ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

STL SYSTEMTECHNIKLABOR

Development of Voice User Interfaces and their Impact on the User Experience of Mobile Applications

Kevin A. Münch

Technical Report – STL-TR-2017-03 – ISSN 2364-7167

stl.htwsaar.de

Technische Berichte des Systemtechniklabors (STL) der htw saar
Technical Reports of the System Technology Lab (STL) at htw saar
ISSN 2364-7167

Kevin A. Münch: Development of Voice User Interfaces and their Impact on the User Experience of Mobile Applications
Technical report id: STL-TR-2017-03

First published: October 2017
Last revision: October 2017
Internal review: Ramon Mata-Toledo, André Miede

For the most recent version of this report see: https://stl.htwsaar.de/

Title image source: Flavio Takemoto (flaivoloka), http://www.freeimages.com/photo/1213712

htw saar – Hochschule für Technik und Wirtschaft des Saarlandes (University of Applied Sciences)
Fakultät für Ingenieurwissenschaften (School of Engineering)
STL – Systemtechniklabor (System Technology Lab)
Prof. Dr.-Ing. André Miede (andre.miede@htwsaar.de)
Goebenstraße 40
66117 Saarbrücken, Germany
https://stl.htwsaar.de

ingenieur
wissenschaften
htw saar

JAMES MADISON UNIVERSITY.

Master's Thesis

For the degree of
Master of Science (M. Sc.)
At University of Applied Sciences, Saarbrücken
And in cooperation with James Madison University
In the major Applied Computer Sciences
At Fakultät für Ingenieurwissenschaften

Development of voice user interfaces and their impact on the user experience of mobile applications

Submitted by
Kevin Alexander Münch

curated and surveyed by
Prof. Dr.-Ing. André Miede
Prof. Ramon Mata-Toledo, Ph. D.

Harrisonburg, VA, 09/30/2017

Statement of Authorship

I hereby declare that I am the sole author of this master thesis and that I have not used any sources other than those listed in the bibliography and identified as references. I further declare that I have not submitted this thesis at any other institution in order to obtain a degree.

Harrisonburg, VA, 09/30/2017

Kevin Alexander Münch

Abstract

Developing voice user interfaces is a complex process that requires steps to accept text as input, process this input and synthesizing an adequate output. Understanding the meaning of the vocal words and generating an answer is the key part of this process. It is handled by a dialog manager. The dialogs that the system creates must be designed in a way that the user can accept it easily. Otherwise, the cognitive load is too high for most users to interact with the interface. However, in hands-free situations, visually distracted situations or for disabled people voice user interfaces help users to interact with the system and are therefore in special interest of the current software development processes.

There are several ways to design dialogs and the interface in a way that the user can easily use them. The dialogs need to be kept simple and structured well, list should be filtered and kept short. The design of the interface can combine visual and audible parts to help the user keeping track of the data. Using more than one modality is one of the easiest ways to create a user interface which requires little cognitive load. This solution is known as multimodal interface.

This thesis explains these rules by example. Therefore, an Android application called TaleTime is developed. This application lets the user experience interactive fiction. Stories are read out chapter by chapter. After each chapter, the user can decide how to the plot of the story goes on. This can either be done by tapping buttons on a screen or using speech recognition. Dialogs must be kept simple because the target audience of this application are young children. The development process is documented and the design of dialogs can be seen in example. This includes example dialogs, flow diagrams and the concrete wording of the dialogs. In addition, it is explained how a simple dialog manager can work. Visual component are used to keep the cognitive load low if the user missed the auditory output of the voice user interface.

This thesis summarizes existing literature and collects best practices to develop voice user interfaces. Additionally, other types of user interfaces and typical use cases of voice user interfaces are mentioned as well. These best practices are then applied to the example of the development of the voice user interface for TaleTime.

Great design simplifies a very complicated world.

— Platon Antoniou

Acknowledgments

First, I would like to thank my thesis advisor Prof. Dr.-Ing. André Miede of the Fakultät für Ingenieurwissenschaften at Hochschule für Technik und Wirtschaft des Saarlandes. Even that Prof. Miede was in Germany during the work on the thesis, he helped me whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it.

I would also like to acknowledge Prof. Ramon Mata-Toledo, Ph.D. of the Department of Computer Science at James Madison University as the second reader of this thesis and local advisor during the work on this thesis. I am gratefully indebted for his very valuable comments on this thesis.

Furthermore, I would like to acknowledge Prof. Steven Frysinger, Ph.D. of the Department of Computer Science James Madison University who set up the contact and made it possible for me to study in the US.

In addition, I would like to thank the students and their advisors from htw Saar and Saarland's university for art and design who worked on the application (TaleTime) as their project work before and during the research and the work on this thesis. Without their passionate participation and input, this project could not have had such a successful result.

I would also like to acknowledge Sebastian Düppre as second reader of this thesis, and I am gratefully indebted to his for his very valuable comments on this thesis.

Finally, I must express my very profound gratitude to my parents and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Kevin Alexander Münch

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem description . . . 2
- 1.3 Thesis structure . . . 3

2 Background **5**
- 2.1 User interfaces and user experience . . . 5
  - 2.1.1 Types of user interfaces . . . 5
  - 2.1.2 User experience . . . 7
- 2.2 Multimodal interfaces . . . 8
  - 2.2.1 Multimodality . . . 9
  - 2.2.2 Usage to lower cognitive load . . . 11
  - 2.2.3 Designing for mobile devices . . . 12

3 Conversational Design **17**
- 3.1 Linguistic basics of conversations . . . 21
- 3.2 Turn taking . . . 25
- 3.3 Avatars . . . 26
- 3.4 Chatbots and personalized assistants . . . 27
- 3.5 Impact to the user experience . . . 28

4 Design of voice user interfaces **29**
- 4.1 Requirements analysis . . . 29
- 4.2 High level dialog design . . . 34
- 4.3 Detailed dialog design . . . 39
  - 4.3.1 User confirmations . . . 41
  - 4.3.2 Stop detection . . . 42
  - 4.3.3 Helping the user . . . 43
  - 4.3.4 Error handling . . . 44
  - 4.3.5 Voices and intonations . . . 46

5 Technological design **49**
- 5.1 Speech recognition . . . 50
  - 5.1.1 General technologies . . . 50
  - 5.1.2 Comparison of existing speech recognition software . . . 53
- 5.2 Natural language understanding . . . 56
- 5.3 Dialog Management . . . 57
  - 5.3.1 Finite state based . . . 58
  - 5.3.2 Frame based . . . 58
  - 5.3.3 Agent based . . . 59
  - 5.3.4 Plan based . . . 60
- 5.4 Speech synthesis . . . 61
  - 5.4.1 Methodology . . . 61
  - 5.4.2 Intonation . . . 62

5.5 Summary . . . 63

6 Implementation **65**
6.1 Existing prototype . . . 65
6.1.1 Used technologies . . . 68
6.1.2 Open tasks . . . 68
6.2 Requirements definition . . . 69
6.3 Definition of grammar . . . 71
6.4 Development . . . 73
6.4.1 Technological basics . . . 74
6.4.2 Architecture . . . 75
6.4.3 Components . . . 76
6.4.4 Open problems . . . 83
6.5 Summary . . . 84

7 Conclusion **85**

Bibliography **87**

List of Figures **97**

List of Tables **97**

Listings **97**

Abbreviations **99**

A package.json of the TaleTime project **105**

B English grammar file for the TaleTime VUI **107**

C Results of Google Speech API as JSON **111**

D Results of Microsoft's Speech API as JSON **113**

1 Introduction

The development of voice user interfaces (VUI) is a complex process. It includes many steps and requires a lot of work. This thesis therefore explains the process and gives inside information about how VUIs can successfully be developed.

Besides a description of the necessary development tasks, this thesis also includes an example for the development of a VUI. This example exists as an application initially developed by htw Saar and is described in Chapter 6.

1.1 Motivation

In the past few years, the use of VUIs has been on the rise. For a long time, VUIs were only the interest of researchers and developers. The reason for this is because VUIs are a good, usable alternative to graphical user interfaces (GUI). The main reason why people tend to use their voice instead of a keyboard is because it is a more natural form of input. Humans are used to conversations. They use their voice every day to communicate. In addition, the usage of spoken language is highly perfected. Although scientists are not exactly sure about when humans began to speak and develop a language, it is assumed that they started at least 100,000 years ago [58]. Compared to written language, this is a very long time. Humans started writing approximately 5000 years ago [28]. This proves, that humans are used to spoken language and they were able to learn it a long time ago. Almost all humans all over the world are able to speak. Although most people also know how to express themselves using written language, the number of people who know how to write is still only at 85% [113]. As a result of these statistics, it is easier to see why people prefer speaking as a means to communicate.

However, when interacting with a computer the most common way to enter data is via the keyboard. The keyboard was invented for using it with typewriters in 1868 [52]. Even if this also seems to be a long time ago, many people are still not used to the usage of a keyboard or feel uncomfortable using it especially when writing long texts. This is another reason where VUIs can be helpful. A VUI accepts a human's voice, which everyone is used to, as input. This obviously eliminates the use of keyboards and increases the amount of input because it is a more natural way to communicate. Also, it helps young children to interact with a VUI even if they do not know how to use a keyboard or write yet. Due to the increase in use of VUIs, it is very important to create them with the highest quality possible. If users begin to struggle using these interfaces, they quickly get disappointed and will avoid their use. This thesis also summarizes ways to keep the quality of VUIs high, so users feel confident in using them.

Besides the more natural and easier way to input data as a main argument to use VUIs, there are other reasons why the development of VUIs is an important subject of research. There are situations where users cannot use their hands to use a keyboard. The most popular example for this situation is having the need to text or memo while driving. With VUIs people can simply dictate what they want to achieve instead of using touch based input methods. Other examples show that the usage of a VUI instead of touch based input methods can increase their usability. One of these examples is the usage of a smartphone application to cook. Users can talk to their phone while using their hands to cut, knead or

accomplish other tasks of the cooking process. Giving a user the opportunity to interact with the phones applications using his voice increases the user experience enormously. The user does not have to stop working and can accomplish multiple tasks at once.

The usage of the voice also helps expressing emotions. Although GUIs can use different forms of highlighting to indicate the importance of certain pieces of text, voice can only use the intonation to indicate different meanings. This is still part of research, as the synthesis of voices expressing emotions is a tough challenge. However, it opens a huge variety of possibilities to express data in a more detailed way. Using emphatic voices, the method of input and output carries not only the information described by the words themselves but the importance or urgency that the speaker gives to the message [90].

In combination with recent achievements in technology and especially in the field of artificial intelligence, VUIs have become more usable. Another reason why the VUIs are also more popular is due to smartphones which provide a speaker and also a microphone; the main components needed to create a VUI. Moreover, they are widely used today. In addition, most phones sold today provide enough processing power to make VUIs widely usable. In fact, 65% of the phones sold in 2015 provide the ability to use virtual assistants [74].

There are also a few negative aspects that will be discussed in section 2.1.1. This includes the usage of VUIs in public spaces and to enter private data. However, VUIs will only be used if the quality of dialog is high enough. This thesis concentrates on theories that help developers keeping the quality of a conversation high. One of the key aspects of a good VUI is natural language understanding (NLU). Especially through the progress in the research regarding artificial intelligence the quality of NLU was highly increased. Based on this fact and the previously mentioned fact that smartphones are now widely used, the usage of VUIs is increasing heavily. Smart watches and the digital interaction in cars and other products pushing the usage of VUIs as well. As a result, users are trying to use VUIs in all different situations now. Unfortunately, the management of dialogues is still often not very helpful. Keeping track of previously mentioned facts and creating a conversation based on this knowledge is often not possible for many applications. This thesis therefore covers methods to build more conversational interfaces also as methods that help to develop highly useful VUIs. These topics are important, because the usage of VUIs will increase in the future as well.

1.2 Problem description

This thesis addresses two main issues. First, it summarizes the existing literature concerning the creation of usable voice interfaces. Therefore, only the best practices will be described here. Since the development of VUIs started in the 1960s and became very popular in the 1990s, the methodologies have changed over time. Because of this reason, this thesis mainly covers modern topics. These changed because of the rise of personal assistants used on smartphones and on devices for connected homes such as Amazon's Echo or Google's Home. These assistants offer more functionalities to build VUIs than older devices. Therefore, different workflows are needed which requires to include the creation of fluent and useful dialogs. Dialogs must kept simple because users can only handle a certain amount of information, especially not if only heard. Designers use flow diagrams and example dialogs as a first step of the design process. Then, more technical problems like the usage of pauses will be discussed as part of this work. Pauses are very important in a conversation. They may not only indicate that the speaker has finished speaking but also give the participants of a conversation more time to understand the given information. Moreover, pauses influence the implied meanings of a conversations.

Studies have shown, that different lengths of pauses can influence the emotions, participants of a conversation may have [26]. This needs to be considered while creating a VUI and will be discussed in section 4.3.5.

Secondly, this thesis considers multimodal interfaces. With the rise of smartphones and similar devices, multimodal interfaces are becoming very common. Smartphones include a screen, so data can be presented and altered using the screen (GUI). In addition, Smartphones contain a speaker and a microphone through which data also can be displayed and entered, respectively (VUI). So, it is important to make sure that users can interact as easy and as intuitive with these two types of interfaces. This thesis considers ways to reduce the cognitive load for the user through multimodal interfaces and suggests guidelines to create devices where both the GUI and VUI work together well.

The most important part of a VUI is the creation of fluent conversations. Therefore, the VUI needs to keep information about what has been said. That is, it needs to keep a history. Human conversations are usually based on such histories. Each participant remembers what was said earlier, at least for a certain time. This is what the VUI also needs to be able to do. There are multiple strategies on how to keep track of the information already given. This thesis will try to present the most recent aspect of this important issue. It is very critical to manage the given information, because people use different words to refer to what was said. These can be pronouns such as *he*, *she* or *it* to refer to people mentioned before, but there are of course many other possibilities. Often, places and dates are replaced by other words. The VUI system must be able to recognize and associate them with the correct information in the same way humans do it. If it is unclear, which piece of information was mentioned, the VUI must be able to figure that situation out and disambiguate it by asking the user for clarification. Disambiguation is a very important part of a conversation. It is used very often in human conversations and helps to let the VUI seem more confident. The methodologies used to keep track of the system's states will be described in section 5.3. This is the core of the thesis. These methodologies will also be used while development in a simplified version.

Although these are the main topics of the thesis, the development of VUIs covers many more topics such as the voice recognition itself, NLU or the synthesis of speech. These topics will not be covered in detail by this thesis. However, some of their main ideas will be described briefly and some references will be given for the interested reader.

1.3 Thesis structure

This thesis is structured in seven chapters. These chapters follow the workflow of the development of VUIs. After the introduction, Chapter 2 describes all background information needed to understand the context of VUIs. This includes an overview over user interfaces and an explanation of the user experience in general. Furthermore, the term of multimodality is explained because VUIs are often used in a multimodal context. Chapter 3 describes the the basics of conversations. This chapter reviews linguistic basics as well as turn taking strategies. In addition, typical use cases of VUIs such as chatbots and personal assistant are described. The development process of VUIs is described in Chapter 4. It sequentially describes all steps in a typical development process starting from the requirement analysis through the detailed dialog design. Whereas Chapter 4 describes the design of VUIs, Chapter 5 gives some technological background about the underlying methodologies. Each distinct part of a typical VUI system is described from a technical point of view here. Chapter 6 describes the development work done during this thesis. It documents the implementation and gives more information about how the technologies can be applied. Finally, Chapter 7 summarizes the whole work and gives a final overview.

2 Background

This chapter introduces the reader to the work of this thesis. It explains the basics regarding user interfaces. This knowledge is necessary to understand why VUIs are needed and what typical use cases can look like. Furthermore, this chapter explains multimodality and especially, how interfaces can be combined in a useful way.

2.1 User interfaces and user experience

This chapter will set the user up with general knowledge about user interfaces. This is important, because VUIs are often used in combination with other interfaces (multimodality). Additionally, this section shortly explains the difference between a user interface and user experience.

2.1.1 Types of user interfaces

This section gives an overview about the types of user interfaces currently available. This thesis main concern is about VUIs, but because all user interfaces have weaknesses, it is important to know what other kinds of user interfaces exists. This is especially relevant, because different types of user interfaces can be combined. The resulting interface is called multimodal interface and is explained in more detail in section 2.2.

There are three main groups of user interfaces: Visual or graphical user interfaces (both abbreviated with GUI), VUIs and gesture based user interfaces. This chapter explains the basics of all of them.

Visual user interfaces

Visual user interfaces, often also called GUIs, are the most popular types of interfaces. They rely on a screen, which is used as the primary output interface. Today, screens often offer touch functionalities. This enabled the user to directly alter the data on the screen. Older versions of GUIs needed another form of input. This is usually a keyboard and / or a mouse.

There are different versions of GUIs. First, there is the usual GUI which interacts with the user through touch or mouse and keyboard. The key elements are windows, buttons, text and icons that can be altered directly by interacting with them.

Then, there are command line interfaces which are only based on text. Those were the first versions of visual user interfaces that use a screen. The user cannot click on icons or use windows, the only way of manipulating data is to use commands typed via a keyboard. Especially for inexperienced users, graphical user interfaces are easier to use. They are more vivid. However, text based command line interfaces are still popular. They are easier to develop and sometimes, depending on the use case, faster to use than GUIs. There are guidelines, which explain when to use which interface [78].

The third kind of visual interfaces are menu driven interfaces. They can offer graphical symbols but they do not allow the user to freely alter data. In most cases, they offer an entry of menu items which the user can select from using physical buttons or touch. A

popular use case are automatic teller machines (ATM). They offer only a few functionalities. The user can only make selections on the options available and no others. Limiting the possibilities of input results in higher security. Menu driven interfaces can be seen as a mix of command line interfaces and graphical user interfaces. They seem to be graphical and can display icons, text and color but they only offer a certain number of commands which are similar to command line interfaces [29].

Voice user interfaces

VUIs interact with the user by accepting the voice as input and responding with synthesized or prerecorded voice through a speaker. These kinds of user interfaces can therefore be used for both input and output. Since VUIs are the main topic of this work, they will not be explained in detail in this chapter. However, the main reason to use a VUI instead of another interface is that it is a better option in certain situations. Those include situations which only have limited space for a keyboard or limited space on the screen itself. Furthermore, VUIs can help creating interfaces for disabled users. Sometimes, when the interaction requires the input of long texts, most users prefer speech as an input method because speech is always faster than typing on a keyboard. Lately, there are many situations where users cannot be distracted or are limited in the use of their hands or their eyes. Examples are while driving, cooking or doing sports [98].

Gesture based interfaces

This kind of interfaces can recognize the movements or gestures of the user. Compared to other types of interfaces, gesture based interfaces are often used for input only instead of both input and output. This is the case because gestures can be recognized easily with a camera, but creating gestures requires a much more complex system. However, the field of robotics is still very interested in creating and using gestures as an output system. Also, gestures are important for multimodal applications which use an avatar [90].

Using gestures as an input method can simplify the interaction for the user. This is the case, because designers can use knowledge that the user already has. With gestures, the content directly becomes the object to be controlled. The designer can borrow one-to-one interactions from the real life to interact with the interface. For example, moving clockwise can be used to go forward, whereas moving counter-clockwise can be used to go backwards. Most users would expect a gesture to do exactly this. Also, designers should apply physics to objects. Most users interact with virtual objects in the same way as if the object would be physical. A good example is a list. To create a new list item, there has to be more space. Designers can use this fact to create a gesture that virtually separates list items to create more space for a new item. Gestures are usually used as shortcuts. They are simple and can only express simple interactions. Furthermore, designers should be careful to not implement too many gestures as the user can easily get confused. This is especially the case if the recognition systems quality is not high enough to separate between nuances of different gestures. Not interpreting the correct gesture can annoy the user quickly [16].

Other interfaces

The three types of interfaces are most important and popular ones, but naturally there are many more types of interfaces.

One of these interfaces are tactile interfaces. They are usually used as an output system. Often, their main function is to warn users. This can happen in cars, either to prevent the driver from falling asleep, when the driver hits the outer boundary line of the road or if the driver wants to swap lines and does not see another car in the blind spot. Also, aircraft manufacturer make use of this system to alert pilots about a possible stall [98].

2.1.2 User experience

Apart from the term user interface (UI), there is another term often used in this context. This term is the user experience (UX) which describes the whole experience a user has while making use of an application. As Jan Weddehage points out, "UI is the vehicle, but UX should drive" [117], the user experience is the concept of how to convey the content to the user but the UI is the method how this is done. In conclusion, this means that the UX does not make use of a certain UI. Which UI should be used depends on what the UX designer decides is best.

Nevertheless, it should be mentioned that the term UX is often used with different meanings. Sometimes, UI and UX are mistakenly used interchangeable. There are many different definitions of UX. The major ones are the following:

- A consequence of a user's internal state (predispositions, expectations, needs, motivation, mood, etc.), the characteristics of the designed system (e.g. complexity, purpose, usability, functionality, etc.) and the context (or the environment) within which the interaction occurs (e.g. organizational / social setting, meaningfulness of the activity, willingness of use, etc.) [48].
- A set of material rendered by a user agent which may be perceived by a user and with which interaction may be possible [116].
- Comprehends all aspects of digital products and services that users experience directly - and perceive, learn, and use - including products' form, behavior, and content, but also encompassing users' broader brand experience and the response that experience evokes in them. Key factors contributing to the quality of users' experience of products are learnability, usability, usefulness, and aesthetic appeal [32].
- A person's perceptions and responses that result from the use or anticipated use of a product, system or service [106].

The definitions given above show that there are many different opinions about the meaning of the term user experience. The website *allaboutux.org* [114] gives an overview of over 27 different definitions.

However, most of these definitions share similarities. The definitions described above show, that user experience is not only about a certain way of designing the user interface. It is also about understanding the user's needs and the users feelings. This results in guidelines that can be completely different than the usual design guidelines used by a company. User experience also includes that the user, in most cases also a customer, experiences a story, not only a simple website. The brand, the design, the language, the images but also the clarity and the learnability must fit together and create a pleasant experience for the user.

It is obvious, that the customer experience does not only include the design of the GUI, but of all interfaces used. As described in section 2.2 this can be a combination of multiple UIs. Especially when multiple user interfaces are used, it is important that all together create a joint user experience. Therefore, this also includes the design of the VUI. The way,

a device talks to a user or even the way internal data is used for suggestions can influence the user experience with a particular brand.

2.2 Multimodal interfaces

Interfaces, which use more than one of the user interfaces described in section 2.1.1 are called multimodal interfaces. Today, many applications are multimodal. The main reason to use more than one interface is due to the weaknesses of the primary interface. A good example for this are car manufacturers. In cars, displays are widely used today. It is also no technical challenge, to use touchscreens or knobs to control the functions of a car. However, if a user, most likely the driver, wants to use the interface, he or she has to look at the touchscreen or at the knob to see whats going on. Therefore, a simplification of user interfaces used in cars can be seen in the last few years. This was necessary, because the functionalities cars offer are growing rapidly and can be distracting for drivers.

In 2015, Ryu et al. developed a multimodal interface for cars to find out which combination of interfaces distracts drivers the least [60]. Therefore, they combined a voice user interface, a visual interface with touch functionality and knobs for additional control and also added an interface for gestures as an input. They also found out that American and Chinese users prefer natural language systems for speech rather than command and control systems while driving. To keep the system simple and not too distracting, they subtle combined the interfaces so that they compensate the weaknesses of other interfaces. For example, all major functionalities can be triggered using speech. The weakness with this system is that the user cannot really keep track of the state of the application. This general weakness is even worse while driving, because the user should concentrate on driving, not on the state of the application. Hence, Ryu et al. use the display to keep the user updated. This is also dangerous for the driver, because a display can show detailed information which takes a long time to be understood. In the cited paper, two seconds are defined as a dangerous distraction time. They also noted, that if there is detailed content on the screen, two seconds are a very short amount of time to understand the amount of data. Ryu et al. therefore decided, to keep the displayed menu simple. Their result can be seen in figure 2.1. They achieved this simple design by using a layered design with symbols and sparse use of text. Additionally, they decided to color code the screen. The user can quickly figure out in which state the application currently is by simply recognizing the color. Also, they kept the gestures simple. Only four gestures were implemented. This has the advantage that the user does not have to learn many new ways to interact with the application, but also makes it easier for the application to understand the gesture. Especially in the car, error handling can be difficult. This is because the application has to tell the user first what went wrong, the user has to understand that and figure out a solution to fix the situation. If the application misunderstands a gesture, it can be complicated for the user to understand that something went wrong. This distracts the user enough to create a dangerous situation. In the described paper, Ryu et al. suggest to combine multiple interfaces in a smart way to keep

Figure 2.1: Visual interface of a multimodal car entertainment system

distraction low and still please the user.

The automobile sector is one of the major industries pushing the development of VUIs. This is because entertainment functions in cars are growing rapidly. Also, the dangers of distraction from the view to the road are very obvious, so customers are actually willing to pay more for features like speech recognition in the car. Because of this reason, all major car manufacturers develop multimodal interfaces for their vehicles. Additionally, major technology companies try to port their systems for mobile devices to cars [35] [5]. This pushes VUIs used on smartphones as well, because it is usually the same system used in the car and on the phone.

Smartphones and personal assistants are the second major industry pushing the development of VUIs. All of them have in common that there are particular situations that require different inputs for multimodal applications using also a VUI. Smartphones are used all the time, so there are more use cases. However, not all of these situations match use cases for every single users. A helpful example would be a cooking application [36]. The user must be able to browse recipes using the display before start cooking. While preparing and cooking, the user most likely prefers to not use the screen to get the next required step. This is especially the case, when the user does not only has to look, but also provide input or scroll to see the next step. During the cooking process, many users do not have clean fingers or cannot stop doing what is currently required. This is only one of many use cases, where another form of input can help. In this particular use case, the smartphone can either recognize a gesture or use a VUI to communicate with the user.

Multimodal interfaces are important for this thesis, because they often use VUIs to compensate the weaknesses of other interfaces. However, VUIs have weaknesses of their own which are described in Chapter 4. For example, long lists cannot be presented well using speech synthesis. Therefore, other interfaces should be used. Furthermore, multimodal interfaces often provide uses for today's applications using VUIs. This is the case, because VUIs still have a high error rate. To compensate this, application designers often include an alternative way of presentation and / or input. This chapter first explains the term multimodality in more detail. Then, methods are shown of when to use multimodal interfaces. This chapter also includes helpful information about which interface to use for which type of content.

2.2.1 Multimodality

The term multimodality describes the fact, that there is more than one interface that can be used by the user. Multimodality does not limit the number of user interfaces but generally are presented using only two different interfaces. Using more than two UIs normally seems to be too much to handle and distracting for most users.

There are two main groups of multimodal interfaces. The first one covers a combination of a variety of input methods, such as speech, touch, pen, gestures or body movements. This first group is the most popular. This is because there are usually problems when entering data into a device. There are situations that require a different form of input, for example hands-free situations.Hands-free situations are situations, in which the user is unable to use her / his hands. The second group of multimodal interfaces covers combinations of output systems. They are rarely used combined with the first group but are starting to become more popular, too. Often, the combination of VUIs and GUIs is due to their positive synergistic effects, redundancy and an increased bandwidth of information transfer. Also, it is said to improve the mapping between the communication medium and the content [98]. Multimodal interfaces also help to support time-sharing and attention management in many different situations [10] [50] [66] [105] [84].

Good multimodal interfaces should consider the CARE properties [115]:

- **C**omplementary

  Ideally, all used user interfaces must work together efficiently. It should be possible to use all user interfaces simultaneously. This is often accomplished by using different interfaces for input and output. The user interfaces must support each other and cannot use completely different styles or functionalities. Otherwise, the cognitive load is too high for the user and, as a result, the user gets distracted and confused. In the worst case, the user stops using the application.

- **A**ssignment

  It must be clear which UIs the user is currently interacting with. There are different ways of how a user interfaces can be selected. In some situations, only one interface is available. There are also differences in how the active user interface can be selected. Sometimes, the application chooses an interface, in other situations the user selects the currently active user interface. In all kinds of situations, it must be clear which the currently active interfaces are.

- **R**edundancy

  A big advantage of multimodal interfaces is that information can be show redundant. This should be done carefully. However, in some situations it can be required to display data in a redundant way. VUIs for example are very volatile when it comes to the display of information. Therefore, important information should be shown in redundant ways. As a result, the user can still perceive the information without performing an extra action even if he / she missed it in the first place. If information is shown in a redundant way, it must be made sure to show the information only in the same temporal window. This meets also the requirements of the complementary property.

- **E**quivalence

  This property describes the ability to accomplish any task by using any interface. It is often difficult to meet this requirement and it depends on the situation, if it is important to the user experience to support it. The user should be able to choose from different interfaces if they are offered.

The CARE properties do not have to be met, but they help creating usable, multimodal interfaces.

In case of multimodal interfaces, "The process of integrating information from various input modalities and combining them into a complete command is called multimodal fusion". The opposite, a multimodal system that disaggregates data on multiple interfaces is called multimodal fission [21]. Usually, a well-designed multimodal system aims for multimodal fusion regarding both the input and the output interfaces.

Moreover, there are many research papers considering the use of multimodal UX engines which handle the states of more than one UI at the same time. Those research papers aim to obtain multimodal fusion. Multimodal fusion can be seen as an additional layer between the UI and the application itself. One research that considers this aspect is the research conducted by Jinwoo Kim, Jae Hong Ryu and Tae Man Han who suggest a new system to handle multiple UIs in in-car-based computer system [60]. Research in this field is required because car manufacturers have still not found a good solution for a non-distracting UI. As they ascertained, the trend goes towards non-touch based systems because they distract the user less than touch-based systems.

The system of Kim et al. can handle four different UIs. There is an interface for touch, voice, gestures and knobs. The result of their work is a multimodal UX engine which

behaves as a single interface to the application level but can handle all interfaces separately. It can also be multitasking. They tested the system in a special laboratory to measure the distraction of the driver from the street with their system combined to common touch based systems. The result shows, that the average time of distraction has decreased using the system proposed from Kim et al. As a result, the system would possibly reduce crashes in real cars.

As shown through this example, multimodality helps to increase the UX. It can prevent the user from too many distractions while increasing the amount of information delivered to the user (or to the customer). These are the main goals that can be achieved using multimodal interfaces.

2.2.2 Usage to lower cognitive load

In most use cases, applications are built to help solving a problem. However, the user still needs to concentrate and control the application using an interface. The degree of the concentration required to do this is known as cognitive load [88]. Sometimes, the terms cognitive factor or cognitive or mental effort are also used [67] [30]. Since these terms are generally used interchangeable, only the term cognitive load will be used in this document.

Due to the design of a user interface, the interaction with an application can cause high cognitive load. This happens in different situations which depend on the type of the user interface. In general, users should not have to remember too many details while interacting with an application. Additional factors that may cause cognitive load are too many choices [45], an unclear interface design or time. The term cognitive load does not include a stress factor but too much cognitive load can cause stress. “Stress represents a psychological response state to a perceived threat or task demand and is in general signalized by specific emotions such as frustration, anxiety and tenseness” [47]. The two phenomena can occur independently. However, stress while using an application is often related to cognitive load.

Cognitive load is required to process everything in the world around us. It can be described as the amount of mental resources needed to process a certain task. Auditory interfaces challenge the human memory in particular, because the information is non-persistent. As a result, the user has to remember what was said before. Otherwise, the context is unclear.

Therefore, VUI designers must respect the human cognitive resources. Users are sometimes new to topics, which requires more cognitive load to process the information. Furthermore, they have to remember the context of the application and may are distracted by other interfaces (in case of multimodal interfaces). To lower the cognitive load, designers can use constants. Such constants are words that are used in the same way in every conversation. They are not depending on a particular application or on a context. An example for a constant is the word *help*. A user always expect it to provide more detailed instructions on how to go on.

Additionally, consistency helps to lower the required cognitive load. Obviously, this always relies on the context. It helps, if users only have to remember a few universal commands. The number of commands should be small. Commands should be named clear and work as expected. The VUI should use a consistent dialog structure and a consistent terminology to give the user a more secure feeling while using the VUI. As result, users feel more confident and exactly know what to expect.

Context also helps to lower the cognitive load. Therefore, the VUI should make sure, there is enough contextual information [118] [9]. It is easier for people to connect new information with existing concepts.

Users, who are currently processing complex information may not be able to answer questions. In normal situations, user only can remember seven items [80]. Therefore, UI designers should never create UIs that require the user to remember more then seven items. However, this is the general case. For VUIs, this number needs to be smaller, because users cannot navigate back as easily as on GUIs. Cohen et al. suggest to limit it at already three items. Also, grouping items helps in this case. Broadbent found out that the recall is best when items are grouped into groups of three or four items [11] [33] [104]. The UI can make use of the fact that people remember best what was said most recently [8].

