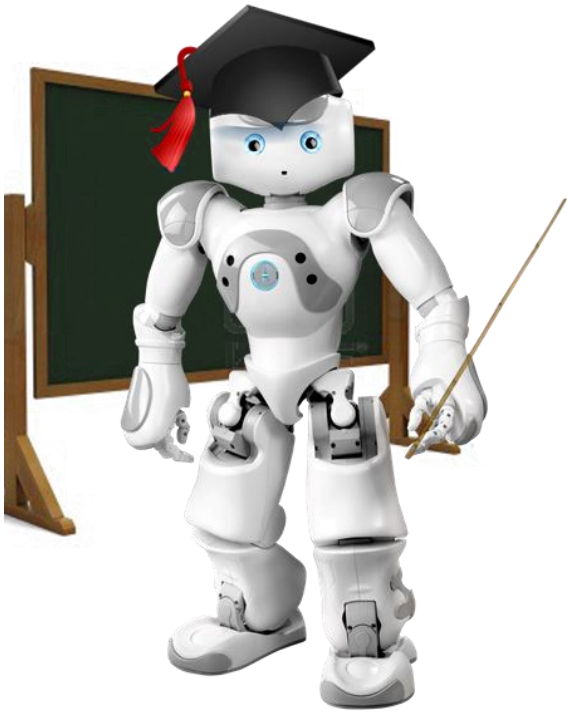
Internship report:

RoboTutor

Speech Recognition



Date:	**05-02-2016**
Internship tutor:	Mr. J.J. van der Burg
Internship company:	EWI TU Delft
Job coach:	Mr. Koen Hindriks
Internship coach 1:	Mr. Joachim de Greeff
Internship coach 2:	Mr. Ruud de Jong
Student name:	Yi Xun Cheng
Student number:	07004338
Study:	Technische Informatica
School:	Haagse Hogeschool

Acknowledgement

During my internship period at RoboTutor of TU Delft EWI I've learned a lot. It was great to finally use the skills I've learned during my study at HHS Delft Technische Informatica in a professional way.

With this acknowledgement I want to thank Mr. K.Hindriks, Mr. J. de Greeff and Mr. R. de Jong from RoboTutor and the PAL team for helping me during my internship period. Especially Mr. R. de Jong who helped me by giving suggestions and answers regarding RoboTutor.

Yi Xun Cheng

The Hague, February 2016

Abbreviations

Term	Definition
TU Delft	Delft University of Technology
EEMCS/EWI	Faculty of Electrical Engineering, Mathematics and Computer Science
SR	Speech Recognition
RT	RoboTutor
PAL	Personal Assistant for healthy Lifestyle
GSR	Google's Speech Recognition API

Table of Contents

1 Introduction

The following internship report describes the results of a 20 week internship at TU Delft EWI within the research groups of the RoboTutor and PAL projects, executed by Yi Xun Cheng, student of Technische Informatica at Haagse Hogeschool Delft.

In the third year of the Technische Informatica program an internship is a compulsory part of the program. The HBO Bachelor degree student performs this internship at a commercially established company. The purpose is to gain experience with a commercial company and to learn how to apply knowledge to a practical and challenging assignment using the skills acquired during the 3 years of the Bachelor program.

However, the internship was performed at TU Delft, which is not a commercial company.
But the opportunity to handle academic working and thinking and to combine research with development was still possible. The student was still able to gain new knowledges and experiences.

Besides this, the main motivation was my ambition to work on robotics, preferably in the research sector. For this reason this internship was a great opportunity for me to contribute to this interesting field of research and eventually help robots communicate with kids.

The first part of the report offers an overview of the organization, followed by the working plan initially agreed upon with TU Delft and approved by The Hague University as a suitable internship (see *Appendix C - Stageplan*).
Following, it proceeds to describe in some detail the most relevant activities carried out and their respective analysis of the speech recognition server and how this was achieved. Finally, the report wrap-ups with a few closing remarks and conclusions from the experience.

2 Organization

2.1 Background

TU Delft EWI/EEMCS
TU Delft EWI is the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology. The education programs are designed for the engineers of the future. This faculty play a big role to achieve this.
Nowadays you can see more technological and innovational solutions blending in our society and this faculty help future engineers to be part of it.
Electrical Engineering and Computer Science together account for 80% of the ICT research at TU Delft. They currently focus on sustainability, high-level effectiveness and energy savings which indicates to the fields of expertise: electrical power engineering and power supply.

Mission
The Faculty develops and maintain knowledge and expertise at the highest international level, and distribute this knowledge to third parties. Social problems we encounter today and the predicted ones of the future will be challenged and solved. The faculty provide aid at a Bachelor's, Master's and doctoral level for students to become responsible engineers who will challenge the future. With a stimulating and inspiring environment the faculty prepares the students for their future careers and their responsibility to the community.

Vision
TU Delft's unique social position, as set out in 'Challenging the Future', means that the programs are designed with the engineers of the future in mind. These young engineers will be challenging global problems in the year 2040. With an expectation in world population of 9 billion people by 2050, it's ravaging how the living standards of the vast majority of the population is still far behind compared to the ones in the rich developed world. Rising living standards in the developing world represents the greatest challenge of the future and will unleash complex issues. Increasing energy consumption and dwindling natural resources, a worsening climate problem, globalization, structural poverty, urbanization, war and international migration would seem to be in store.

The expectations of these problems plays a big role for the vision of the role and responsibilities of the Faculty of EEMCS. Electrical Engineering, Computer Science and Mathematics will make vital contributions to global welfare, well-being and security. ICT will play an essential role in the future society and it will be the key to greater globalization, connecting a world which will become totally networked at all possible levels. Expanding virtualization will also lead to new paradigms. The faculty will make a real contribution to developing sustainable products and systems and address the challenges of energy production, storage and distribution.

Strategy
With a faculty that hosts 3 programs in one building, it's easier for these programs to work together. The research facilities in this faculty works on international levels and some of these research groups are even on the top of international levels. The faculty strives for great quality and teamwork between the 3 programs so together they can research on ambitious targets. There will also be focus on quality of teaching, both within the educational programs and the educational process and environment.

The Faculty's main objectives can be summarized as follows:

- Supplying excellent BSc, MSc and PhD graduates
- Conducting leading research programs
- Ensuring high visibility for research findings
- Outstanding research facilities
- Creating a pleasant working environment for staff
- Helping alumni to occupy important national and international positions

Policy
The Faculty's excellent researchers and outstanding facilities in three important ICT disciplines, energy power engineering and applied mathematics will enable them to meet these objectives. The fact that they participate in so many research centers and institutes, interfaculty research projects and national and international research projects demonstrates the relevance of their fields of study.

To achieve the mission and objectives set out above, the main thrusts of the Faculty's policies in the near future are:

- Modernize and enhance the quality of teaching: the starting point for this will be to ensure complete coherence between the Bachelor's, Master's and post-graduate level programs.
- Enhance the attractiveness and input of the Bachelor's degree programs.
- Focus more closely on the sustainability of research: anticipate new research areas and expand new research which involves solving today's major global questions.
- Enhance the quality and visibility of the Faculty's research.
- Stimulate socially engaged and responsible entrepreneurship in the broadest sense.

2.2 Projects Overview

The internship period consists of a concurrency with two research projects at TU Delft EWI. Both of the projects have a research goal in the field of robotics.

2.2.1 RoboTutor project

RoboTutor originated with the idea of applying modern and innovative technology in education. Modern humanoid robots are used to help teachers in class. The robots can teach standardized curriculum to the audience and this means that the teacher will have more time to focus on personal attention of the crowd. To achieve this, a lot of human robot interaction have to take place.

RoboTutor uses a NAO robot from Aldebaran Robotics which is equipped with special software written in C#. The main purpose of the robot is to give a presentation to a group of people (see Image 1). The robot uses slide sheets of PowerPoint and accompany the presentation with narration.

[Image 1: RoboTutor giving a presentation in a classroom]

However, the robot cannot yet come up with its own presentation and therefore the software uses scripts to accomplish it. These are simple text files that contain the text to be spoken by the robot at a certain slide moment.
Another important feature of RoboTutor is the multiple question feature. This feature uses a TurningPoint system. It consists of 'response cards' that is distributed to the audience. With these cards the audience can make choices during a multiple question session and based on the chosen answers the robot will generate a reply.

The RoboTutor project also has a mood expression module. This module is the PhD thesis of J. Xu (Xu, 2015) which is developed for the robot to be able to show emotional behaviors according to its mood. This gives the robot an extra social ability and be more human like. With the human like body language of the robot, humans see the robot as trustworthy, reliable and life-like.

2.2.2 PAL project

The PAL (Personal Assistant for healthy Lifestyle) project is an ongoing (started 1st of March 2015) 4 year project which involves the research partners TNO (as coordinator), DFKI, FCSR, Imperial and Delft University of Technology, end-users (Gelderse Vallei hospital and Meander, and Diabetics Associations of Netherlands and Italy), and SME's Mixel and Produxi. This project is granted 4.5M euro by the EU (ref. H2020-PHC-643783).

This project develops a system to help children between 7-14 years old with Type 1 Diabetes Mellitus (T1DM). Besides the children, it's also an assistant for the children's families and the related medical staff. The children can acquire knowledge, skills and habits about diabetes with this assistant. It is also possible to keep track of health condition, development and behaviors of the disease, abilities, gender and social environment.

