowed to store medical data by the machine learning module.

Analysis parameters Many analysis processes are dependent on parameters that were chosen by programmers, such as thresholds. Currently, these parameters are hardcoded in every analyser and if two analysers need the same result with different parameters, the only way to do this is to make two instances with different parameters. A better solution would be to allow parameterised timeline event so the same Micro Analyser instance can produce differently gauged results.

File server The STL and MAC guarantee that all data that is convertible to text is immutable. This means that, for example, file paths are immutable, but data that is referenced by such a file path is mutable. Once a Micro Analyser has the file it can change the file at its own discretion. To ensure immutability for linked file, a file server could be built where Micro Analysers can request files and push files but not override existing files.

Visualisation of alternative medical terms Every report has a list of medical words that are part of the summary with alternative medical terms (see Section 2.7.7). The purpose of this list was to show options to the care provider for the medical terms in the final report. However, as of now these are not shown anywhere, as there was not enough time to figure out the best way of showing them, while at the same time having the care provider be able to edit the summary of the report by typing. In the future, the options should somehow be shown to the care provider when editing a report.

Build new sentences In the current system, the report is generated using the classified sentences from the transcription. A report features differently structured sentences in comparison to sentences that occur in a conversation. To improve the system, new sentences should be built from the classified ones that would fit in a report. To realise this, the semantic meaning of the classified sentences should be extracted.

Classify groups of sentences As another means of improving the system, a way to classify groups of sentences in a conversation could be created. Currently, only sentences can be classified. This confines the classification to sentences that hold information on their own, but some sentences, such as questions and answers inherently need each other to represent information. Question-answer pairs frequently occur in conversations between a care provider and a patient, which indicates that an increase in quality of reports could be gained from this innovation.

Use background knowledge of patient To improve analysis, the system could use the previous medical records of the patient to predict the cause of the consultation. This prediction could be used to adjust the analysis of the session. It provides the system with more information at the start of the analysis.

Graph generation for specific medical domains Some medical domains would benefit from a specialised system, instead of the current, generic one. To implement this analysis procedure would have to change according to the type of complaint the patient describes. The system can then generate a new analysis graph that is specialised for this complaint.

Feedback of changes by care provider Another way to possibly improve the system, is to have the changes a care provider makes to a generated report sent back to the server and to use this data to train one or more analysers. By following this approach, the system would keep improving the more it is used, based on real information. In this case the privacy of the patients should be kept in mind when deciding on how and where to store the data. As of yet privacy has not been an issue, because of the systems status as a prototype.

Optimise system per individual Every care provider has their own mannerisms. As a care provider uses the system the analysis and classification could be trained to more effectively correspond with the mannerisms of the care provider. This would increase the accuracy of the system as it would be specialised for the care provider.

Bibliography

[Autoriteit Persoonsgegevens, 2018] Autoriteit Persoonsgegevens (2018). Autoriteit persoonsgegevens. https://autoriteitpersoonsgegevens.nl/.

[Banarescu et al., 2013] Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., and Schneider, N. (2013). Atssi: Abstractive text summarization using sentiment infusion. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

[Base24, 2014] Base24 (2014). Medische database. http://www.code24.nl/Medische-database.

[Bhargava et al., 2016] Bhargava, R., Sharma, Y., and Sharma, G. (2016). Atssi: Abstractive text summarization using sentiment infusion. *Procedia Computer Science*, 89:404–411.

[Bird et al., 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition.

[CBG, 2018] CBG (2018). Geneesmiddeleninformatiebank.

[Clifton, 2014] Clifton, M. (2014). Unit testing succintly: Strategies for unit tests. https://code.tutsplus.com/articles/unit-testing-succintly-strategies-for-unit-tests--cms-22414.

[CMUSPhinx, nd] CMUSPhinx (n.d.). Postprocessing framework – cmusphinx open source speech recognition. https://cmusphinx.github.io/wiki/postpframework/.