In general, designers should make sure that users are able to describe actions in their own words. This way, users do not have to think about commands that the VUI uses. Teaching users how to use a VUI is a process that takes much more time. Worse, this technique can upset users and make them quit using the application.

The ultimate goal for a simple UI is to keep the cognitive load low. Achieving this goal results in a better user satisfaction. Especially today's mobile applications have to be simple. If they stress the user, the user will uninstall it very quickly [109]. This is mostly the case, because users have a wide range of options for applications to solve whatever problem they have [107].

Multimodality is often used to reduce the amount of cognitive load. This is especially the case, when the main interface is not suitable to present the data. A good example can be seen in figure 2.2. The Google Assistant was asked to provide a list of the presidents of America. The response of the Google Assistant is as follows: *Presidents of the United States of America include Donald Trump, Barack Obama, George W. Bush and others*. As can be seen with this answer, there are not all names read out. This would have been too much cognitive load for most users because it would require to remember a large number of names. Also, reading out the whole list would take much time. To solve this problem, the Google Assistant combines a VUI with the existing GUI. There is a list of all presidents, which the user can scroll through. Additionally, Google uses the GUI to provide a picture for each president, rather than just the name. This makes it much easier, if the user is looking for a particular list entry without knowing what exactly to look for. It is also noticeable that the user gets suggestions on the screen on how to move on.

This example shows how the cognitive load can be lowered. The Google Assistant does not read the whole list out. In this case, the user would have to remember all of the lists entries. It is very likely that this takes too much time for the user. Like lists, there are other forms of data that are not suitable to be presented using a VUI. Another prominent example are maps and locations. Maps and also pictures are difficult to explain, even in a human-to-human conversation. These are examples to show, how GUIs can support VUIs if they are used as a multimodal interface. Also, VUIs can support GUIs. For example, if a user is looking for a song, the GUI can only describe the song using text. An auditory output is not possible. Changing to a VUI in that moment, creates a more fluent user experience, because the user does not have to switch between the graphical description on the GUI and the auditory channel.

As can be seen from the examples above, multimodal interfaces are usually used to compensate the weaknesses of other types of interfaces. This is not only the case for GUIs and VUIs, but also for all other types of user interfaces.

2.2.3 Designing for mobile devices

Multimodal design for mobile devices usually includes a GUI and a VUI. Due to the existence of a front camera, an interface to recognize gestures would be possible as well, but it is rarely used. Mobile devices have perfect preconditions for multimodal support.

This is because in most cases, they have a touchscreen, a microphone, a speaker and a front camera.

This chapter concentrates on multimodal applications combining a VUI and a GUI. Since this thesis describes the VUIs in more detail in the following chapters, this sections main objective is about the GUI design.

When designing a multimodal application, it is very important that the different interfaces support each other and fit together. Often, the GUI supports the VUI. Because the user has to concentrate on two different interfaces at the same time, the design of the GUI should be kept simple. It supports the VUI by presenting the key words or displaying data that cannot be displayed using a VUI. In addition, GUIs can be used very well to present confirmations or for the purpose of disambiguation. For confirmation, the user can just click *OK* rather than confirming it each time using the voice. This is a very quick and easy way of confirmation. Because all of this is done in the same time, the GUI should follow guidelines to not be too complicated.

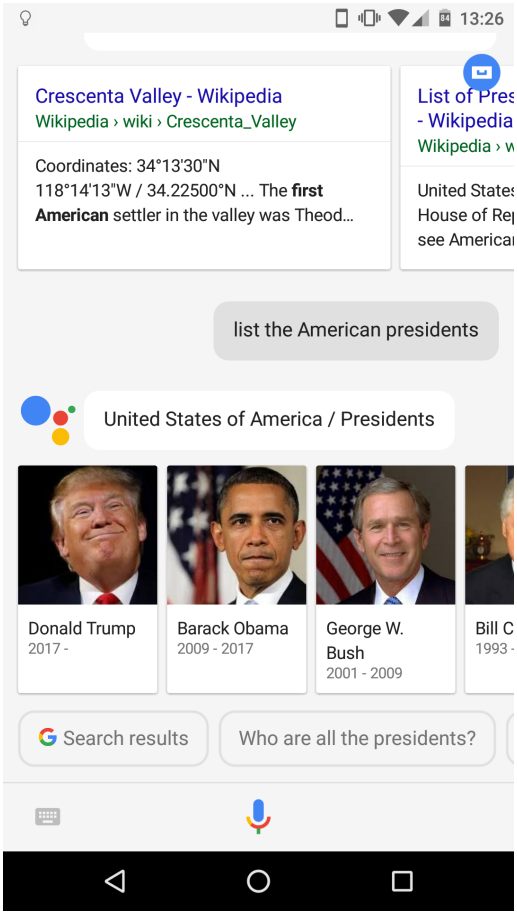


Figure 2.2: Response of the Google Assistant to request to show a list of the American presidents

In general, the design of a GUI supporting a VUI should follow the same guidelines as every other GUI should follow. However, simplicity is even more important in this use case. Most smartphone users are very comfortable with touchscreens, because they were using them for years. Even on hybrid devices with a physical keyboard, users prefer touch. When designing the GUI, developers need to keep some rules in mind. The most important fact is that users use their thumb in most cases to control the device. In conclusion, this means that the important functions can easily be reached by the thumb. As Josh Clark explains, the placement depends on the size of the screen. Nevertheless, all screen sizes have similarities, as can be seen in figure 2.3. It is best to place the main functionalities in the bottom, middle part of the screen. Unfortunately, this place is already occupied by main system functions of most operating systems. However, the room above these main system function (a bar on Android, the home button and a menu on iOS) can still be used. It additionally has the advantage to be well reachable by both left handed and right handed users. Potentially harmful actions such as posting something in public or transferring money should be put out of the thumbs range. Then, they are more difficult to reach and are not hit by accident very often.

Because the GUI is often not the main interface in a multimodal application, it is already kept simple. In most cases it is used as secondary confirmation. Besides the rules mentioned above, designers should also keep in mind that the thumb will cover the bottom parts when using functions at the upper part of the screen. Especially when an interac-

tion results in changes on the screen, this is an important aspect. Users sometime do not recognize that something changed as they cannot see it.

Another aspect which needs to be considered is the touch size. There are suggestions made by each operating system on how large a touch target should be. Because of different rendering and different screen sizes, this is often different depending on the operating system. Microsoft for example suggests 33px, whereas Google suggests a minimum size of 29px on Android. Apple suggests 33px for iOS applications [64].

Furthermore, Clark points out to create interfaces that require the least possible number of clicks to achieve the goal the user wants to reach. Especially when using the GUI as additional interface for a VUI, this should be a very important design goal as the user must manage to switch interfaces. Additionally, requiring more clicks from the user can be disappointing for the user [16].

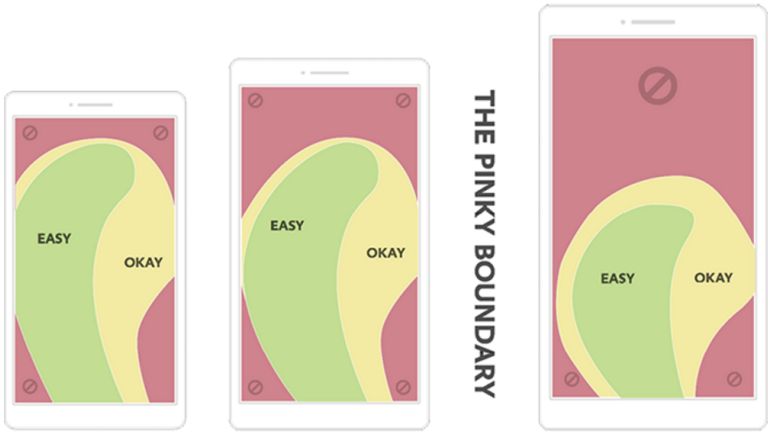


Figure 2.3: Locations which can be easily reached with a thumb on a phone [16]

The quality of VUIs can be increased significantly by anticipating what the user wants to do next. This is not only helpful for the VUI itself, it can also be used for suggestions presented on the GUI. Because a tap on a button is always quicker than using the voice again, this can save much time. It also increases the confidence the user has into the application because a button is often very clear.

The Google Assistant makes heavy use of that. An example can be seen in figure 2.4. The Google Assistant was asked to show the closest parks nearby. There are a multiple noticeable things in the response provided.

First, the Google Assistant provides a map. It is helpful to display this kind of data on the screen, because it cannot properly be presented using a VUI.

The second noticeable fact is the presentation of the list of results. There are only three results shown. The reason for this is because showing more results would confuse the user. Because the application combines a map and VUI, so a longer list would be too much cognitive load for most users. However, the list ends with the entry *More places*. This suggests that the search can be continued if the result is not satisfying yet. Each list entry provides the name of the location, a rating and a picture. The picture is used as a simplification, especially for users who are looking for a particular place they already know. Additionally, the places are ordered by their distance to the current location. Google assumes the user searches for places, because he / she wants to go there. This is the case for most location searches, so sorting the entries by distance is useful. All results are displayed in a place reachable by the user's thumb, so a navigation can be started with only a few taps.

The third aspect which should be noticed in figure 2.4 are the suggestions of what to do next. They are shown in a very good position at the bottom of the screen. It can be difficult to anticipate those suggestions, depending on the situation. However, they simplify the interaction a lot. One of the major problems VUIs have, is that it is not clear what a user can do, because there is no hint like a graphical menu that list all actions. This leads to the problem, that users may not know, what to say. This situation can be solved by suggestions on the screen. The VUI can also help the user with reading them out as examples. However, this should only be done to help the user when it seems that she / he is stuck. More information about how to help the user and handle such kind of errors can be found in section 4.3.3 and section 4.3.4. In figure 2.4, the reader can see that the Google Assistant suggests to search further or to select the closest one and get directions on how to get there.

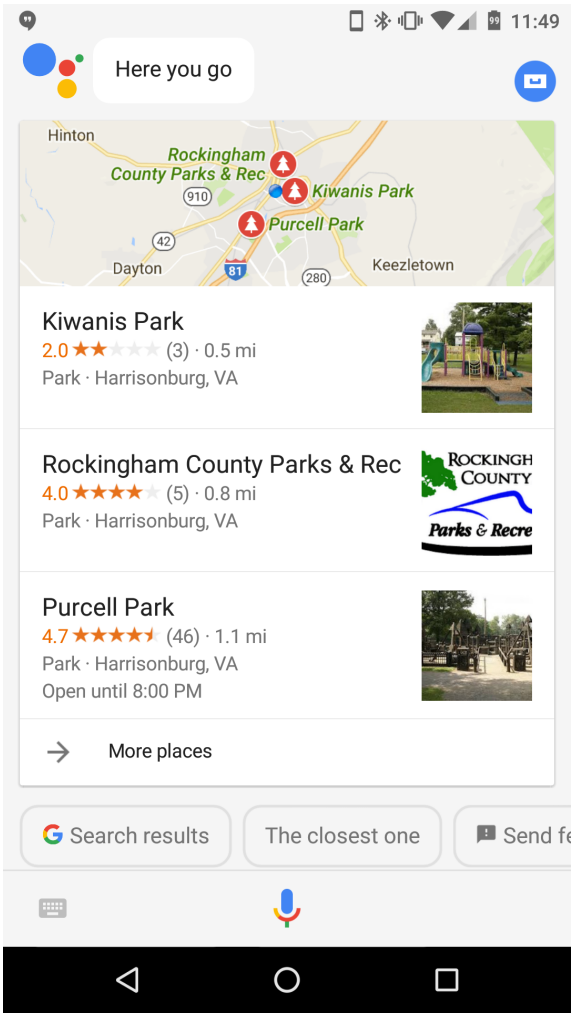


Figure 2.4: List of closest parks provided by the Google Assistant

Single tap alternatives like buttons or suggestions like this are very important parts of the simplification of a GUI. Also, autocompletion and type ahead functions should be used whenever possible. A single tap is always faster than typing or using the voice. So, if assumptions can be made, they should be used. Another example are input boxes for numbers like the quantity. Adding plus and minus steppers heavily reduce the effort to increase or decrease the amount by single steps. It is obvious that this only helps in certain use cases. It is up to the interface designer to decide whether it is useful or not. Especially when using multimodal interfaces, it depends on the current use case which user interface should be the primary used one. As Clark points out, single taps are a very good solution, but sometimes it is not possible. Although many users are comfortable with touchscreens, they are still not the best solution for writing long texts. Paul Kinlan even suggests to minimize writing in every situation by saying that "I love typing on a phone" was never said by a real person ever [64]. Deciding when to use which interfaces and managing the transitions between the different interfaces is the main task of the interface designers [16].

However, using a GUI and a VUI together creates new possibilities to interact with an application. Cathy Pearl suggests a combination of both interfaces as follows: "For example, if I ask, "What's the capital of this state?" while pointing to Kansas on a map of the United States, anyone observing me will know which state I'm referring to. A VUI could do this as well by using the location of where the screen is tapped in conjunction with the speech input. For a chess game, for example, the user could tap a space while saying "move my knight here" [90]. In such use cases, the quality of the VUI needs to be on

a very high level. Errors weight more, because it seems to be very clear what was meant to the user. In conclusion, this means there is not much understanding for confirmations or disambiguations. Applications using both interfaces like the ones describe by Cathy Pearl are still pretty rarely used today. However, there are many uses cases where a combination like this makes sense and creates an attracting user experience.

3 Conversational Design

Modern VUIs need to be conversational. Today, most VUIs offer only to understand spoken commands and translate them into actions. Conversational design introduces a VUI design that keeps track of what was said before. A conversation that relies on the knowledge of all participants about what their history is one of the major aspects of creating a more human conversation. Stanford professor Herbert Clark describes that the theory of a common ground of a conversation is, that "individuals engaged in conversation must share knowledge in order to be understood and have a meaningful conversation" [15].

A task that does not rely on other tasks and can be run on its own is called *one turn task*. One turn tasks are the opposite of conversational design. There can be a sequence of one turn tasks, but each task can be performed on its own. It does not rely in any way on what was said before or on any knowledge about the user currently logged in. These commands are what today's systems usually offer. They are still useful to perform commands a normal computer system would do. But since artificial intelligence systems are growing rapidly, it is possible to integrate this intelligence into VUIs and create more advanced systems.

The following conversation, dialog 3.1, shows how a user could possibly interact with the Google Assistant. The data provided here depends on a real conversation conducted for the purpose of this thesis. In this case, the Google Assistant was used on a Google Nexus 5X mobile phone. Notice that each expression starts with the command *Ok Google*. The expression was said after the Google Assistant responds with a sound indicating that it is ready to listen to the user. To simplify the conversation, this command and the sound are only transcribed the first time, when it is used to start the conversation. Furthermore, the Google Assistant sometimes provides additional data shown on the phones screen. If this is the case, there are additional annotations in the conversation (italicized and surrounded by hyphens).

The conversation above shows a sequence of one turn tasks. None of the asked questions relies on knowledge of what was said before in the conversation. In other words, each of the questions could have been asked without being part of this particular conversation. Additionally, the time of when the questions was asked does not make any difference. The result would always be the same. All information needed to answer a question is provided within the question.

Conversational means that there is a knowledge about what was said before in the conversation. In addition, this knowledge is used to understand the question and / or improve the answer. There are two levels of conversational design. First, there are conversations that consider the short term history. This means, everything said in the existing, currently ongoing conversation is considered for answers and can be used to improve the answers responded to the user. Second, conversations can be based on what happened days or weeks ago. This is more human, because humans usually remember the experiences they had, even a longer time ago. Statistics about what was said earlier help as well. Mathematical statistical processes help to reveal more details about the user's preferences. Those can be used to select better answers when responding to the user. Especially during disambiguation, this data is very helpful [90].

The Google Assistant already supports these kind of conversations in some situations. Figure 3.1 shows a sequence of an example dialog performed for this thesis. It shows, that

Me
Ok Google

Google
–SOUND–

Me
How is the weather in Harrisonburg, Virginia?

Google
In Harrisonburg, it's 23°C and mostly cloudy. Today will there be scattered thunderstorms with a forecasted high of 27 and a low 18.

Me
Where is Harrisonburg, Virginia?

Google
Here is a map of Harrisonburg, Virginia.
–Providing a map on screen–

Me
How far is Harrisonburg, Virginia away from Dulles International Airport?

Google
Dulles International Airport IAD is 118 miles away from Harrisonburg by car.
–Providing a map with route on screen–

Me
What's the population of Harrisonburg, Virginia?

Google
The population of Harrisonburg was 52,478 in 2014.

Me
What's the zip code of Harrisonburg, Virginia?

Google
Harrisonburgs zip codes are 22801, 22802 and 22807.

Me
What are the most popular things to do in Harrisonburg, Virginia?

Google
Attractions in Harrisonburg include Explore More Discovery Museum, Shenandoah Valley and others.
–Providing list with pictures–

Dialog 3.1: Sample dialog for a successful dialog

there are good approaches. However, it also shows that the Google Assistant sometimes

struggles with tasks. There are six different tasks performed by the Google Assistant. All of them can be seen in figure 3.1. Each task relies on the previous one. Firstly, the Google Assistant is asked who James Madison is (a). It responds by reading out the beginning of the related article from Wikipedia [59]. Secondly, the next question asked is about the height of James Madison (b). It should be noticed, that the question does not directly refers to James Madison by his name. Instead, the user refers to him as *he*. The Google Assistant correctly assumes that *he* refers to James Madison, as this was the last male person the user was talking about. This only works because the user was talking about James Madison earlier. In contrast, starting the Google Assistant and asking the question *How tall was he?* without any context of earlier conversations, Google responds by answering the question who the tallest men currently alive is. The reason for this behavior is the design of the Google Assistant. Because there is no context, it answers the question which is closed to one asked and can be answered. The third question in figure 3.1 is about the birthplace of James Madison (c). Google answers this question with both the place and the time when he was born. Fourthly, the user asks how far this place is away (d). The screenshot shows the second attempt asking directly for the place that was read out as James Madisons birthplace earlier. The first attempt was the question *How far is that away?*, which was directly asked after the answer where James Madison was born. It did not respond with the correct answer. The pronoun *where* was not correctly mapped to the birthplace in this case. Because of this reason, a more direct question had to be used. For the user, this means that two separate questions are necessary. Moreover, to ask the second, more direct question, the user has to figure out, what went wrong in the first place and also needs to remember the exact place where James Madison was born. This is heavy cognitive load for the user which does not result in a usable interface. The two remaining questions are related to the place where James Madison was born. The fifth question aims to find restaurants there (e). Google responds with a restaurant and additionally uses the screen to show more information and quick links that help the user to navigate there, call or visit the website. Lastly, the Google Assistant is asked for the opening hours of the restaurant (f). Again, the pronoun *it* is replaced by the restaurant correctly.

As can be seen in this sequence, the replacement of pronouns already works well for easy conversations. Such replacements are done in every conversation by humans. If they do not work as expected, a user feels quickly annoyed. Moreover, the user has to keep every replacement in mind and carefully replace possible pronouns by himself. This work should be done by the VUI. Designers should therefore include strategies to replace pronouns whenever possible.

Interaction which goes beyond one turn tasks becomes more human. A VUI designed to act with this behavior is called conversational. To achieve this, there are some abilities required. The anticipation of what could possibly be asked next must be accurate. Therefore, a good knowledge of the history what was asked before is important. Moreover, the VUI system should be able to replace pronouns correctly. The replacement of pronouns relies on the history of data hold by the system. This data is required because the system needs to know what to replace the pronoun with. There are more items that can be important for the replacement of pronouns. To replace a personal pronoun, it can be useful to have a look to the contacts a user has saved. Also, this helps while disambiguation. The application needs to keep track of every aspect that was said and save as much information as possible. Therefore, multiple strategies can be applied. Section 5.3 gives an overview over those strategies.

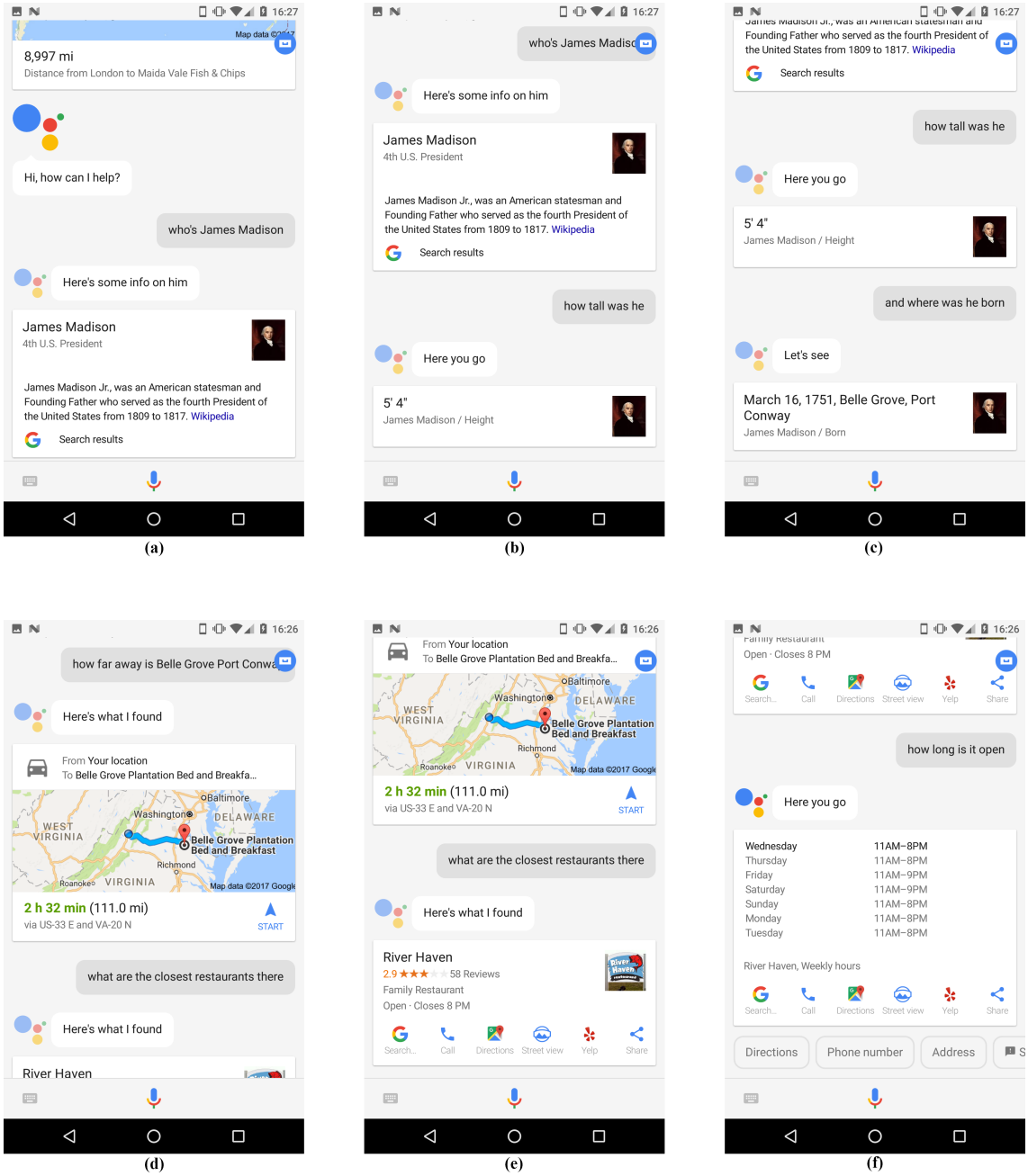


Figure 3.1: Series of screenshots taken when using the Google Assistant to get more information about James Madison

Another important aspect are questions. To keep a conversation going, questions are a very helpful instrument to do so. As Cathy Pearl points out, it is relevant for a good conversation to keep the questions clear. Instead of asking *Do you want to change it or send it?* a system should better ask *What would you like to do: send or change?*. By using the latter one, the user knows that there currently are two options which can be decided from. Staying with clear questions is especially important so the user does not feel lost and does not know which options are currently available. To keep the questions clear, a designer should never use rhetorical questions, because they most likely confuse users. When designing a VUI, the designer has to keep in mind that the user has to remember all options. This is required because the screen cannot be used to get an overview, in case the user missed an option or forgot about one of them. Therefore, it is also necessary to keep the number of options small. If the system offers many options, then it is better to give the user freedom about what to say and mapping what was said to the available options than reading all options out. On the other hand, the user can get lost with a design like this. Therefore, VUIs carefully have to be designed to help lost users. Section 4.3.3 gives more information about how to help a user [90].

Not only questions help to create a more fluent conversation. A VUI should prevent overlapping speech and not use any filler words. Moreover, conversation markers help user to navigate through a VUIs menu structure. There are different kinds of such markers. First, there are markers for the timeline such as *first, last, halfway there* and others. Second, a VUI should make use of acknowledgments like *Thanks and Alright* to give the user the feeling that what said was understood. Lastly, positive feedback as for example *Good job* can help to increase the confidence into the system as well as creating a fluent conversation. Because they are often used in real human conversations, these markers help to create more fluent conversations. Since the main benefit of VUIs is that users do not have to learn how to interact with them, a more human VUI becomes automatically more usable.

3.1 Linguistic basics of conversations

To be able to create a usable VUI, designers must understand the principles of human conversations. A lot of research has been made to analyze, how humans communicate with each other and how they build up conversations.

According to Cohen et al. [17], conversations are a discourse with the language describing the understanding beyond words. Conversations always provide a context and everything said in a conversation is therefore context-sensitive. The meaning of a sentence often relies on what else was said in the conversation. By design, conversations are communicative. Their language follows certain principles in grammar, words and rules how to create new words. These rules are different for each language, but they exist in every language. Knowing these rules helps users to understand the meaning of a utterance. Often conversations are based on conventions. Sometimes, conversations also include a certain level of unconsciousness. This is the case because words are sometimes used differently by several people. Richards says that a conversation "consists of exchanges which are initiated and interpreted according to intuitively understood and socially acquired rules and norms of conversational cooperation, which can in turn be manipulated to create a wide range of meanings beyond the level expressed directly by the utterances in the conversation themselves" [96].

Paul Grice defined four maxims that are required for a good conversation [45]. A good conversation is based on the cooperative principle, which was introduced by Paul Grice as well. The cooperative principle "refers to the fact that listeners and speakers, in order to have a successful conversation, must act cooperatively" [90]. The four maxims defined

by Grice are:

- **Quantity** This maxim relates to the amount of information provided in what is said. It summarizes two other maxims. Firstly, a contribution should be as informative as required for the current situation. Secondly, it should also not be more informative than required. However, this second maxim is sometimes seen disputable. Grice says, too much information is more a waste of time than a transgression of the quantity maxim. Nevertheless, being over informative can be confusing or it can mislead the recipient.
- **Quality** Each contribution should be true. This maxim also includes two sub maxims. The first one relates to what the speaker believes. According to Grice, the speaker should not say anything that she / he does not believe is true. The second maxim which the quality maxim is based regards the evidence. Grice claims, that the speaker should not say anything, which lacks evidence.
- **Relation** This can be simplified by the fact that each contribution has to be relevant for the conversation. This is a very open maxim which still opens a lot of questions about what is relevant.
- **Manner** This maxim does not describe what was said, it describes how something is said. This is a supermaxim of a set of different other maxims. Those include for example to avoid the obscurity of expression, to avoid ambiguity, to be brief and to be orderly. Depending on the individual and the situation, there might be others that can be added to the list. In general, the speaker should be clear with what she / he wants to say.

These maxims are, according to Grice, the basis for a good conversation. However, he also points out, that, depending on the situation, some maxims can be more important than others.

Especially for the understanding and the synthesis of natural languages, designers need to understand the mechanics of human conversation. Natural language processing (NLP) concerns the understanding of the speech itself, not of the action items correlating to it. As described, NLP only considers the speech and therefore does not distinguish between a single sentence and a whole conversation. The process of analyzing a whole conversation is called conversation analysis (CA). Although the general opinion around 1990 was that CA cannot be done by deterministic computers [69], this has changed since then [82].

The result of the CA can be seen in Schegloff's adjacent pair model [102]. This model depends on adjacent pairs which are connected to a more complex sequence. This sequence is the representation of a conversation. Following Schegloff's model, an adjacent pair is characterized by:

- A sequence of two utterances, which are
- (preferable) adjacent
- produced by different speakers
- ordered as a first-pair part (1PP) and second-pair part (2PP) and
- typed, so that a first pair part requires a particular second pair part (or range of second parts)

A simple example would be a greeting. Both participants greet each other and create a greeting-greeting sequence. Similar, goodbye-goodbye would be another example for a simple adjacent pair. A simple question and answer situation, which is familiar from frequency asked questions (FAQ) or search engines can be represented by using adjacent pairs. This is because the question can be represented by the first-pair part and the answer can be represented by the second-pair part. To create a conversation, the adjacent model can be extended. Schegloff calls this process expansion. The sequence can be expanded to an arbitrary length using additional adjacent pairs. There are three different positions where in the structure that are possible for the expansion: *pre-expansion*, *insert-expansion* and *post-expansion*.

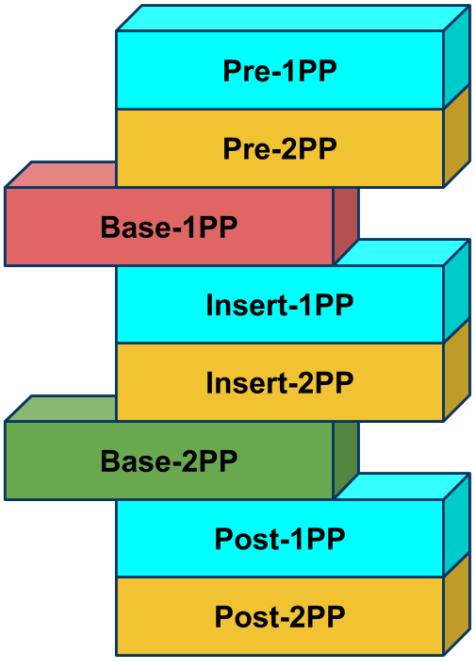


Figure 3.2: Expansion with adjacent pairs [102]

Moore et al. give a good example for the expansion [82]. Their example can be seen in figure 3.2. It is noticeable, that there is a base adjacent pair which represents the action defined by the utterance. The pre-expansion defines the dependencies for the first-base part, which is then produced by the speaker. The recipient then either manages the dependencies or repairs the base first-pair part. With doing so, the recipient generates the response to the main action which was initialized by the speaker. The speaker then has the chance to manage dependencies or to perform repairs based on the base first-pair part or on the base second-pair part.

Dialog 3.2 gives a practical example of this process. The first adjacent pair (lines 01 and 02) describes the preconditions. The question cannot be answered directly, so there is an expansion necessary to clarify the situation. This is done by asking another question. In line 03, the confirmation indicates that the action can be performed. The action results in a way to the requested location, which can be seen in line 04. Both the lines 05 and 06 are used as the post-expansion pair.

It is obvious, that this is only a simple example of an expansion. This example contains only a single expansion. However, each of the adjacent pairs, meaning the pre-expansion pair, the inserted pair and also the post-expansion pair can be expanded to arbitrary lengths.

01 Hello, what is the shortest way to the bus stop?
02 Do you mean the bus stop in Main Street?
03 Yes!
04 Okay. Just follow this street and turn left. It is right at the corner.
05 Thank you.
06 You're welcome.

Dialog 3.2: Example dialog for adjacent expansion

Each of the pairs usually helps to create dependencies necessary to create the action items. If something was not understood correctly, any participant (speaker and recipient)

can initiate a repair. A repair aims to correct troubles might occur during the utterance. It is represented by different wording which repeats something that was actually already said [101].

As Moore et al. point out, the usage of adjacent pairs which make a conversation able to be expanded, is the major difference between dialog systems and information retrieval systems. In comparison to information retrieval systems "only dialogue systems, with sequence expansions and repairs, can provide support for user understanding of that information" [82].

The adjacent model is one of most important models regarding conversations. Nevertheless, there are many other theories bothering the subject of conversations. To help VUI developers, Google simplified the process of a conversation into six steps [43]:

1. *Opening a channel and setting up common ground*

   This happens when participant A sends a message to participant B to initiate the conversation.

2. *Commit to engage*

   Participant B agrees to have a conversation with A.

3. *Construct the meaning*

   Both participants share ideas and connect through them. They are might using different contexts in this step. This can be a previous conversation but also any other type of context. Context are often unspoken.