The PAL system consists of a social robot (Aldebaran NAO), a mobile avatar and an extendable set of mobile health applications (on-line diary and educational quizzes). These modules are connected to a common knowledge-based and reasoning mechanism. Health professionals (diabetologists, nutritionists, nurses and psychologists) instruct and supervise the PAL system by setting self-management goals and tasks for the children. They can also monitor the situation and developments of the child and provide support when it's needed. PAL helps the children adopt to the goals and perform activities (together with their families) that help them achieve these goals.
For a global overview (see *Appendix A – PAL Cloud*).

3 Job description

3.1 Motive

RoboTutor expands its features as much as possible. The goal for the future is a robot that can support a teacher in class, so the teacher can focus on different tasks. To achieve this the robot has to educate the audience in an entertaining way.
The software of RoboTutor provides the ability to be further expanded with new features. There is now a need for a new feature for RoboTutor: to create more interaction with the audience through speech.

3.2 Problem Description

Currently it is not possible for the software of RoboTutor to understand what the audience is saying to the robot, it can only give a presentation and cannot receive feedback.
To make it more interactive the robot has to be able to understand the audience and give a reply or perform an action. This is achieved with the new feature of speech recognition.

3.3 Objective of the Assignment

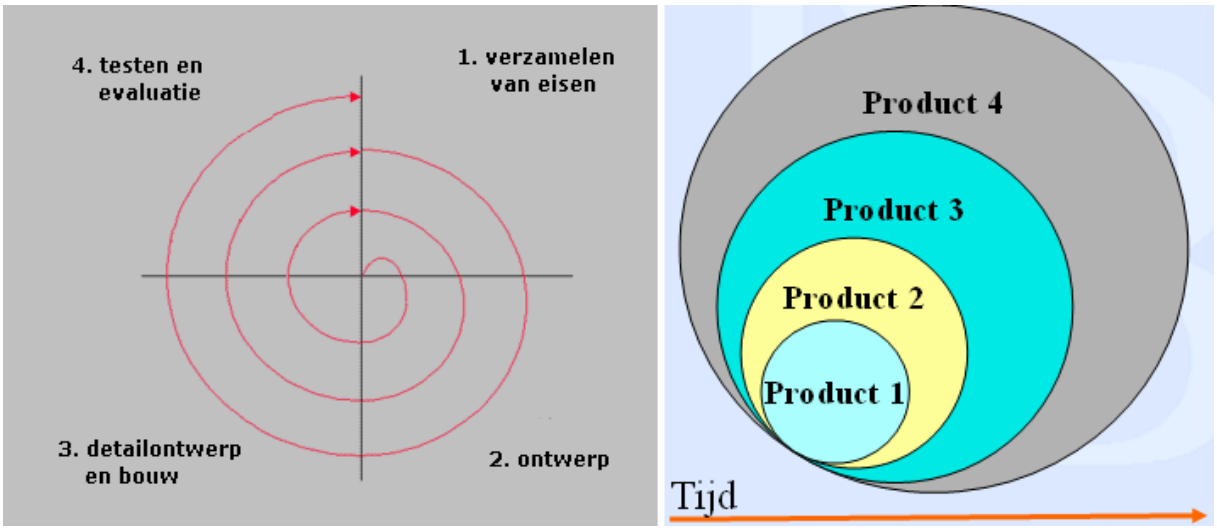
RoboTutor is extended with a speech recognition feature. The robot is able to hear sentences now, recognize them and perform a command. Commands are added to RoboTutor and the robot can generate a logical reply and/or perform an action. With this new technique of human-robot communications, RoboTutor is one step closer to be more human like and assist a teacher in class.

3.4 Results

Assignment is accomplished when the RoboTutor speech recognition system works using a microphone in a room with more than 10 people that can choose from 10 different predefined commands. The selected commands must be executed successfully by the robot. Only one command can be performed at the same time and after executing this command, the audience can choose for a new command.

3.5 Approach

For this project, it is very clear from the beginning that only the end results are determined. There are no demands on how this should be accomplished as long as the end results are achieved. The evolutionary/cyclic software process model (Craig Larman, 2003) makes this possible. This methodology consists of several short cycles after each other. During each of those cycles the project is analyzed, designed, realized and tested (see Image 2).



[Image 2: Cyclic software development methodology cycles]

This is different from the waterfall method (TutorialPoint, 2016), where the project is analyzed, designed, realized and tested once. Agile's Extreme Programming (Amber, 2012) also comes to mind, because of constantly changing of the requirements and design. The biggest problem of XP is the beneficial feature of pair programming, but this cannot be done if there is only one programmer working on the speech recognition feature. Besides pair programming, XP also has the feature to write unit tests before actually coding. With a research project this is very difficult, because this project is an experiment on the end users.

Evolutionary software process model is the best match, because features of evolutionary methodology connects with the needs of this project. During this project, it is not defined from the start how the result will be achieved. This makes it possible for the programmers to have more freedom in their own design choices. The only big picture is the end goal, which is a working speech recognition feature that translates audience speech to text for the robot to understand. When the robot understands the text, it generates a reply or executes a movement. It is expected that the requirements can change at any moment: because experiment can show that the end users (primary school students) dislike it, or new techniques and approach has been found to implement the feature. We as technicians see the robot as a high tech machine, while the kids see the robot more than just something which has features: they see it as a human being, maybe even a friend and treat it as one.

Each cycle during this project can be described in detail as:

1. Analysis (Analyze problems and change requirements)
2. Design (Add/update design according to the requirements)
3. Realization (Implementation of the new features for the system)
4. Test (Perform test to validate that the new feature is working)

4 Analysis

Before starting with the project, it is very clear that a profound research needs to be done about the available speech recognition methods. This chapter describes the research on SR in detail and also the route to accomplish the project.

4.1 Research on speech recognition

Speech recognition is the technology that enables computer to translate spoken language into understandable text. A profound research on speech recognition is needed to gather knowledge and understanding on how it works. This is needed to make it easier to implement the speech recognition module in a correct way.

During the research, there are some requirements to keep in mind with:

- The speech recognition engine has to be crossplatform, so future programmers will not be platform dependent.
- The engine has to support the Dutch language, because the primary use of RoboTutor is for primary school kids and most of them do not understand English yet.
- The engine does not need to be connected to the internet to work.

Techniques

There are different techniques to apply speech recognition. Some SR systems are speaker dependent while others are speaker independent (voice-commands.com, 2005). Speaker dependent means that the SR system is "trained" by having an individual person speak to the SR system. This results in increased accuracy of the SR system, but the system is only trained for that specific voice.

A speaker independent way is applying the technique of big cloud data (Mozer, 2015). This means that an individual speaker is not needed to train the system. The SR engine is trained with a lot of cloud data which has been gathered in the past. The benefits of this technique compared to speaker dependent is that the "training" data is stored on a server computer while speaker dependent stores all the speech data locally. But it does have a disadvantage too: to apply big cloud data, one has to be connected to the internet for the SR engine to work.

Another great method to apply speech recognition is Deep Learning in Deep Neural Networks (BaiduUSA, 2015). This is a very new technology in speech recognition and probably solves a lot of problems that current SR engines have. Right now it specializes in noisy environments, which is a big problem for a lot of SR engines. Deep Learning is a bit different than cloud based big data: it does also use big data for training, but what makes it amazing is the use of virtual neural networks. This technique is still ongoing a lot of researches to make it work and therefore cannot be used during this project.

Engines

There are a lot of ongoing researches about speech recognition. Some even have their own engines which are available and easy to use while some other SR engines requires an expert in this field.

Research on the possible SR engines put focus on certain aspects:

- Open Source vs Commercial
- Whether the SR engine is still supported and updated.
- Programming language, favorable crossplatform compatible.
- Advantages and disadvantages

CMU Sphinx (CMUSphinx, 2015) is an Open Source speech recognition toolkit with various tools used to build speech applications. CMU's Sphinx4 and Pocketsphinx is described, because these are the researched engines from the toolkit. Those engines has a common way to recognize speech: take waveform, split it on utterances by silences and then try to recognize what's being said in each utterance. This is achieved by matching the audio with all the possible combinations of words. The Hidden Markov Model (Ghahramani, 2001) is used in these SR engines as the model of speech to describe the sequential process. The HMM model consists of a sequence of states which change each other with certain probability.
Speech recognition uses three models to do the match: An acoustic model (acoustic properties for each senone, short sound detector), a phonetic dictionary (mapping from words to phones) and a language model (restrict word search, defines which word could follow previously recognized words).

Commercial based speech recognition engine Dragon Nuance (Nuance, 2015) is the best working SR engine currently available, voted as most accurate engine in 2015 (Knoder, 2015). It is possible to use with different programming languages: C, C++, and Visual Basic, etc., easy to use and have a great support center. Besides being the best, it is also the engine used by Apple's Siri (Wildstrom, 2011). But it's not a fairy tale engine either, from the greatness it offers it comes with a very expensive price (169 Euros minimum).