[Codeship, nd] Codeship (n.d.). Continuous integration: What is ci? testing, software & process tutorial. https://codeship.com/continuous-integration-essentials.

[Dasgupta, 2017] Dasgupta, P. B. (2017). Detection and analysis of human emotions through voice and speech pattern processing. *CoRR*, abs/1710.10198.

[Fellbaum, 2005] Fellbaum, C. (2005). Wordnet and wordnets. In *Encyclopedia of Language and Linguistics*, pages 665–670. Elsevier, Oxford, 2 edition.

[GigaSquid Software, nd] GigaSquid Software (n.d.). Speech act classification for text with clojure - squid's blog. http://gigasquidsoftware.com/blog/2015/10/20/speech-act-classification-for-text-with-clojure/. (Accessed on 01/18/2019).

[GitLab, nd] GitLab (n.d.). Gitlab continuous integration & deployment. https://about.gitlab.com/features/gitlab-ci-cd/.

[Google, nd] Google (n.d.). Getting punctuation | cloud speech-to-text api | google cloud. https://cloud.google.com/speech-to-text/docs/automatic-punctuation.

[Guru99, nda] Guru99 (n.d.a). Integration testing tutorial: Big bang, top down & bottom up. https://www.guru99.com/integration-testing.html.

[Guru99, ndb] Guru99 (n.d.b). What is non-functional testing? https://www.guru99.com/non-functional-testing.html.

[Guru99, ndc] Guru99 (n.d.c). What is system testing? types & definition with example. https://www.guru99.com/system-testing.html.

[Hamdan and Alramouni, 2015] Hamdan, S. and Alramouni, S. (2015). A quality framework for software continuous integration. *Procedia Manufacturing*, 3:2019 – 2025.

[HL7, nd] HL7 (n.d.). Over hl7. https://www.hl7.nl/overhl7.html.

[ISO/IEC JTC 1/SC 7, 2011] ISO/IEC JTC 1/SC 7 (2011). ISO/IEC 25010:2011. https://www.iso.org/standard/35733.html.

[Khan and Salim, 2014] Khan, A. and Salim, N. (2014). A review on abstractive summarization methods. *Journal of Theoretical and Applied Information Technology*, 59(1):64–72.

[KNMG, 2018] KNMG (2018). Knmg-richtlijn omgaan met medische gegevens.

[KNMP, 2018] KNMP (2018). Apotheek.nl. http://www.apotheek.nl.

[Lantronix, 2010] Lantronix (2010). Medical device networking for smarter healthcare. Technical report, Lantronix, Inc.

[Medipim, nd] Medipim (n.d.). Medipim. http://www.medipim.be.

[Nederland, nd] Nederland, Z. (n.d.). Gipdatabank. http://www.gipdatabank.nl.

[NHG, 2013] NHG (2013). Richtlijn nhg. https://www.nhg.org/sites/default/files/content/nhg_org/uploads/richtlijn_adequate_dossiervorming_met_het_elektronisch_patientendossier_1.pdf.

[NHG, 2016] NHG (2016). Richtlijn gegevensuitwisseling huisarts-centrale huisartsenpost — nhg. https://www.nhg.org/themas/artikelen/richtlijn-gegevensuitwisseling-huisarts-centrale-huisartsenpost.

[NVMO, 1990] NVMO (1990). Gespreksvoering en anamnesetraining. http://www.nvmo.nl/resources/js/tinymce/plugins/imagemanager/files/109._Pols_BMO_1990_-109.pdf/.

[Postma et al., 2016] Postma, M., van Miltenburg, E., Segers, R., Schoen, A., and Vossen, P. (2016). Open Dutch WordNet. In *Proceedings of the Eight Global Wordnet Conference*, Bucharest, Romania.

[Richtlijnendatabase, 2017] Richtlijnendatabase (2017). Richtlijn informatie-uitwisseling tussen huisarts en specialist - richtlijn - richtlijnendatabase. https://richtlijnendatabase.nl/richtlijn/richtlijn_informatie-uitwisseling_tussen_huisarts_en_specialist_hasp/richtlijn_informatieuitwisseling_tussen_huisarts_en_specialist.html.