4. *Evolve*

   Based on their interaction, either A or B or both obtain something new based on their conversation

5. *Converge on agreement*

   Describes the result of the conversation. The participants either agree on something and have a positive result or they might try to repair the conversation if not.

6. *Act or interact*

   As a result of the conversation, there is always a functional action or the achievement of an unconscious goal.

Those guidelines simplify conversations, but they are the main parts of each conversation. Although the steps described above are defined by Google, they are similar to Schegloff's definition. Opening a channel and agreeing to participate in a conversation (steps 1 and 2) can be seen as pre-expansion of the actual conversation. The construction of the meaning (steps 3) can be seen as the inserted pairs between the main action items. Because this step may requires more information, it can be expanded as well. When the participants reach the evolvement stage (step 4), they basically reach the point which is described by Schegloff as the base second-pair part. Finally, the agreement and the action items (steps 5 and 6) can be seen as post-expansion. They can be expanded again, because they may require repairment processes.

There is more research about the mechanics of conversations, but most of them are based on Schegloff's adjacent pair model. Further studies concerning the behavioral roles of members [97] [51] [22].

3.2 Turn taking

When designing a VUI, it is very important that the system recognizes when it is it's turn to speak. To start the interaction, most VUIs use a keyword. Examples are Amazon's Echo with the command *Alexa* [1], Google's Assistant using the command *OK Google* [38], Apple's Siri can be activated using *Hey Siri* [6] or Microsoft's Cortana with the command *Hey Cortana* [75]. Some can also be activated by tapping a button on a GUI. For example, this is not possible with Amazon's Echo, because this device does not have a screen.

Besides that the initiation can be seen as turn, a turn in a conversation is usually the point, when the role of the active speaker changes. Therefore, one participant of the conversation has to stop talking followed by another participant who starts talking. In a simple version with only two participants the participant who is listening has to realize that the speaker has stopped. Then, the floor is open. So the listener is now able to become the speaker. In a conversation with more than two people, the participants have to master another challenge because it is often not clear who the next speaker will be. This will be discussed later in this chapter.

It is possible, that a participant starts to talk even if there was no turning point reached. In most cases, this is considered to be rude and impolite [90]. However, there are other situations that justify the interruption of another speaker.

Based on Sacks, Schegloff and Jefferson, the theory uses Turn-Constructional-Units (TCU) to mark turning points. After each TCU, there are three different possibilities how the conversation can go on. Firstly, nothing can happen and the conversation comes to an end. Secondly, another person can continue the conversation. In this case, the current speaker can select a person to continue. This can happen by asking a direct question to another participant of the conversation. Moreover, the current speaker can only address another person by using their name. It is obvious, that the speaker also can leave it open who should speak next. Finally, the conversation can be continued by the same person as before. In other words, the current speaker selects himself as the new speaker. The possibilities described above are from a speakers point of view. However, listeners can also influence the conversation flow.

Each listener has choices and can influence the turn taking of a conversation. First, each participant can simply wait until the turn is offered. Second, a listener can wait at a possible turning point for the speaker to continue (no change of the speaker). Third, a listener can decide to speak when the floor is open. Fourth, any listener can interrupt the current speaker, even if this is considered as rude behavior. Lastly, any participant can also decide to speak even if the floor was offered to someone else. Similar to interrupting someone, this can also be considered as impoliteness.

Who takes the turn is in real conversations often also influenced by non-verbal cues. Obviously, these cannot used when building a VUI, except the interface uses an avatar. An avatar is a visual representation of the VUI. It can be a still image, a prerecorded video or an animated cartoon. As Cathy Pearl says, there are also nonhuman avatars which look like monsters or other familiar shapes [90]. Pearl also points out reasons, why to use an avatar and why not. There are positives and negatives listed in her book "Designing voice user interfaces" (see chapter 3 in [90]). Edlund / Beskow show, that non-verbal cues are very important and result in longer turns. Additionally, people who are using non-verbal cues take the turn much faster than people who do not use non-verbal cues [25].

Before someone takes the turn, there are differences in behavior as well. People can decide to start speaking as soon as possible, even if that means that they interrupt other speakers. They can also wait for the current speaker to stop or add a pause, before they start speaking themselves. As Gerritsen and Claes found out, this behavior can be different

in other cultures [26] [14]. For example, Germanic speakers often start when another speaker stopped, while Roman cultures tend to start as quickly as possible. In contrast, Japanese speakers usually add a short pause before starting to speak. This behavior should be considered when designing a VUI. Fortunately, this behavior is strongly connected to the language the user is currently using.

In case, any participant decides to start speaking before the current speaker has finished his point, the conversation overlaps. This phenomenon is called overlapping speech. Ter Maat and Heylen point out, that overlapping speech is not always a bad situation [71]. Overlapping speech can result in a breakdown of the conversation, but it does not have to crash the conversation. It is often used as a backchannel, mainly to ask questions, express agreement or to make suggestions. As Schegloff says, there are different options for the current speaker to handle the situation. Firstly the current speaker can stop speaking, which is often considered to be more warm, passive and submissive. Secondly, the speaker can continue normally with his speech. Lastly, the speaker can raise the voice and speak louder. This is mostly considered to be unfriendly, aroused, disagreeable and rude [100]. To find out, what feelings participants of the conversation have, Ter Maat and Heylen built a simulator, to create different conversations [71]. The behavior of the participants can be described using an extended markup language (XML) file. The results were not evaluated automatically. Instead, the results were evaluated by real people who were listening to the audio files.

To summarize the situation, turn taking is an important topic and should be considered while designing a VUI. The turn taking behavior has a great influence to the UX. It can also influence the persona of the VUI. As a result the way people think about the VUI and their feelings may be influenced by the turn taking behavior. It is important to create a VUI that behaves as human as possible. In conclusion, the VUI needs to meet the expectations of a certain culture for each language (if the VUI should support different languages).

3.3 Avatars

Whenever people hear a voice for the first time, they imagine how this person could look like. Obviously, this only happens, if only the voice is present. This is important, because the voice needs to fit to the person it belongs to. In nature, this is almost always the case. But when designing an avatar, it is usually designed for a voice and a certain use case.

A persona, or avatar, is the mental image of a person. They can be used as visual representation of a VUI. They usually appear on a screen. If the team decides to provide an avatar, it is important, to carefully design avatar and voice together. The wrong persona will seem awkward and confuse users.

It is a complex decision, to decide whether a VUI should be visualized or not. To decide, it could be a good idea to imagine, how a system could be visualized. It also depends, on what the system recognizes. If the VUI often turns requests into search requests, it is not recommended to user a very human avatar. Instead, an abstract representation can be used. This indicates that something is going on, without being too personal. The Google Assistant or Siri implementing such visual representations. But often, avatars are not required. In contrast, they sometimes even distract people [90].

If designed correctly, an avatar acts as a metaphor of the system and as the role of the VUI. Its design heavily depends on the context. If the VUI serves a business aspect, the avatar must be designed to seem trustworthy. A young girl for example would be the wrong choice. Designers also need to keep in mind, that the brand and also its image are closely connected to the persona. An avatar should also designed to serve the end users. The user need to talk to the persona and therefore has to like it. As a result, designers

should keep in mind, who their target audience is and what the mindset of the users is.

Ideally, a persona should be developed before the VUI. This ensures the consistency with the application content. The definition of a persona should start with a predefinition, followed by a biographical sketch. Then, designers can add vocal attributes. Looking at a photograph while designing the avatar is often helpful [17].

3.4 Chatbots and personalized assistants

The core of a VUI is the script. This text has to be generated. Obviously, a VUI also covers the aspect of understanding spoken words and replying to them, but the core is about understanding the meaning of what was said and replying to that in a proper way. As a result, VUIs can be simplified by skipping the speech recognition and replacing it with text as input. Similar, the speech synthesis, which is the normal output for VUIs, can be replaced by generating text. The text has to be generated anyways, but a VUI additionally reads it out and therefore uses a voice.

In contrast, chatbots only understand the text and produce text as an output. In other words, chatbots are simplified VUIs. Chatbots are text based tools which let users accomplish tasks in a conversational manner. Therefore, bots (short for robots) use, in most cases, artificial intelligence to understand what was said, perform a task and generate a proper response to the user. If artificial intelligence is not sufficient enough, applications are sometimes supported by humans to complete tasks.

Bots became very popular, because they are the first step towards a conversational UI. Because many people are used to text and especially to typing on a phone (over 2.5 billion people have at least one messaging application installed on their smartphone [23]), they are used to a text based interfaces. This is one of the reasons why many users are willing to use chatbots. Another situation that leads to the current good position of text based bots is the enormous amount of mobile applications. As David Marcus, Facebook's vice president of messaging apps, says: "People don't want apps for every single business that you interact with. They want the ones on your homescreen and that's it" [92]. This situation gets worse, as the average application size gets bigger [111], but space on phones is still limited. This is where chatbots can help. Because messaging applications already provide all the requirements a chatbot needs, they are the first candidates to be extended to chatbot applications. Examples for this are WeChat [72], Telegram [108] or Facebook's Messenger [27]. Instead of sending a message to a real human, a bot answers. As a result, users do not have to switch to another application to accomplish their tasks. As Google's messenger Allo shows, they also can be included in a conversation with real humans and support with tasks [37].

For developers, it is also easier to develop chatbots than developing a whole application. This is because a developer only needs to define the content. The UI and the design of the interaction with the user is already designed by the chatbot application itself. This results in easier and faster development processes. The downside is that developers lose the opportunity to customize the look and feel of their application.

Unfortunately, bots are not always a solution. Not every use case can fit into a text based application. The fact, that chatbots are now able to not only support text as output, does not change so much [46]. Chatbots suffer from the downside, that they require a lot more taps than applications that are made for a special use case. Dan Grover compares the use case *Ordering pizza* by using a chatbot and an application, which is made to order pizza. The result is 73 taps in the chatbot versus 16 taps in the application. Also, using an applications lets users accomplish their tasks faster, as everything is clearly structured. In a chatbot, the user always has to read text, which slows down. Therefore, chatbots are

obviously more flexible compared by their content [46].

Chatbots can still be useful in many use cases, mainly because people are used to type. Also, chatbot applications like WeChat try to support users by designing more possible types of answers. They also try to improve the interaction by giving the developers more options to design the interface, so users do not have to type every action and can be offered with customized buttons instead.

The term personal assistants collects different varieties of VUIs and chatbots, but all of them have the ability to respond with personalized responses. Therefore, data about the user is necessary. This data can come from existing user profiles or from previous conversations. It does not make any difference, if the input and output is covered by speech (as in case of VUIs) or by text (as in case of chatbots).

3.5 Impact to the user experience

VUIs change the ways how users interact with modern applications. Not all applications are qualified for the usage of VUIs. This is heavily based on the underlying data. Today, after VUIs have gained a lot of attention, many applications just try to implement VUIs only for the purpose of having a VUI.

However, with a good strategy and a thoughtful design, almost every application can be improved by offering VUI interactions. Designers should be careful and only use VUIs when really needed. If applied correctly, VUIs change UX so that the user has a real advantage.

The first major advantage for most users is that they can describe tasks in their own words. This simplifies user interfaces, because users do not have to search in overstuffed menus for the action they want to use. Applications can therefore kept simpler and still offer many functionalities. In addition, this helps users that are new to applications because they do not have to search for actions, which they sometimes do not even know the name of. The interaction with more complex tasked can also be simplified using VUIs. Secondly, VUIs enable applications for hands free usage. This enables the user to use the application even when using the hands for other purposes such as driving or cooking. This enables many applications to be more present in every life situation. Especially personal assistants, such as Amazon's Alexa, that aim to be available whenever needed benefit from this fact. Finally, applications can make use of multimodal interfaces to interact with users in the best way possible. Users can even chose their preferred way of interaction.

In contrast, the usage of VUIs changes the branding strategy for companies. Their main branding option is now the voice and not a logo. Although, commonly used sounds can be reused to identify the product efficiently. In contrast to commercials, VUIs often only offer the possibility to brand their product by the voice. The choice of the voice is therefore significantly important.

4 Design of voice user interfaces

The design of a VUI is a complex process. This chapter explains the main steps of this process. Additionally, this chapter will help the reader to figure out what good VUI design is about. This chapter describes the design of a VUI in a high level of abstraction. This design step includes the description of what should be said, which words should be used and which scenarios are supported. This chapter therefore does not cover the technical implementation. Detailed thoughts about the implementation process can be read in Chapter 6.

4.1 Requirements analysis

The decision to develop a VUI can be made because of different reasons. Similar to any other product, the requirements need to be defined before the development starts. Requirements result in a specification, which is important so that each member of the team knows what exactly should be the product. The first part of this section is about general requirements analysis. However, the latter part is mostly about specific topics that concern the development of VUIs.

Requirements are defined as attributes that a software must satisfy. These attributes are defined prior to development. They not only concern the functionality of a software, they can also describe different other characteristics that the product can have. Examples for this are security, performance or availability.

The organized and structured methodology followed to identify the resources a system need to serve these requirements and the necessary characteristics for these resources is called system requirements analysis (SRA). This process includes the design and selection of these resources. The motivation for this process is driven by the interest of management to keep the risk of failure low.

In general, SRA can be seen as the translation between the user needs and the actual design of the application. This not only includes the interfaces design, it regards every decision made towards the architecture of the application. This process is necessary because the field of information technology (IT) has grown too much. As a result, people needed to specify on certain areas. SRA decomposes the user's needs in a way, that special tasks can be created. These tasks can then be organized and assigned to engineers who are specified on this particular field. This is also necessary because software products become more complex. In almost every case, they cannot be handled by a single person anymore. As a result, tasks are grouped by their specification and assigned to teams. The big problem is consequently divided into multiple smaller problems, which can be handled by fewer people. Jeffrey Grady describes this process as follows: "[...] we decompose because we are organized into specialized engineering organizations driven by competition and our limitations and because we are limited in the scope and complexity (breadth and depth) of a problem that any one person can master" [44]. Not only for software projects, this has resulted in a three-step production process: First, the problem needs to defined. After the problem is clear, the second step is to find a solution for the problem. This second step is often divided into three sub processes. Solving the problem mostly starts with finding an engineering solution. This technical solution has to be translated in manufacturing

needs and supplier needs then. The final sub process then manufactures the solution. The third and final step of the overall process is to prove, that the solution actually works. This includes quality assurance and the evaluation that the solution solves the problem as planned [44].

System requirements analysis concentrates on the whole system. However, the most important aspect of a VUI is the user. Therefore, this section concentrates on user requirements analysis. This process is highly influenced by user centered design (UCD).

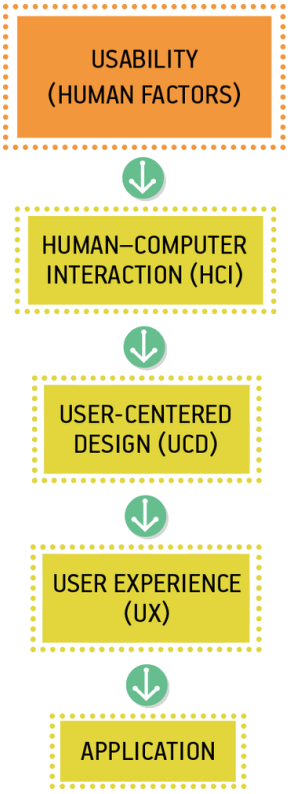


Figure 4.1: UCD in context [68]

UCD emerged from the awareness that usability is not a subjective view of a single developer. However, UCD is a methodology emerged from human computer interaction (HCI) that helps both designers and developers. It puts the user in the center of the requirements analysis and consequently in the focus of the development. This ensures good usability and removes ambiguities. It is also not subjective. To the contrary, it is based on ergonomics, psychology, anthropology, and many other fields. In general, it relies on data which is determined by studies. This data additionally helps to limit assumptions and statistically proves what users are doing. Also, UCD is about more than design. Aesthetics are important, but UCD mostly cares about the effectiveness for the designed purpose. Towards most managing opinions, UCD is not a waste of money. If applied correctly, it helps saving money. However, also developers feel bad about UCD. This is because caring and observing about users results in the feeling, that the product is not moving forward. In addition, hearing criticism is hard for most people. This is why developers often see UCD as a distraction. Instead, many developers would prefer to write new code or learn about new application programming interfaces (APIs) and frameworks. But UCD can help saving money by implementing the user needs and not something else. It is as important as writing new code. Additionally, conducting a survey creates a distance from the current task. This can help seeing problems from a different perspective and, as a result, solves them in an easier way. User centered design takes care of the problems at their root. Only looking at bug reports does usually not help. Instead, developers have to stay in contact with users and think of other possible problems. Sometimes, users simply misunderstand the application. Therefore, the requirements must be defined in close contact to actual users. Furthermore, UCD helps to concentrate on the user's needs instead of distracting the developers. As a result, the planning moves from user requirements to technical requirements and not the other way around. This often happens, as developers tend to see the technical problems first. As can be seen in figure 4.1, UCD is a sub topic of HCI, which concentrates on the interaction between humans and computers. It is a sub topic of usability, which concerns the human interaction with every device. The result of UCD is an increase of the UX. This is because developers concentrate on what the user wants and not how to solve only the technical problems. The user often does not care about how the problems are solved. For users, it is more important to have an easy solution. A good UX represents an easy solution. If UCD is used correctly, it saves much money because it prevents big change request in a later development cycle [68].

The combination of system requirements analysis with UCD results in user require-

ments analysis. As the previous explanations suggests, it concentrates on the requirements important for the user. This is the basis for the development of a VUI.

For VUIs, the user requirements analysis is slightly different. Due to the different technologies and ways of interaction, the designers need to plan in other ways and consider other constraints. Cohen et al. define three main topics that VUI designers need to take care of: Understanding the business, understanding the user and also understanding the application [17]. These topics were initially defined for interactive voice response (IVR) systems, however, they are still valid and widely applicable across the software industry.

At first, designers must look at the business they are designing for. There are many topics and questions that need to be considered:

- **Business model** This part is all about the motivation why the company has to build the VUI. Understanding the business model can help to improve the UX. The motivation can be divided in the primary motivation (the main reason for development). However, there are usually secondary motivations as well. An example for a secondary motivation would is replacing an existing system.
- **Corporate context** Designers have to think about how the VUI fits into the product portfolio of the company. How can it be combined with other services the company offers? Are there overlapping functionalities?
- **Corporate image** Companies always trying to build a corporate image. This image is perceived by potential customers and mainly directed by the marketing strategy. If a new VUI is created, it has to fit into the corporate image.
- **Schedule and rollout planning** Obviously, the management's schedule has to be considered when analyzing the user needs and defining the requirements. Not all requirements can be met in a defined time frame.
- **Metrics** From requirements, metrics can be defined. These metrics are then used during the development process to measure, how good the requirements are being met.
- **Planning of beta test** Especially for VUIs, an early beta test is essential. Only real users can give designers an impression on how good the design of the conversation actually is. Therefore, the beta test phase has to be planned early.

Besides the business itself, the most important issue for VUI designers is to understand the users. The necessary data to understand the users is in many companies already present and can be provided by the marketing department. If this is the case, designers should start with this data. Sometimes, especially when developing a new product or when developing without a big company, this data is not available yet. Then, the designer have to think about their potential customers even more precisely. Cohen et al. also defined categories of questions that designers have to think about when analyzing the requirements. They divided these categories into two groups: First, the user profile. This means, everything that is known about the future users. The second groups concerns the usage of the VUI. The following list gives an overview about questions designers should think about, when analyzing the user profile:

- **Target group** Who is the target group of the VUI? There are more details about the target group that matter. For instance, this can be the demography, and, if there is a specific demography, what matters to this group. It is also important to know if the

target group is only a single group or if it can be segmented into different segments, which have to be treated differently.

- **Dialect or jargon** If all the users have a common background, it is sometimes the case that they use a specific jargon or lingo. The VUI has to be prepared to understand these specific words. If all users come from a certain region, dialects can be expected as well. This can also give hints as how to address or meet the interest of special age groups.
- **Tech sophistication** Designers should think about the sophistication of their users about technology. If it is the case that users are expected not to be familiarized with technology then, more explicit explanations are necessary. It is good to think about the users experiences with technology as well.
- **State of mind** Users will sometimes only use the VUI in a particular situation. If this is the case, it is important to think about this situation. Is it time-critical? Is it mission-critical? Are users dealing with money or valuable transactions? On the other hand, users can also only use the VUI for entertainment. The amount of seriousness needs to be resolved and this, in turn, influences the dialogs and the design of the VUI.
- **Self image** Almost all people have a self image. To prevent unexpected behavior, this image should be considered while designing the VUI. If the target groups understands itself mostly as young and hip, the VUI has probably a different wording than if the target group are mostly people understanding themselves as business people.
- **Mental model** In some cases the VUI serves a functionality that users already know. This can be because of another VUI system, but also because of other modalities. It should be clear, that the users have to know how to interact with the data. This helps, figuring out how they predict the VUI to work.
- **View of brand** This point concerns the users current perception of the brand. This is mostly the case, if the VUI is part of a larger software product and also if the user already knows the brand. This often leads to expectations which the designers should be aware of.

The second category concerns the usage of the VUI. This is very important, because designers have to make sure that the user feels comfortable while using the VUI. The following are categories that should be considered when analyzing the requirements:

- **Reason for usage** The main reason for the usage of a VUI is the most important thing to understand. It helps designing menus, error messages, help messages and more. Additionally, it gives information about how often features are used and how important they are.
- **Single usage vs. repeated usage** How often a system is used has direct impact on the design of the VUI. Use cases that are rarely used usually need more explanation, whereas a user who repeatedly uses a system gets familiarized with the system quicker and learns what to do.
- **Level of attention** For instance, a VUI designed to use while driving, must consider that people may not fully concentrate on the VUI system. They should concentrate on driving. Therefore, the designers need to keep everything even simpler to avoid distractions. Considerations like this one always needs to be a part of the design process.

- **Channel and environment** Today, most VUIs run on a phone. But they are also available on desktop computers or on devices like Amazon's Echo, which do not have a screen. The channel therefore needs to be considered. It also helps to figure out how the environment will be. Sometimes, the designer already knows, that there are noises to be expected in the background.
- **Voluntary vs. involuntary usage** Especially when people call and use IVR systems, designers should keep in mind that customers sometimes expect to talk to a real person. This especially is the case if their concern is serious.
- **Other systems** Modern VUIs make heavy use of multimodality. As a result, the VUI depends on other systems as well. Obviously, other systems on which the VUI depends on need to be considered during the planning process as well.

It is obvious, that the view of the company is very objective. Not all questions can be answered and, if they can be answered, they only can be answered from the perspective of the company. This problem can be solved when using surveys, focus groups and studies. However, this is very expensive and takes much time. Nevertheless, if the data is available, designers should use it. Often, this data is available in big companies that are already spending much effort on marketing. Observational studies are also often a good resource of information. They can be used, if call centers are available. Then, designers can listen to phone calls to get an impression on what people ask, how people describe their issues and what their preferences are. Additionally, it helps to interview the customer service representatives. Designers do not get an unbiased impression this way, but these people can provide data observed over a much longer duration. They can help with typical confusions or misunderstandings of users.

Besides understanding the business and the users, the design team also needs to understand the application itself. This means, all the details of the application must be clear to everyone. Therefore, designers should think about the following questions:

- **Tasks and sub tasks** A detailed description of each task that the application supports. Additionally, the input given by the user and the output produced by the system should be documented.
- **Task complexity** For each task, it also recommended to analyze the complexity. This result in the amount of commands needed to accomplish a certain task. Additionally, the complexity includes how much the user needs to learn to be able to use the application.
- **Recognition challenges** Tasks can sometimes be challenging. The reason for this can be the data itself. Problems like large list or long sequences of numbers should be identified early and be documented in the user requirements. Often, there solutions are already available. It just needs to be considered how to develop them. For instance, splitting long numbers like credit cards in groups of four digits is commonly used. The VUI should know this. Also, it is recommended to use check sums whenever they are available. This always helps to validate data.
- **Application environment** Especially if the application depends on other system such as database systems or server calls that require an internet connection, designers need to think about the expected latency (delay). The application needs to handle cases, when the connection takes longer than expected.

- **Other technologies** VUIs may be used for different methodologies that can influence the design. For example, a VUI can also be used to identify the user currently talking. This result in different requirements for the application. In cases where other systems may be involved they also need to be thought of when defining the requirements.

The requirement analysis is the first step of the development process. Defining the attributes of a software is essential and so it is for VUIs as well. This section points out questions and aspects, designer should think about while defining the requirements. The questions help, to think about all potentially important facets that can might influence the application's design. It is clear, that not all topics are relevant for each application. This is always depended on the particular use case. The requirements definition is the input for the next step: the design of dialogs on a high level.

4.2 High level dialog design

This section describes the second step of the development process of a VUI. The result of this process is a script of all possible dialogues that the VUI should cover. Therefore, the user requirement analysis is used to figure out, which functionalities are needed. The purpose of the high level dialog design is to find a way to cover these requirements using speech.

There are three main methods to create dialogs: sample dialogs, visual mockups and flow diagrams.

Sample dialogs are a very cheap and easy to use design tool. Such a dialog represents a single task of the application. It consists of a dialog between two parties, the VUI and the user. The dialog goes back and forth between those two participants. Cathy Pearl compares it best with a movie script [90]. Dialog 4.1 is an example for a sample dialog, picked from the development process of the application which is also part of this thesis. This process is discussed in more detail in Chapter 6. The following example starts with the VUI, reading out a chapter of the story. For the purpose of better readability, this story is shortened.

VUI
[...] looked at his friend and then back to Joni, who was in front of him and blocking him. How do you want the story to continue: shoot or pass?

User
Shoot!

VUI
Joni was not a good player and it was easy for Ahmed [...]

Dialog 4.1: Sample dialog for a successful dialog

This example represents a situation, where the user acts as expected. Not every single use case needs to be covered by sample dialogs, but the main functionalities should be covered. Sometimes, it is difficult for designers to start, because there are many dialogs that can possibly occur. In this case, a good way to start is to think about the five most common use cases. These can then be scripted as sample dialogs. In addition to every successful dialog, designers should also think about what happens in case the user is not

understood or simply does not say anything. The next example, dialog 4.2 is also taken from the development process referred to earlier. It shows a sample dialog if the user does not say anything.

VUI
[...] looked at his friend and then back to Joni, who was in front of him and blocking him. How would you like to continue: shoot or pass?

User
–silence–

VUI
Oops! Please say that again.

User
–silence–

VUI
How do you want the story to continue: shoot or pass?

User
–silence–

VUI
Tap a button on the screen to continue

Dialog 4.2: Sample dialog for a failed dialog, in which the user simply does not say anything

Dialog 4.2 shows a situation that require error handling. Section 4.3.4 describes these kind of situations more precisely. This sample dialog is an example for an unsuccessful dialog. As can be seen in this example, it was the plan of the designer to ask the user first to repeat the answer given before. If the user still does not say anything, which is the case in this example, the VUI repeats the possibilities the user can answer. This is done, because it is may possible that the user missed or misheard the options that are available. If the user still does not respond, it is most likely that the user is not in front of his device anymore, just does not want to answer or cannot answer because the situation does not allow speaking loudly anymore. To prevent the VUI ending up in an endless loop of asking the user for questions, it just asks the user to use one of the buttons shown at the GUI. In this situation, a graphical hint is also shown on the GUI to indicate that the user should use the visual buttons. This helps, if, for example, the speaker was muted and the user cannot hear any of the help given through the VUI.

These are only two variants of this dialog, which is the core functionality of the application developed during this thesis. There are many options, how the dialog can change. For instance, the user can start speaking again after the VUI asked to repeat the first time. Additionally, there are a lot of options on how to word the answer. The user can simply say the command suggested by the VUI, but can also use her / his own words to paraphrase what should be said. Mapping these answers is a problem that will be covered in section 5.3.

Sample dialogs are very flexible, because they can applied to the development of every VUI. It does not matter if the VUI only understands commands or if it is able to understand

and manage whole conversations. Sample dialogs can always be used.

After writing dialogs, it is best to read them out loud. This gives a first glimpse of how the VUI possibly sounds like. In some cases, spoken language sounds much different than it looks like when it is written. At first, this can be done by the designer on her / his own. After they are finalized by the designer, it is important to read them out with another person. This is relevant to get a feeling for the interaction with the VUI. Each of the people involved in this conversation imitates either the VUI or the user [90].

The second, very important tool for designing dialogs on a high level are flow diagrams. They are best used after a few sample dialogs were written. Flow diagrams generalize the sample dialogs in many cases. They illustrate the flow of the application state and can be compared to call flow diagrams [53]. As a result, flow diagrams display all ways a user can take to accomplish a task using the VUI. The generalization therefore treats all possible answers, a user may give. Usually, they are grouped in only one way of the flow diagram. An example would be an agreement. The terms *OK*, *yes*, *yea* and *sure* can all be grouped to a single path in the flow diagram representing the agreement. This path is then named with only one of the possible answers. Similarly, the responses given by the VUI are only examples for a group of possible answers. In most cases, data changes. Since data depends on the exact example, it is not a problem to generalize it.

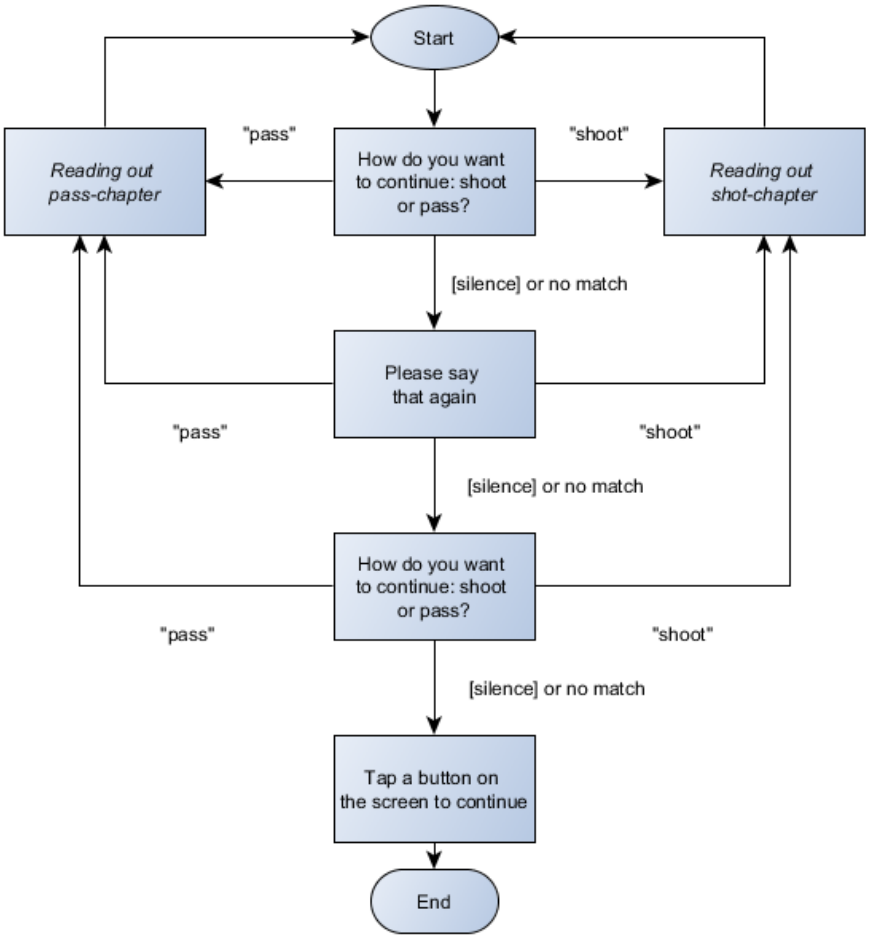


Figure 4.2: An example for a simple flow diagram

Figure 4.2 shows an example of a flow diagram. It represents the sample dialogs described previously in dialog 4.1 and dialog 4.2. As can be seen in this example, flow

diagrams consist of a starting point, a sequence of processes and forks which result in different branches and an ending point. Processes, represented as quadrangle, contain a sample response of the VUI. This is a sample, because the wording can be different due to variation or different data. Paths between processes represent what the user says. Similar to what is said by the VUI, these terms are grouped. One of these groups is *shoot*. The user does not necessarily has to say the exact word "shoot". Depending on the implementation, phrases like "I want him to shoot", "make a shot" or even "kick the ball" are covered by this branch. All of these phrases would recognized the same way by the VUI and lead to the same node. This flow diagram shows only a small snippet of the whole VUI, which is highly interwoven in this case. Therefore, reaching and ending point here means that the process starts again, with the application reading out the next chapter. This is also indicated in this diagram.