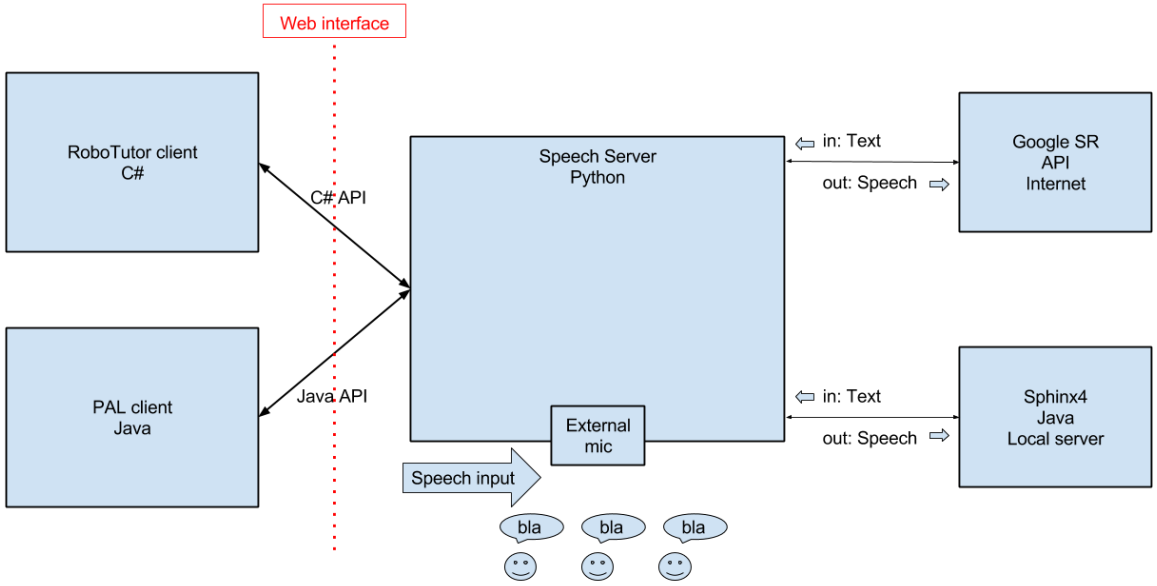
Another commercial based SR engine worth a mention is Google's SR API (Google Inc., 2012). It's the only commercial cloud based engine which offers a free API. The engine is also speaker independent, because it's trained with clouded big data from Google. One big disadvantage of this engine is the reliability on the long run: Google might decide to close the API in the future and this result mean a non-working speech recognition module for RoboTutor.

Only some of the researched SR engines are described in this document. More detailed information about the other researched engines can be found at *Appendix B – Speech Recognition Research*.

Conclusion
This thorough research on speech recognition makes it clear that a perfect speech recognition engine does not exists yet. It is to be expected that the speech recognition engine can be faulty at times and at certain circumstances (noisy environments, multiple speakers, etc.). But it is still possible to make it work as good as possible and have RoboTutor understands the audience.
From the research the best option is Baidu's Deep Speech (BaiduUSA, 2015), which uses neural networks based on deep learning to specialize in noisy environments. This is perfect to use in a primary school classroom full of screaming kids. The technique is unfortunately still very new and there are no available API's yet.
The best option for now would be to use CMU's Sphinx4, because it's open source (possible to configure to the project's requirements), still updated, programmed in Java and it has great and understandable documentation. CMU's Sphinx4 has also been installed and tested which confirmed the easy usage of the speech engine.

4.2 Project cycles

This chapter describes the planned project cycles during this internship. Even though the research concludes that CMU's Sphinx4 is the best option during this project, this is not the case. The requirements are changed by the time the research is completed because another research project in the organization comes to sight. PAL also needs a speech recognition module and a speech recognition module is built for both of the projects.
The speech recognition module is not firstly implemented on the whole organization, thus both projects. It is first implemented on RoboTutor and after confirming with tests it is implemented on PAL and tested.
The project result can be split in smaller parts. The software is divided into a number of more or less independent parts as seen in the project diagram (see Image 3).



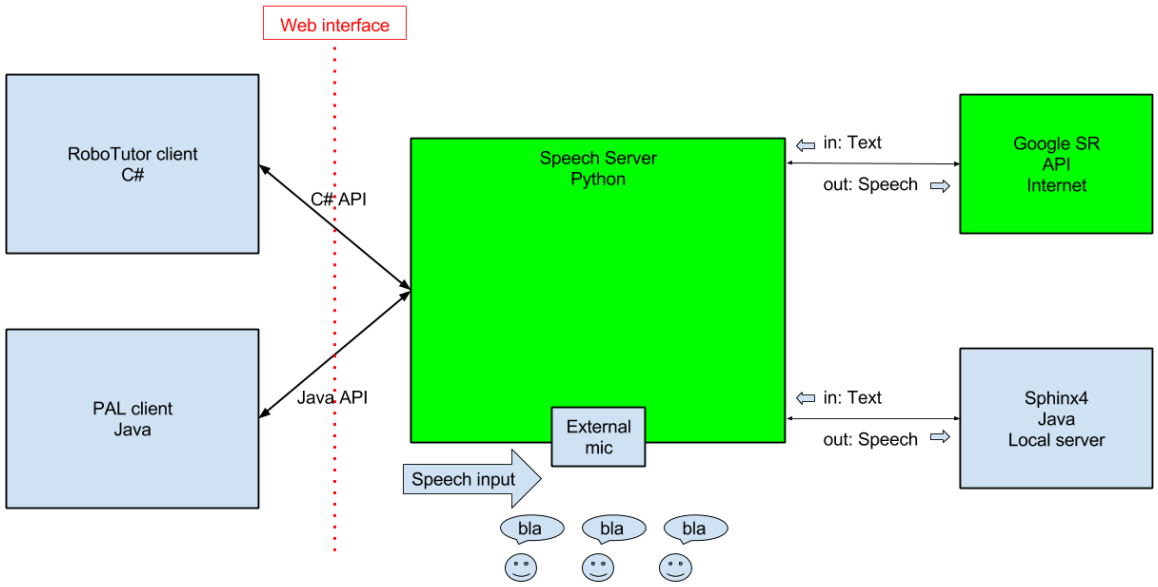
[Image 3: Full project diagram]

The speech recognition module is the Speech Server. Both the clients of RoboTutor and PAL can make use of the speech server and translate speech input into computer understandable text. It's also very easy to attach/detach a speech recognition engine, as seen in the project diagram. The speech server can be configured to use any speech recognition engine to transcribe speech data.

The project diagram does not determine the end result of this project. It is first implemented and tested with the speech engine of Google's SR API and if there is more time and priority another speech engine (Sphinx4) will be implemented to use. Even though the SR research concluded the usage of Sphinx4 during this project, GSR is chosen. The reason why Google SR API is chosen is the fact that it saves a lot of time to have the engine running compared to Sphinx4. With Sphinx4 the SR engine needs to be trained by an individual speaker and this takes a lot of time. Even after it is trained, the engine can only be optimized for that individual speaker. Google's SR API is trained with cloud based big data, the whole engine training process can be skipped and the engine can be used for multiple speakers.

5 Speech Server

Analysis



[Image 4: Project diagram with highlighted parts of the Speech Server]

During this project cycle the speech server is implemented. The speech server consists of a server which communicates with clients and uses the GSR to transcribe speech data to text (see Image 4).

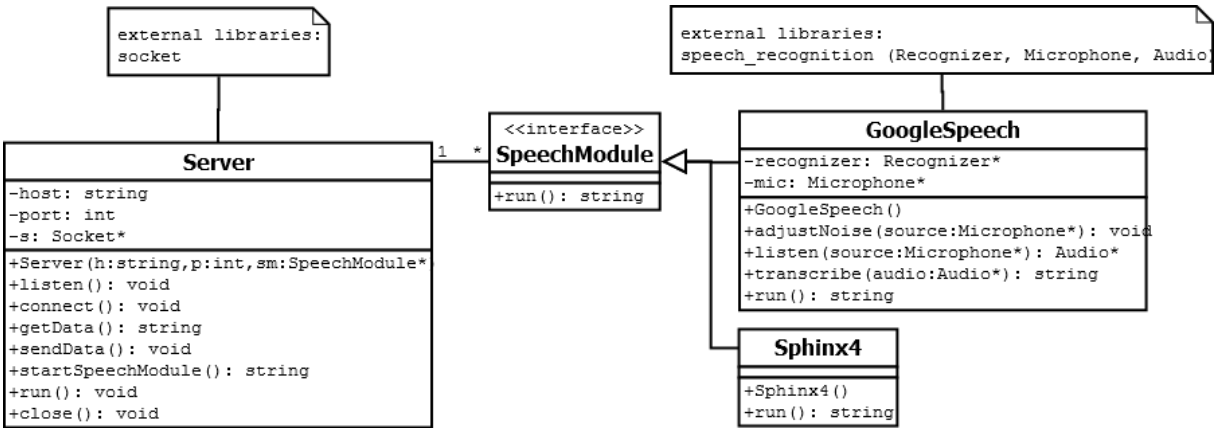
Google's SR API does not have an official ready to use module, therefore the engine can be achieved in different ways. The first question is which programming language the speech server uses to communicate with the engine. Java and Python are the possible choices, because they are both crossplatform. The decision went for Python, because it is faster and light weighted compared to Java and in Python it's possible to use wrappers within the python interpreter to import a library from a different programming language (Jython, 2011). Thinking about the future this is very smart, because the speech server is engine independent. It should be possible to attach either Google's SR API (also Python), Sphinx4 (Java) or any speech engine to the speech server.

Google SR Plugin

The Google SR API Plugin (Zhang, 2015), provided by the internship coach Mr. J. de Greeff is used during this project to apply speech recognition. This plugin supports Google SR, Wit.ai, IBM Speech to Text and AT&T Speech to Text engines. Only the Google SR API feature is used during this project. Besides recognizing speech data from the microphone, the plugin can also record and save audio to WAV format, transcribe a WAV audio file and have several other speech recognition functions which are not related to this project.