[Rodriguez et al., 2014] Rodriguez, I., Llana, L., and Rabanal, P. (2014). A general testability theory: Classes, properties, complexity, and testing reductions. *IEEE Transactions on Software Engineering*, 40:862 – 894.

[Shami and Verhelst, 2007] Shami, M. and Verhelst, W. (2007). An evaluation of the robustness of existing supervised machine learning approaches to the classification of emotions in speech. *Speech Communication*, 49(3):201 – 212.

[Snomed, nd] Snomed (n.d.). SNOMED International. https://www.snomed.org.

[Software Testing Fundamentals, nda] Software Testing Fundamentals (n.d.a). Acceptance testing - software testing fundamentals. http://softwaretestingfundamentals.com/acceptance-testing/.

[Software Testing Fundamentals, ndb] Software Testing Fundamentals (n.d.b). Integration testing. http://softwaretestingfundamentals.com/integration-testing/.

[Software Testing Fundamentals, ndc] Software Testing Fundamentals (n.d.c). Integration testing - software testing fundamentals. http://softwaretestingfundamentals.com/integration-testing/.

[spaCy, nd] spaCy (n.d.). Facts & figures - spacy. http://www.spaCy.io/.

[Speer and Havasi, 2012] Speer, R. and Havasi, C. (2012). Representing general relational knowledge in conceptnet 5. In *LREC*.

[Van den Bosch et al., 2007] Van den Bosch, A., Busser, B., Canisius, S., and Daelemans, W. (2007). An efficient memory-based morphosyntactic tagger and parser for dutch. In Dirix, P., Schuurman, I., Vandeghinste, V., and Van Eynde, F., editors, *Computational Linguistics in the Netherlands: Proceedings of the 17th Meeting of Computational Linguistics in the Netherlands.*, pages 99 – 114. LOT.

[V&VN, 2011] V&VN (2011). Richtlijn verpleegkundige en verzorgende verslaglegging.

[VZVZ, nda] VZVZ (n.d.a). Netwerkleveranciers — vzvz. https://www.vzvz.nl/ict-dienstverleners/netwerkleveranciers.

[VZVZ, ndb] VZVZ (n.d.b). Over het lsp — vzvz. https://www.vzvz.nl/over-het-lsp.

[VZVZ, ndc] VZVZ (n.d.c). Over vzvz — vzvz. https://www.vzvz.nl/over-vzvz.

[VZVZ, ndd] VZVZ (n.d.d). Softwareleveranciers — vzvz. https://www.vzvz.nl/ict-dienstverleners/softwareleveranciers.

[Zorgvisie, 2017] Zorgvisie (2017). Wet cliëntenrechten bij elektronische verwerking van gegevens komt er aan - zorgvisie. https://www.zorgvisie.nl/wet-clientenrechten-bij-elektronische-verwerking-van-gegevens-komt-er-aan/.