This is a very simple example. Especially for more complex VUIs, flow diagrams help to structure the dialogs. It is then helpful, to organize functionalities into different flow diagrams.

High level dialog design also includes steps to decide whether a dialog should be able to use multimodality or not. Especially if the designer judges that it is suggestive to use multimodality, this process also answers decisions on when to use another modality and how to use it. Because this is mostly the screen, decisions have to be made on what to show on the screen in certain situations so that all modalities fit together.

Especially if VUI and GUI teams are separated, it is important to use visual mockups and wireframes, to clearly exchange the ideas. VUI and GUI designers must work closely together to ensure a fine user experience. GUI designers usually care about the layout first. This is often done by creating wireframes. They represent the skeleton of a GUI, only concentrating on the layout. This is done without the usage of concrete content. Instead, boxes and sample texts are used. Often, wireframes are designed without color. They are only about the rough layout [34]. Figure 4.3 (a) shows a wireframe for the player of the application developed during this thesis. The player is the visual representation of the VUI in this application. As can be seen in this wireframe, there are no images selected yet, neither are texts. It only shows the position of the images, headings and buttons. This can be used as basis for discussions about the layout and for further, detailed work.

In contrast, a visual mockup shows the product closer to how it should look like when it is developed. There is a wide range of mockups. They can be professionally designed and look exactly like a screenshot or designed similar to wireframes but with real content. Obviously, there are many levels in between. Which one should be used always depends on the situation. Often, changes on existing GUIs can be shown by a professional mockup, because screenshots of the existing software interface can be manipulated easily. In case of a new GUI, this requires much more work. Therefore, it is often only used for larger professional projects.

A mockup is a more detailed wireframe with concrete data and images close to the final product design. It is obvious that there is a correlation between the wireframe and the mockup. The mockup is based on the wireframe and extends it with more details. Of course, designers can differ from the wireframe, if they realize, that the user experience suffers while designing the mockup [34].

The contrast between a wireframe and a visual mockup can be seen in figure 4.3. This figure is divided in two sections, (a) shows the wireframe and (b) the visual mockup. The reader can easily realize that both are correlated to each other. However, the visual mockup (b) shows many more details. Real texts are used, icons are used and, the style of buttons is more detailed and includes colors. It is noticeable, that a mockup is closer to the final design and therefore closer to a specific platform as well. For example, the

design guidelines between Android and iOS differ dramatically [87] [81]. This starts at the layout, but also includes different icon and font styles. As a result, different mockups are necessary for different platforms.

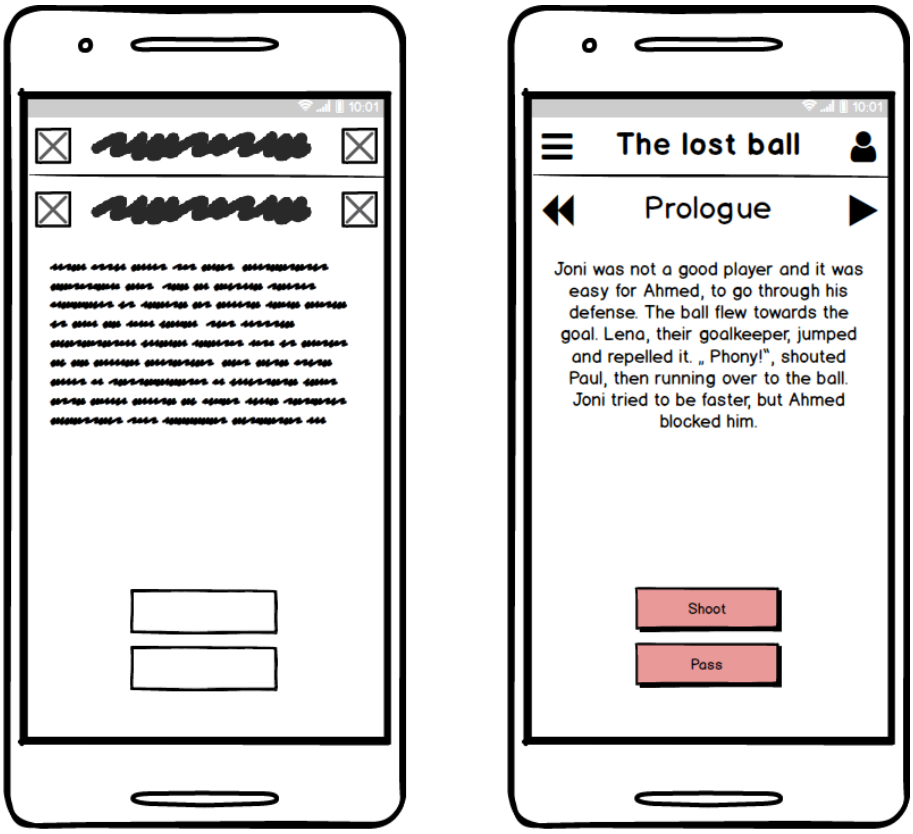


Figure 4.3: Difference between a wireframe and a mockup

Mockups and wireframes are important for VUI designers, because they need to know how the GUI will look like to make sure that the VUI fits to the GUI. The teams need to work closely together and mockups are one of the most important ways to exchange ideas.

Cathy Pearl additionally mentions prototyping tools in her book [90]. These tools help setting up a simple prototype. They are just emerging. Starting from examples, the designer provides the input on how to create messages. The designer can add parameters and an actions that should be executed. This is a quick and simple alternative to test, how a VUI feels like. Noticeable tools are api.ai [123], wit.ai [124] and Nuance Mix [18].

This section covered the basic tools to create dialogs at a high level. The results of this step of the development process are the input needed for the next steps. These include the detailed design of the dialogs. This step will be explained in the next chapter. High level design decisions are also important as a basis of decisions regarding technical questions. A more technical overview will be given in Chapter 5.

4.3 Detailed dialog design

After the dialogs are designed at a high level, they need to be elaborated. Therefore, the designers have to take a closer look to the dialogs themselves. The goal is always to create dialogs that are as natural as possible so that the user does not have to learn anything new. However, people are constantly using new ways to interact with UIs. Although, this always takes a while until users are familiar with new interaction ways. This happened to common gestures used with touchscreens and will also happen with VUIs [16]. In the meantime, VUI designers have to make sure people can use what they are used to when communicating with other humans. The overall goal of a VUI must be to provide a benefit. Therefore, good quality is necessary. Until now, the main focus was on the speech recognition. Today, this shifts more towards conversational dialog design and NLU [90]. The design principles aim to minimize cognitive load, maximize efficiency and clarity and ensure a high accuracy [17].

Detailed dialog design concentrates on the correct wording at first. This is even more important if the VUI does not have a visual component. Especially in this case, it is very important that dialogs are kept short and easy to understand. Therefore, data and options should be displayed short. If a screen is available, it can be used to display more complex data. But if not, the user should only have a few options to choose from. Options can be shortened by reading out only two items and telling the user that there is more. Another strategy to avoid this situation is to simply do not offer any options at first and only give the user options on how to go on as help. Then, the VUI obviously must be able to parse much more possible inputs. It also depends on the users knowledge. If the user already knows enough about the application, then she / he can easily interact with it in the absence of any given options.

For lists, the number of items can easily be reduced by asking a few questions before displaying them. As a result, lists are filtered. Additionally, the lists can be read out faster, if the user only has to review the data. If the user should be able to select or edit an item, lower reading speed helps the user to keep track of the data [91].

A good dialog design makes it clear to the user what to do next. Therefore, if the VUI asks a question, this question should be placed at the end of the response. For instance, there is a big difference between “What should Ahmed do? Shoot or pass?” and “Ahmed can either shoot or pass. What do you want him to do?”. The latter output puts the question to the end. This prevents the user from directly answering the question. This is the main problem. Because most users tend to answer questions straight away, they interrupt the VUIs output.

If options are available, they should be clearly presented. Designers should therefore not present more than three choices and keep the explanations brief. However, the explanation does not have to be short each time, because designers cannot assume that the user knows what to do (especially for new users). If complex tasks can be accomplished through the VUI, they need to offer enough possibilities to get help. Users should be able to ask anything and get sufficient explanation. To not confuse the user, only questions necessary should be asked by the VUI. As a result, designers should find out, if the data really needs to come from the user or if it can be received in a different way. An example for this can be the type of a credit card. By asking for the card number first, the system often can infer the type of the card because it is coded into the number. Additionally, a question for data should only refer to a single piece of information at a time. This helps to not confuse the user. Instead of saying “Please tell us your credit card information. We need the credit card number, the expiration date, the name of the cardholder and the security code.” the VUI should ask: “OK, we need your credit card details. What is your credit card number?”.

Note, that the latter expression also includes a question whereas the first one does not. Questions help the user to know when the system's turn is over and when the user's turn starts. User prompts also help, if it is not possible to ask a question. Those expressions also should not contain any special texts or shortcuts. If these are necessary, they need to be explained. Tech or legal terms should be avoided.

When reading information out, it should be done so in consumable pieces. For example, letting the user review the credit card information, the VUI should not read out all the information at once followed by the question "Is that correct?". Instead, similar to the input, each piece of information can be checked at once. This could be something like "The credit card number is 1234 5678 1234 5678. Is this correct?".

The VUI should also never blame the user for doing something wrong. Since there are no limitations, VUI designers need to expect the unexpected. Speech is an unconscious activity. As a result, users do not think about the choices they make, their pronunciation or their sentence structure. Additionally, the user should always have a way out. On a desktop, the user can easily close the window. Because this is not possible using a VUI, the user must have an easy way to return to the main menu or start over. Also, designers should expect users to ask for help.

Furthermore, it is noticeable, that designers should always rely on text, not on intonation. Intonation can change due to the used text to speech (TTS) system [1].

The biggest design challenge when interacting only through sound is that the data is non persistent. This means that after the data is presented via the speaker, it is gone. In contrast, a screen shows data for a much longer time. Also, the user has no possibility seeing what the system has understood. As described in section 2.2, this can be compensated by the usage of multimodal interfaces. Even a small screen can significantly reduce cognitive load, if it is combined effectively with the VUI [17].

The details of the dialog design is mostly about the language. It is important that words are carefully picked, because they are always context sensitive. Especially when connecting utterances to a full conversation, the cohesion between single expressions is important. It is the glue that keeps the conversation together. In spoken language, it is very important that pronouns and time adverbs are replaced with information given before. Also, it is important to use discussion markers. These help to keep track of where the conversation currently is. There are a few categories which should be used to structure dialogs [93]:

- **Enumerations** like first, second, for one thing, to begin with, next, lastly ...
- **Reinforcing** such as furthermore, moreover, in addition, above all ...
- **Equative** for example equally, likewise, similarly, in the same way ...
- **Transitional** like by the way or now
- **Summative** like for example then, in conclusion, to sum up ...
- **Apposition** such as for example, for instance, that is ...
- **Result** like hence, so, therefore, somehow ...
- **Inferential** like else, then, in that case ...
- **Reformulatory** like for example better, rather, in other words ...
- **Replacive** for example alternatively, on the other hand ...
- **Antithetic** like instead, in contrast, by comparison ...

- **Concessive** such as anyway, besides, however, still, yet ...
- **Temporal** like for instance meantime, meanwhile ...
- **Attitudinal** actually, strictly speaking, technically ...

Making sure dialogs contain those words help the user to orient in the dialog structure. Cohen et al. call this conversation management. It can be used to gently indicate the user what the next step is. Additionally, expressions like *by the way* can be used for just in time instructions. Using the word *Oh* indicates a cognitive mismatch and can be used to indicate that something was misunderstood. If people recognize the word *Oh* they feel not blamed and accepting that something went wrong. Designers should make sure to not use it too often [103]. Expressions like *actually* are considered to be the real truth and can be seen as a reaction to go forward. It is also used to indicate other options and to recover from errors. If more than one option is available, designers can make use of the term *otherwise*. It mostly indicates unpredictable options. User feedback is very important, telling the user that what she / he said was understood, can simply be done by starting with the word *Okay*. Letting the user know that she / he was understood is considered polite and so is being sorry. If something goes wrong, the VUI should be sorry, as humans would be as well in this situation. Designers should be careful with the word *must*. It indicates a social, interactive meaning and carries also a logical, probability. It is very strong. Often, it can be replaced so it is still available for very important sentences. In contrast, *may* socially means asking for permission. Its logical meaning is to indicate possibilities. Therefore, designers should be aware of its usage.

These words are all English. However, it is important to note that similar words exist in every language.

Not only the words are important, the sentence structure is also. Open-class sentences leave room for other information and speculation, whereas closed-class sentences do not. The designer should tend to use close-class sentences, but this is up to the particular situation. In general, the old information known from previous parts of the conversation is called “topic” and the new information in a sentence “subject”. What comes last in an expression is automatically in the focus of the listener (End-Focus Principle), so designers should make sure that the subject comes last. As a result, listeners remember the new information better. A tool to achieve this is the usage of active or passive voice.

Designers should be aware of differences between spoken and written language. There are usually words in every language that are only used either written or spoken. In English, one of these examples is *that* and *this*. *That* is used in spoken language, but *this* points forward and is therefore not usable in spoken language, because the listener does not know what comes next. Additionally, especially in spoken English, contractions should be used because they are more natural. There are also differences in grammar which need to be considered. For example, in English there is a difference between the future forms *will* and *going to*. Whereas *will* indicates a spontaneous action, *going to* usually points out planned intentions and shows, that the future is on the way.

These guidelines help to create better dialogs in general. However, there are some specific aspects designers have to consider. These will be explained in the next sections.

4.3.1 User confirmations

The user needs feedback. Because the user cannot see what the input was, the VUI must be able to confirm what was understood. The problem is, that not every transaction should be confirmed. Over-confirming drives people crazy and should always be prevented. As a result, designers should make sure that actions of high consequence (publicity visible

/ money or other people involved) will be confirmed by the user [1]. This means, that the system explicitly has to double check the action or at least will be asked without a reaction.

A good work flow for designer is not to use a confirmation and then think about what would happen if the action goes wrong. Depending on how bad the consequences would be, designers can then decide to provide feedback on different forms. An additional screen can help providing feedback in an explicit, but not too bothersome way. A screen could also be the best choice, if the type of information that should be confirmed cannot be described good using speech.

In general, there are two types of confirmations: explicit and implicit confirmations. Explicit confirmations asked the user directly to confirm, such as the expression *I heard set the timer, is that correct?* does. Only if the user confirms, that what the VUI understood was what the user actually meant, the action will be performed. In contrast, implicit confirmations encourage the user to interrupt, if something was misheard. An example for an implicit confirmation would be: *OK, setting the timer*. This expression repeats what was understood and performs the action. If the user did not want to set a timer, this can be easily undone. In this case, the user could also quickly say, that the action should be canceled.

Deciding when to use which type of confirmation is usually up to the designer. This decision can be made for every single dialog, but can also be decided depending on the confidence of the speech recognition system. The results of the speech recognition are usually ordered by their confidence or contain a value of confidence. For example, the system can ask for implicit confirmation if the confidence is higher than 80 percent; use explicit confirmation if the value is between 80 and 45 percent and tell the user that nothing was understood if the confidence is lower than 45 percent. These values obviously need to be adjusted depending on the situation.

Depending on the system, there are other ways of confirming the action. For instance, if the system controls the home, turning off the lights can just result in turning of the lights. If this goes wrong, due to misunderstanding, it is not a problem for the user. If everything is as expected, the user can actually see that the lights turned off. This does not have to be confirmed separately. If it takes a few seconds, no confirmation is needed in such cases. In case there is a delay for more than a few seconds, the user should still get a confirmation.

Generic confirmations like *Okay* already help to create a more conversational VUI. Developers can randomize them and add them to many dialogs. This gives the user a more conversational feeling and also the safety that what the user said was understood.

Especially if the user inputs complex information, it is helpful to use a screen to display the data and let the user confirm. Users mostly cannot remember more than seven items, even if they said them themselves. The visual confirmation gives the user also more time do decide, if the data was correct [90].

4.3.2 Stop detection

To create a natural and fluent conversation it is very important to detect when one participant has finished speaking. Then, it is required to take turns in the conversation. The timeout in which nobody speaks and which marks the turn is called “stop”. Unfortunately, detecting stops is not always easy. Human speakers vary the duration of steps depending on the conversation. A good VUI has to recognize this.

The VUI needs to be trained on the duration of pauses and when to detect stops. Otherwise, this results in an awkward dance of starting and stopping the speech recognition. The normal timeout should be around 1.5 seconds. However, it should be flexible since it is different in various states. Sometimes, users need to read out complex data and pauses

to group the data. This is for example the case when reading out a credit card number, which is usually naturally grouped into four digits. The VUI need to be able to wait longer in such cases, so the time between stops needs to be flexible. The timeout itself therefore depends on the data.

A no speech timeout (NSP) describes the end of the conversation. This is the case, when nobody says something for a longer time. A recommended duration is ten seconds. Sometimes, NSPs are unexpected and should result in error handling. If the VUI detects multiple NSPs in an unexpected step of the dialogs, this indicates that the user is stuck. Then, the VUI should provide some additional help. Developers should try to keep track of NSPs and figure out, why they happen. Designers also can think of their dialog design and imagine, what would happen if the user simply does not say anything.

In contrast to a NSP, the user also can speak too much in a short amount of time. This is normally the case because of limitations of the speech recognition software. In general, this is called too much speed (TMS). If this happens, the VUI should gently try to give the user hints to shorten the input [90].

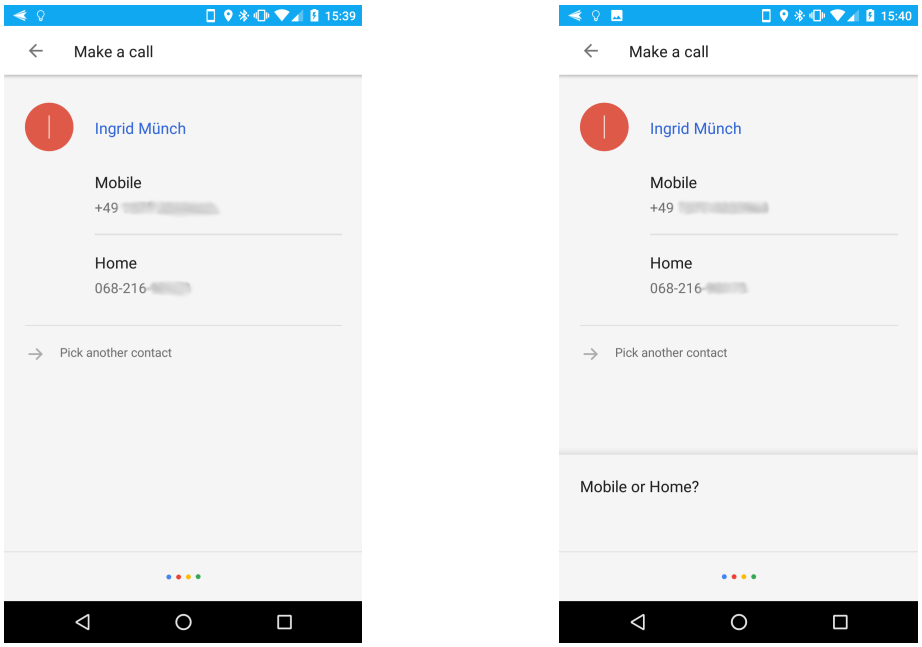
4.3.3 Helping the user

One of the major problems of VUIs is that information is volatile. The user cannot look at it as long as she / he wants to. After the information is displayed, it is gone. This requires special actions as the user might not get all of the information which was presented or could not understand everything. In many cases, the user therefore needs to get help. This can be done by providing more information about what is expected or by providing examples.

Another way of helping the user is by using disambiguation. This is mostly necessary because humans are not always clear and VUIs run into the same problem. There can be different cases, in which the VUI needs to disambiguate. First, there can be a lack of information. The VUI then simply has to ask for more information. Second, there can be too much information or multiple answers that might match. Designers then have to decide, how to handle these situations. They can decide to simply take the first matching answer, reject all possible answers or disambiguate by focusing on one answer and asking the user again for clarification. Which way to choose is up to the data and to the confidence of the accepted answer.

In general, the user should not be told every specific constraint. Instead, the VUI should give the user the freedom so that the user can indicate her / his answer defined in the user's own words. If something goes wrong, the VUI can try to automatically correct the input or disambiguate. Figure 4.3 shows an example of such a case. The user tried to call a person, but the VUI realized that there are two different phone numbers that match. Consequently, the VUI does not know which one to call. There is a lack of information. The VUI then asks the user to provide more information: "Call Ingrid, sure. Mobile or home?". Additionally, this disambiguation is supported by displaying both options at the screen. The user has also the option to use the screen as an input method by tapping the preferred number. In case the user does not say anything or does not provide the needed information, the VUI tries to help the user with a more detailed help message. In this case, Google Assistants output was: "To keep going, just say which phone number you want to use. For instance, you can say the first one". If the user still does not answer in a helpful way, the VUI tells the user to use the GUI by saying "Go ahead and tap one of the options on the screen".

If a user does use the words *either*, *neither* or *nor*, the VUI has hints to decide on which answer it should decide for. It can also use these words while outputting information which increases the credibility of the system.



(a) Audible disambiguation with simple support on the screen

(b) Disambiguation with hint on the screen after no input

Figure 4.4: Disambiguation on the screen

Also, the user should never have the feeling that she / he has been left hanging. If the VUI needs information, it should ask for it. As long as the dialog continues, the VUI always needs to give the user a hint for what is expected [90].

Helping the user also includes making sure, that the user can ask for help in all situations. Expressions like *I'm confused*, *Uggh* or *Where am I?* should immediately trigger help for the user.

4.3.4 Error handling

Another result of volatile information is that a VUI needs high-quality-error-handling capabilities. In the best case, it should not be visible to the user that an error has occurred. However, today's speech recognition technologies still do not have high enough accuracy. As a result, good error handling is essential for an acceptable user experience. In addition, designers should keep in mind that errors count much more than successful workflows. Therefore, designers have to understand that neither humans nor machines are perfect and this should be understood by the user. In fact, every error is recoverable. Abi Jones, design lead at Google, describes this fact as follows: "When you talk to a human being, there is never an unrecoverable error state" [90].

There is a set of common errors. First, the VUI can reach a state, where no speech was detected. The solution is then to ask for more input (speech) explicitly. This should be done, if the VUI is audio only and not supported by a screen. If there is a GUI available, it might be an option to do nothing and wait for the user to react. Second, speech can be detected, but not recognized. Then, the user should be asked to repeat what was said or the VUI can remain silent, so the user repeats her- / himself. Third, a common error is to recognize the speech but not being able to match it and not having an answer. This should be avoided in all cases. Lastly, the VUI can recognize the speech in a wrong way. An incorrect recognition can happen, if the automatic speech recognition tool returned the wrong text. This can sometimes be avoided by working with N-best lists as results,

so the VUI does not depend on a single result from the speech recognition. However, in this case, the user must be able to correct the VUI. Most users do anticipate the next step and therefore realize, if the VUI recognized something incorrectly and behaves in another way than expected.

If an error occurs, it needs to be escalated. There are several ways to do so. For example, in a first step, the user could be asked to say the city and state. If this does not result in an acceptable answer, the VUI can give an example. For instance, the VUI could help by saying *Please provide the city and the state, like in Los Angeles, California.* For most users, it is easier to copy patterns than to understand them without examples. If this does not work either, the next step is to ask the user to write (if a GUI is available) or spell it. In all cases, the user should never be blamed [90].

The first challenge is to find out, what the problem initially was. A common cause for errors is an expression, which is not grammatically correct. This means, that whatever the user said, could not be matched to the grammar defined by the VUI designer. If the VUI still accepts false speech recognition results that can cause serious problems. Obviously, the best case is to prevent errors, which is described in section 4.3.1. If the cost of a fail is low, the worst that can happen is that the user loses a few seconds. If the user is able to barge-in, which means to interrupt the VUI speaking, this is most likely even less. Recovering from errors is then easily possible, because the user tries to correct the VUI by her- / himself. From time to time, it is also recommended to confirm grouped items. This increases efficiency and also the confidence in the system itself. Designers and developers also must make sure that the VUI recognizes if the user is doing the same mistake over and over again. This should result in a more detailed error message, so the user realizes what was done wrong. Additionally, VUI developers should keep track of cases like this (e.g. in log files) so designers can react to possibly misleading dialogs.

The most important aspect is to understand the root cause of an error. This can be the dialog where the problem is discovered. However, these kinds of errors can usually be corrected by the user itself. It is more problematic if the root cause lies in previous dialogs. Then, it is often not very easy to figure out where the problem comes from.

If the user runs into a problem, it is not very helpful to describe the problem in more detail. Instead, the VUI should offer the user help so the process can go on. This can be done by showing another way out of the problem.

In general, errors can be handled by giving the user more specific help about what is part of the grammar definition and what went wrong. Then, the escalation is for more details. It is helpful, to provide examples in these cases. This procedure is called progressive prompting [121] [118]. If it is clear, what to say, then the VUI can also perform a rapid reprompt. This is a short expression indicating that the system did not understand what the user said and none of the options which were possible could be matched. The easiest way to indicate this is the expression *I'm sorry*. This gives the user no detailed information about what went wrong and therefore sometimes requires an extra step. It should also not be used in open questions because the user has too many options to answer. However, rapid reprompting can be used as general first level error handling. If the user still has trouble, the VUI can then give a more detailed help message. Whenever rapid reprompts are used, their wording should be diversified. Rapid reprompts are in most cases preferred by users as shown by some experiments run by nuance. The results of these experiments are shown in figure 4.5. It shows, that 80% of the users prefer rapid reprompts.

The user should always have the possibility to escape, with commands that are already known. If an error is already known, a second error of the same type, which is occurring again, should be handled in a different way. This can be alternative wording. After an error occurred for more than the second time in a row, the VUI should offer another way

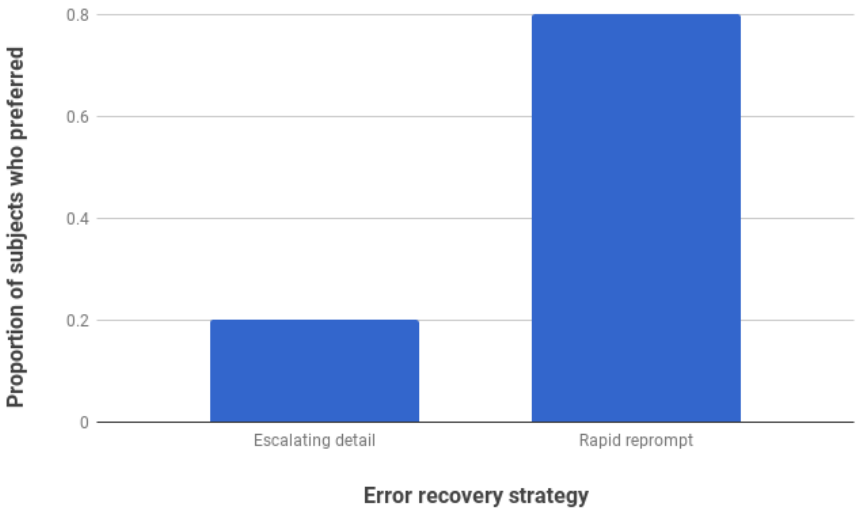


Figure 4.5: Result of user's preference regarding error recovery [17]

out of the situation [17].

4.3.5 Voices and intonations

The output of the VUI is not only about the wording, it is also about how something is said. Today, speech synthesis is good enough to create different voices and / or intonations. These should be used, so that the VUI fits in the use case it should fit in.

As David Crystal [20] points out, language carries both a verbal and a nonverbal meaning. The nonverbal meaning describes the prosody. It helps the user focus on what will be heard and also helps to get the right context. This is done by giving aid to identify the grammar structure. It is often the case, that the same words have different meanings. Then, intonation and / or grammar can be used to clarify the meaning. Also, the intonation can mark the important parts in a sentence.

Stress in the context of intonations can change the meaning of words. It is defined as the variation in loudness that differentiates strong and weak syllables. It helps the users keeping track of the conversations. This is because stress can be used to highlight the new information in a sentence. As a result, it gives designers more options to structure their output.

There are many default patterns, which help to create a more natural VUI. These patterns vary from language to language. In this document, only the most important ones will be mentioned. The most common patterns are:

1. **Rising-Falling, final:** usually indicates declarative sentences
2. **Rising:** indicates questions, that can only be answered with yes or no (polar question)
3. **Rising-Falling, non-final:** this pattern is used in more complex sentences

The intonation is especially important when concatenating messages. Lists, for example have different intonations depending on the position of the item. If another item is following, the intonation rises. For the last item, it falls. Polar questions, which can either be answered with yes or with no, are usually indicated by a rise of the intonation.

In contrast, questions asking for more complex information are implementing pattern 1 (rising-falling, final) when asked for the first time, but using pattern 3 (rising-falling, non-final) when asked for the purpose of repetition or clarification. Either-Or questions are basically the same than lists with only two items and behave in exactly this way regarding their intonation.

In written language, pauses are indicated by commas. Obviously, this is not possible in spoken language. As a result, there is additional silence needed after nonrecognition messages or if there are commas or colons. No silence should be used if between the end of a paragraph and the beginning of another one. Also, some colons are purely graphical and should not cause a pause. This is for example the case, if a time should be read out.

5 Technological design

This chapter describes the technological background of a VUI system. Because this system is very complex, it can be divided into multiple sub systems. Each of these systems will be described in this chapter. It is important to have a basic understanding of how these systems work in order to put them together to the whole VUI system. Due to the focus of this thesis, the components will not be explained in detail. However, the basic underlying theory will be explained.

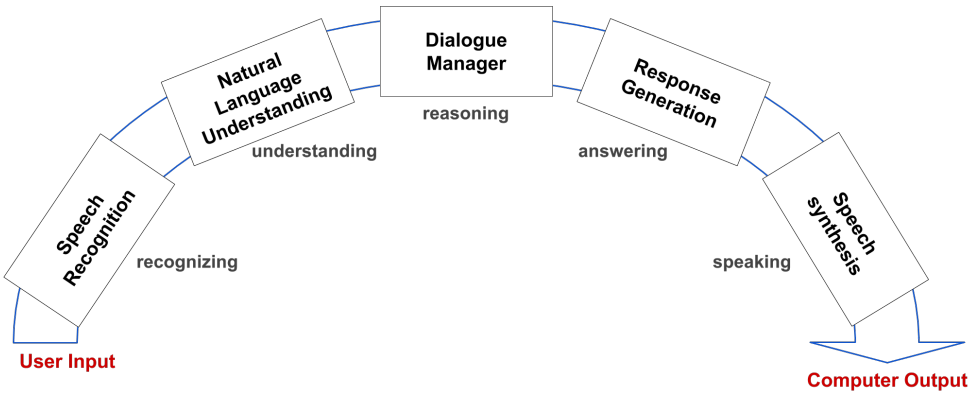


Figure 5.1: Overview of parts of a VUI application [13]

Figure 5.1 gives an overview about all the components that are typically part of a VUI. Before the VUI starts its work, the user input must be captured. This can be done by the application itself, but can also be part of the VUIs functionality. The audio result, in figure 5.1 referenced as user input, is the data that the VUI starts its work with.

The first step is the speech recognition system. It analyzes the audio file and outputs the understood words. It recognizes the words. Usually, it has knowledge of a vocabulary. Therefore, it is often necessary to configure the speech recognition module with the language that should be understood. Modern speech recognition systems also analyze the intonation of the speaker and return data about the speaker's feelings. After the words themselves were recognized, the next module takes over. The NLU module tries to understand the words and checks, if the understood words make sense. This module can correct the speech recognition, because it knows about the grammar and the semantic rules of the language. Additionally, this module tries to understand the meaning of the words. Therefore, the module combines the feelings and the words. The third step is reasoning. The dialog manager (DM) is the heart of the VUI. It has a strategy to understand what was said and decides how to respond to the input. The strategy therefore represents how the VUI acts. Its result is the input for the next step: the response generation. The output is the answer to the user. The response generation includes the creation of a text that will be used as a response to the user. The last step is the speech synthesis. It turns the generated text into audio. This audio is the output produced by the VUI and the response to the user [13].

The following sections describe these modules. Therefore, their internal functionality will be described. Furthermore, these sections give the reader an idea of what can currently be done and, more important, what not.

5.1 Speech recognition

The purpose of the speech recognition module is to translate auditory input into text. Speech recognition is not speech comprehension. It only gets the text out of an acoustic signal [13]. Therefore, the recognized signal must be interpreted. In this section, the word signal will always be used to describe the auditory input of the speech recognition module. The speech recognition only translates the signal into text. It does not check any semantic or syntactic language rules. Due to this fact, it can occur that the result is not a proper sentence that is based on the languages grammar. A correction of this is of course done in later processing steps.