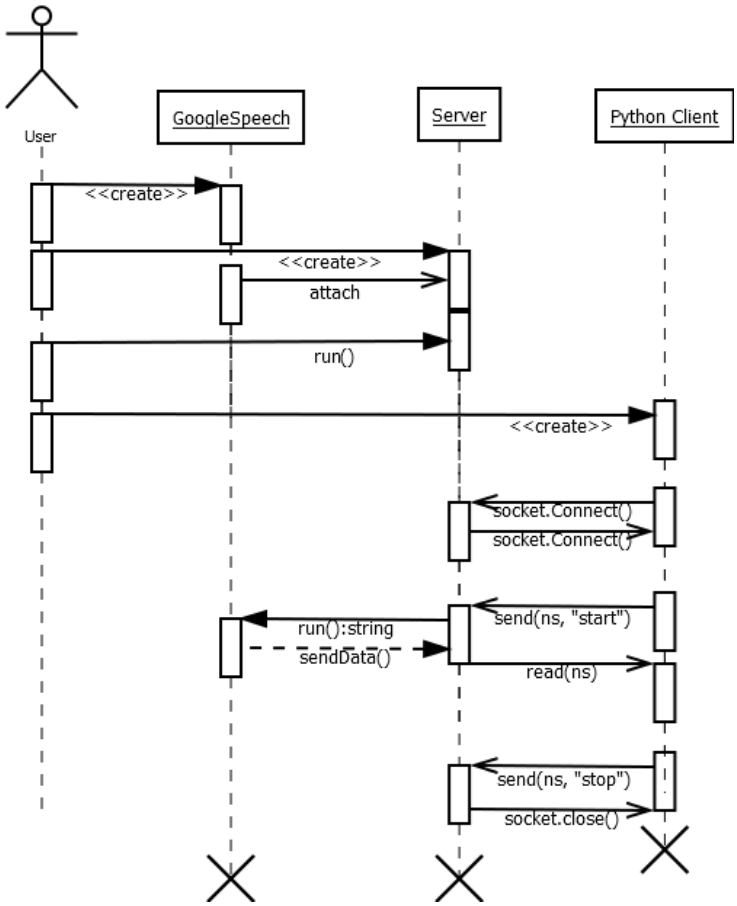
The plugin uses a FLAC encoder to encode the audio data which is sent to the Google SR API. It can also calibrate a threshold value for the background noise: the higher the threshold value, the less sensitive the speech recognition will work. Good values are expected to be in the range of 50 to 4000.

Design



[Image 5: Class diagram of the speech server]

The class diagram (see Image 5) shows how the speech server can be realized. This diagram also shows the abstract server design pattern (Gamma, Johnson, Helm, & Vlissides, 1994) used for the class SpeechModule. With the abstract server, the system can be attached with the class GoogleSpeech or Sphinx4 and it's easy to add a new class for a new speech module. The SpeechModule also acts as an interface in the code of the class Server which means if the speech module changes, the code in class Server does not have to be modified.
The class GoogleSpeech uses an external library called 'speech_recognition', this is the plugin that communicates with GSR.



[Image 6: This sequence diagram shows how a python client works with the speech server]

A working scenario is described as follow (see Image 6):

1. The speech server is started and attached with GSR.
2. The python client is started.
3. Python client is connected to the speech server through socket communications.
4. The python client requests the server to listen to speech input.
5. The server listens, transcribes the audio data and send back text data to the client.
6. The client receives the text data.
7. Connection is terminated between client and server.

Realization

For the realization of the speech server, Python 3.4 is used because of compatibility reasons with the Google SR API Plugin. The Plugin only supports PyAudio till 0.2.8 and that version of PyAudio only works for Python 3.4 or lower.

PyAudio provides Python bindings for PortAudio, which is a crossplatform audio I/O library. With PyAudio you can easily use Python to play and record audio. This is needed to make use of the internal microphone where the speech server is run. In this case, the server is run on the intern's laptop and it has a built in microphone. 'Special' external microphones will only be used if the tests show that the built in microphone are not good enough for speech recognition. The data sent through socket from client to server and vice versa is encoded/decoded to ASCII value when sending/receiving.

The following routine is executed when the speech module is run:

1. Calibration to set a threshold value for background noise, this increases the accuracy of the SR engine because it's taking notice of the background noise.
2. Create audio data by listening to the microphone until a moment of silence is detected. The default value of a moment of silence in the GSR plugin is a pause of 0.8 seconds.
3. Send the audio data to Google's SR API to transcribe.
4. If understandable text is transcribed, send the text to the client.

A simple class of Sphinx4 is created in Python to show that the Speech Server is modular and that the speech server can be attached by different speech engines.

Test

The speech server is tested with two different tests. One with WAV files and the other with real time speech input. The speech sentences tested during this chapter are defined by their potential of being a speech command for RoboTutor, because those commands can be executed by the robot.

The first test is with prerecorded data from an individual speaker. The WAV files are recorded with a python script and saved into a folder. With another python script the WAV files in the folder are translated to speech text and shown on the terminal. Recording from mobile phone generates an error, because the wav files generated from the mobile are not compatible. These two functionalities are implemented apart from the speech server. Their purpose is to show in a short period of time how usable the GSR is.

The test results of the first test are show in Table 1.

Audio Spoken Text	Transcribed Text	Correct
‘bodybuilder’	‘bodybuilder’	Yes
‘dans’	‘Daens’	No
‘doe een dansje’	‘Doe een dansje’	Yes
‘ga zitten’	‘Gazette’	No
‘hallo’	‘hallo’	Yes
‘laat je spieren zien’	‘laat je spieren zien’	Yes
‘loop naar achter’	‘loop naar achter’	Yes
‘loop naar links’	‘loop naar links’	Yes
‘loop naar rechts’	‘loop naar rechts’	Yes
‘loop naar voren’	‘loop naar voren’	Yes
‘politie’	‘politie’	Yes
‘sta op’	‘sta op’	Yes
‘zit’	‘zet’	No

[Table 1: Test results with prerecorded data from an individual speaker]

The second test is recording and sending real time speech input to the speech server. During this test a simple python client is created to start the speech server, so a client for RoboTutor and PAL can be implemented later. The speaker records speech audio by speaking into the built in microphone. The speech audio is sent to the speech server who transcribes the audio with Google’s SR API and return text data when it understands a sentence/word.
The same sentences are tested 4 times to show consistency and accuracy of the module, with the results shown in Table 2.

Spoken Text	Transcribed Text	%
‘bodybuilder’	X, ‘Barbie dood’, X, X	75
‘dans’	‘dan’, X, ‘Daens, ‘dames’	25
‘doe een dansje’	X, X, X, X	100
‘ga zitten’	‘Gazette’, ‘NZB’, X, ‘gratis zitten’	25
‘hallo’	X, X, X, X	100
‘laat je spieren zien’	X, ‘zet je virus’, X, ‘laat je vieren’	50
‘loop naar achter’	X, X, X, ‘Jansma Drachten’	75
‘loop naar links’	X, X, X, ‘Love Me Again’	75
‘loop naar rechts’	‘loop naar apps’, X, ‘loop naar apps’, ‘loop naar ex’	25
‘loop naar voren’	X, ‘loop naar noorden’, X, X	75
‘politie’	X, X, X, X	100
‘sta op’	X, ‘startup’, X, ‘start’	50
‘zit’	X, ‘zet’, X, ‘zet’	50
Average:		63.5

[Table 2: Test results with real time recorded data]

As expected, the tests shows that the speech recognition module is not perfect. The speech server has a lot of problems to understand ‘zit’ and ‘rechts’ and mostly transcribes it as ‘zet’ and ‘apps.

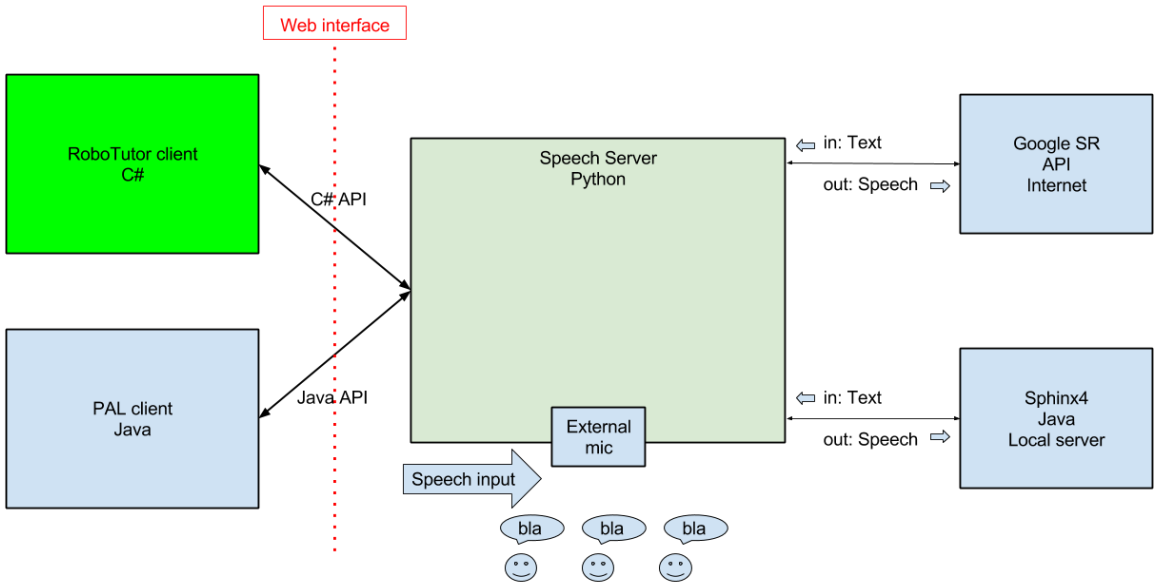
Evaluation
The achieved results with the usage of the build in microphone is better than expected. The speech server works as expected: not perfect, but useable and possible to understand speech text with a 63.5% correct rate. One serious problem for the system is the calibration of a threshold value for the background noise. This function requires that it's silent in the room but there is no user feedback on the speech server which means the user won't be notified. When there is a lot of noise during calibration, the threshold is set very high and no speech can be recognized.