Part IV

Appendices

Appendix A

GUI screenshots

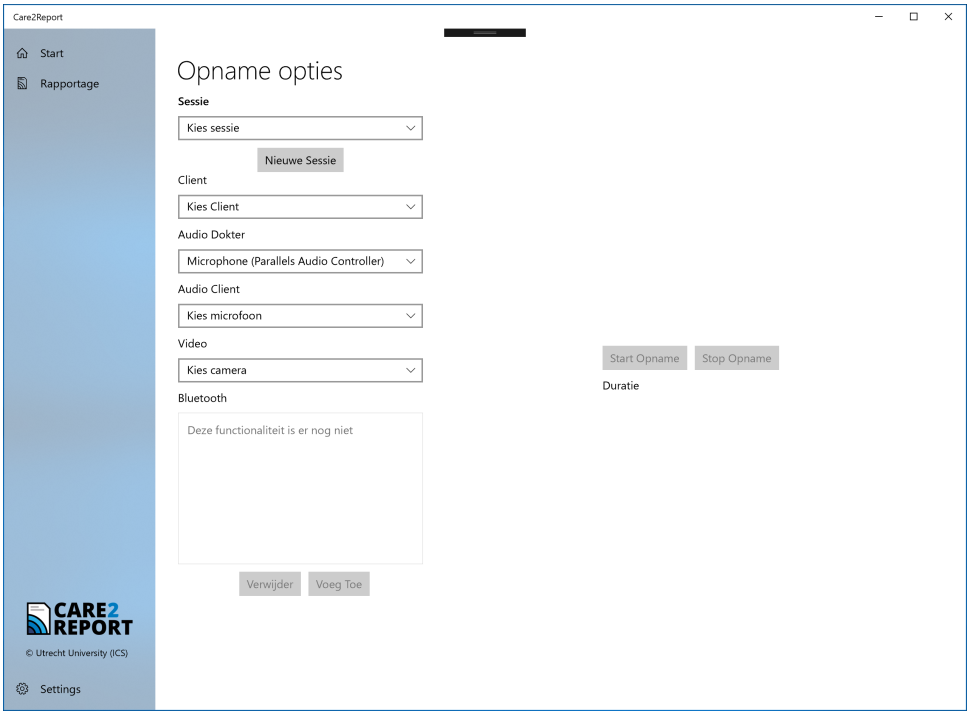

Figure A.1: Home page

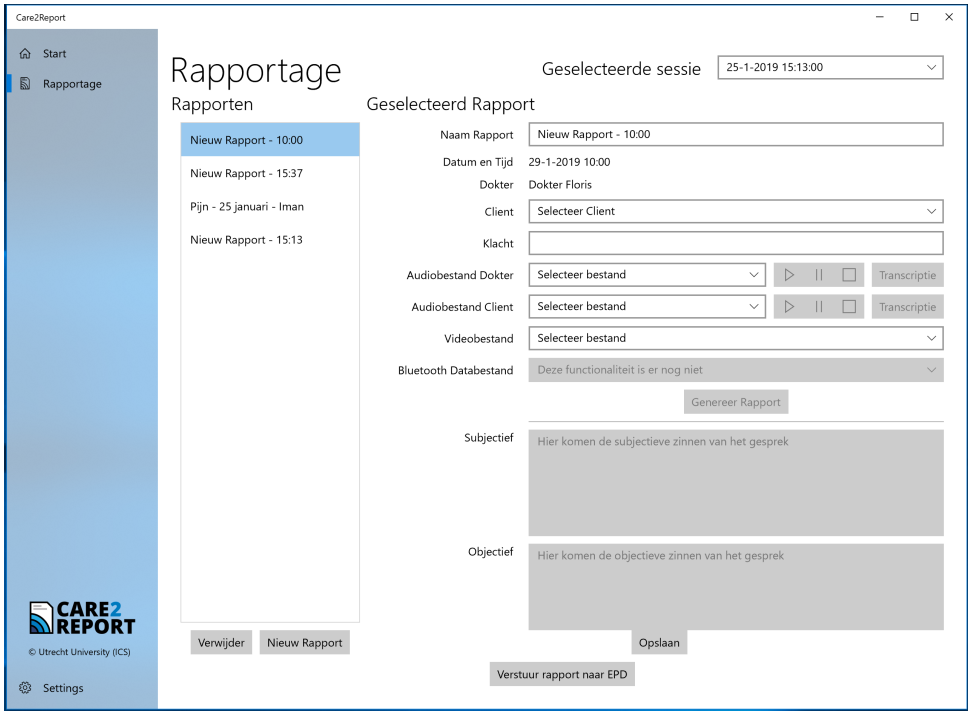

Figure A.2: Report page before generating report

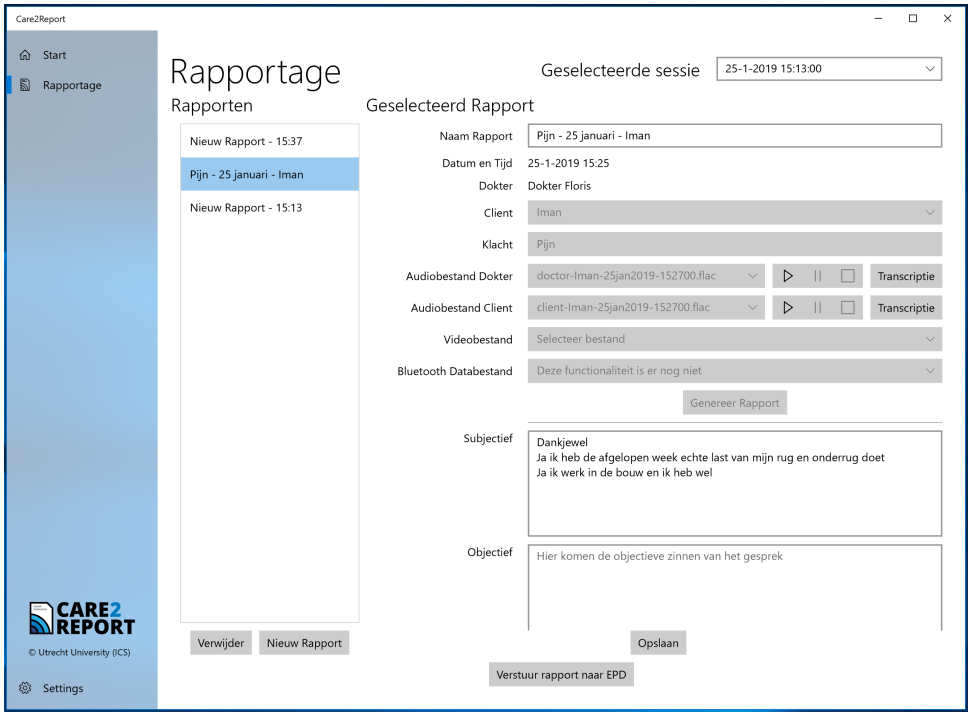

Figure A.3: Report page after generating report

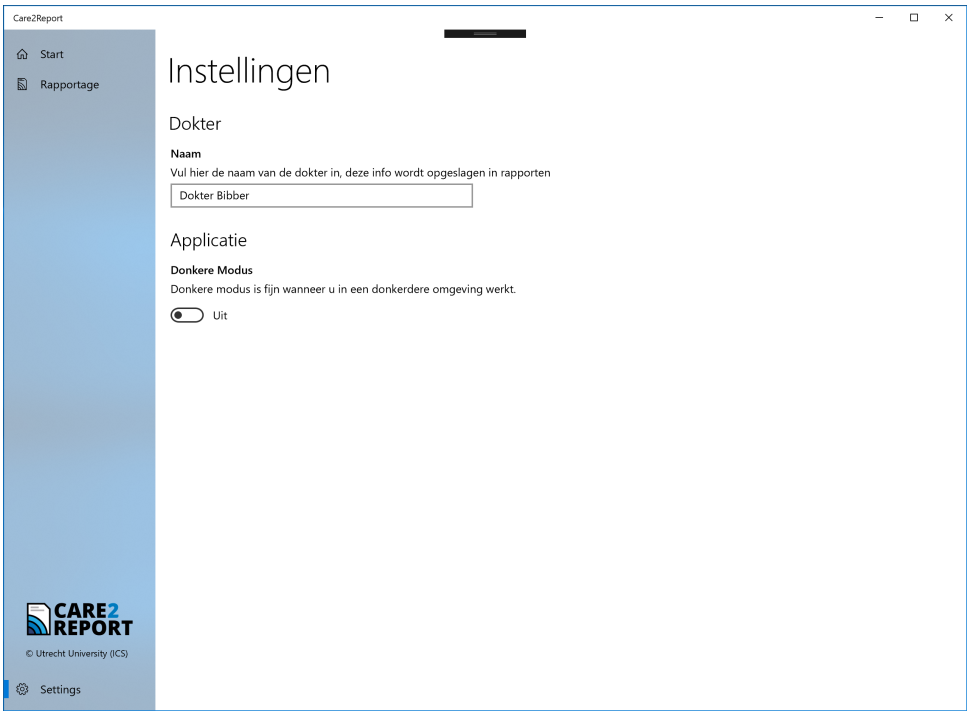

Figure A.4: Settings page