This chapter first explains the general technologies that are used to analyze the audio signal and match the signal to words. This is the main purpose of the first subsection. In a later part of this section, different speech recognition services will be compared. This includes not only a comparison of their pricing and availability, but also a comparison of their performance. This comparison is done due to the fact, that one of these providers has to be used for the project described in this thesis (see Chapter 6).

5.1.1 General technologies

This section describes the general technologies that are the basis of most modern speech recognition modules. Different speech recognition engines rely on different technologies. First, there are differences which acoustic units engines use for recognition. Some use words (especially if the vocabulary is small) and others use syllables to interpret the phrases. Syllables are especially necessary for recognition if the vocabulary is large. The underlying mathematical model is often statistically. Most modern systems are based on hidden Markov models (HMM).

HMMs are one option to create a signal model. The term signal is here used to describe every observable output of real world scenarios. A model helps to process the signal, learn about the signal source without having the actual source available or to realize practical systems, such as speech recognition. Signal models can be deterministic and use specific properties of the signal. If not, the models are mostly statistical. Since most of the models for speech recognition are statistical, these are in the main focus of this section. Other types of statistical models are Gaussian and Poisson processes. However, this section concentrates on Markov and hidden Markov processes.

Markov processes describe a relationship between N distinct states. In a simple model, each state is connected to each other state. This can be seen in figure 5.2. If this is the case, the model is called fully connected Markov model. Each edge is captioned with a probability a_{ij}. It describes the probability of going from state *i* to *j*. Obviously, the sum of all a_{ij} per state must be one. In addition, each a_{ij} has to be bigger than zero.

In a Markov model, the probabilistic description is only based on the current and the previous state. This example describes an observable Markov model. In an observable Markov model, the output is the set of states at each time where each state corresponds to a physical event.

In contrast, the stochastic process of a HMM is not directly observable. It can only be observed through another set of of stochastic processes that produce a sequence of observations. If this case, the main questions on how to design the model are: What are

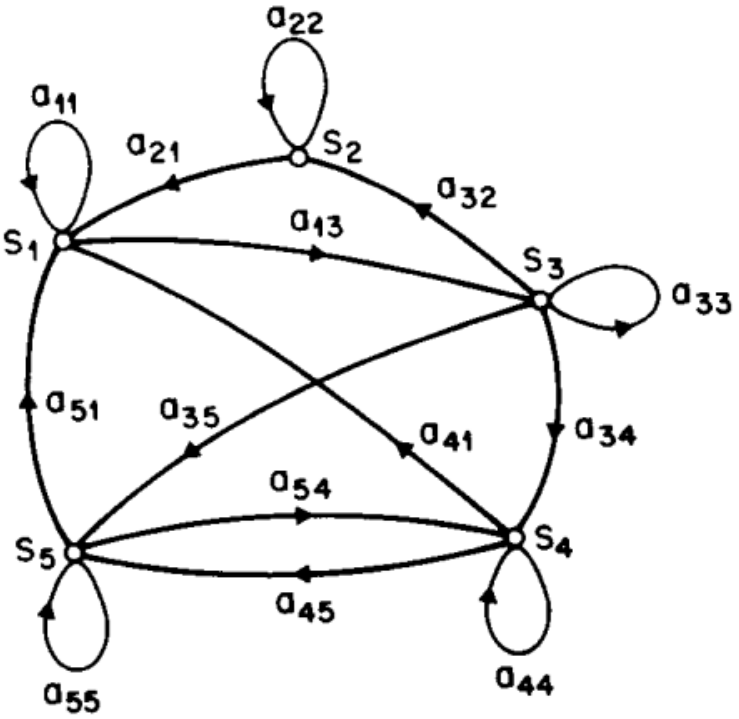


Figure 5.2: A fully connected Markov model [94]

the states and how many states should be in the model. A HMM can be described through five elements:

- **N** describes the number of states in the model
- **M** is the number of distinct observation symbols per state
- **A** is the definition of the state transition probability distribution
- **B** describes the observation symbol probability per distribution in state *j*
- π is simply the initial state distribution

A HMM can be used to generate observation sequences like $O = O_1 O_2 ... O_T$. There are a lot of different ways because of many different states and many different probabilistic values on how to change from one state to another. A compact notation for a HMM is therefore often described as $\lambda = (A, B, \pi)$.

In speech recognition, each word in the vocabulary gets its own N-State HMM. The signal is recognized as spectral vectors. These vectors are represented by codes, which can be mapped to the states. Training data helps to learn about the physical meaning of each state. By comparing a model to all other models, new words can be recognized. The word is then represented as the word whose model fits the best.

There are different types of HMM. The example described above shows a fully connected HMM. This type of HMMs are characterized by the fact that each state is connected with every other state. Depending on the use case, other types of HMMs may be more useful. The left-right-model (or Bakis model) is often used for speech recognition. In this type of model, no transitions to states whose indices are lower than the current states index are allowed. In other words, there are no backward moves possible. Sometimes, there are additional constraints added. One of these can be the limitation of the jump

size. That means, that a forward move can only be made to indices lower than the current index plus the maximum jump size. This type of model includes many more variations.

Other types of models are continuous observation densities, autoregressive HMMs and more. An overview of them can be found in cite [94].

There are a few implementation problems, which could be solved in the past decades. First, HMMs have problems with scaling. The computation quickly exceeds the precision range of computers. Second, much training data is needed. Especially left-right models cannot be trained with a single observation sequence. The third problem describes the difficulty to estimate the initial parameters. HMMs have many local maximums. The best initial parameter would be the global maximum, but it is difficult to set it to the local maximum so that this local maximum is also the global maximum. Unfortunately, there is still no sufficient answer. However, random initial states are adequate in most cases. Another problem is the choice of the model size and the type. There is no answer for this problem, each choice has to be made for every signal being modeled. The last problem is missing training data. In the past few years, an enormous increase of training data was observed. Today, much training data is available and this problem is obsolete.

While designing an HMM for speech recognition, there is another challenge: background silence. This can be be solved by detecting it and discriminating it, building a model that includes the background silence or subtracting the background silence from the signal [94].

There are other modeling approaches as well. Deterministic grammar approaches try to describe the words as a graph. Unfortunately, the model is not very flexible. As a result, the recognition hardly understands malformed sentences. Additionally, designing the grammar is very complex. Another approach is large vocabulary continuous speech recognition (LVCSR) [12]. This approach has no constraints and is based on a large dictionary. It is often used for specialized tasks. LVCSR makes use of N-grams, which are explained in section 5.2. They give the user the most freedom, which means the speech recognition is very flexible.

It is extremely important, that the speech recognition provides a good performance. VUIs need to recognized speech almost in real time. Because other steps are also necessary, the speech recognition cannot take too much time. Especially if the user should feel like in a real conversation, the system cannot wait for a long time on the result. There are two major ways to improve the performance: optimization and parallelization.

To optimize the recognition, the system should only compute the top N alternatives. N can be a parameter. However, a value around five is more than sufficient. Speech recognition systems can then approximate other words due to their positions in the sentence. Therefore, they use beam search (see [85] for more details about beam searches) to reduce the options.

Parallelization can be done by distribute the computation. Fortunately, the computation of HMMs can be split very easily. Furthermore, graphical processing units (GPU) are optimized for such mathematical calculations. Therefore, the speed can be increased by a factor of five when dividing the problem and compute it in a distributed environment using GPUs [13].

Today's speech recognition already produces high quality results. However, the speech recognition modules still have to deal with the same problems and they are still difficult to solve. Almost 90% of the use cases used while developing a VUI are ideal situations. This means an adult male speaking in a quiet room with no background noise and a good microphone. However, this is almost never the case in real world scenarios. Instead, VUIs have to deal with constant background noise and often even worse: side speech. In such cases, Karen Kaushansky shares the opinion to not tell the user to move to a less noisy

place. Instead, the problem should be escalated and the user should be offered help. If this does not help either, the user should use the GUI.

Sometimes, another problem for the speech recognition occurs: multiple speakers. Users are usually not alone. Then, the speech recognition must be very careful with its interpretation. However, modern systems are able to also analyze the voice and make sure only a single voice is interpreted. Additionally, children are a problem. They are often harder to understand. Furthermore, there is less training data for the speech recognition system to train it for children's voices. If the VUI is not completely sure, it should simply move on or show graphical alternatives.

Spelling can be extremely difficult for speech recognition systems. Especially names are very hard to understand for the system. One problem is, that they are often differently spelled, but sound the same. Another problem is that names are often not part of the vocabulary. It can then be helpful to check the users address book. If accessible, more user data can be very helpful for the speech recognition process and increase its quality significantly. If there is no chance to understand, what the user was saying, the GUI is the best solution. Sometimes, the speech recognition module can make use of additional data. This includes checksums of credit cards or a simple check for postal codes [62].

5.1.2 Comparison of existing speech recognition software

There are a number of existing services available. This section compares them. As a result, the best available software will be used in the application developed during this thesis. The comparison focuses on web services. For the comparison, two different audio files are used. The first audio file is professionally recorded and available from Google Cloud Storage [40]. The spoken text is *How old is the Brooklyn bridge?*. In this writing, it will be named as sample 1. The second audio file was recorded using a built-in microphone from a laptop in a normal environment. There are no significant background noises, but the quality is worse than the professionally recorded audio file. Especially the volume level is lower than the one of sample 1. It will named sample 2 for the following comparison. The text which was recorded is *The quick brown fox jumps over the lazy dog*.

Cordova plugin

The plugin provided by Cordova uses the speech recognition classes provided by Android or iOS (Siri), respectively. It wraps the functionality of the operating system into a commonly usable API. The usage is free of any cost, but it requires an active internet connection because the system communicates with Google or Apple servers.

This module is used in the implementation part of this thesis. Therefore, a description of the API can be found in paragraph 6.4.3. The following information is based on the Android version of the module. The reason for this is that the development was only done for Android, because developing an iOS application requires a more complex and more expensive development environment. This environment was not available for the thesis.

The plugin can be configured to return a number of N-best results. How many results should be returned can easily be configured by setting a parameter. The default is five.

The Cordova plugin internally calls the APIs of the underlying operating system. Because this thesis only includes the development of an Android application, the API called is always Google's Speech API [7]. Since this API is already part of the comparison, the Cordova plugin will not be tested as an additional API.

Google Speech API

Google also offers a representational state transfer (REST) API for it's speech recognition API [42]. Unfortunately, the usage of this REST API is not free. The prices, by the time of this writing, can be seen in table 5.1.

Table 5.1: Prices of Google's REST speech API

Monthly usage	**Price per 15 seconds**
0 - 60 minutes	Free
61 - 1,000,000 minutes	$0.006

To access the Speech API, it is required to create a project in Google's Cloud Platform console. The speech API must be enabled and a service account has to be created. Then, in a first request, the project has to be authenticated. This request returns an access token, which can then be used to send a request to the actual API. A detailed guide can be found at [41].

Both sample audio files were sent to the REST API. A response from this is formed in JSON and contains a set of alternatives. Each alternative contains a transcript containing the text that was recognized. In addition, the confidence level is also returned. The exact results as JSON can be seen in appendix C.

Table 5.2 gives an overview of the results. For sample 1, the API returned the exact text without a question mark. This indicates, that the understanding of the rising intonation at the end of the audio is not perfect. The confidence score of 0.987629 indicates that the API is very sure about what was understood. In fact, this result is very close to the original text. It is still noticeable, that the sentence starts with a lowercase character. The API returns very good results for sample 2 as well. The recognition is only slightly different to the original. Instead of the present version of the verb *jump*, the past version (*jumped*) was recognized. However, the confidence score indicates a very high confidence, which can be misleading. Although, the confidence level is high enough to properly process the answer in most cases.

Table 5.2: Result of sample audio files from Google's REST Speech API

Recognized text	**Confidence score**
how old is the Brooklyn Bridge	0.987629
the quick brown fox jumped over the lazy dog	0.97975445

Bing Speech API

Similar to Google, Microsoft provides an option to use speech recognition as well. The Bing Speech API is part of Microsoft's cognitive services which can be used through Microsoft Azure [77]. Like Googles API, Microsoft's service is not free. Their prices can be seen in table 5.3. It should be noted that each transaction can only contain utterances up to 15 seconds.

This API was also used to recognize both audio samples. Microsoft's API does not require to set up a project, but the API needs to be activated using an account. A subscription key can be assigned to this account. From then, this key can be used to request an

Table 5.3: Prices of Microsoft's Bing Speech API

Monthly usage	**Price per 15 seconds**
5000 Transactions	Free
Additional 1000 Transactions	$4

access token. Similar to the Google API, this access token must be included in the request to the actual Speech API. A detailed guide is available at [76].

Microsoft's API also returns it's response as JSON. The complete result can be found in appendix D. In contrast to Google's API, this API contains a parameter to define how detailed the result should be. This parameter is called *format* and can either be set to *simple* or *detailed*. If set to *simple*, the response only contains a single transcription. If set to *detailed*, it contains a set of results as N-Best list, including confidences. For both samples, both response formats can be seen in appendix D.

The responses of the Microsoft API can be seen in table 5.4. It shows both the *simple* and the *detailed* results. For sample 1, the *detailed* result only returned one result. For sample 2, there are five entries in the list. These five results can also be seen in the table. They are listed sequentially. This API has no trouble recognizing sample 1. Even the question mark is recognized correctly. As can be seen from the detailed result, the confidence is still only at a value of around 95%. In contrast, the second sample is not recognized very well. None of the returned examples contains all of the words of the original text. The closest result is the fifth one, which is indicated with the lowest confidence score. However, applications can react to a low confidence score. The score is never higher than 82%, which is mostly considered not high enough.

Table 5.4: Result of sample audio files from Microsoft's Speech API

Sample	**Recognized text**	**Confidence score**	**Format**	**Comment**
1	How old is the Brooklyn bridge?	-	simple	
1	How old is the Brooklyn bridge?	0.95739913	detailed	Only one result
2	Brown Fox jumps over.	-	simple	
2	Brown Fox jumps over.	0.8211723	detailed	First entry
2	Brown Fox jumps.	0.8211723	detailed	Second entry
2	Call Brown Fox jumps over.	0.7018736	detailed	Third entry
2	Call Brown Fox jumps.	0.7018736	detailed	Fourth entry
2	Brown Fox jumps over the lazy.	0.7018736	detailed	Fifth entry

wit.ai

Using wit.ai, developers can create entities. Whenever the application sends audio to the wit.ai interface, the service tries to understand this audio and maps it to the defined entities. Entities are defined by example. As a result, the developer only needs to provide a name for the entity and a set of example utterances. These can be added as written text or as audio (because wit.ai turns the audio into text and maps it to the entities). The more examples are provided, the better is the accuracy of the service. An example set could be

What's the temperature?, *How high is the temperature in here?* and *How warm is it?*. All of these phrases are questions asking for the current temperature. From a technical perspective, this can be described as a get request for the value of the temperature. If the user now asks *What's the temperature in here?*, the system will map it to the entity which was trained with the given utterances. This works even that this specific phrase was not in the training set.

The application offers a web API (there are clients for NodeJS, Python and Ruby available). The API can receive text and audio. If received, the application created in wit.ai tries to map the phrase to one of the defined entities. It returns the entity including a confidence level.

However, the real value of this service can be seen when the developers start to mark variables. For example, instead of getting the current temperature, the user could also try to set the temperature. Then, most user would use a phrase similar to *Set the temperature to 75 degrees.* In this example, the number 75 can be exchanged with other values as well. Wit.ai suggests to use it as a variable. In addition, developers can help with their knowledge and define phrases, values, locations and more as variables. Therefore, wit.ai has many built in types of variables. If an entity supports variables, the resulting response contains this value. As a result, the response contains the entity, the service's confidence and the recognized variables [124].

5.2 Natural language understanding

In the section 5.1, the speech recognition part of a VUI was described. The output is usually a set of possible words. Whenever a sentence is recognized, the speech recognition tries to recognize all words contained in this sentence. To create a higher accuracy, techniques of NLU are used. Sometimes, the two modules speech recognition and NLU are merged together and provided as a single module.

NLU describes the module responsible for analyzing the words that were understood by the speech recognition system. It tries to understand, what the user said. The NLU system is the front end to the DM, which is described in section 5.3. Most systems start with putting the words provided by the speech recognition into a legitimate word sequence. Therefore, probabilistic models based on the language are used. Additionally, each language provides a set of rules that describe the grammar of the language. These sets can also be used to verify the consistency of the phrase, especially if the language is known. In the rare case that the system does not know, which language the user speaks, this process is obviously more difficult. The step of applying the grammar rules is often denoted as semantic parsing. A third part of the understanding process is the dialog decoder. It collects information about the previous parts of the dialog and uses this information to figure out the most likely version of the recognition [13]. Adding information that is also available can increase the recognition rate of this step enormously. For example, McGraw et al. suggest to use the contacts of an user's phone book. Names are very hard to recognize, but it is very likely that the user mentions names that is already known. A list of all contacts can therefore help to figure out, what the user meant [73].

As mentioned earlier, NLU systems often use probabilistic models to figure out what the user said. The most commonly used model are N-Grams. An N-Gram mode, estimates the likelihood of a sequence of words of the length N by computing the conditional probability of observing the Nth word given the previous N-1 words. Unigrams estimate the likelihood of atomic words. Similar, bigrams estimate the likelihood of pairs of words and trigrams estimate triplets of words. This can be used for any length of words. N-Grams are trained by large corpora of texts, such as collections of newspaper articles [61] [49]. Google used its collection of books to create a very large set of N-Grams. They can

be viewed using the Google Books N-Gram viewer [39].

Unfortunately, N-Grams cannot always be used. They work very well for the English language, but each language has its own difficulties. Grouping languages helps. If languages are similar to each other (the same family of languages), then the same systems can be used. This is the reason why earlier systems only supported languages like English, German, French, etc. As Nouza et al. are pointing out, N-Grams cannot be used for Slavic languages. The reason for this is their lexicons. They are up to ten times larger than for the English language, due to the more complex grammar of the language. Therefore, more text is needed, which is barely available. In addition, the free word order makes it harder to understand the sequences. However, the techniques can still be used, but the confidence is significantly lower [86].

The output of most NLU systems are N-best lists. Each list entry contains a possible string that was understood. Additionally, most systems provide a value that describes the level of confidence. This value is calculated by the system and describes, how likely it is that the string correlated with it was what the user actually said. This list is the input for the DM, which is described in the next section.

5.3 Dialog Management

After the input was converted from audio into text, the DM starts its work. The DM is the heart of a VUI, because it decides how the dialog goes on. Therefore, it analyzes what was said and maps it to a certain use case. Then, it decides how the dialog should go on. This decision is based on the current phrase, but also on all the information available from previous dialogs. As a result, the DM must hold and manage all the information that the application has about the user and the interaction.

In general, there are two challenges. First, the DM has to figure out, which action should currently be done. Therefore, the text must be mapped to the available actions. Second, the manager must make sure that all needed information is available. If this is not the case, it has to ask the user for the missing information. This information can be a single term. An example for a single term could be *Set the temperature to 75 degrees.* The DM can figure out that the user wants to set the temperature. There is only a single additional value needed: the temperature. Since this value is already given, the manager can start the action. However, actions can require much more information and therefore be more complex. An example for a complex action could be booking a flight. The user typically does not provide all necessary information in a single utterance. Instead, the user would most likely say something like *I want to book a flight from Washington DC to Frankfurt.* The DM can now already figure out, which action should be done. In addition, two locations are given: Washington DC as the departure location and Frankfurt as the destination. To book a flight, there is a lot of more information necessary. For example, an application would need the desired date, payment information, passenger names and more. Some of this information can might be already available in the system. Depending on this status, the DM has to plan the next output of the system. It controls the dialog flow from the VUI perspective.

From a design perspective, it is better to let the user try to use her / his own words to describe the action instead of trying to teach users the exact wording. Therefore, the DM must be able to understand many different phrases and map them to the possible actions [91].

To figure out, which action should be taken, there are two general strategies: keyword spotting and full sentence spotting. The first strategy, keyword spotting, is the most common used strategy. It looks for keywords in the result of the speech recognition respec-

tively NLU module and interprets them. In contrast, full sentence spotting tries to interpret full sentences. It is more complicated and less flexible. However, it can be used to get more information out of the input phrases [13].

Furthermore, this section gives an overview about different strategies of DMs. These will be described in the following subsections. These strategies can be used separated from each, but mixtures are also possible. Whatever strategy is best, depends on the particular use case.

5.3.1 Finite state based

One of the simplest dialog managers is the finite state based dialog manager. It controls the dialog flow through a predefined set of states, which are connected. An example for such a control flow can is show in figure 5.3. This figure is used for a phone based customer control system. The states are connected. There are options to repeat the question for a state or the system can go on to another state. However, going on to another state is only possible if the necessary information was provided. Forward movements are possible as well. In this case, a state can also lead to transfer the customer to a service representative, which is here indicated as transfer [122].

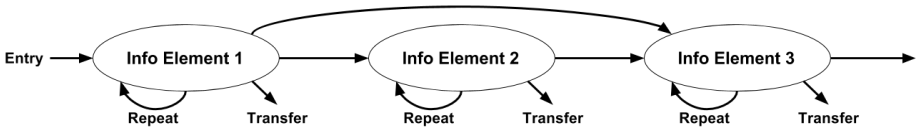


Figure 5.3: Simple example of a finite state based dialog manager [122]

This type of dialog manager always follows predefined states. Transitions may refer to different paths. Because all paths are predefined, this type is good for well-defined and structured tasks. The structure must be present during development time. However, finite state based dialog managers are not very flexible and inhibit the user's initiative [13].

5.3.2 Frame based

If the information needed can be stored in a form, frame based dialog managers can be used. They act as slot filling applications that try to fill all available slots with the information necessary to perform a given task. It can store information from previous tasks and ask for the information needed [13].

Kim et al. used this system and extended it to create a more useful DM. They used a frame based state representation to keep the complexity of the belief low. Furthermore, they used dialog examples to train the system. If the user's intention was figured out, the goal can be represented as a frame. This frame is in the beginning usually empty or only sparsely filled. Because there exist a large number of potential options to fill the empty frame sets, these options were grouped by Kim et al. Then, only groups of equivalent states are possible. Figure 5.4, shows, how such a frame can look like. The task given in this example is to get the weather forecast for a certain date. Therefore, three distinct pieces of information are necessary in this system: the time, the requested weather type (e.g. temperature, humidity) and the location. As can be seen in figure 5.4, the left example shows an incomplete frame. The location is missing here. The DM therefore needs to recognize, that the action is currently not possible and ask the user to provide

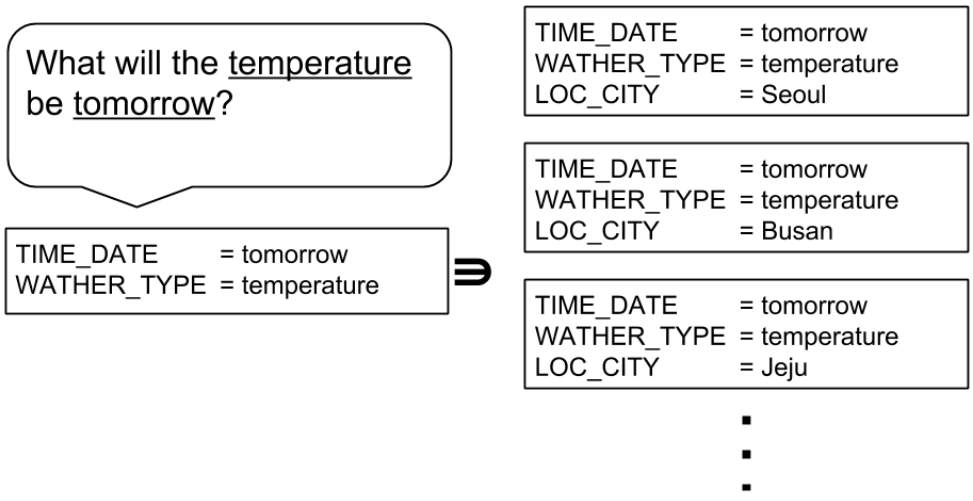


Figure 5.4: A frame based information representation [63]

the necessary information. In contrast, the examples given in the right of figure 5.4 show complete frames that can be used to perform the action [63].

5.3.3 Agent based

The main purpose of a DM is to maintain the conversational context and control the dialog. Today, there are many personal assistant existing which help user with simple, constrained tasks. Maintaining them requires significant effort. Additionally, they often cannot provide the desired level of sophistication. However, if the domain becomes more complex, the DM can act as an agent. The agent can do the reasoning and delegate special tasks to specialized assistants. In other words, it acts as a single point of contact for a set of assistants. In addition, it manages the state of the application and the dialog. This is especially useful if the application has to handle more natural dialogs, as their domains are more complex. In an agent based DM, the user can easily switch back and forth between the assistants. In addition, it is possible to change the physical context by using different devices.

Nguyen and Wobcke create an agent based DM which manages emails [83]. They describe the architecture of an agent based DM by example. These examples will be used in the following explanations.

Dialogs are mostly user driven. However, system initiative is necessary to recover errors and lead the user. The DM decides when to talk. Also, the DM as an agent coordinates the specialized agents. As a result, the architecture is modular which means that it can be extended easily. New functions or updates can be installed without much effort. This is one of the major advantages of the agent based design.

According to Wooldridge and Jennings, an agent should always have at least one information attitude [119]. Therefore, each supporting assistant must serve a purpose. Rao and Georgeff additionally define an assistant through three properties: belief, desire and intention (BDI) [95]. These properties represent the informational, motivational and deliberative states of the agent. They should be defined for each assistant and define its behavior. One example implementing these properties is the Procedural Reasoning System (PRS). It was developed by Georgeff and Lansky and is used in Nguyen and Wobcke's

email example [89]. It relies basically on a loop that queries events and chooses predefined plans based on the stored data. The result is then stored as well and the loop starts with the next query entry.

In the given example, the input events are basically the conversational acts by the user. Then, the system tries to identify the user's intention. After that, a task will be performed using an appropriate assistant. The output is a generated response. The loop starts over with the next input by the user.

The DM keeps track of the dialog data. Therefore, it identifies the user's intention and groups it into a class. This is done to group different utterances and manage them more easily. Nguyen and Wobcke also simplify the situation by assuming that there is only one speech act per utterance, even if it is possible that utterances contain more speech acts. A speech act is the class of the intention including its subjects, objects and other data that might be helpful. It is easier to identify the intention, if the DM has domain specific knowledge of the domain. Then, a domain specific dictionary can be created.

The dialog data contains also the dialog history. This is save as a stack of all conversational acts both by the system and the user. In addition, a second stack is held. This stack is called focus stack and contains all objects mentioned earlier in the conversation. This can be people's names, folder names or email items. Key phrases are also possible. This stack can not only be used to understand the user but also to create context sensitive responses.

Nguyen and Wobcke's DM is built out of five different parts that are called after each other. Firstly, the conversational act determination and domain task classification identifies the task domain. This is necessary, so that the required task plan can be loaded. Then, in a second step, the intention identification determines the actual task. This may requires more information than in the current utterance available. Therefore, a connection to dialog history stack and to the focus stack is necessary. For example, after first saying *Show me my emails from today* the system shows the emails from today (or reads them out). If the presumed answer contains an email from Paul, the user could possibly ask next: *Read the one from Paul*. The DM must then refer to the previously mentioned time object (today), which still has to be stored in the focus stack. Otherwise, it would might be unclear, which email the user is talking about. After an intention was identified successfully, its data is also added to the focus stack. The third step then performs the action by calling the assistants. After they are done, the DM lastly generates a response and outputs it to the user.

As can be seen from Nguyen and Wobcke's implementation, an agent based DM adds an additional layer to a set of existing assistants. As a result, more complex domains can be covered.

5.3.4 Plan based

Besides predefined, finite state based dialog models, there are also local managed based approaches. These approaches are defined through the fact that they do not try to pre determine dialog paths. As a result, they are more flexible. One of these approaches is the plan based dialog management. It divides a specific task into small goals and plans and tries to fulfill them. This is done by controlling the dialog interaction. By fulfilling all small tasks, the bigger overall task can be accomplished. This method especially supports a good structured way of defining the application state.

Wu et al. suggest a system that stores the data in trees. These trees are called topic trees. The root of a tree is called topic node and describes the overall goal. Middle nodes label logical relations and leaf nodes contain the actual information. Such a representation can be seen in figure 5.5. Here, the *Flight information* is one of the root nodes and characterizes

therefore a particular topic tree. Its middle nodes are connected via logical operators like *AND* or *OR*. Additionally, they are marked as *Primary Property (PP)*, *Secondary Property (SP)* or *Additional Property (AP)*. PP nodes store dominant information for this topic, SP nodes store detailed information about the topic and also need to be discussed with the user. However, nodes marked with AP store additional information and are therefore optional for the discussion with the user. Topic trees in this system additionally contain a bit that indicates if there is information stored. For leaf nodes, this is meant to indicate whether there is information in this leaf node or not. For middle nodes, this is done recursively to all leaf nodes. This simplifies the checking for nodes that still have to be discussed.

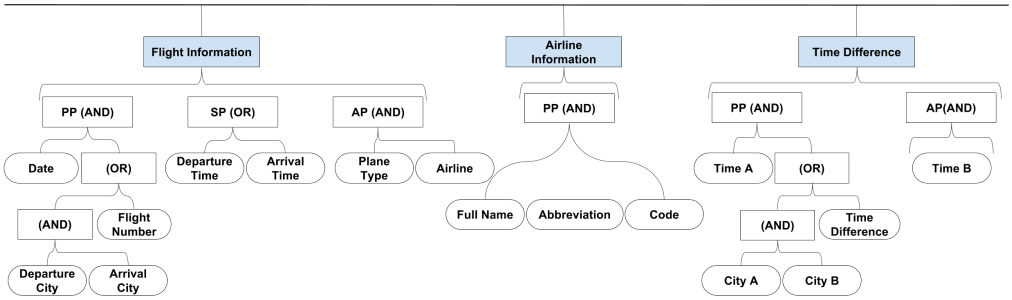


Figure 5.5: The information representation in a plan based DM [120]

Also, each node has response generation function attached. This function can generate text depending on the nodes information. This text can then be displayed or read out.

After such a tree was build, the system suggested by Wu et al. starts the reasoning engine. It traverses all nodes through all trees and finds similar information. If this is the case, these nodes will also be filled so that the system does not have to ask the same question for a different topic in the future. This step can also overwrite existing information. If this is the case, the system forgets old data. The reasoning also looks for action items that can be performed whenever all data is known. Then, the corresponding action will be performed [120].

As can be seen from Wu et al. suggested system, this concept suggest a good way to perform more complex tasks that use different domains. Its data storage system is very efficient and structured. It can easily be connected to an agent based DM.

5.4 Speech synthesis

After the DM has chosen how the dialog should go on, the speech synthesis module is responsible for turning the outputted text into audio. This is the exact opposite of the speech recognition module described in section 5.1. This section describes first general methodologies how speech can be synthesized. After that, the role of intonation will be described. This is a very important part of the output, because it can transmit information in addition to only the meaning of the words.

5.4.1 Methodology

Today's speech synthesis systems are either based on the concatenation of acoustical units such as syllables or on HMMs. Earlier systems were almost always based on the acoustic unit concatenation principle. However, using this technology makes it very hard to vary the voice of the outputted system. To vary the speaking style, the emotions and

other parameters of a voice, this system requires a huge database of predefined syllables. In contrast, the voice can be adjusted more easily using HMMs, which are described in section 5.1.1. Through the usage of HMMs for speech synthesis, it is possible to influence the voices pitch, spectrum, and pace. Therefore, the same HMM can be used. Only the parameters need to be adjusted. This can also be used to personalize the voice. Especially if avatars (see section 3.3) are used, this can be useful to separate the characters [13].

Tokuda et al. first suggested to use HMMs for speech recognition. They used the Festival framework and adjusted it to the English language (it was first designed for Japanese) and added their HMM functionalities [24]. Their engine is very small, so it can be used on small devices such as mobile phones as well (less than 1MB).

Their system is divided into two parts: the training part and the synthesis part. The training part extracts the spectrum and the excitation parameters from a database that contains training data audio. Using this extracted data, the HMMs are trained. The spectrum is stored using the mel-sepstral analysis technique, which enables the software to re-synthesize the data directly using the Mel Log Spectrum Approximation (MLSA) filter (both techniques are explained in [31]). Additionally, the duration of the samples is analyzed so that the system learns how to vary the pace of the output. Furthermore, contextual factors such as the stress relation or location factors are stored as well. Tokuda et al. use decision trees to store such information.