There is also a problem with the moment the speech server stops listening. Now it stops until it detects a moment of silence (pause of 0.8 seconds), therefore when a speaker talks unlimitedly long the server keeps listening and sends a very large audio file to GSR. This results in crashes or long waits of the server.

Next cycle
For the next cycle user feedback is added to the terminal to notify the user that the speech server is calibrating the background noise. Besides the user feedback, the listening function is changed to listen for a maximum period of 5 seconds.

6 RoboTutor

Analysis
The following diagram shows in green color which parts of the system are implemented during this project cycle:

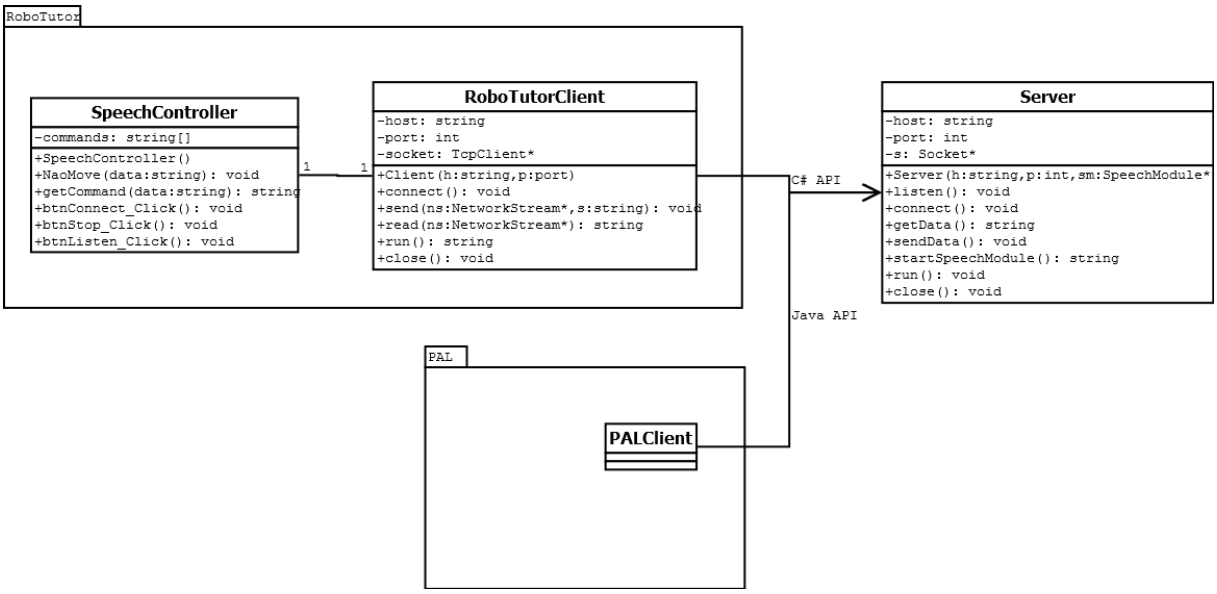


[Image 7: Project diagram with highlighted parts of the RoboTutor C# client]

The tested Python client in the previous project cycle shows that a client can be connected to the speech server and use GSR to translate audio to text. In this cycle the Python client is removed and a C# client from RoboTutor comes in its place which means RoboTutor is extended with a speech feature (see Image 7).

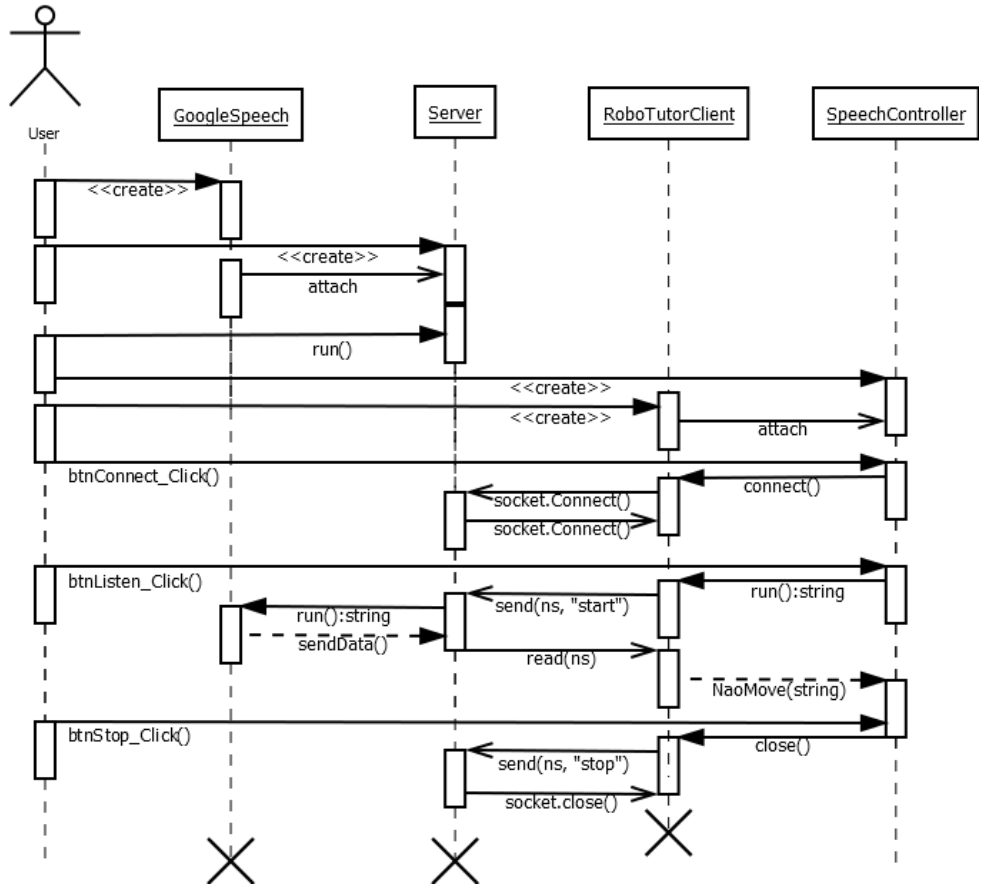
RoboTutor's current software is written in C# and works in Visual Studio 2010. The system works with different modules who can communicate with each other such as a Camera Interface, PowerPoint controller, a script engine and also a Mood Expression Behavior module.
Analyzing the current system is very difficult, because there is no documentation about the system. The only 'documentation' available is the source code and the comments inside the source code. Besides using RoboTutor's software, Choregraphe Suit 1.14.5 can also be used to work with the robot. The reason for this is faster and easier use of the Aldebaran NAO robot and be familiar with its possibilities and abilities. But the disadvantage of using Choregraphe is the modules RoboTutor provides. All the features like PowerPoint controller, script engine, mood behavior, etc. cannot be used in Choregraphe.
With the received text from the speech server, the system checks for a predefined commando and if this is the case the commando is executed. There is no logical reply algorithm in this system, maybe in the later stages of the project. A very interesting logical reply algorithm is the use of a smart chatbot. The idea is to send machine understandable text to the chatbot and the chatbot can generate a logical reply back.

Design



[Image 8: Class diagram of the RoboTutor system linked with the speech server]

The client communicates with the Speech Server through socket communication (see Image 8). The class SpeechController is the main part of the client. It communicates with other modules inside the RoboTutor project and it has a graphical user interface where a user can connect and close connection to the server and request the server to listen to a command.



[Image 9: This sequence diagram shows how the RT client executes a speech command with speech recognition]

A working scenario is described as follow (see Image 9):

1. The speech server is started and attached with GSR.
2. The RT client is started.
3. User clicks on the button 'Connect' and the client is connected to the speech server through socket communications.
4. User clicks on the button 'Listen' and the client requests the server to listen to speech input.
5. The server listens, transcribes the audio data and send back text data to the client.
6. The client receives the text data and forwards it to the Speech Controller.
7. The Speech Controller notifies another module in RT to execute the command.
8. User clicks on the button 'Stop'.
9. Connection is terminated between client and server.

Realization

Working on Visual Studio 2012 is not the best option. Even though it clearly states that the system is compatible with Visual Studio 2010, it has been run and tested on VS2012. This gives a lot of compatibilities errors, such as the wrong installation version of F# 2.0.0.0 which is needed for some of the behaviors of the mood behaviors engine.

To implement the speech recognition feature on RoboTutor, the Dashboard (where all the modules can be seen in their own respective tabs) is extended with a new tab called 'Speech'. In the speech tab the buttons 'Connect', 'Listen' and 'Stop' are added to allow the user to use the speech recognition module.
A 'simple' approach is implemented to make the robot execute a command by speech. The Speech module connects to the server and request the server to listen to a speech command. The speech server listens to speech audio and send an understandable text to the client. The speech module of RoboTutor has a string array of predefined commands that it can execute. If the received text from the server is found inside the array of commands, the speech controller calls the function NaoMove() as seen in the sequence diagram of this cycle. What NaoMove() do is send a 'Say' or 'ExecuteBehavior' message to the module NaoManager. 'Say' message is the command to let the robot speak a certain sentence while 'ExecuteBehavior' is the command to allow the robot to execute a movement. NaoManager is the RoboTutor main controller, in this case it receives a message from the Speech module to notify the module who controls the Physical robot in RoboTutor to speak a sentence or execute a movement.