Appendix B

Final Dependency Graph

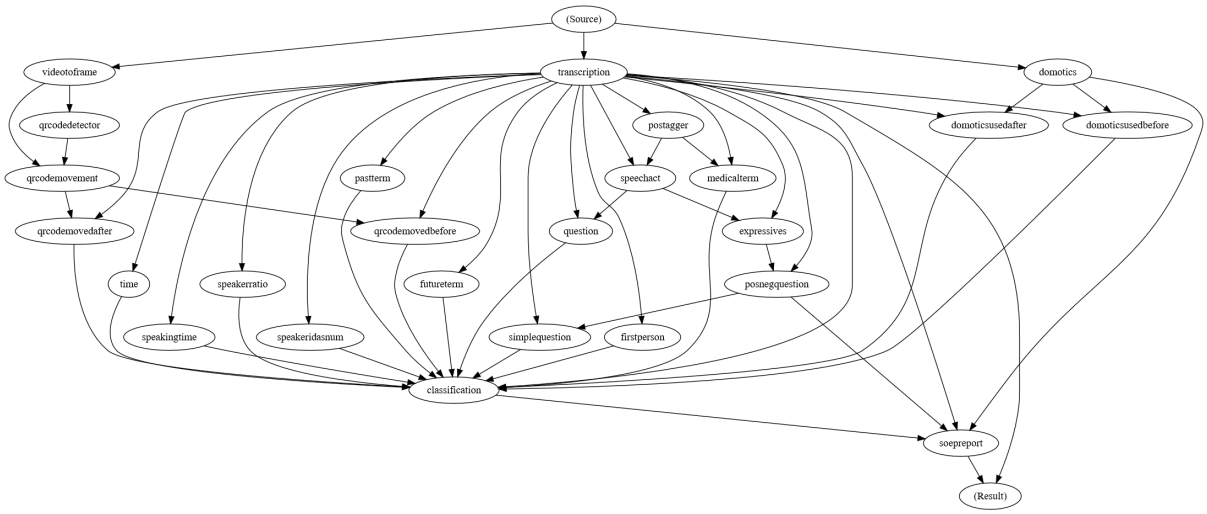

Figure B.1: Final dependency graph

Appendix C

Session Timeline Example

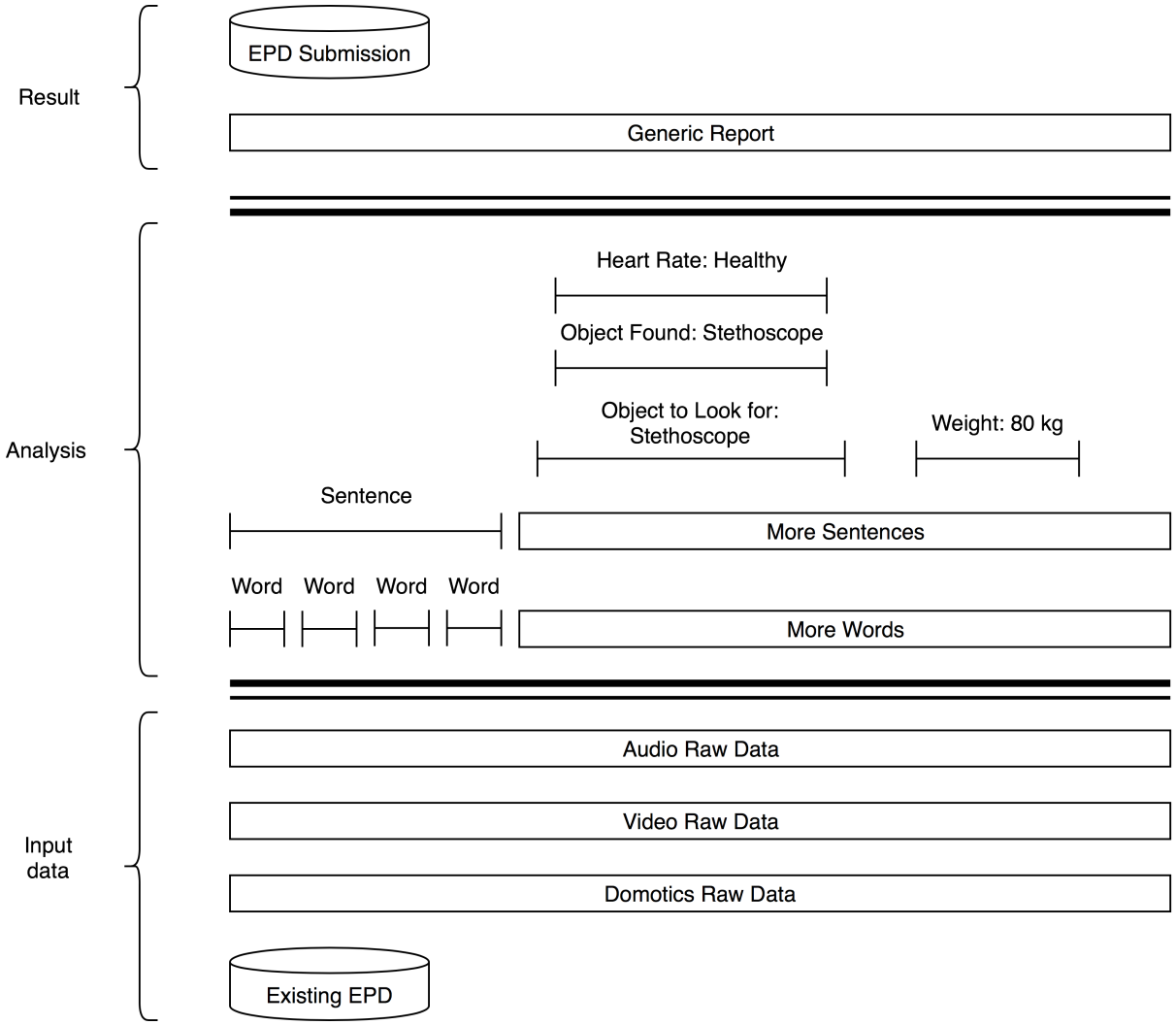

Figure C.1: Session Timeline example

Appendix D

GANTT Chart

PROJECT TITLE	Speech2EPD
PROJECT MANAGER	Ruben Verboon
COMPANY NAME	Kiéli Ltd.

Task Title	Start Date	Due Date	Days Work
Minimal Viable Product	14/9/18	8/11/18	40
Architectural Design	14/9/18	31/1/19	50
Refinement	9/11/18	3/1/19	40
Domotics	12/10/18	31/1/19	80
Completed Project	4/1/19	31/1/19	20

Sprint 1 (14/09 - 27/09) | Sprint 2 (28/09 - 11/10)
WEEK 1 | WEEK 2 | WEEK 3 | WEEK 4
F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th

Task Title	Start Date	Due Date	Days Work
Minimal Viable Product	14/9/18	8/11/18	40
Architectural Design	14/9/18	31/1/19	50
Refinement	9/11/18	3/1/19	40
Domotics	12/10/18	31/1/19	80
Completed Project	4/1/19	31/1/19	20

Sprint 3 (12/10 - 25/10) | Sprint 4 (26/10 - 08/11)
WEEK 5 | WEEK 6 | WEEK 7 | WEEK 8
F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th