The second part of the system is the synthesis part. As an input, it gets an arbitrary text that should turned into audio. Therefore, it concatenates context dependent HMMs according to the text that should be synthesized. First, it turns the text into a label sequence. This label sequence is then turned into a speech waveform using the MLSA filter. It should be noticed that the system has the ability to vary speech parameters so that the output sounds realistic [110].

Today's systems can create audio files that are easily customizable by varying a few parameters. Using HMMs, a small engine can be used to generate different voices without having a large database of samples.

5.4.2 Intonation

Human conversation is more than just words and sentences. It carries additional information which is not written down. This is mainly based on different intonations. It is important, that the speech synthesis module creates a correct intonation. First, this is important because it helps to reduce the cognitive load for a user. Section 4.3.5 describes, how the intonation helps to reduce the cognitive load. Second, the intonation can also change the meaning of a whole expression. As can be seen in table 5.5, the sentence *I should go* will be understood differently by the user depending on the intonation.

Table 5.5: The meaning of a sentence can change depending on its intonation [17]

	Prosody	**Meaning**
I should gò.	Falling tone on stressed "go"	[Neutral]
I should gó?	Rising tone on "go"	Is that your advice?
Ì should go.	Falling tone on "I"	Not you!
I shòuld go.	Falling tone on "should"	And I defy to deny it!
I shôuld go.	Rising-falling tone on "should"	But I don't think I will.

As can be seen in table 5.5, the meaning of the expression can change drastically when-

ever the intonation changes. Especially when requesting more data from the user, which is usually done by asking questions, the intonation is the only way to mark the expression as a question. In written output on graphical user interfaces, symbols such as question marks or explanation marks can be used. Obviously, these need to be translated by the speech synthesis engine into the correct intonation. This is the first and easier step, because those symbols can be added by the DM. It should know what should be expressed with the current utterance. The second problem is more complicated. In case there are no symbols indicating how the intonation needs to be, additional meta information is necessary. In a few cases this information can be inferred from the sentence structure or the context. However, in most cases, the intonation needs to be influenced manually. By today, this is still one of the major problems of speech synthesis.

The speech synthesis module also needs to add pauses between sentences or when commas occur. This gives the user more time, to think and understand the sentence outputted by the system. Especially for long phrases the addition of pauses is essential. In contrast, some symbols do not need to be read out. One popular example would be a colon between the hours and minutes when reading out a time.

5.5 Summary

This section gave an overview over the technical background of a VUI. The main components are described. First, there is the speech recognition, which is today heavily based on HMMs. Additionally, various popular service providers for speech recognition are compared. This comparison is needed because the implementation part of this thesis requires a speech recognition module. The comparison includes the speech plugins of Android respectively iOS, Googles REST API and Microsoft's Bing Speech API. In addition, a new, upcoming service wit.ai is included in the comparison as well. The second important module of a VUI is NLU. This section described its purpose and the main concepts of NLU, which are today mostly probabilistic models such as N-Grams. The heart of a VUI is the third module, the dialog management. There are multiple strategies used by DMs. These strategies, namely finite state based DMs, frame based DMs, agent based DMs and plan based DMs are described as well. Often, these strategies are used as a mixture of more than one. They can be used together to compensate others weaknesses. Finally, this sections described speech synthesis modules and pointed out the importance of intonation in human speech.

6 Implementation

This thesis describes the theoretical background of speech recognition. In addition, it also describes the development of an application using speech recognition. This application is a project of htw Saar. It was originally developed by a group of Master students. The development stopped right at the time, when the work on this thesis started. In conclusion, the work of this thesis extends the existing functionalities.

The application can be used to experience interactive fiction on a mobile device. During first development phases, the application was called MTGA (this name can still be seen in the project). With development going on, the name changed to *TaleTime*. This name will be used in further writing to reference the application which is developed in this thesis.

Starting with the idea that stories are used for thousands of years, the project of htw Saar had the idea to go new ways with interactive fiction. Classical stories are told in books or movies. They are usually straight forward. New technologies offer possibilities to encourage the user to interfere into the storyline. This type of storytelling is called interactive fiction. There are already a few examples. Some of them are available as books, as computer games or mobile applications. Additionally, tools to create interactive stories are also already available. The goal of the project was, to explore new ways of how interactive stories can be consumed. It also included the development of a prototype [79].

This thesis has the goal to extend the existing prototype so it can be controlled by the user's voice. Details about what exactly is required, can be found in section 6.2. During the development of this thesis, a group of Bachelor students is working on the project as well. They work on the same basis, but aim to extend TaleTime with completely different and mostly independent functionalities. This group reviewed the existing prototype and decided to port it to the new, modern and recent Angular platform. Unfortunately, this requires to completely rewrite most of the code. For this reason, some work required to fulfill this thesis requirements consists in rewriting existing features. Mostly, this is the case, because some features are necessary to develop and validate the speech recognition. These features can therefore not be developed in a parallel process by the Bachelor group. Although some of these features could easily be mocked, the actual development was sometimes easier.

The following section describes the basis, which this thesis depends on. This is followed by a definition of requirements and details about the development process and the implementation itself.

6.1 Existing prototype

Due to great work of a group of students of htw Saar, there is already an existing prototype. The project includes the application itself and also a generator, which is used to produce an XML file that contains the stories. This generator uses the Hypertext Markup Language (HTML) as input and produces an XML file that is consumed by the application. Stories are written using Twine [112]. It is an open source tool to create nonlinear, interactive stories. Twine publishes directly to HTML. This HTML is then used as input for the generator. The stories are not written by the project team. For this reason, htw Saar cooperates with Saarland's university for art and design. Students of this university are

also part of this project. They participate by writing stories and therefore generating input for the application.

The prototype itself exists only as an Android application. It offers the following functionalities:

- **Story management** The prototype gives an overview about the available stories. This overview is shown as a list. Each story has a picture which is shown as well. A screenshot of how the actual application looks like can be seen in figure 6.1a. This list acts as entry point for the prototype and gives the user the choice to select a story which can then be played. The story management also displays meta data about the story itself. This meta data contains the author, available languages and a short description.

- **Voice selection** There are different voices available. Technically spoken, these voices are different audio files. The prototype offers a dialog to select one of these voices. As a result, the user can pick her / his preferred voice. Figure 6.1b shows, how this dialog looks like in the application.

- **Player** The player is the main part of the application. It reads out a chapter of the story. As can be seen in figure 6.1c, there are buttons to control the audio. Play, pause and stop are available. In figure 6.1c, it can also be seen that there are two different possibilities of how to go on in the story. This is the interactive part. The user can control the storyline by choosing one of the available options. Sometimes, the story is linear and only offers one choice to go on. If this happens, the user has to tap a button to go on. This situation can be seen in figure 6.1d.

- **User management** Another core idea is, to use the application with other people, for example the family. For this reason, the prototype already supports multiple users and a simple user management. Users can be created and deleted. For each user, the progress on each story is saved individually.

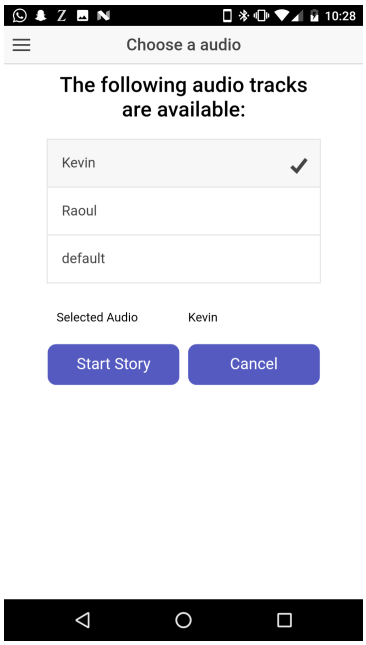
- **Settings** There are a few settings the user can use to customize the application. First, the application supports multiple languages. Currently, only English and German are supported. Second, the user can change the font size. This is useful, because children or disabled people may need a larger font. Lastly, the user can choose to go through stories using the interactive mode. If it is disabled, the application simply picks the first choice and goes on. In this case, the user experiences a linear story.

The audio files used to output the stories are available within the story. This means, that the packaged installation file contains all necessary audio files (in this case MP3). Because of this reason, the application package is quite large. In case of Android, which requires an Android Application Package (APK) file to install an application, the resulting package measures about 115MB.

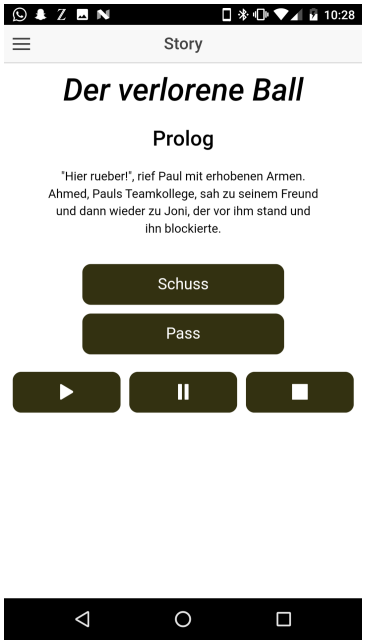
As mentioned earlier, there are multiple audio files for the same story. These files contain recorded audio of professional speakers, who are trained to read stories out. As a fallback, there is always one audio file generated with a TTS software. It is not generated by the application itself. Instead, the files are pre-generated and also part of the application package.

The application is also prepared to support stories which are available in multiple languages. Tags to indicate the languages available are already available. However, the application itself does not support any selection of a language for the story yet. In contrast, the UI language can be changed.

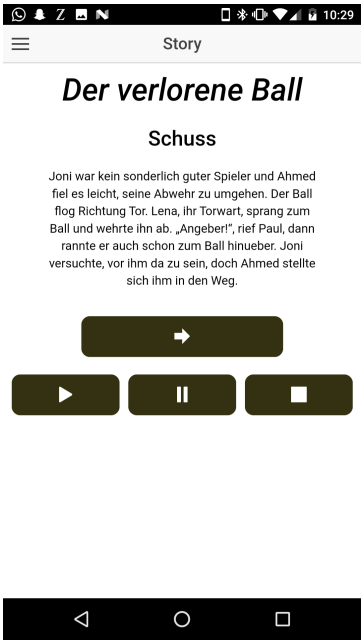
(a) The story overview



(b) Dialog to select voice



(c) The player with two options



(d) The player with only a single answer

Figure 6.1: Screenshots from the existing prototype

6.1.1 Used technologies

The existing prototype is based on the Ionic framework [54]. Ionic is a free and open source software development kit (SDK) for native and progressive web applications. The existing prototype was built using Ionic 1. It supports the development of mobile applications for many operating systems. The development itself is done using web technologies. In fact, an application is developed using HTML, Cascading Style Sheets (CSS) and JavaScript. Ionic 1 requires AngularJS (version 1) for development [57].

In order to run, Ionic requires Cordova. Apache Cordova is the underlying framework. It wraps the applications written with web technologies for each platform. Therefore, it offers a unified API, which is then matched on the devices platform API. Using this technology, the web application can access the devices capabilities such as sensors, data, network status and other functions provided by the operating system [19].

The prototype was built with the following library versions:

- **Ionic App Lib:** 2.1.2
- **Ionic CLI:** 2.1.4
- **NodeJS:** 6.9.1
- **AngularJS:** 1.5.3
- **Cordova:** 6.4.0
- **Cordova CLI:** 6.4.0

By 25th of January, 2017, Ionic 2 was released [70]. The new version is now based on the new version of Angular, which was released a few months earlier [65]. The Angular team also announced a new pattern of versioning. As a result, major releases will occur twice a year. For this reason, the framework is always referred as Angular in this thesis. If it is important to know the major version, the version is explicitly mentioned. Starting with Ionic 2 and Angular 2, it is recommended to use TypeScript for development (instead of JavaScript). Also, the required Angular framework was completely restructured. Existing code can be run in parallel, but it cannot easily be upgraded to the new version of Angular.

Due to this reason and the fact, that another group of students from htw Saar continues the work on the application, this thesis is not based on this prototype from a technical point of view. However, the design, the requirements definition and the ideas described in section 6.2 are based on this prototype. During the work on this thesis, the earlier mentioned group of bachelor students from htw Saar works on porting the prototype to the new version of Ionic and Angular. All development tasks, which are part of this thesis, are therefore based on the work of this group. Technical details about the current development will be given in section 6.4.

6.1.2 Open tasks

As mentioned in the previous section, the application is currently only a prototype. Also, it was mentioned, that another group of students currently works on this project. This section gives an overview about weaknesses of the prototype and functionalities that are not yet implemented. These ideas are mentioned in this thesis, because the student group of htw Saar works on these tasks in parallel to the work of this thesis. The resulting application will contain both work from this thesis and the project group.

The following ideas are not implemented in the existing prototype yet and will be covered by either this thesis or the project group:

- **Port to current Ionic version**

  As mentioned in the previous chapter, one of the major tasks of the project group is to port all existing functionalities to the current, modern version of Ionic and Angular. By the time of this writing, the goal is to base the development on Ionic 3.7.0 and Angular 4.1.0.

- **Redesign of GUI**

  A useful GUI is very important. The existing prototype did not focus on a user interface, which is why it's UI is only built upon the bare minimum. An important task to make the application more usable is therefore to redesign the whole application. A new style will be implemented as well as a new layout.

- **Story data**

  Currently, the audio files are part of the application package. As a result, they cannot be changed. This means, stories cannot be deleted and new stories cannot be added. This is one of the most important tasks which need to be worked on. The idea is to store the data using the file system of the device. However, this also requires a source, where stories can be downloaded. The idea includes creating a marketplace, where users can purchase new stories.

- **Link to Google Account**

  In the future, the application should be able, to connect a local account to a Google account. Using Google Drive, the user data can be stored online. As a result, the user's data can easily be synchronized on multiple devices. This is especially useful for the usage with multiple users.

- **Different node types**

  Interactive fiction enables the user to influence the storyline. It is possible, to extend the story by adding conditions to the storyline. As a result, the story becomes more dependent on the users choices. An example could be, that the user must have made the choice earlier in order to go on with a certain node. A simple example for this could be a story, where the user first can decide whether to pick up a key or not. In a later chapter, this key could possibly be required to open a door. If it is not, the user cannot pick this storyline.

- **Speech recognition**

  This is the feature developed in this thesis. At the moment, the user needs to pick an answer by tapping a button on the screen. This represents a break in media, because the story is read out. To improve the user experience, the user should be able to input the data in the same way it is presented: via audio. Therefore, a useful feature is to create an interface to control the application with the voice.

Except the speech recognition tasks, all other tasks will mainly be accomplished by the bachelors group. However, since most of the features are necessary to start with the development of speech recognition, some supporting work is also done during this thesis.

6.2 Requirements definition

This section describes the requirements that should be covered by the development work of this thesis. The requirements all regard the speech recognition. This technology is

mainly used to control the player, which is the core functionality of TaleTime. The new design of the player can be seen in figure 6.2.

Figure 6.2: Mockup of the new player

As can be seen in figure 6.2, there are buttons to toggle play and pause, picking an answer and going back to the previous chapter. These are also the main functionalities that need to be covered by the speech recognition. The following list gives an overview about which functionalities will be covered by this thesis:

- **Naming answer**

  The user should be able to pick an answer be simply reading it out or repeating one of the options. This is the simplest way of picking an answer. It is similar to a command pattern, because the user does not use her / his own words to describe which answer should be taken. All possible answers can be seen on the screen and they are additionally read out after each chapter was read out.

- **Describing answer**

  Instead of reading the answer out, the user should be able to describe the answer using his / her own words. The system must then match the correct answer. Obviously, there are limitations. But if similar words are used, the system must match the correct answer. Examples include “I want to him to kick” instead of “kick” and also possibly instead of “shoot”. Also, this matching depends on how similar the answers are to each other. If the nodes are more distinct from each other, than the matching can be done more accurate.

- **Picking index**

  Another way to pick an answer is to define it by its index. For example, the user could say: "I want to choose the first answer". This should also be recognized by the system and matched to the correct answer. This pattern also includes phrases like "the last" or "number three".

- **Controlling the flow if only one answer is available**

  Sometimes, the story is linear. Then, the user basically only has the choice to go on. The system should still confirm this by asking "Do you want to continue?" or a similar dialog.

- **Picking one for the user**

  In case the user does not care on how to go on, or expresses that she / he does not want to make a choice, the system should first provide some additional explanation. If the user still does not choose, then the system should go on with a random answer. It should be noticed, that this way is different than not saying anything. This case must also be handled, but in a different manner. This requirement only covers the user explicitly saying that she / he does not want to make a choice.

- **Repeat current chapter**

  The user should also be able to repeat the current chapter by saying "Repeat!", "Play this chapter again" or something similar. This can then be matched by the application and result in the correlating action.

- **Go back to previous chapter**

  Instead of tapping the button, the user should also be able to go back to the previous chapter by using her / his voice. For example, the user could say: "I want to listen to the previous chapter again" or "Go back". If this is the case, the system needs to load the previous chapter.

The VUI also must provide additional help, if the user was not understood or if the user did not say anything. This will be done in different steps. First, the possible answer will be read out again. Second, if the user still did not say something that matches, the VUI tries to assist by telling the user that she / he should pick an answer and then reading the answers out again. The flow of this way through the VUI can be seen in the earlier used figure 4.2.

In general, the VUI needs to try to help user whenever something was misunderstood. To prevent going into an endless loop of repeating the answers, which could happen if the user for instance simply walks away, the VUI will only try to help the user three times. After that, it will ask the user to use one of the buttons instead of the VUI.

In general, the wording of the VUI will be kept simple. This is because of two reasons. First, the target group of TaleTime are young children. Therefore, the language should be simple, so each child can understand the VUI easily. Second, users also have a screen in front of them. The screen distracts them. As a result, the cognitive load increases. To prevent confusing the users, a simple language is required.

6.3 Definition of grammar

The grammar of TaleTime VUI is defined in a JSON file. For each language, there is another JSON file. This leads to the advantage that more languages can be added easily. This

JSON file contains both the output produced by the VUI to interact with the user and also phrases that are used to match the user's input against. It is obvious that the answers that are possible for each chapter of a story are not contained in this file. Instead, the algorithm tries to match what was said to the set of answers define in the story itself.

A snippet of the English JSON file can be seen in listing 6.1. It only shows parts of the section used for the output of the VUI.

```
...

  "answers": {
    "multiple": [
      {
        "id": 0,
        "value": "How do you want the story to continue: "
      },
      {
        "id": 1,
        "value": "How would you like to continue: "
      },
      {
        "id": 2,
        "value": "You have the choice: "
      }
    ],
    "single": [
      {
        "id": 0,
        "value": "Do you want to continue?"
      },
      {
        "id": 1,
        "value": "Continue?"
      }
    ]

...
```

Listing 6.1: Part of output section of English grammar JSON

As can be seen in listing 6.1, a definition of multiple versions is possible. As a result, the VUI system has different expressions to choose from. For each statement that can be expressed, there is an entry in the correlating JSON file. Listing 6.1 shows expressions used for output. This means, whenever the VUI needs to say something, it reads the expressions out of the JSON file and picks the best matching expression or simply a random one. Having multiple options helps to let the VUI sound less mechanic. For instance, if the VUI of TaleTime comes to the step where it needs to read the answers out, it reads the JSON file. Depending on how many answers are available, there are different expressions. These are grouped into different arrays within the JSON file. Hence, the VUI uses different keys depending on how many answers are available to read out of the JSON file.

In case the VUI realizes that there are two possible answers, it uses the key *multiple* and gets three different expressions that can be read out. It is then up to the VUI to pick one

of these. The JSON file is designed to only contain interchangeable expressions per array.

Similar, there are expressions that the VUI can use to match the users input against. These expressions obviously do not contain any data of the answers from the story. This data is only stored within the story. However, there are several use cases where a general matching can be done. One of these examples can be seen in listing 6.2.

```
...

"enum": [
    {
      "id": 0,
      "value": "first",
      "index": 1
    },
    {
      "id": 1,
      "value": "second",
      "index": 2
    },
    {
      "id": 2,
      "value": "third",
      "index": 3
    }

...
```

Listing 6.2: Part of the input section of English grammar JSON

The data in listing 6.2 is used to find out, if the user was referring to an index. For example, if the user wants the pick the first number, an expression like “Let’s pick the first one!” is presumably. The VUI can then compare this statement against words usually used to express positions and pick the correct answer for the user. Data like this is also stored in the JSON file.

It not only contains data about enumerations. Use cases like expressing that the user does not care will also be compared against data in the JSON file. Other use cases include agreement, negations and control sequences for the general behavior of the application.

The full JSON file can be found in appendix B.

As mentioned earlier, this procedure ensures simple multilanguage support. The only changes necessary are the usage of other JSON files. TaleTime currently supports English and German. The JSON files are loaded whenever the user changes the language in the settings. This ensures that not only the UI language changes, but also the language used and understood by the VUI.

6.4 Development

This section describes technical details concerning the development of the VUI which is part of TaleTime. To illustrate this, this section will first describe the technological basics of the prototype. Second, this section gives an overview of the architecture designed. This will point out all components used in the VUI and helps to understand, how they

work together. Followed by this, each component is described in more detail. The term component expresses any part of the software, which can be individually described. Lastly, open problems that could not be handled during this thesis will be pointed out.

6.4.1 Technological basics

The project is based on Ionic 2, which uses Angular (version 2 or newer). By the time of this writing, the following versions are used:

- **Ionic core:** 3.7.0
- **Angular:** 4.1.0
- **Cordova:** 6.2.3
- **NodeJS:** 7.7.3

This list does not contain all packages that are used during this project. For example, Angular itself consists out of many modules. This makes it easier to only include the necessary parts of Angular into the application. However, a complete overview of all packages used, including their versions, can be found in appendix A.

Ionic scaffolds the structure for an application. Generally spoken, the main components are pages. Each page represents a screen. The content of the page is described using HTML, the appearance can be changed using Syntactically Awesome Style Sheets (SASS) [99]. Internally, Ionic uses an Angular component to represent a page. Angular components consist out of a template (HTML), stylings (CSS) and application code. It is recommended to write the application code in TypeScript. However, JavaScript and Dart are also supported by Angular. This project uses TypeScript. The application code should only contain code that is necessary for this exact component.

Since Angular is a framework whose original purpose is to develop single page applications (SPA), components can also be used on a web page. However, this thesis concentrates on mobile development with Ionic. Ionic uses Angular and abstracts it for its own purposes. Hence, an Angular component and an Ionic page are used interchangeable in this thesis.

There is a strong connection between the application logic of a page and the HTML template. It is possible to use values of variables in the application code and output them directly into the template. Angular takes care of keeping them updated. Also, there are structural directives that change the document object model (DOM), depending on different states of the application. Additionally, there are more options to change the appearance of the application depending on the state. Angular offers more directives for this. For example, the color of a button can be changed depending on the content of a variable. Angular also takes care of keeping this updated. Because this thesis is not about the basics of Angular, only the core concepts will be explained whenever necessary. A good overview of Angular and its functionalities can be found in the official documentation [4].

Ionic also brings additional functionalities needed on a mobile device. It is easily possible to stack pages and go to other pages, for example when buttons are tapped. This helps constructing the UI. The stack can also be used to navigate back to last visited page. Pages do not contain application logic.

More complex actions or actions that are used in multiple pages can be extracted into providers. The Angular documentation describes providers as services, but Ionic defines them as providers. Therefore, they are called providers in this writing. A provider can

be injected using Angular's dependency injection mechanism [3]. As a result, application logic can easily be shared across different components.

Ionic also offers to add more complex functionalities to an application. To keep the application code small, these are optional. Ionic calls these additional functionalities Ionic Native. This thesis makes use of two important Ionic Native modules: Speech Recognition [55] and Text To Speech [56]. These modules are used for input and output of the VUI.

6.4.2 Architecture

The architecture of the VUI, which is developed for TaleTime, consists out of four important providers:

- **LanguageFileProvider** This provider encapsulates access to the JSON files, that contain the grammar. Whenever the VUI requests texts, this provider is used.
- **TTSTextProvider** Using the LanguageFileProvider, this provider generates texts that can be used by the VUI to output messages.
- **TTSProvider** Is used from the Ionic Native plugin and directly called to synthesize text, whenever written text needs to be read out by the VUI.
- **SpeechRecognitionProvider** This provider performs the speech recognition. It is imported from the Ionic Native Speech Recognition plugin. When called, this provider returns a list of phrases that were understood.
- **AnswerMatchingProvider** This provider matches the results of the speech recognition to the answers possible to go on in the story. It contains the logic to match phrases to the answers that are possible or, if no answers matches, to return an error.

These providers are described in more detail in section 6.4.3. Figure 6.3 illustrates, how the different providers work together. The VUI is controlled by the player page. Its application logic calls the providers whenever needed. Figure 6.3 demonstrates how a normal procedure works. After the player finished reading out a chapter, it requests a text wrapper to read the answers out. Depending on the situation, different wrappers may be called. In this case, the player separates between situations, where only one answer is available and situations, where more than one answer is possible. After the wrapper text was received, the player puts the wrapper and the answers together and creates a text that can then be outputted by the TTSProvider. After the TTSProvider outputted the answers, the speech recognition is started. It waits for the user to say something and returns what was understood. This result is then matched by the AnswerMatchingProvider. As a result, this provider returns either the answer which was matched or an error. If an answer was matched, the player loads the next chapter and starts reading it out, followed by the same process described above. If the AnswerMatchingProvider did not match an answer, the player tries to provide help. This procedure is described in more detail in the next section (6.4.3).

During development, one major architectural problem occurred. Usually, the application uses TTS to output the answers that are possible. However, sometimes the audio files do not only read the chapter out, but also the answers. Unfortunately, there is currently no possibility to find that out. As a workaround, each story needs to provide this information. In a future version, it would make sense to also have recordings of the answers. This would also give designers the possibility to provide further explanations instead of only the answer text itself.

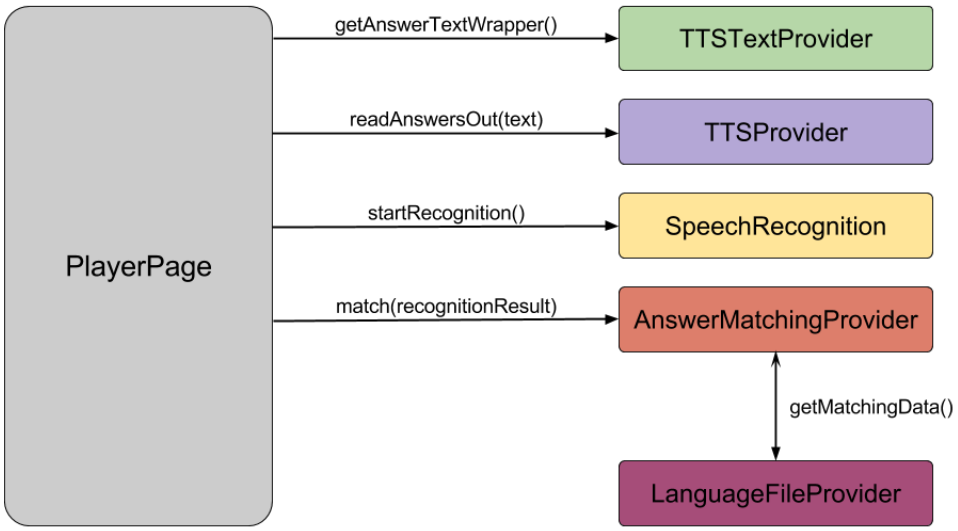


Figure 6.3: Overview of the VUI part of TaleTime

6.4.3 Components

Section 6.4.2 gave an overview about the architecture and how all components work together. This section describes each component in more technical detail.

Player The player is in full control of the VUI. It has access to the StoryProvider, which contains all the data of the story itself (except the audio), an AudioProvider to play the provided audio files and all providers mentioned above.

To describe, how the player works, the reader must have an overview about the most important methods contained in the player. The following list gives an overview about them and shortly describes their function. However, this list only describes the signature of each method. In case further details matter, they are given later. Also, this list represents a subset of the methods available, because only methods are mentioned that are important for the VUI.

- **play()**

  This method starts playing the audio track of the current chapter. If there is no audio track for this story, TTS is used as a fallback. After TTS or the AudioProvider is done reading out the chapter's text, *readAnswersOut(counter)* is called. The counter of this function represents, how often the speech recognition was started. Because the player just finished reading the chapter out, this counter is set to zero.

  The player requests the path to an audio file from the StoryProvider. If reading the audio file does not work, for example because the file is not available anymore, *switchToTTS()* is called.

- **pause()**

  No matter if TTS or an audio track is used for output, this method stops the output. Currently, there is no difference between pause and stop. Calling this function always stops the output.

- **switchToTTS()**

  In case the player needs to switch to TTS, this method sets the necessary conditions and starts the output again with TTS. This method is mainly used when skipping to TTS as a fallback.

- **readAnswersOut(counter: number)**

  First, this method requests the answer text from the TTSTextProvider. This provider also puts the answers in the wrapper, which it reads out of the grammar file. Then, it triggers the TTSProvider to read this text out. The counter, which is given as parameter, helps keeping track of how often the speech recognition was started. In the beginning, this counter is zero. If the method was called and there was no parameter provided, it is also assumed as zero. After the TTSProvider finished its output, *startSpeechRecognition(counter)* is called.

- **startSpeechRecognition(counter: number)**

  To start the speech recognition, this methods first sets up the environment for the speech recognition. This includes setting the language. Then, it triggers the recognition. The speech recognition either returns an array of matches or an error. If there were matches, the AnswerMatchingService is used to try and find a matching answer. In case, no answer matches or there was an error, *repeatSpeechRecognition(counter)* is called. Again, the counter is used to keep track of how often the speech recognition was called.

- **repeatSpeechRecogniton(counter: number)**

  In case the speech recognition did result in an error or the result could not be matched to one of the answers, this method is called. The behavior depends on the counter, which represents how often the speech recognition was used. Currently, the behavior is as follows, depending of the state of the counter:

  - **0:** Speech recognition is simply restarted.
  - **1:** Answers will be read out again and speech recognition is restarted.
  - **2:** An output is generated to tell the user that she / he should use the GUI instead of the VUI. The VUI then stops to try to use speech recognition until the user selected an answer. In addition, the GUI highlights the answer buttons to indicate that the user should use them to pick an answer.

  Those values can be adjusted. It is important, that the VUI keeps track of how often the speech recognition tried to understand the user. This prevents the VUI from ending in an endless loop asking the user over and over again. Additionally, the output can be adjusted this way.

The description of the methods above contains also the control flow, which the VUI goes through. Figure 6.4 illustrates this control sequence and the calls of each method in a clearer way. The VUI starts its work, when the player is done reading a chapter out.

The method *readAnswersOut(int)* is passed as a callback function to either the TTSProvider's speak function or the AudioProvider's play function. *readAnswersOut(int)* creates a text and reads it out. Then, it calls *startSpeechRecognition(int)*. The result of this method can be successful or not. If it is not, an error is returned. In this case, *repeatSpeechRecognition(int)* is called. If the result of *startSpeechRecognition(int)* was successful, the player tries to match the result to one of the possible answers. This is done by calling the AnswerMatchingProvider's *match(Answers[])* method. If it's result is successful, the player loads

the next chapter by calling the *loadNodeFromAnswer(int)* method. If there was no match which means that the AnswerMatchingProvider either returned *null* or an error, then *repeatSpeechRecognition(int)* is also called.

Depending on the parameter passed into *repeatSpeechRecognition(int)*, it's behavior is different. After each decision of what to do next, it increments the counter. If the counter is zero, it starts the speech recognition again, so that the user has the option to repeat what was said earlier. In case the counter is one, which means the speech recognition tried to recognize what the user was saying twice, the answers are read out again. This is done, because the user apparently did not understand or forgot the answers. Reading the answers out again acts as a help message for the user. It is most likely that the user simply misunderstood the options at this point. If *repeatSpeechRecognition(int)* is called again and the counter is two, it creates a help message which tells the user to skip to the GUI. This is done, because after the user tried to say her / his decision for three times, it is not very likely that the VUI will understand the user by the fourth time. Also, it is possible that the user was not saying anything the whole time. In this case, not stopping at a particular counter would let the VUI end up in an endless loop and keep telling the user to repeat her / his decision.