Some problems from last cycle is fixed during this cycle. The speech server is modified and user feedback is added to notify the user whenever it's calibrating a threshold value. The speech listener is also been updated with a record function which listens for 5 seconds and then send the audio data to GSR.

Test

This cycle's test is a lot like the one from the previous cycle. With a RoboTutor client in place of a Python client, the speech server is tested again. But this time focusing on the robot understanding the commands and executing them correctly. The 10 predefined commands are chosen because they can be executed by the robot and is something practical or fun to show off. Some of the commands even have multiple ways to be executed, which makes the speech recognition module feel more dynamic and smart. Each command is tested 3 times for consistency. The test results are shown in Table 3.

#	String Command	Execution	Passed (3x)
1	'je naam' 'jouw naam' 'hoe heet'	'Say' Mijn naam is Nao - 'ExecuteBehavior' Bow	Yes, Yes, Yes Yes, Yes, Yes Yes, Yes, Yes
2	'hello' 'hallo' 'hoi'	'Say' Hallo, hoe gaat het? - 'ExecuteBehavior' Waving	Yes, Yes, Yes Yes, Yes, Yes Yes, Yes, Yes
3	'spier' 'sterk'	'Say' Ik voel me sterk - 'ExecuteBehavior' ShowBiceps	Yes, Yes, Yes Yes, Yes, Yes
4	'push up' 'pushup' 'fitnes'	'Say' 1, 2, 3 - 'ExecuteBehavior' Pushups	No, Yes, Yes Yes, Yes, No Yes, Yes, Yes
5	'voetbal' 'schop'	'Say' Goal, 'ExecuteBehavior' Kick	Yes, Yes, Yes Yes, Yes, Yes
6	'sta op' 'opstaan'	'Say' Laten we gaan spelen, 'ExecuteBehavior' StandUp	Yes, Yes, Yes Yes, Yes, Yes
7	'squat' 'zit' 'rust'	'Say' Ik ben moe, 'ExecuteBehavior' Squat	No, No, No Yes, Yes, No Yes, Yes, No
8	'macarena'	'ExecuteBehavior' Macarena	Yes, Yes, Yes
9	'gangnam'	'ExecuteBehavior' GangnamLong	Yes, Yes, Yes
10	'vlieg'	'Say' Geef me eerst een kaap, 'ExecuteBehavior' LookAround	Yes, Yes, Yes

[Table 3: Test results of the available predefined commando's for RoboTutor]

The tests are executed on hard ground in TU Delft EWI by two different speakers. Besides on hard ground, it is also tested in carpet environment, which was not a good idea. The carpet adds friction to the robot's behaviors and it fell down when executing the 'Kick' behavior and 'GangnamLong' dance.

Evaluation

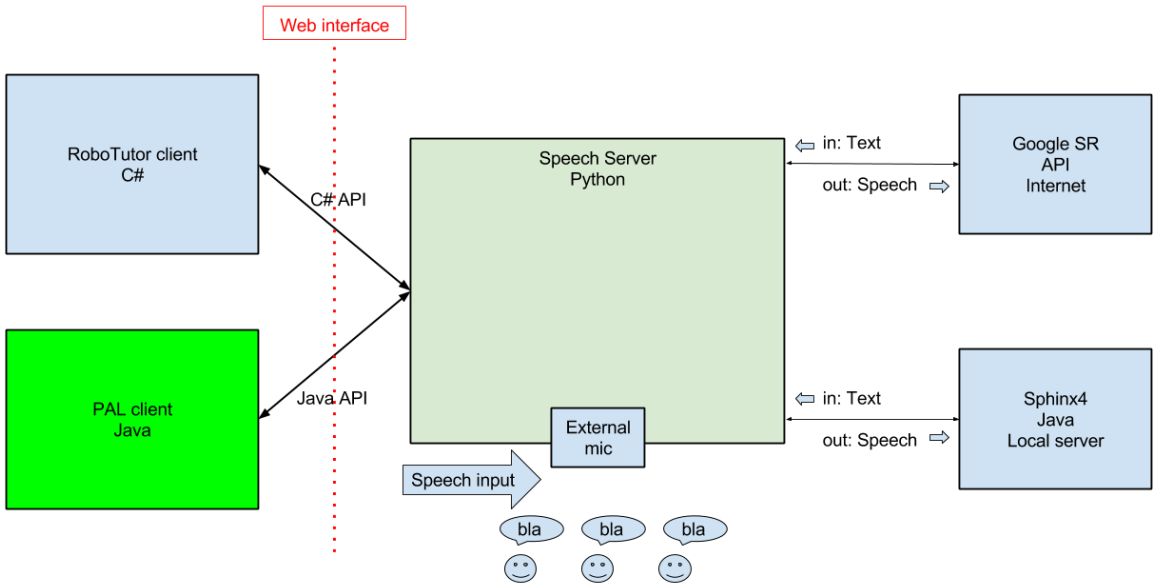
The test results in this cycle shows the speech server achieving a high score: 56 of 63 (89%). The main reason for this is the way the commands are asked. Not only the command are spoken, but a whole sentence is used and this results in better accuracy of the speech recognition module. For example 'Wat is je naam', 'Laat je spieren zien', 'dans de Gangnam style', etc. The command 'squat' scores 0% in the test, because the SR module keeps thinking the speaker said 'squad'.
The problems of last cycle is fixed in this cycle. With user feedback the user knows when the server is calibrating and that he/she has to be quiet. Implementing the record function that listens for 5 seconds in the speech server helps with removing the problem from last cycle where big sized audio data is sent to GSR. But there is a problem, if the user speaks for 2 seconds, he/she has to wait 3 extra seconds for the speech server to transcribe audio data.

Next cycle

After confirming that the speech server works with a C# RoboTutor client, in the next cycle the server is tested with a Java PAL client. The cycle has a lot of similarities with this cycle.

7 PAL

This cycle looks a lot like the previous cycle and is described less because they have a lot in common.



[Image 10: Project diagram with highligthed parts of the PAL client]

In this cycle a Java PAL client is attached to the speech server to use speech recognition (see Image 10).
The PAL system gets access to the speech server with the help of a client. There is no stable version of the PAL system yet. The start of this cycle starts exactly during the PAL hackathon week. In this week, all the PAL developers and teams gathered around to show their developments and test the current modules together.
The implementation of PAL is easier than in RoboTutor. RoboTutor's client uses the transcribed text to execute a command while PAL's client receives the transcribed text and forward it to another module which decides the logic of the data and what to do with it.
The language of the speech server has to be set to English because PAL uses an international approach.

Test
During the last day of the PAL hackathon, when all modules are stable the system is tested.

The following modules has to be running: Logical Manager, Dialog Manager, Speech Server, Speech Client, Physical Robot, Robot Avatar and the Quiz module.

And in the following order, the system is tested:

1. The robot asks the user for his/her name.
2. The user replies by saying his/her name.
3. The robot repeat the name in a logical way (example: 'Hi, how are you doing Peter') and asks if the user wants to play a quiz
4. The user replies with 'yes' and the robot starts a quiz
5. During each question, the robots asks the question and the user can choose a multiple choice answer

This test setup uses speech recognition and the quality of the Speech Server is tested.