Task Title	Start Date	Due Date	Days Work
Minimal Viable Product	14/9/18	8/11/18	40
Architectural Design	14/9/18	31/1/19	50
Refinement	9/11/18	3/1/19	40
Domotics	12/10/18	31/1/19	80
Completed Project	4/1/19	31/1/19	20

Sprint 5 (09/11 - 22/11) | Sprint 6 (23/11 - 06/12)
WEEK 9 | WEEK 10 | WEEK 11 | WEEK 12
F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th

Task Title	Start Date	Due Date	Days Work
Minimal Viable Product	14/9/18	8/11/18	40
Architectural Design	14/9/18	31/1/19	50
Refinement	9/11/18	3/1/19	40
Domotics	12/10/18	31/1/19	80
Completed Project	4/1/19	31/1/19	20

Sprint 7 (07/12 - 20/12) | Kerstvakantie (21/12 - 03/01)
WEEK 13 | WEEK 14 | WEEK 15 | WEEK 16
F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th

Task Title	Start Date	Due Date	Days Work
Minimal Viable Product	14/9/18	8/11/18	40
Architectural Design	14/9/18	31/1/19	50
Refinement	9/11/18	3/1/19	40
Domotics	12/10/18	31/1/19	80
Completed Project	4/1/19	31/1/19	20

Sprint 8 (04/01 - 17/01) | Sprint 9 (18/01 - 31/01)
WEEK 17 | WEEK 18 | WEEK 19 | WEEK 20
F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th

Figure D.1: The project GANTT

Appendix E

Stakeholder Table

Table E.1: Stakeholder table

Stakeholder	**Why**	**Interest(s)**	**Constraints**	
			Functional	**Non-functional**
Client	The client is the primary stakeholder. They took the initiative, came up with an idea and want to have their idea realised.	Demoing a proof-of-concept. Possible extensibility.	Use existing state of the art speech to text software. Create loosely coupled components so that the product can evolve and stay relevant with new developments. Make an easily changeable/expandable program.	Create project within time frame. Create project within budget.
Development team	The development team creates the product and does this on a voluntary base as academic development is the primary goal.	A good grade. Self development. Experience. A clear goal.	Use of languages developers are familiar with.	Have a limited amount of time. Cannot have contact with EPD suppliers.
Project bureau	The project bureau facilitates the project for the students and has contact with the client.	A project students learn from. A good relationship with the client.		Project needs to be done within time frame.
Patient	The product will be used in the presence of the patient.	Good documentation of the conversation for future medical care. Quicker health care because of shorter waiting lists.	Takes into account the privacy.	Non intrusive. Does not take up more time than usual. Does not influence the quality of the medical care in a negative way.
Medical practitioner	The medical practitioner will use the product.	Decreases time spend on administration and increases time that can be spent with patients. Provides administration that is shareable.	Ease of use. The quality of EPD entries has to be retained.	Does not influence the medical care given to patients in a negative way.

Table E.2: Stakeholder table (cont.)

Stakeholder	**Why**	**Interest(s)**	**Constraints**	
			Functional	**Non-functional**
Health institutions	The medical practitioners using the system are part of an institution.	Medical practitioners can see more patients in the same time, saving the institution money. Improve employees' experience by lessening administrative workload.	Quality has to be good, a poor product could bring the institution in a negative light.	Does not influence the patient's treatment in a negative way. Does not disrupt medical practitioners during their work.
Government	The government is responsible for medical care and provides the budget for medical professionals.	Lower medical costs. Higher overall satisfaction across the sector.	Patient privacy must be guaranteed.	Quality of health care must be retained.
Taxpayers	In the end a portion from the salary of the medical practitioners comes from tax money.	Less tax money used for administration.		Project should save money, not cost extra.
Future developers	Future developers will use the architecture, code and documentation from the original team to further develop the product.	Be able to further develop the project by the original team.	Use of well-known languages, frameworks and technologies which are likely to stay in use for the foreseeable future. Extensible code. Understandable code. Usable technical and code documentation.	
Utrecht University	The students represent the name of the university.	A good representation of the university.		Positive client-development team interaction. Product usable after project.