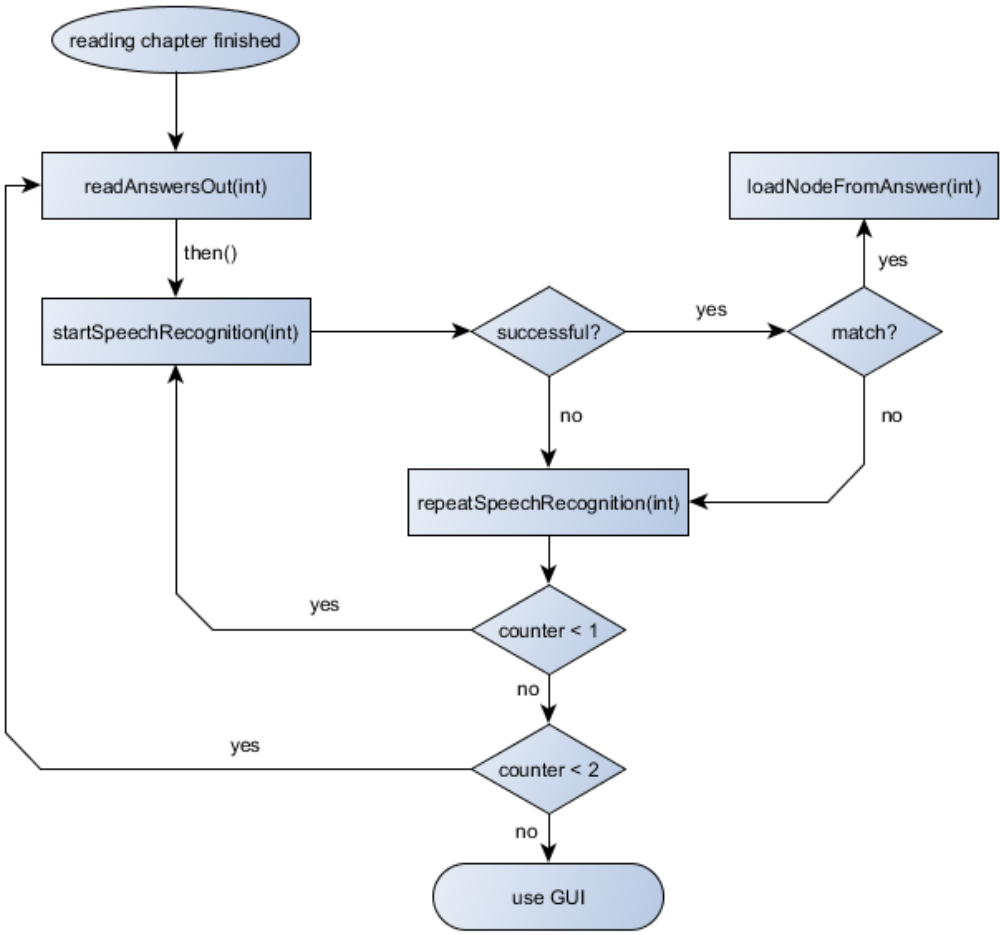


Figure 6.4: Control flow of the VUI and the called methods

The player controls the core functionalities of the VUI. The reason why the player is in charge of this, is because it also controls the GUI. As a result, it can provide an integrated UX using both the GUI and the VUI.

TTSTextProvider

The main purpose of this provider is to generate texts that are used by the VUI as output. Listing 6.3 demonstrates how this provider works. It shows the method used to generate the text when the VUI is reading the answers out.

```
1  public createAnswersText(answers: MtgaNextStoryNode[]): string{
2      let text: string;
3
4      if (answers.length == 1){
5        // only one possible answer
6        console.log("There's only one possible answer");
7        let i = this.generateRandomNumber(this.languageFileProvider.
             preDefinedTexts.answers.single.length);
8        text = this.languageFileProvider.preDefinedTexts.answers.
             single[i].value;
9      } else {
10       // multiple possible answers
11       console.log("There are multiple possible answers");
12       let i = this.generateRandomNumber(this.languageFileProvider.
             preDefinedTexts.answers.single.length);
13       text = this.languageFileProvider.preDefinedTexts.answers.
             multiple[i].value;
14       for (let i=0;i<answers.length;i++){
15         if (i != answers.length-1){
16           text = text + " " + answers[i].value;
17           if (i < answers.length-2){
18             text = text + ",";
19           }
20         } else {
21           let j = this.generateRandomNumber(this.
                 languageFileProvider.preDefinedTexts.linking.or.length
                 );
22           text = text + " " + this.languageFileProvider.
                 preDefinedTexts.linking.or[j].value + " " + answers[i
                 ].value + "?";
23         }
24       }
25     }
```

Listing 6.3: Method to create the answer texts, used as output by the VUI

As can be seen in listing 6.3, the method call provides a parameter. This parameter contains the possible answers that can be matched. If there is only one answer, the TTSTextProvider requests a randomly chosen text from the LanguageFileProvider. It already knows that it has to look into *answers.single*. If there are multiple answers, the TTSTextProvider reads a random text from the LanguageFileProvider. This time, *answers.multiple* is used. Then, the string is expanded with the possible answers. It is noticeable, that there are different words for linking available. Here, a link meaning *or* is used. These linking words are also part of the grammar.

LanguageFileProvider

The LanguageFileProvider wraps the access to the JSON files which contain the grammar. When this provider is first instantiated, it loads the default language file (currently English). It basically only offers two methods.

The method *loadLanguageFile(language)* reads the JSON file containing the grammar for the specified language. Then, it holds the read data for further access. The second method is a getter to access the data. Other components accessing the data therefore must have knowledge about the structure of the grammar files.

TTSProvider

The TTSProvider, which is part of the Ionic Native module *Text To Speech* offers two important methods: *speak(string | TTSOptions)* and *stop()*.

The *speak(string | TTSOptions)* method starts the speech output. If only a string is passed as parameter, it tries to read it out using English. Using an options object, the output can be customized. If an object is passed, the following options are available:

- **text:** contains the text, that should be read out.
- **language:** the language, that should be used for the output. This is especially important for the intonation of the output.
- **rate:** defines a rate between zero and one. The rate describes the pace that the TTS system uses to speak.

As can be guessed from the name, the *stop()* method stops the current output.

Both methods return promises. A promise is a return value for asynchronous execution. Whenever a promise is returned, the application can register tasks that are performed when the execution was successful. This is done by calling the *then()* function of a promise and passing a function as a parameter. If the asynchronous execution of the promise was successful, the passed function will be called. In case an error occurs, the function passed to the *catch()* function of the promise is called.

SpeechRecognitionProvider

Similar to the TTSProvider, the SpeechRecognitionProvider is important from an Ionic Native module. It offers a set of methods. Only a few of them are used by TaleTime's VUI. The following list gives an overview about them:

- **startListening(options)**

  This method starts the speech recognition process. On Android, it stops itself when the service noticed, that the user has stopped speaking. If iOS is used, there is another method to stop the recognition manually. Since this thesis only concentrates on the development of TaleTime for Android, this method is not considered.

  Similar to the TTSProviders *speak(int)* method, this method also returns a promise. If the recognition was successful, the result returned is an array of strings. The strings are ordered by their confidence. However, the exact confidence value is not given.

  Via the options object that can passed as parameter, a few adjustments for the recognition can be made. These include the language through a language code, the number of matches, if the module should use a graphical popup when listening, the caption for this popup and if partial results are allowed.

- **requestPermission()**

  In order to use the speech recognition service, the provider needs to access the devices microphone. Most operating systems require the user to approve an application accessing the microphone. This methods brings up a dialog and asks for the user's permission to use the microphone.

- **hasPermission()**

  This method checks if the user already has given the permission to use the microphone. TaleTime uses this method to make sure, the user does not have to give the permission more than once.

Internally, this plugin uses Android's speech recognition API [2].

AnswerMatchingProvider

The AnswerMatchingProvider is the core of the VUI. It matches the results of the speech recognition to one of the answers that are possible for the user to answer for this chapter.

To do so, this provider offers one important method, which will be explained in the following text. This function is called *match(string[], MtgaNextStoryNode[])*. It has two parameters. The first is the array of strings that contains the result of the speech recognition. The second parameter is the current node of the story. It holds the information about the current chapter and the possible answers. The method either returns another node, which can then be loaded by the player, or null. If null is returned, no match was found.

```
1  public match(result: string[], answers: MtgaNextStoryNode[]):
       MtgaNextStoryNode | string | null {
2
3      if (answers.length != 1){
4        // find exact math considering the hierarchy
5        for (let i = 0; i < result.length; i++) {
6          for (let answer of answers) {
7              if (AnswerMatchingProvider.checkContent(answer.value,
                   result[i])) {
8              console.log(answer.value + " matched to answer");
9              return answer;
10           }
11         }
12       }
13     } else {
14       // only one answer possible, system asks if user wants to continue
15       // therefore check if user agrees
16       for (let i = 0; i < result.length; i++) {
17         for(let a of this.languageFileProvider.preDefinedTexts.
               agree){
18           if (AnswerMatchingProvider.checkContent(a.value, result
                   [i])) {
19             console.log(a.value + " matched to answer");
20             return answers[0]; // it is only one possible here
21           }
22         }
23       }
24     }
25
```

```
26      // nothing found so far, check for numbers
27      for (let i = 0; i < result.length; i++) {
28        for(let e of this.languageFileProvider.preDefinedTexts.enum){
29            if (AnswerMatchingProvider.checkContent(e.value, result[i
                  ])) {
30            return answers[e.index-1]
31          }
32        }
33      }
34
35        // navigate backwards
36      for (let i = 0; i < result.length; i++) {
37        for (let e of this.languageFileProvider.preDefinedTexts.
              backwards) {
38            if (AnswerMatchingProvider.checkContent(e.value, result[i
                  ])) {
39            return ANSWER_CHAPTER_BACKWARDS;
40          }
41        }
42      }
43
44        // repeat current chapter
45      for (let i = 0; i < result.length; i++) {
46        for (let e of this.languageFileProvider.preDefinedTexts.
              repeatChapter) {
47            if (AnswerMatchingProvider.checkContent(e.value, result[i
                  ])) {
48            return ANSWER_CHAPTER_REPEAT;
49          }
50        }
51      }
52
53        // does not matter
54      for (let i = 0; i < result.length; i++) {
55        for (let e of this.languageFileProvider.preDefinedTexts.
              doNotCare) {
56            if (AnswerMatchingProvider.checkContent(e.value, result[i
                  ])) {
57              return answers[AnswerMatchingProvider.
                    createRandomNumber(answers.length)];
58          }
59        }
60      }
61
62      return null;
63    }
```

Listing 6.4: The method used for matching

Listing 6.4 shows the algorithm which is used by the provider to match the answers. This algorithm tries to match the exact wording first. That means, that each possible result of the speech recognition is compared against the list of possible answers. Therefore, it considers the order of the results from the speech recognition. If the provider finds a match, it returns the node. If not, it goes on.

The next attempt to match answers is to compare, if one of the results is contained in one of the answers. This means, the result could be a substring of one of the answers.

Similar to the first attempt, the provider keeps track of the order and tries to match each result to one of the answers. Again, if an answer is found, it returns the corresponding node. If not, it goes on with the next matching strategy.

This strategy tries to read other information. Currently, the providers tries to find out if the user is talking about a particular index. If the user is talking about a number, then it checks if the numbers meaning could be the index of the answer. If this is the case, the provider returns it. The next attempt tries to match the answer to a navigation command. It is possible, that the user wants to repeat the chapter or navigate backwards. If the answer still is not matched, there is one more thing that can possibly be matched by the VUI. It tries to find out, if the user does not care on how to go on. If this is the case, the provider randomly picks one of the answers. If not, it attempts that it is not possible to match the result to an answer. It then returns null.

With this strategy, most of the answers can be matched. In fact, during a test with the whole story, 15 out of 17 tries were matched successfully. There were 15 nodes, the user tried to answer one question two times. Therefore, there were 17 tries. Overall, this is a result of 88% during a normal test run.

6.4.4 Open problems

There are currently some problems that are still open and cannot be solved in this thesis. The following list gives an overview about those problems. Some of them occurred during development, some during the design phase.

- **MP3 files contain answers**

  As already mentioned earlier, some of the MP3 files, which were professionally recorded, already contain the answers. This means, that the reader not only read out the chapter itself, but also the possible answers. The VUI can currently not know, if the MP3 file contains the answers or not. As a result, it is possible that the answers are read out twice. First, the reader reads them out when the application plays the audio file. Then, the VUI reads the answers out for a second time. Fortunately, it is clear for each reader, if the answers are part of the audio file or not. In other words, differences do not exist between different files recorded by the same reader.

  A workaround used for this thesis is to store the data in the story file. There is now a marker in the story file for each reader, indicating if the answers are part of the audio file or not.

  Future versions should aim to solve this problem by dividing the audio files. For each chapter, there should be at least two audio files. The first contains the audio of the chapter itself. The second contains the audio of the answers. Another option would be to also provide a single audio file per answer. Most important is, that this procedure is done consistently.

- **High data usage of TTSProvider**

  Unfortunately, the TTSProvider currently consumes much data. Especially, if whole texts are read out, this is the case. The previous group working on this project solved this problem with prerecorded audio files created by the TTS software. This cannot be done for the answers that are read out by the VUI, because these texts are dynamically generated. However, if TTS is only used to read the answers out, the usage is not too high. Also, this can be solved by providing audio files for the snippets that are contained in the grammar JSON files.

Most of the problems above also contain suggestions to solve them in their description. However, the solution requires more work than affordable in this thesis. As a result, solving the problems can be part of future projects working on TaleTime.

6.5 Summary

As shown in the previous sections of this chapter, the development of the VUI for TaleTime started with the prototype that was already existing. This prototype was responsible for the tasks that were planned.

However, since the project group, which was also working on this project, decided to move the project to the current version of the Ionic platform, all of the existing code had to be rewritten. This lead to an organizational problem, because it would not have made sense to develop the VUI based on the older platform and also migrate it to the new version. Instead, this thesis includes support for the migration to the current version of Ionic or Angular, respectively. All of the tasks regarding the VUI were then based on the migrated prototype. This is also the code basis, the project group was working with for any further development.

The development of the VUI is completely done in TypeScript. For input and output, plugins provided by the Ionic framework were used. The design of the VUI is based on the theory described in previous chapters. It is designed to be as easy to use as possible, which particular care of the wording. Because one of the main target groups of TaleTime is young children, it is important that the wording is easy enough so they can understand it.

As a fallback, the VUI uses the GUI. Whenever it gets stuck, it stops keep asking the user for more voice interaction. Instead, it tells the user to use the GUI.

There are still a few open problems described in this section. These problems can be solved in future projects. They are described in section 6.4.4. Additionally, some of these descriptions already contain suggestions on how to solve these problems. However, most of these solutions require more work to be done than possible in this thesis.

7 Conclusion

As this thesis has shown, the development of VUIs is a complex process. However, it is important for today's IT systems and will gain more importance in the future, as system will evolve and become more mature. People already started using personal assistants. Once their more used to the interaction via voice, they will automatically increase the usage. In addition, more and more devices will be connected to each other. As a result, it will be possible to control many more devices via voice. This is also possible, if the devices themselves do not have their own microphones and / or speakers. Through personal assistants, many more functionalities will be able to use voice as an input.

As a result, the user experience will change. Users are able to use applications more intuitively with VUIs. They simplify the usage because users do not have to learn how to interact with complex menus. Furthermore, user can use their own words to describe the function they want to use. In addition, the hands-free usage of VUIs give them a big advantage in certain situations such as driving a car. Overall, VUIs increase the user experience by far.

Especially in combination with other UIs, VUIs can be used in almost every use case. Such multimodal applications can make sure that the VUI is only available, when it is actually useful. Multimodal applications help to use the strengths of each UI and avoid their weaknesses in the same time.

Furthermore, multimodal design helps to lower cognitive load. Particularly when using VUIs, the cognitive load is naturally higher. Because VUIs read information out, the information is volatile. Therefore, it is required for the user to keep everything in mind. VUIs also only support a sequential output. As a result, users have no possibility to see what an earlier output was. This enormous weakness can be covered by using a screen. The multimodality helps the user to remember items used beforehand. In addition, VUIs cannot be used to display some kind of data. Maps and images for example cannot be displayed using a VUI very well. Instead, screens are more useful. However, in some cases VUIs are the best choice. The best UI always depends on the data.

All in all, VUIs must be built with a high quality so users accept them. If the recognition or the matching is implemented poorly, users get annoyed very quickly. To prevent this, designers need to follow a few rules in their dialog design.

These rules include the structure of dialogs. They should always be clear and straightforward. Options need to be pointed out clearly. Disambiguation should kept low. Analogous, question should be clear as well. The usage of rhetoric questions should generally be avoided. Questions also should be placed at the end of a sentence because users tend to answer them directly. When placed in the middle of a sentence, the user would interrupt the output. In general, users remember best what is placed in the end of the output.

Developers can also make use of checksums whenever possible to validate the accuracy and the VUIs quality. This is all part of the high level dialog design.

In contrast, the process of detailed dialog design takes care of the exact wording. Designers make sure, that everything sounds good and that the brands image is kept. Additionally, it should be made sure that the user can get enough help whenever needed and that the error handling catches all possible errors.

From a technical perspective, VUIs consist of the speech recognition, NLU, the DM and the speech synthesis. Speech recognition and speech synthesis heavily rely on HMMs

today. The DM is the heart of a VUI, because it keeps track of everything that was said and connects this data to create the dialog from the VUIs perspective.

This thesis includes the implementation of a VUI in an existing application. TaleTime, the application that is developed in this thesis, is used to explore interactive fiction. Because the output of this application is audio, it makes sense to use audio for the input as well. As a result, there is no break between the modalities. The dialogs in this VUI are created following the thoughts expressed in this thesis. It allows the user to navigate through an application completely based on voice. The implementation is based on Ionic and uses Androids internal speech recognition API. It then matches the answers and returns text to the output system.

This thesis gave an overview over the development procedures used when developing VUIs. They are explained using a practical example and can be used in small and larger projects. Following the guidelines in this thesis helps to create a VUI with the highest quality possible. For more interested readers, there are many citations that lead to more detailed sources.

Bibliography

[1] Amazon. *Alexa Skills Kit Voice Design Best Practices - Amazon Apps & Services Developer Portal*. Apr. 13, 2017. URL: https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/alexa-skills-kit-voice-design-best-practices (visited on 04/13/2017).

[2] Android. *SpeechRecognizer | Android Developers*. Aug. 12, 2017. URL: https://developer.android.com/reference/android/speech/SpeechRecognizer.html (visited on 08/12/2017).

[3] Angular. *Angular - Dependency Injection*. Aug. 11, 2017. URL: https://angular.io/guide/dependency-injection (visited on 08/12/2017).

[4] Angular. *Angular - What is Angular?* Aug. 11, 2017. URL: https://angular.io/docs (visited on 08/12/2017).

[5] Apple. *iOS - CarPlay - Apple*. June 6, 2017. URL: https://www.apple.com/ios/carplay/ (visited on 06/06/2017).

[6] Apple. *iOS - Siri*. Apple. June 18, 2017. URL: http://www.apple.com/ios/siri/ (visited on 06/19/2017).

[7] Peter Bakondy. *cordova-plugin-speechrecognition: :microphone: Cordova Plugin for Speech Recognition*. original-date: 2016-09-29T16:23:24Z. Aug. 25, 2017. URL: https://github.com/pbakondy/cordova-plugin-speechrecognition.

[8] Bruce Balentine. „Re-Engineering the Speech Menu." In: *Human Factors and Voice Interactive Systems*. The Springer International Series in Engineering and Computer Science. DOI: 10.1007/978-1-4757-2980-1_10. Springer, Boston, MA, 1999, pp. 205–235. ISBN: 978-1-4757-2982-5 978-1-4757-2980-1. URL: https://link.springer.com/chapter/10.1007/978-1-4757-2980-1_10 (visited on 07/14/2017).

[9] John D. Bransford and Marcia K. Johnson. „Contextual prerequisites for understanding: Some investigations of comprehension and recall." In: *Journal of Verbal Learning and Verbal Behavior* 11.6 (Dec. 1972), pp. 717–726. ISSN: 00225371. DOI: 10.1016/S0022-5371(72)80006-9. URL: http://linkinghub.elsevier.com/retrieve/pii/S0022537172800069 (visited on 07/14/2017).

[10] Bart J. Brickman, Lawrence J. Hettinger, and Michael W. Haas. „Multisensory Interface Design for Complex Task Domains: Replacing Information Overload With Meaning in Tactical Crew Stations." In: *The International Journal of Aviation Psychology* 10.3 (July 1, 2000), pp. 273–290. ISSN: 1050-8414. DOI: 10.1207/S15327108IJAP1003_04. URL: http://dx.doi.org/10.1207/S15327108IJAP1003_04.

[11] DE Broadbent. „The magic number seven after fifteen years." In: *Studies in long term memory*. Ed. by A Kennedy and A Wilkes. London: Wiley, 1975, pp. 3–18.

[12] CallMiner. *Phonetics vs. LVCSR: Under the Hood of Speech Analytics*. CallMiner. Apr. 6, 2013. URL: https://callminer.com/blog/phonetics-vs-lvcsr-hood-speech-analytics/ (visited on 09/03/2017).

[13] Gérard Chollet, Asmaa Amehraye, Joseph Razik, Leila Zouari, Houssemeddine Khemiri, and Chafic Mokbel. „Spoken Dialogue in Virtual Worlds.“ In: *Development of Multimodal Interfaces: Active Listening and Synchrony*. Ed. by Anna Esposito, Nick Campbell, Carl Vogel, Amir Hussain, and Anton Nijholt. Lecture Notes in Computer Science 5967. Springer Berlin Heidelberg, 2010, pp. 423–443. ISBN: 978-3-642-12396-2 978-3-642-12397-9. URL: http://link.springer.com/chapter/10.1007/978-3-642-12397-9_36 (visited on 04/21/2017).

[14] Marie-Thérèse Claes and Marinel Gerritsen. *Culturele waarden en communicatie in internationaal perspectief*. 3., herz. dr. OCLC: 930883864. Bussum: Coutinho, 2011. 294 pp. ISBN: 978-90-469-0304-9.

[15] Herbert Clark. „Language Use and Language Users.“ In: Gardner Lindzey and Elliot Aronson. *Handbook of social psychology*. 3rd. New York : Random House, c1985., 1985, pp. 179–231. ISBN: 978-0-394-35049-3.

[16] Josh Clark. *Designing for Touch*. A Book Apart, Sept. 2016. URL: http://proquestcombo.safaribooksonline.com/9781492017851 (visited on 04/13/2017).

[17] Michael H. Cohen, James P. Giangola, and Jennifer Balogh. *Voice User Interface Design*. Addison-Wesley Professional, Feb. 2004. ISBN: 978-0-321-18576-1. URL: http://proquestcombo.safaribooksonline.com/0321185765 (visited on 04/04/2017).

[18] Nuance Communications. *Nuance Developers*. Aug. 7, 2017. URL: https://developer.nuance.com/public/index.php?task=mix (visited on 07/08/2017).

[19] Apache Cordova. *Architectural overview of Cordova platform - Apache Cordova*. Aug. 10, 2017. URL: http://cordova.apache.org/docs/en/latest/guide/overview/index.html (visited on 08/10/2017).

[20] David Crystal. *The Cambridge Encyclopedia of the English Language*. Google-Books-ID: Kh_RZhvHk0YC. Cambridge University Press, Aug. 4, 2003. 226 pp. ISBN: 978-0-521-53033-0.

[21] Arianna D'Ulizia. „Exploring multimodal input fusion strategies.“ In: *The Handbook of Research on Multimodal Human Computer Interaction and Pervasive Services: Evolutionary Techniques for Improving Accessibility* (2009), pp. 34–57. URL: http://books.google.com/books?hl=en&lr=&id=08CqMtIKSWwC&oi=fnd&pg=PA34&dq=%22with+embedded+software+and%22+%22the+universal+accessibility+concept,+are:%22+%22which+refers+to+the+simultaneous+or%22+%22of+usability,+accessibility,+flexibility+and%22+%22well+as+users+with+disabilities.+Finally,+it%22+&ots=NZfqTSFGYb&sig=HC82ik_FuQ8eu7Wg1KmhxLLf3Ts (visited on 06/12/2017).

[22] H Kirk Downey, Don Hellriegel, and John W. Slocum. *Organizational behavior : a reader*. St. Paul : West Pub. Co., c1977., 1977. ISBN: 978-0-8299-0137-5.

[23] The Economist. *Bots, the next frontier*. The Economist. Sept. 4, 2016. URL: http://www.economist.com/news/business-and-finance/21696477-market-apps-maturing-now-one-text-based-services-or-chatbots-looks-poised (visited on 06/28/2017).

[24] The Univerity of Edinburgh. *Festival*. Sept. 1, 2017. URL: http://www.cstr.ed.ac.uk/projects/festival/ (visited on 09/01/2017).

[25] Jens Edlund and Jonas Beskow. „Pushy versus meek-using avatars to influence turn-taking behaviour.“ In: *Eighth Annual Conference of the International Speech Communication Association*. 2007. URL: https://www.researchgate.net/profile/Jens_Edlund/publication/221485203_Pushy_versus_meek_-_Using_avatars_to_influence_turn-taking_behaviour/links/0fcfd506ad6ca6c27d000000.pdf (visited on 06/27/2017).

[26] Birgit Endrass, Matthias Rehm, and Elisabeth André. „Culture-specific communication management for virtual agents.“ In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 281–287. URL: http://dl.acm.org/citation.cfm?id=1558052 (visited on 06/27/2017).

[27] Facebook. *Messenger Bots for Business & Developers*. June 28, 2017. URL: https://messenger.fb.com/ (visited on 06/28/2017).

[28] Brian M. Fagan and Charlotte Beck. *The Oxford Companion to Archaeology*. Google-Books-ID: ystMAgAAQBAJ. Oxford University Press, 1996. 865 pp. ISBN: 978-0-19-507618-9.

[29] Usability First. *Usability First - Usability Glossary - menu-driven interface | Usability First*. 2017. URL: index.html (visited on 06/10/2017).

[30] John C. Foster, Rachael Dutton, Mervyn A. Jack, Stephen Love, Ian A. Nairn, Nathalie Vergeynst, and F. W. M. Stentiford. „Interactive Speech Technology.“ In: ed. by Christopher Baber and Janet M. Noyes. Bristol, PA, USA: Taylor & Francis, Inc., 1993, pp. 167–175. ISBN: 978-0-7484-0127-7. URL: http://dl.acm.org/citation.cfm?id=210140.210161.

[31] Toshiaki Fukada, Keiichi Tokuda, Takao Kobayashi, and Satoshi Imai. „An adaptive algorithm for mel-cepstral analysis of speech.“ In: *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*. Vol. 1. IEEE, 1992, pp. 137–140. URL: http://ieeexplore.ieee.org/abstract/document/225953/ (visited on 09/01/2017).

[32] Pabini Gabriel-Petit. *UXmatters :: Glossary :: Terms*. Dec. 6, 2017. URL: http://www.uxmatters.com/glossary/ (visited on 06/12/2017).

[33] Daryle Jean Gardner-Bonneau. „Human Factors Problems in Interactive Voice Response (IVR) Applications: Do we Need a Guideline/Standard?“ In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 36.3 (Oct. 1, 1992), pp. 222–226. ISSN: 1541-9312. DOI: 10.1518/107118192786751899. URL: http://dx.doi.org/10.1518/107118192786751899.

[34] Jesse James Garrett. *The elements of user experience. [electronic resource] : user-centered design for the Web and beyond*. Voices that matter. Berkeley, CA : New Riders, c2011., 2011. ISBN: 978-0-321-68865-1.

[35] Google. *Android Auto*. Android. June 6, 2017. URL: https://www.android.com/auto/ (visited on 06/06/2017).

[36] Google. *Design Principles and Methodology*. May 6, 2017. URL: https://developers.google.com/actions/design/principles (visited on 04/19/2017).

[37] Google. *Google Allo - A smart messaging app*. June 28, 2017. URL: https://allo.google.com/ (visited on 06/28/2017).

[38] Google. *Google Assistant - Your own personal Google*. Google Assistant - Your own personal Google. June 18, 2017. URL: https://assistant.google.com/ (visited on 06/19/2017).

[39] Google. *Google Ngram Viewer*. Aug. 29, 2017. URL: https://books.google.com/ngrams (visited on 08/29/2017).

[40] Google. *How old is the Brookly bridge?* Sept. 4, 2017. URL: https://storage.googleapis.com/cloud-samples-tests/speech/brooklyn.flac (visited on 09/04/2017).

[41] Google. *Quickstart | Google Cloud Speech API Documentation | Google Cloud Platform*. Aug. 31, 2017. URL: https://cloud.google.com/speech/docs/getting-started (visited on 09/04/2017).

[42] Google. *Speech API - Speech Recognition*. Google Cloud Platform. Aug. 29, 2017. URL: https://cloud.google.com/speech/ (visited on 08/29/2017).

[43] Google. *Understanding How Conversations Work:
The Key to a Better UI | Actions on Google*. Google Developers. May 17, 2017. URL: https://developers.google.com/actions/design/how-conversations-work (visited on 06/18/2017).

[44] Jeffrey O. Grady. „Introduction to Systems Requirements.“ In: *System Requirements Analysis, 2nd Edition*. 2nd ed. Elsevier, Sept. 19, 2013. ISBN: 978-0-12-417130-5. URL: http://proquestcombo.safaribooksonline.com/book/operations/9780124171077/1dot-introduction/st0015_html.

[45] H. P. Grjoe. „Logic and conversation.“ In: *Cole, P. and JL Morgan (cds), Syntax and Semantics* 3 (1975), pp. 41–58. URL: https://www.researchgate.net/profile/Barbara_Partee/publication/274129468_Formal_Semantics_Origins_Issues_Early_Impact/links/574614d408ae9ace8424375f.pdf (visited on 04/24/2017).

[46] Dan Grover. *Bots won't replace apps. Better apps will replace apps.* Apr. 2016. URL: http://dangrover.com/blog/2016/04/20/bots-wont-replace-apps.html (visited on 04/08/2017).

[47] John H. L. Hansen and Sanjay Patil. „Speech Under Stress: Analysis, Modeling and Recognition.“ In: *Speaker Classification I*. Ed. by Christian Müller. Lecture Notes in Computer Science 4343. DOI: 10.1007/978-3-540-74200-5_6. Springer Berlin Heidelberg, 2007, pp. 108–137. ISBN: 978-3-540-74186-2 978-3-540-74200-5. URL: http://link.springer.com/chapter/10.1007/978-3-540-74200-5_6 (visited on 06/08/2017).

[48] Marc Hassenzahl and Noam Tractinsky. „User experience - a research agenda.“ In: *Behaviour & Information Technology* 25.2 (Mar. 2006), pp. 91–97. ISSN: 0144-929X, 1362-3001. DOI: 10.1080/01449290500330331. URL: http://www.tandfonline.com/doi/abs/10.1080/01449290500330331 (visited on 06/10/2017).

[49] Julia Hirschberg. „N-Grams and Corpus Linguistics.“ Lecture. CS 4705. Oct. 2, 2007. URL: http://www.cs.columbia.edu/~rambow/teaching/lecture-2007-10-02.ppt (visited on 08/29/2017).

[50] Chih-Yuan Ho, Mark I. Nikolic, and Nadine B. Sarter. „Supporting timesharing and interruption management through multimodal information presentation.“ In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 45. SAGE Publications, 2001, pp. 341–345. URL: http://pro.sagepub.com/content/45/4/341.short (visited on 06/12/2017).

[51] L. Richard Hoffman. „Applying Experimental Research on Group Problem Solving to Organizations.“ In: *The Journal of Applied Behavioral Science* 15.3 (July 1, 1979), pp. 375–391. ISSN: 0021-8863. DOI: 10.1177/002188637901500311. URL: http://journals.sagepub.com/doi/abs/10.1177/002188637901500311.

[52] Computer Hope. *When was the first keyboard invented?* Apr. 26, 2017. URL: https://www.computerhope.com/issues/ch001802.htm (visited on 05/18/2017).

[53] IBM. *IBM Knowledge Center*. Call Flow Diagrams. July 7, 2017. URL: https://www.ibm.com/support/knowledgecenter/en/SS8PJ7_8.5.1/com.ibm.xtools.ngn_toolkit.doc/topics/c_callflow.html (visited on 07/08/2017).

[54] Ionic. *Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular*. Ionic Framework. Aug. 10, 2017. URL: https://ionicframework.com/ (visited on 08/10/2017).

[55] Ionic. *Ionic Framework*. Ionic Framework | Speech Recognition. Aug. 11, 2017. URL: https://ionicframework.com/docs/native/speech-recognition/ (visited on 08/11/2017).

[56] Ionic. *Ionic Framework*. Ionic Framework | Text To Speech. Aug. 11, 2017. URL: https://ionicframework.com/docs/native/text-to-speech/ (visited on 08/11/2017).