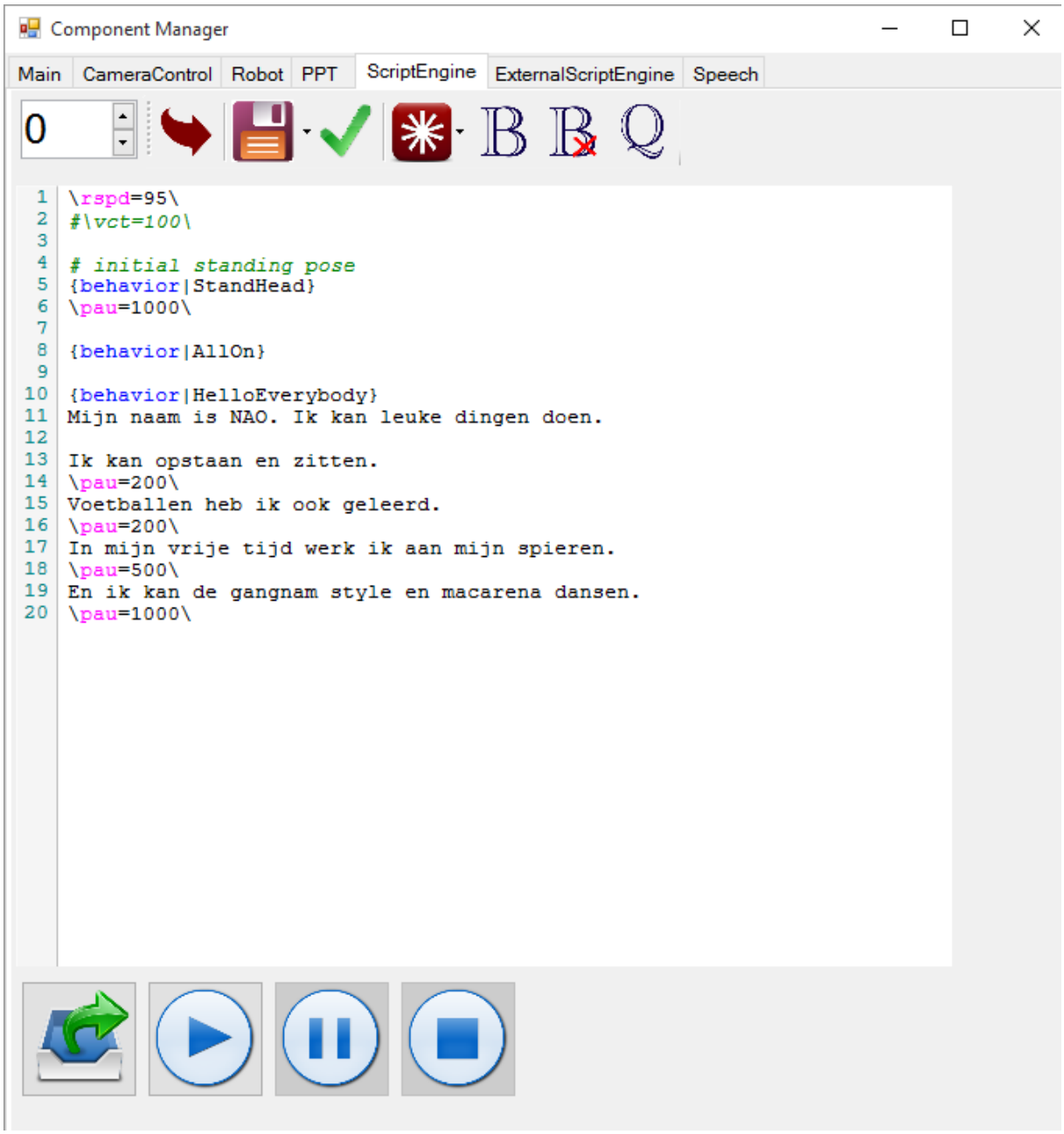
Evaluation
This cycle concludes that the speech server also works for the English language. English language actually works better than the Dutch language on the speech server. It is expected, because the English language is more international and there is more research and data about it.
Therefore the tests have a bigger success rate. The usage of SR is very simple and repeated as seen in the test setup. The speech server has to transcribe mostly yes/no answers or A-D because of the multiple questions. The only part where speech recognition is used dynamic is the name of the user.

Next Cycle
After this cycle the project plan has been achieved. Because there is still more time for one cycle, another feature can be challenged and added to RoboTutor.
There are a lot of challenges possible, such as:

1. Quiz for RoboTutor with speech
2. Number game (robot generates random number and the user must guess it, resulting in mood changes of the robot when it's incorrect)
3. Implementation of SR in the Script Engine
4. Implementation of a simple Sphinx4 Java speech recognition engine

From this list, the most interesting one is the implementation of SR in the Script Engine. This feature is the one with the highest priority and most logical to have. Otherwise the speech server can only be used to execute one of the 10 commands and not dynamically used for other modules such as mood behavior and PowerPoint presentation. This is achieved by implementing SR into the Script Engine of RoboTutor.

8 Speech Recognition in Script Editor



[Image 11: Screenshot of an example script in the Script Editor]

The Script Editor module makes it possible to use different modules of RoboTutor in a chronological order. With the script editor, the user can define the presentation of the robot by controlling what the robot should say and execute (see Image 11).

This chapter describes two cycles taken to implement SR in the Script Editor. A static use of the SR module is implemented first and then this feature is expanded to be more dynamic.

8.1 Static

In this cycle, the SR module is implemented in the Script Editor in a static way to use, but the functionality of speech recognition is available.

{listen|parameter}
The script 'listen' has the same functionalities as the graphical user interface with buttons from RoboTutor. But here the functionalities are executed by script and not with buttons.
The script 'listen' connects the speech server with 'connect' as parameter and runs it with parameter 'run' to listen to a command. The 'close' parameter terminates the connection with the speech server. Example of the script is shown in Image 12.

```
1 {listen|connect}
2 Wat wil je dat ik doe?
3 {listen|run}
4 {listen|close}
```

[Image 12: Example how the script 'listen' is used in the Script Editor]

8.2 Dynamic

The big challenge here is using SR to change the flow on how the Script Editor runs. This makes the robot more dynamic during presentations. A practical scenario is when the robot finishes PowerPoint slide 1 and asks the crowd if they understand it before going to the next slide. If the audience answer 'no', then slide 1 is explained in a different or more detailed way and if the answer is 'yes' the robot continues with slide 2.
There are 2 ways considered to accomplish this dynamic feature:

1. Jump to another line in the Script Editor.
2. From the current script, open another script and execute it.

Both methods have their advantages and disadvantages. In method 1 it's very easy to lose track of the situation which causes confusing while in method 2 the script is still run chronological and easier to track.
Method 2 is chosen and is implemented in this project, because method 1 is very confusing to use.

{script|parameter1} or *{script|parameter1|parameter2}*
The script feature takes 2 parameters in. The first parameter is the directory of the follow up script that needs to be executed and the second parameter is optional. With the second parameter the user can let the robot ask a question which request a yes/no answer and the follow up script can only be executed if the user say 'yes'. That means if there is no second parameter, the Script Editor just executes the follow up script.
The follow up script is executed by stopping the current script, loading the follow up script from the directory and executing it. Example of this script is shown in Image 13.

```
1 \pau=500\
2 Wat wil je dat ik doe?
3 {listen|run}
4 \pau=1000\
5 {script|speech/command|Wil je meer zien?}
6
7 {listen|close}
```

[Image 14: Example how the script 'script' is used in the Script Editor]

9 Usability Test

During the end of the project a usability test is executed as an experiment to see how well the speech recognition module of RoboTutor works in 'real life'.

The following test setup is being tested:

1. The speech server is started with GSR as SR engine.
2. The RT system is started and connected to the speech server.
3. A virtual robot is run in Choregraphe Suite and connected to RT.
4. Participants for the experiment

This experiment consists of 6 participants: one child aged 6, two adults aged 25-35 and 3 young adults aged 18-25. The participants have a variety of knowledge base: logistic consultant, software developer, junior app developer and students of computer science and marketing. The participants are not chosen, they just happen to be the available ones to participate with the experiment.

A virtual robot is used, because the NAO robot is not available during the test. All the behaviors of the physical robot has to be loaded inside the virtual robot to be able to execute the predefined commands.

The predefined commands are shown on screen for the participants. They have no knowledge how the system works and this test shows how usable the system is to new users.

The group can shout out a command for the virtual robot to execute. Their interactions are being monitored during this process.

The following are being observed:

1. How fast the user performs the tasks.
2. Irritation when the speech server is faulty.

Evaluation

Watching the users interacts with the robot is better than interviewing them about their own behaviors, because users do not realize some unconscious behaviors.

Observed points during this usability test:

1. The participants are very quiet and observant to the robot during the start.
2. The tasks are executed mostly by just reading the command, only the technicians used a creative approach. Example when executing command 'pushup', most users read the word while the creative ones used a sentence 'Can you please do 10 pushups'. The robot does not do 10 pushups, but it can still execute the behavior pushup.
3. The participants are mostly amazed by the dances of Gangnam style and Macarena.
4. During faulty times of the speech server, participants tend to lose patience and start to raise their voice and shout the command repeatedly. This is the case with the command 'squat', which scored 0% during the test of RT commands (see Table 3).
5. During faulty times of the speech server, participants move their head closer to the robot which results in better accuracy. But this only works because a virtual robot is used. If the physical robot is used, then the accuracy won't get higher because the speech server listens to the microphone on the laptop. During this test both the speech server and virtual robot are run on the same laptop.

10 Reflection

My internship was very beneficial for me. I improved my technical skills and gained knowledge in new areas. During the start of the project I had no knowledge on speech recognition. Research helped me understand how speech recognition works, how it needs to be applied and how to use a speech engine.
I also gained experience in the research field which was provided less during my Bachelor study. Working on a research topic for the first time introduced me on how difficult and important speech recognition is.
The speech server was built in Python, this was a new programming language for me and a great opportunity to add it to my skills.
Besides working alone on RT, I was also able to work as a team in PAL. Therefore I gained working experience as an individual and also in teamwork. I also learned during this internship how important responsibilities and commitments are in an organization or team.
My education at Technische Informatica helped me a lot during this internship, one course who helped me was Design Patterns. With the pre knowledge of Design Patterns I was able to apply an Abstract Factory design pattern (see Image 5) in the correct way.

Competences
This section proves that the pre-defined competences has been achieved during this internship period. Only 2 of them are being described.

C8 – Designing a technical information system
A throughout analysis had to be done to search for an appropriate software development methodology to use during this project. This resulted in better efficiency in the working process and saving time and risks.
The design for this system was minimal, because more focus was set on programming and experimenting speech recognition. Therefore it was expected that the requirements changed constantly.
The system had to be designed for an Object Oriented system, because Object Oriented languages are used to realize the software. UML diagrams such as class- and sequences diagrams are used to design how the system works before actually implementing it. Applying design patterns in the design helped with achieving the requirements changes that are expected and also the possibility to easy attach/detach a speech engine. If the SR engine is removed and replaced by another, no code has to be changed for the class 'Server'. The rest of the system will not be affected by these changes. The design also shows how the systems communicates with each other: through client-server communications.