[57] IonicV1. *Download - Ionic Components*. Aug. 10, 2017. URL: https://ionicframework.com/docs/v1/overview/#download (visited on 08/10/2017).

[58] Nina G. Jablonski and Leslie Aiello, eds. *The origin and diversification of language*. Wattis Symposium series in anthropology no. 24. San Francisco, Calif: California Academy of Sciences : Distributed by the University of California Press, 1998. 202 pp. ISBN: 978-0-940228-44-3.

[59] *James Madison*. In: *Wikipedia*. Page Version ID: 785434326. June 13, 2017. URL: https://en.wikipedia.org/w/index.php?title=James_Madison&oldid=785434326.

[60] Jinwoo Kim, Jae Hong Ryu, and Tae Man Han. „Multimodal Interface Based on Novel HMI UI/UX for In-Vehicle Infotainment System." In: *ETRI Journal* 37.4 (Aug. 2015), pp. 793–803. ISSN: 12256463. DOI: 10.4218/etrij.15.0114.0076. URL: http://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,cookie,url,cpid,uid&custid=s8863137&db=iih&AN=108832367&site=eds-live&scope=site&authtype=ip,uid (visited on 04/17/2017).

[61] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson, 2014. 939 pp. ISBN: 978-1-292-02543-8.

[62] Karen Kaushansky. *I hear voices: Explorations of multidevice experiences with conversational assistants: UX, IoT & interaction conference: O'Reilly Design, January 19 - 22, 2016, San Francisco, CA*. Jan. 20, 2016. URL: https://conferences.oreilly.com/design/ux-interaction-iot-us-2016/public/schedule/detail/45414 (visited on 08/28/2017).

[63] Kyungduk Kim, Cheongjae Lee, Sangkeun Jung, and Gary Geunbae Lee. „A frame-based probabilistic framework for spoken dialog management using dialog examples." In: *Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue*. Association for Computational Linguistics, 2008, pp. 120–127. URL: http://dl.acm.org/citation.cfm?id=1622088 (visited on 04/24/2017).

[64] Paul Kinlan. *Mobile Web UX - Chrome Dev Summit*. 2017. URL: http://mobile-ux.appspot.com/#56 (visited on 04/17/2017).

[65] Jules Kremer. *Angular: Angular, version 2: proprioception-reinforcement*. Angular. Sept. 14, 2016. URL: http://angularjs.blogspot.com/2016/09/angular2-final.html.

[66] J. D. Lee, B. Caven, S. Haake, and T. L. Brown. „Speech-based interaction with in-vehicle computers: the effect of speech-based e-mail on drivers' attention to the roadway." In: *Human Factors* 43.4 (2001), pp. 631–640. ISSN: 0018-7208. DOI: 10.1518/001872001775870340.

[67] Stephen Love, RT Dutton, John C Foster, Mervyn A Jack, and FWM Stentiford. „Identifying salient usability attributes for automated telephone services." In: *Third International Conference on Spoken Language Processing*. 1994.

[68] Travis Lowdermilk. *User-Centered Design*. O'Reilly Media, Inc., Apr. 4, 2013. 154 pp. ISBN: 978-1-4493-5980-5. URL: http://proquestcombo.safaribooksonline.com/9781449359812.

[69] Paul Luff, David Frohlich, and Nigel G. Gilbert. *Computers and Conversation*. Elsevier, June 2014. ISBN: 978-0-08-050264-9.

[70] Max Lynch. *Announcing Ionic 2.0.0 Final*. The Official Ionic Blog. Jan. 25, 2017. URL: http://blog.ionic.io/announcing-ionic-2-0-0-final/ (visited on 08/10/2017).

[71] Mark ter Maat and Dirk Heylen. „Generating Simple Conversations." In: *Development of Multimodal Interfaces: Active Listening and Synchrony*. Ed. by Anna Esposito, Nick Campbell, Carl Vogel, Amir Hussain, and Anton Nijholt. Lecture Notes in Computer Science 5967. DOI: 10.1007/978-3-642-12397-9_7. Springer Berlin Heidelberg, 2010, pp. 92–101. ISBN: 978-3-642-12396-2 978-3-642-12397-9. URL: http://link.springer.com/chapter/10.1007/978-3-642-12397-9_7 (visited on 06/27/2017).

[72] Chatbots Magazine. *China, WeChat and the Origins of Chatbots*. Chatbots Magazine. Mar. 12, 2017. URL: https://chatbotsmagazine.com/china-wechat-and-the-origins-of-chatbots-89c481f15a44 (visited on 06/28/2017).

[73] Ian McGraw, Rohit Prabhavalkar, Raziel Alvarez, Montse Gonzalez Arenas, Kanishka Rao, David Rybach, Ouais Alsharif, Hasim Sak, Alexander Gruenstein, Francoise Beaufays, and Carolina Parada. „Personalized Speech recognition on mobile devices." In: *arXiv:1603.03185 [cs]* (Mar. 2016). URL: http://arxiv.org/abs/1603.03185 (visited on 03/26/2017).

[74] Mary Meekers. *2016 Internet Trends Report*. Jan. 6, 2016. URL: http://www.kpcb.com/internet-trends (visited on 05/19/2017).

[75] Microsoft. *Cortana | Your Intelligent Virtual & Personal Assistant | Microsoft*. June 18, 2017. URL: www.microsoft.com/en-us/windows/cortana (visited on 06/19/2017).

[76] Microsoft. *Get started with the Azure Speech REST API using cURL*. Mar. 16, 2017. URL: https://docs.microsoft.com/en-us/azure/cognitive-services/speech/getstarted/getstarted-curl (visited on 09/04/2017).

[77] Microsoft. *Pricing - Bing Speech API | Microsoft Azure*. Aug. 29, 2017. URL: https://azure.microsoft.com/en-us/pricing/details/cognitive-services/speech-api/ (visited on 08/30/2017).

[78] Microsoft. *The GUI versus the Command Line: Which is better? (Part 1)*. Microsoft.com Operations. Dec. 3, 2007. URL: https://blogs.technet.microsoft.com/mscom/2007/03/12/the-gui-versus-the-command-line-which-is-better-part-1/ (visited on 06/10/2017).

[79] André Miede. *Neue Wege im Bereich „Interactive Fiction"*. July 6, 2016.

[80] George A. Miller. „The magical number seven, plus-or-minus two, some limits to our capacity for processing information.“ In: *Brain Physiology and Psychology. Buttenvorths: London* (1967), pp. 175–200. URL: http://books.google.com/books?hl=en&lr=&id=rQGiUlAtpQUC&oi=fnd&pg=PA175&dq=%22Magical+Number+Seven,+Plus+or+Minus%22+%22This+number+assumes+a+variety+of+disguises,+being+sometimes+a+little+larger+and+sometimes%22+%22information.+Since+these+experiments+would+not+have+been+done+without+the+appearance%22+&ots=VWAD4ovhQa&sig=HxgV4dXlZ1dbYR9203gda7nxx_0 (visited on 04/24/2017).

[81] Bruno Müller. *Designing native apps for Android and iOS: key differences and similarities*. Cheesecake Labs. Sept. 20, 2016. URL: https://cheesecakelabs.com/blog/designing-native-apps-for-android-and-ios-key-differences-and-similarities/ (visited on 07/08/2017).

[82] Robert J. Moore, Rafa H. Hosn, and Ashima Arora. *The Machinery of Natural Conversation and the Design of Conversational Machines.pdf*. Jan. 1, 2016.

[83] Anh Nguyen and Wayne Wobcke. „An agent-based approach to dialogue management in personal assistants.“ In: ACM Press, 2005, p. 137. ISBN: 978-1-58113-894-8. DOI: 10.1145/1040830.1040865. URL: http://portal.acm.org/citation.cfm?doid=1040830.1040865 (visited on 04/24/2017).

[84] Mark I. Nikolic and Nadine B. Sarter. „Peripheral Visual Feedback: A Powerful Means of Supporting Effective Attention Allocation in Event-Driven, Data-Rich Environments.“ In: *Human Factors* 43.1 (Mar. 1, 2001), pp. 30–38. ISSN: 0018-7208. DOI: 10.1518/001872001775992525. URL: http://journals.sagepub.com/doi/abs/10.1518/001872001775992525.

[85] Peter Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Google-Books-ID: eH6jBQAAQBAJ. Morgan Kaufmann, June 28, 2014. 975 pp. ISBN: 978-0-08-057115-7.

[86] Jan Nouza, Jindrich Zdansky, Petr Cerva, and Jan Silovsky. „Challenges in Speech Processing of Slavic Languages (Case Studies in Speech Recognition of Czech and Slovak).“ In: *Development of Multimodal Interfaces: Active Listening and Synchrony*. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-12397-9_19. Springer, Berlin, Heidelberg, 2010, pp. 225–241. ISBN: 978-3-642-12396-2 978-3-642-12397-9. URL: https://link.springer.com/chapter/10.1007/978-3-642-12397-9_19 (visited on 08/29/2017).

[87] Chris O´Sullivan. *A Tale of Two Platforms: Designing for Both Android and iOS*. Web Design Envato Tuts+. Apr. 15, 2015. URL: https://webdesign.tutsplus.com/articles/a-tale-of-two-platforms-designing-for-both-android-and-ios--cms-23616 (visited on 07/08/2017).

[88] Sharon Oviatt. „Human-centered Design Meets Cognitive Load Theory: Designing Interfaces That Help People Think.“ In: *Proceedings of the 14th ACM International Conference on Multimedia*. MM '06. New York, NY, USA: ACM, 2006, pp. 871–880. ISBN: 978-1-59593-447-5. DOI: 10.1145/1180639.1180831. URL: http://doi.acm.org/10.1145/1180639.1180831.

[89] Michael P. Georgeff and Amy Lansky. *Reactive Reasoning and Planning*. Jan. 1, 1987. 677 pp.

[90] Cathy Pearl. *Designing Voice User Interfaces*. 2016. ISBN: 978-1-4919-5541-3. URL: http://shop.oreilly.com/product/0636920050056.do (visited on 03/25/2017).

[91] Cathy Pearl and Ian Menzies. „DESIGNING FOR DEVICES WITHOUT SCREENS.“ In: *Designing Voice User Interfaces*. 2016. ISBN: 978-1-4919-5541-3. URL: http://shop.oreilly.com/product/0636920050056.do (visited on 03/25/2017).

[92] Rob Price. *One of Mark Zuckerberg's 'big regrets' shows why he's going all-in on virtual reality*. Business Insider. Nov. 16, 2015. URL: http://www.businessinsider.com/mark-zuckerberg-virtual-reality-big-regret-oculus-rift-messenger-2015-11 (visited on 06/28/2017).

[93] Randolph Quirk and Sidney Greenbaum. *A Concise Grammar of Contemporary English*. Google-Books-ID: CNZ4AAAAIAAJ. Harcourt Brace Jovanovich, 1973. 506 pp. ISBN: 978-0-15-512930-6.

[94] Lawrence R. Rabiner. „A tutorial on hidden Markov models and selected applications in speech recognition.“ In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286. URL: http://ieeexplore.ieee.org/abstract/document/18626/ (visited on 08/13/2017).

[95] Anand Rao and Michael P. Georgeff. „BDI agents: From theory to practice.“ In: (Nov. 2000).

[96] Jack C. Richards. „Conversation.“ In: *TESOL Quarterly* 14.4 (1980), pp. 413–432. ISSN: 0039-8322. DOI: 10.2307/3586231. URL: http://www.jstor.org/stable/3586231.

[97] Rutger Rienks and Dirk Heylen. *DOMINANCE DETECTION IN MEETINGS USING EASILY OBTAINABLE FEATURES*. 2002.

[98] Nadine B. Sarter. „Multimodal information presentation: Design guidance and research challenges.“ In: *International Journal of Industrial Ergonomics* 36.5 (May 2006), pp. 439–445. ISSN: 01698141. DOI: 10.1016/j.ergon.2006.01.007. URL: http://linkinghub.elsevier.com/retrieve/pii/S0169814106000217 (visited on 06/09/2017).

[99] Sass. *Sass: Syntactically Awesome Style Sheets*. Aug. 11, 2017. URL: http://sass-lang.com/ (visited on 08/12/2017).

[100] Emanuel A. Schegloff. „Overlapping talk and the organization of turn-taking for conversation.“ In: *Language in society* 29.1 (2000), pp. 1–63. URL: http://journals.cambridge.org/abstract_S0047404500001019 (visited on 04/24/2017).

[101] Emanuel A. Schegloff. „Repair After Next Turn: The Last Structurally Provided Defense of Intersubjectivity in Conversation.“ In: *American Journal of Sociology* 97.5 (1992), pp. 1295–1345. ISSN: 0002-9602. URL: http://www.jstor.org/stable/2781417.

[102] Emanuel A. Schegloff. *Sequence Organization in Interaction: Volume 1: A Primer in Conversation Analysis*. Cambridge University Press, Jan. 2007. ISBN: 978-0-521-53279-2.

[103] Deborah Schiffrin. *Discourse Markers*. Studies in Interactional Sociolinguistics. DOI: 10.1017/CBO9780511611841. Cambridge University Press, 1987.

[104] Robert M. Schumacher, Mary L. Hardzinski, and Amy L. Schwartz. „Increasing the Usability of Interactive Voice Response Systems: Research and Guidelines for Phone-Based Interfaces.“ In: *Human Factors* 37.2 (June 1, 1995), pp. 251–264. ISSN: 0018-7208. DOI: 10.1518/001872095779064672. URL: http://journals.sagepub.com/doi/abs/10.1518/001872095779064672.

[105] Aaron E. Sklar and Nadine B. Sarter. „Good Vibrations: Tactile Feedback in Support of Attention Allocation and Human-Automation Coordination in Event-Driven Domains." In: *Human Factors* 41.4 (Dec. 1, 1999), pp. 543–552. ISSN: 0018-7208. DOI: 10.1518/001872099779656716. URL: http://journals.sagepub.com/doi/abs/10.1518/001872099779656716.

[106] International Organization for Standardization. *Ergonomics of human-system interaction — Part 210: Human-centred design for interactive system (ISO 9241-210:2010)*. 2010. URL: https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-1:v1:en.

[107] Statista. *Anzahl der Apps in den Top App-Stores 2016 | Statistik*. Statista. Aug. 6, 2017. URL: https://de.statista.com/statistik/daten/studie/208599/umfrage/anzahl-der-apps-in-den-top-app-stores/ (visited on 06/08/2017).

[108] Telegram. *Bots: An introduction for developers*. June 28, 2017. URL: https://core.telegram.org/bots (visited on 06/28/2017).

[109] James Tiongson. *Mobile App Marketing Insights: How Consumers Really Find and Use Your Apps*. Think with Google. May 2017. URL: https://www.thinkwithgoogle.com/consumer-insights/mobile-app-marketing-insights/ (visited on 06/08/2017).

[110] Keiichi Tokuda, Heiga Zen, and Alan W. Black. „An HMM-based speech synthesis system applied to English." In: *IEEE Speech Synthesis Workshop*. 2002, pp. 227–230. URL: http://www.scs.cmu.edu/afs/cs.cmu.edu/Web/People/awb/papers/IEEE2002/hmmenglish.pdf (visited on 09/01/2017).

[111] Liam Tung. *Make room: Android apps get bigger as Google doubles file sizes for developers*. ZDNet. Sept. 29, 2015. URL: http://www.zdnet.com/article/make-room-android-apps-get-bigger-as-google-doubles-file-sizes-for-developers/ (visited on 07/17/2017).

[112] Twine. *Twine / An open-source tool for telling interactive, nonlinear stories*. Aug. 10, 2017. URL: https://twinery.org/ (visited on 08/10/2017).

[113] UNESCO. *Adult literacy rate, population 15+ years, both sexes (%) | Data*. 2010. URL: http://data.worldbank.org/indicator/SE.ADT.LITR.ZS?end=2015&start=1970&view=chart (visited on 05/18/2017).

[114] All Bout UX. *User experience definitions « All About UX*. Dec. 6, 2017. URL: http://www.allaboutux.org/ux-definitions (visited on 06/12/2017).

[115] Kris Van Hees and Jan Engelen. „Equivalent representations of multimodal user interfaces: Runtime Reification of Abstract User Interface Descriptions." In: *Universal Access in the Information Society* 12.4 (Nov. 2013), pp. 339–368. ISSN: 1615-5289, 1615-5297. DOI: 10.1007/s10209-012-0282-z. URL: http://link.springer.com/10.1007/s10209-012-0282-z (visited on 04/25/2017).

[116] W3C. *Glossary of Terms for Device Independence*. 2005. URL: https://www.w3.org/TR/di-gloss/#def-user-experience (visited on 06/12/2017).

[117] Jan Weddehage. *User Experience Dossier – Teil 1: UI versus UX*. Nov. 2015. URL: https://entwickler.de/online/ux/user-experience-dossier-teil-1-ui-versus-ux-185990.html (visited on 03/26/2017).

[118] Susan Weinschenk and Dean T. Barker. *Designing effective speech interfaces*. Google-Books-ID: O1tTAAAAMAAJ. John Wiley & Sons, Mar. 3, 2000. 424 pp. ISBN: 978-0-471-37545-6.

[119] Michael Wooldridge and Nicholas R. Jennings. *Intelligent Agents: Theory and Practice*. Vol. 10. DOI: 10.1017/S0269888900008122. Feb. 1, 1970.

[120] Xiaojun Wu, Fang Zheng, and Mingxing Xu. „Topic Forest: a plan-based dialog management structure.“ In: *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*. Vol. 1. 2001, 617–620 vol.1. DOI: 10.1109/ICASSP.2001.940907.

[121] Nicole Yankelovich, Gina-Anne Levow, and Matt Marx. „Designing SpeechActs: Issues in Speech User Interfaces.“ In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '95. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995, pp. 369–376. ISBN: 978-0-201-84705-5. DOI: 10.1145/223904.223952. URL: http://dx.doi.org/10.1145/223904.223952.

[122] B. L. Zeigler and B. Bazor. „Dialog design for a speech-interactive automation system.“ In: *Interactive Voice Technology for Telecommunications Applications, 1994., Second IEEE Workshop on*. IEEE, 1994, pp. 113–116. URL: http://ieeexplore.ieee.org/abstract/document/341532/ (visited on 04/24/2017).

[123] api.ai. *Building Your First Agent*. API.AI. Aug. 7, 2017. URL: https://api.ai/docs/getting-started/building-your-first-agent (visited on 07/08/2017).

[124] wit.ai. *Wit.ai*. Aug. 7, 2017. URL: https://wit.ai/ (visited on 07/08/2017).

List of Figures

2.1 Visual interface of a multimodal car entertainment system 8
2.2 Response of the Google Assistant to request to show a list of the American presidents 13
2.3 Locations which can be easily reached with a thumb on a phone [16] . . . 14
2.4 List of closest parks provided by the Google Assistant 15

3.1 Series of screenshots taken when using the Google Assistant to get more information about James Madison 20
3.2 Expansion with adjacent pairs [102] 23

4.1 UCD in context [68] 30
4.2 An example for a simple flow diagram 36
4.3 Difference between a wireframe and a mockup 38
4.4 Disambiguation on the screen 44
4.5 Result of user's preference regarding error recovery [17] 46

5.1 Overview of parts of a VUI application [13] 49
5.2 A fully connected Markov model [94] 51
5.3 Simple example of a finite state based dialog manager [122] 58
5.4 A frame based information representation [63] 59
5.5 The information representation in a plan based DM [120] 61

6.1 Screenshots from the existing prototype 67
6.2 Mockup of the new player 70
6.3 Overview of the VUI part of TaleTime 76
6.4 Control flow of the VUI and the called methods 78

List of Tables

5.1 Prices of Google's REST speech API 54
5.2 Result of sample audio files from Google's REST Speech API 54
5.3 Prices of Microsoft's Bing Speech API 55
5.4 Result of sample audio files from Microsoft's Speech API 55
5.5 The meaning of a sentence can change depending on its intonation [17] . . 62

Listings

6.1 Part of output section of English grammar JSON 72
6.2 Part of the input section of English grammar JSON 73
6.3 Method to create the answer texts, used as output by the VUI 79
6.4 The method used for matching 81

A.1 The package.json file of the TaleTime project 105

B.1 The English grammar file of the TaleTime project 107

C.1 The result of the Google Speech API for sample 1 111
C.2 The result of the Google Speech API for sample 2 111

D.1 The simple result of Microsoft's Speech API for sample 1 113
D.2 The detailed result of Microsoft's Speech API for sample 1 113
D.3 The simple result of Microsoft's Speech API for sample 2 113
D.4 The detailed result of Microsoft's Speech API for sample 2 114

Abbreviations

APK Android Application Package

AP Additional Property

API Application Programming Interface

ATM Automatic Teller Machine

CA Conversation Analysis

CSS Cascading Style Sheets

DM Dialog Manager

DOM Document Object Model

FAQ Frequently Asked Questions

IVR Interactive Voice Response

JSON JavaSript Object Notation

GPU Graphical Processing Unit]

GUI Graphical User Interface

HCI Human Computer Interaction

HMM Hidden Markov Model

HTML Hypertext MArkup Language

IT Information Technology

LVCSR Large Vocabulary Continuous Speech Recognition

MLSA Mel Log Spectrum Approximation

NLU Natural Language Understanding

NLP Natural Language Processing

NSP No Speech Timeout

PP Primary Property

PRS Procedural Reasoning System

REST Representational State Transfer

SPA Single Page Application

SASS Syntactically Awesome Style Sheets

SDK Software Development Kit

SP Secondary Property

SRA System Requirements Analysis

TCU Turn Constructional Units

TMS Too Much Speed

TTS Text To Speech

UCD User Centered Design

UI User Interface

UX User Experience

VUI Voice User Interface

XML Extended Markup Language

Trademarks

The following list gives an overview about all trademarks used in this thesis.

Amazon™

Amazon Echo™

Amazon Alexa™

Android™

Angular™

Apache Cordova™

Api.ai™

Apple™

Apple iOS™

Apple Siri™

Facebook™

Google™

Google Allo™

Google Assistant™

Google Books N-Gram Viewer™

Google Nexus™

IBM™

Ionic™

Microsoft™

Microsoft Azure™

Microsoft Bing™

Microsoft Cortana™

Nuance™

Nuance Mix™

O' Reilly™

Sass©

Telegram™

Twine™

WeChat©

Wit.ai©

Appendix

A package.json of the TaleTime project

The package.json contains all dependencies used in the TaleTime application. Additionally, all versions are listed.

```
{
    "name": "TaleTime",
    "version": "1.0.0",
    "author": "htw Saar",
    "homepage": "https://www.htwsaar.de/",
    "private": true,
    "scripts": {
        "clean": "ionic-app-scripts clean",
        "build": "ionic-app-scripts build",
        "lint": "ionic-app-scripts lint",
        "ionic:serve": "ionic-app-scripts serve",
        "ionic:run-ad": "ionic cordova run android --device"
    },
    "dependencies": {
        "@angular/common": "4.1.0",
        "@angular/compiler": "4.1.0",
        "@angular/compiler-cli": "4.1.0",
        "@angular/core": "4.1.0",
        "@angular/forms": "4.1.0",
        "@angular/http": "4.1.0",
        "@angular/platform-browser": "4.1.0",
        "@angular/platform-browser-dynamic": "4.1.0",
        "@ionic-native/core": "3.7.0",
        "@ionic-native/file": "^3.12.1",
        "@ionic-native/globalization": "^4.1.0",
        "@ionic-native/native-audio": "^3.14.0",
        "@ionic-native/speech-recognition": "^4.1.0",
        "@ionic-native/splash-screen": "3.7.0",
        "@ionic-native/status-bar": "3.7.0",
        "@ionic-native/text-to-speech": "^4.1.0",
        "@ionic/storage": "2.0.1",
        "cordova-android": "^6.2.2",
        "cordova-plugin-android-permissions": "^1.0.0",
        "cordova-plugin-compat": "^1.0.0",
        "cordova-plugin-console": "^1.0.7",
        "cordova-plugin-device": "^1.1.6",
        "cordova-plugin-file": "^4.3.3",
        "cordova-plugin-globalization": "^1.0.7",
        "cordova-plugin-nativeaudio": "^3.0.9",
        "cordova-plugin-speechrecognition": "^1.1.2",
        "cordova-plugin-splashscreen": "^4.0.3",
```

```
42          "cordova-plugin-statusbar": "^2.2.3",
43          "cordova-plugin-tts": "^0.2.3",
44          "cordova-plugin-whitelist": "^1.3.2",
45          "inquirer": "^3.2.1",
46          "ionic-angular": "3.2.0",
47          "ionic-plugin-keyboard": "^2.2.1",
48          "ionicons": "3.0.0",
49          "ng2-translate": "^5.0.0",
50          "rxjs": "5.1.1",
51          "sw-toolbox": "3.6.0",
52          "ts-data.stack": "^1.0.6",
53          "zone.js": "0.8.10"
54      },
55      "devDependencies": {
56          "@ionic/app-scripts": "1.3.7",
57          "@ionic/cli-utils": "^1.7.0",
58          "ionic": "3.10.1",
59          "typescript": "2.2.1"
60      },
61      "cordovaPlugins": [
62          "cordova-plugin-whitelist",
63          "cordova-plugin-statusbar",
64          "cordova-plugin-device",
65          "cordova-plugin-console",
66          "ionic-plugin-keyboard",
67          "cordova-plugin-splashscreen"
68      ],
69      "cordovaPlatforms": [],
70      "description": "TaleTime: A project of htw Saar",
71      "cordova": {
72          "plugins": {
73              "cordova-plugin-console": {},
74              "cordova-plugin-device": {},
75              "cordova-plugin-splashscreen": {},
76              "cordova-plugin-statusbar": {},
77              "cordova-plugin-whitelist": {},
78              "ionic-plugin-keyboard": {},
79              "cordova-plugin-file": {},
80              "cordova-plugin-nativeaudio": {},
81              "cordova-plugin-android-permissions": {},
82              "cordova-plugin-globalization": {},
83              "cordova-plugin-tts": {},
84              "cordova-plugin-speechrecognition": {}
85          },
86          "platforms": [
87              "android"
88          ]
89      }
90  }
```

Listing A.1: The package.json file of the TaleTime project

B English grammar file for the TaleTime VUI

```
{
  "answers": {
    "multiple": [
      {
        "id": 0,
        "value": "How do you want the story to continue: "
      },
      {
        "id": 1,
        "value": "How would you like to continue: "
      },
      {
        "id": 2,
        "value": "You have the choice: "
      }
    ],
    "single": [
      {
        "id": 0,
        "value": "Do you want to continue?"
      },
      {
        "id": 1,
        "value": "Continue?"
      }
    ],
    "help": [
      {
        "id": 0,
        "value": "Tap a button on the screen to continue"
      }
    ]
  },
  "linking": {
    "or": [
      {
        "id": 0,
        "value": "or"
      }
    ],
    "and": [
      {
        "id": 0,
```

```
        "value": "and"
      }
    ]
  },
  "repeat": [
    {
      "id": 0,
      "value": "There was no match. Please say that again."
    },
    {
      "id": 1,
      "value": "Please say that again."
    },
    {
      "id": 2,
      "value": "Oops. Please say that again."
    }
  ],
  "agree": [
    {
      "id": 0,
      "value": "Yes"
    },
    {
      "id": 1,
      "value": "Okay"
    },
    {
      "id": 2,
      "value": "Yep"
    }
  ],
  "decline": [
    {
      "id": 0,
      "value": "No"
    }
  ],
  "enum": [
    {
      "id": 0,
      "value": "first",
      "index": 1
    },
    {
      "id": 1,
      "value": "second",
      "index": 2
    },
    {
```

```
      "id": 2,
      "value": "third",
      "index": 3
    }
  ],
  "backwards": [
    {
      "id": 0,
      "value": "back"
    },
    {
      "id": 1,
      "value": "previous"
    },
    {
      "id": 2,
      "value": "last"
    }
  ],
  "repeatChapter": [
    {
      "id": 0,
      "value": "repeat"
    },
    {
      "id": 1,
      "value": "play again"
    },
    {
      "id": 2,
      "value": "start over"
    }
  ],
  "doNotCare": [
    {
      "id": 0,
      "value": "next"
    },
    {
      "id": 1,
      "value": "skip"
    },
    {
      "id": 2,
      "value": "move on"
    },
    {
      "id": 3,
      "value": "don't care"
    },
```

```
144       {
145         "id": 4,
146         "value": "continue"
147       }
148     ]
149   }
```

Listing B.1: The English grammar file of the TaleTime project

C Results of Google Speech API as JSON

Listing C.1 shows the result of sample 1 which was professionally recorded. The original text was *How old is the Brooklyn Bridge?*

```
{
  "results": [
    {
      "alternatives": [
        {
          "transcript": "how old is the Brooklyn Bridge",
          "confidence": 0.987629
        }
      ]
    }
  ]
}
```

Listing C.1: The result of the Google Speech API for sample 1

Listing C.2 shows the result of sample 2. The quality of this example is worse than the one of sample 1. The original text was *The quick brown fox jumps over the lazy dog*.

```
{
  "results": [
    {
      "alternatives": [
        {
          "transcript": "the quick brown fox jumped over the
             lazy dog",
          "confidence": 0.97975445
        }
      ]
    }
  ]
}
```

Listing C.2: The result of the Google Speech API for sample 2

D Results of Microsoft's Speech API as JSON

Listing D.1 shows the result of sample 1 which was professionally recorded. The original text was *How old is the Brooklyn Bridge?*. In this result, the format option *simple* was used.

```
{
    "RecognitionStatus": "Success",
    "DisplayText": "How old is the Brooklyn bridge?",
    "Offset": 1100000,
    "Duration": 16700000
}
```

Listing D.1: The simple result of Microsoft's Speech API for sample 1

Listing D.2 shows the result of sample 1 which. Here, the format option *detailed* was used.

```
{
    "RecognitionStatus": "Success",
    "Offset": 1100000,
    "Duration": 16700000,
    "NBest": [{
        "Confidence": 0.95739913,
        "Lexical": "how old is the brooklyn bridge",
        "ITN": "how old is the Brooklyn bridge",
        "MaskedITN": "how old is the Brooklyn bridge",
        "Display": "How old is the Brooklyn bridge?"
    }]
}
```

Listing D.2: The detailed result of Microsoft's Speech API for sample 1

Listing D.4 shows the result of sample 2. The quality of this example is worse than the one of sample 1. The original text was *The quick brown fox jumps over the lazy dog*. In this result, the format option *simple* was used.

```
{
    "RecognitionStatus": "Success",
    "DisplayText": "Brown Fox jumps over.",
    "Offset": 13700000,
    "Duration": 18900000
}
```

Listing D.3: The simple result of Microsoft's Speech API for sample 2

Listing D.4 shows the result of sample 1 which. Here, the format option *detailed* was used.

```
{
    "RecognitionStatus": "Success",
    "Offset": 13700000,
    "Duration": 18900000,
    "NBest": [{
        "Confidence": 0.8211723,
        "Lexical": "brown fox jumps over",
        "ITN": "Brown Fox jumps over",
        "MaskedITN": "Brown Fox jumps over",
        "Display": "Brown Fox jumps over."
    }, {
        "Confidence": 0.8211723,
        "Lexical": "brown fox jumps",
        "ITN": "Brown Fox jumps",
        "MaskedITN": "Brown Fox jumps",
        "Display": "Brown Fox jumps."
    }, {
        "Confidence": 0.7018736,
        "Lexical": "call brown fox jumps over",
        "ITN": "call Brown Fox jumps over",
        "MaskedITN": "call Brown Fox jumps over",
        "Display": "Call Brown Fox jumps over."
    }, {
        "Confidence": 0.7018736,
        "Lexical": "call brown fox jumps",
        "ITN": "call Brown Fox jumps",
        "MaskedITN": "call Brown Fox jumps",
        "Display": "Call Brown Fox jumps."
    }, {
        "Confidence": 0.7018736,
        "Lexical": "brown fox jumps over the lazy",
        "ITN": "Brown Fox jumps over the lazy",
        "MaskedITN": "Brown Fox jumps over the lazy",
        "Display": "Brown Fox jumps over the lazy."
    }]
}
```


Listing D.4: The detailed result of Microsoft's Speech API for sample 2

Colophon

This document was created using the LaTeX-template for graduation works at htw Saar in the fields of study of applied computer science and mechatronics / sensor technology (Version 2.1). The template was created by Yves Hary and André Miede (with friendly support of Thomas Kretschmer, Helmut G. Folz und Martina Lehser).

Page data: (F)10.95 – (B)426.79135pt – (H)688.5567pt