D16 – The realization of software
During this internship 3 programming languages were applied: C# in RoboTutor, Python in the speech server and Java in PAL. The choice of Python was a great opportunity to learn a new programming language and apply it to the project.
Using GIT as version control during this project was smart. This way it was possible to work on the same software version on different computers, which made it easy to work at home too.
All 3 programming languages has been used Object Oriented during this project. This means the code is implemented in a way that even if the system changes, most of the code is reusable. A good example is the use of the speech server. Because applying the abstract server, changing the SR engine will not affect the class Speech Server.
At the end of each cycle the code was refactored to be able to be reused and easy to extend.

11 Conclusion

In the conclusion the results, process, objective and problem are evaluated to determine what is accomplished after the internship period.

Result
The result is quoted in chapter 3.4 Results as:

Assignment is accomplished when the RoboTutor speech recognition system works using a microphone in a room with more than 10 people that can choose from 10 different predefined commands. The selected commands must be executed successfully by the robot. Only one command can be performed at the same time and after executing this command, the audience can choose for a new command.

The achievement of the result is shown with the test results of RT commands (see Table 3) and the Usability Test (see chapter 9). These tests show that the speech recognition module of RT works in a group, even though the usability test is executed with 6 participants compared to the 10 participants defined in the result quote. 10 predefined commands are also achieved and tested, which scored an 89% success rate.

Process
The evolutionary software development methodology is used during this project (see chapter 3.5). This method is a great choice, because the requirements changed very fast especially in the beginning and the development process uses cycles where the requirements, design and code can be changed after test and evaluation. Using this methodology in research projects is very good, because the field is unknown and hard to predict.

The biggest problem during this internship is the lack of documentation of the previous version of RT. With only the source code as help, it is very difficult to understand how the modules works and why they are implemented that way. During implementation of the RT client with user interface buttons (see Image 9), only one other module in RT has to be examined. But during implementation of the speech recognition module in the Script Editor (see chapter 8 Speech Recognition in Script Editor), the lack of documentation causes cycle delays and bugs.
This can be prevented by using a lot of time during the analysis process to document the current system with help of a technical expert. But the question is whether the documentation will be as accurate as meant by the one who programmed it.

Objective
The objective is quoted in chapter 3.3 Objective of the Assignment as:

RoboTutor is extended with a speech recognition feature. The robot is able to hear sentences now, recognize them and perform a command. Commands are added to RoboTutor and the robot can generate a logical reply and/or perform an action. With this new technique of human-robot communications, RoboTutor is one step closer to be more human like and assist a teacher in class.

Before the start of this project, the robot can only talk but not listen. Now the robot can listen thanks to speech recognition and reply or perform an action. Hearing is one of the 5 senses of a human being and is achieved with speech recognition, but the speech module is not this complex yet to achieve human intelligence.

Problem
The problem is quoted in chapter 3.2 Problem Description as:

Currently it is not possible for the software of RoboTutor to understand what the audience is saying to the robot, it can only give a presentation and cannot receive feedback.
To make it more interactive the robot has to be able to understand the audience and give a reply or perform an action. This is achieved with the new feature of speech recognition.

Besides giving a presentation, the robot can also listen to its audience now. But listening and executing every possibility the audience ask for is not possible. The robot only executes it if the command is predefined in the software. But the problem is solved, the robot is more interactive now and can listen to the audience.

12 References

Amber, S. W. (2012). *Agile Modeling and eXtreme Programming*. Retrieved from http://agilemodeling.com/essays/agileModelingXP.htm

BaiduUSA. (2015, March). *Deep Speech: Lessons from Deep Learning*. Retrieved from http://usa.baidu.com/deep-speech-lessons-from-deep-learning/

CMUSphinx. (2015, February). *CMUSphinx Tutorial for Developers*. Retrieved from http://cmusphinx.sourceforge.net/wiki/tutorial

Craig Larman, V. R. (2003, June). *Iterative and Incremental Development: A Brief History.* Retrieved from http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf

Gamma, E., Johnson, R., Helm, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Ghahramani, Z. (2001). *An Introduction to Hidden Markov Models and Bayesian Networks.* Retrieved from http://mlg.eng.cam.ac.uk/zoubin/papers/ijprai.pdf

Google Inc. (2012, October 19). *Web Speech API Specification*. Retrieved from https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html

Jython. (2011, March). *The Jython Tutorial*. Retrieved from http://www.jython.org/docs/tutorial/indexprogress.html

Knoder, J. (2015). *Voice Recognition Software Reviews*. Retrieved from TopTenReviews: http://voice-recognition-software-review.toptenreviews.com/

Mozer, T. (2015, December 17). *Deploying speech recognition locally versus cloud*. Retrieved from http://embedded-computing.com/guest-blogs/deploying-speech-recognition-locally-versus-the-cloud/

Nuance. (2015). *Dragon Naturally Speaking 12 Datasheet*. Retrieved from http://www.nuance.com/ucmprod/groups/dragon/@web-enus/documents/webasset/nc_008810.pdf

TutorialPoint. (2016). *SDLC - Waterfall Model*. Retrieved from http://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

voice-commands.com. (2005). *Voice-Commands e-Speaking*. Retrieved from http://www.voice-commands.com/103.htm

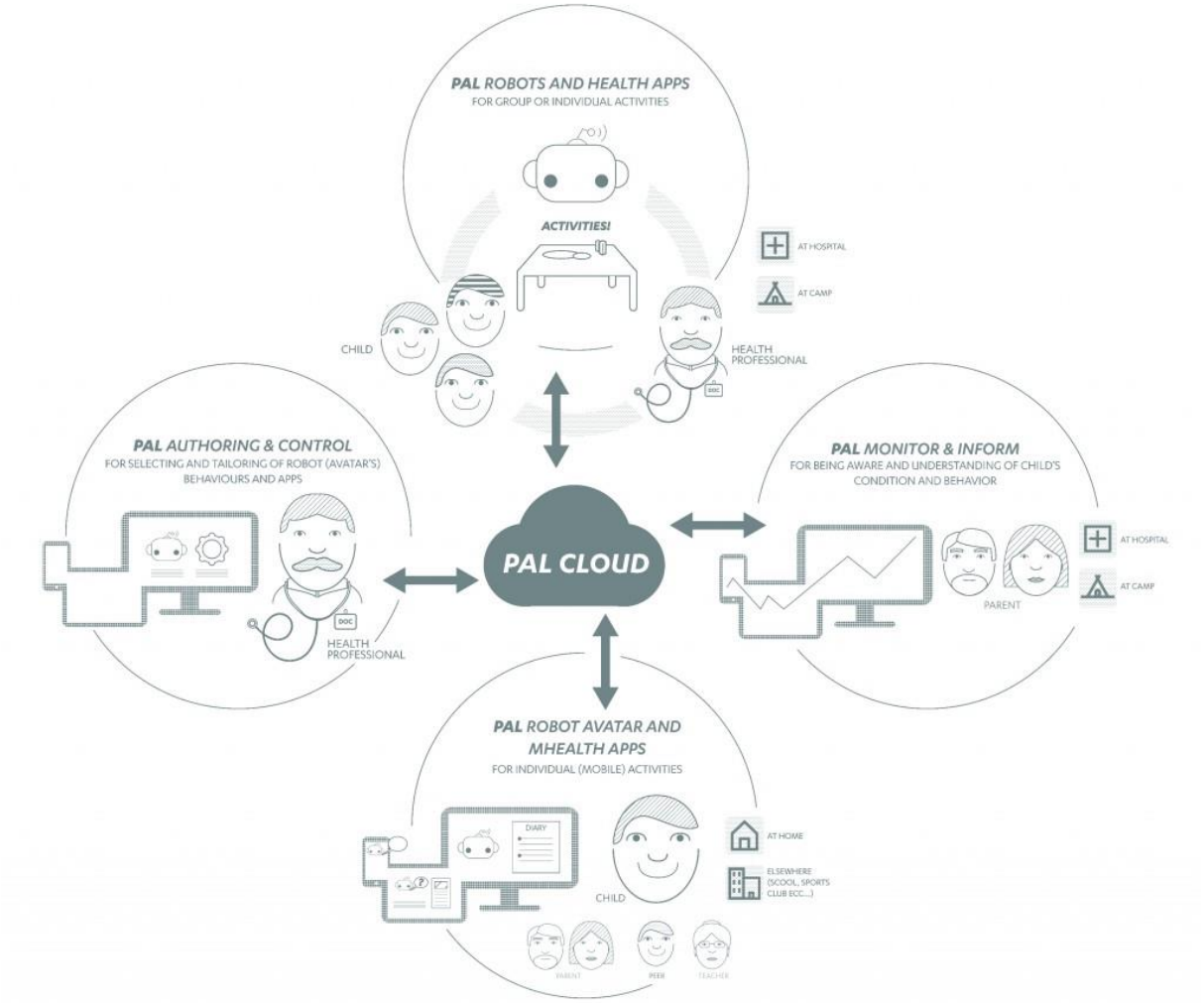
Wildstrom, S. (2011, October 10). *Nuance Exec on iPhone 4S, Siri, and the Future of Speech*. Retrieved from https://techpinions.com/nuance-exec-on-iphone-4s-siri-and-the-future-of-speech/3307

Xu, J. (2015). *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction.* Delft: TU Delft.

Zhang, A. (2015). *GitHub Speech Recognition*. Retrieved from https://github.com/Uberi/speech_recognition/

13 Appendices

Appendix A - PAL Cloud



Appendix B – Speech Recognition Research

Appendix C – Stageplan

Appendix D – Test rapport

Appendix E – Bedrijfsorientatie