An Architecture of a Telephone-based System for Speech Data Collection

Student Research Project at the Cognitive Systems Lab (CSL)
Prof. Dr.-Ing. Tanja Schultz
Department of Computer Science
Karlsruhe Institute of Technology (KIT)

from

Zlatka Mihaylova

Supervisors:

Prof. Dr.-Ing. Tanja Schultz
Dipl.-Inform. Tim Schlippe

Begin: 31. October 2009
End: 30. January 2010

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 30. Januar 2010

Abstract

The work describes the process of building a telephone-based interactive voice response (IVR) system. For the collection of audio data for speech recognition, web-based audio recorders are used so that technical audio equipment does not have to be shipped to the countries with the language in question. However, some parts of the world do not have an Internet infrastructure at all or only a slow one. If these areas provide a telephone infrastructure, an appropriate telephone-based interface is a solution for the collection of audio data. Using existing open source software tools and some classical concepts for implementing conversational systems, we designed and implemented a low cost telephony system for this specific task. Our final system consists of two modules for collecting spontaneous and read speech. Both modules were evaluated with a Wizard of Oz scenario. A second test with the implemented read speech module was performed. As a result from this second evaluation, we can summarize that the test users were able to easily complete their task and are overall satisfied with the system. The system is running and can be accessed via the public switched telephone network on +4972118030681.

Zusammenfassung

Die Arbeit stellt den Prozess beim Aufbau eines Telefon-based Interactive Voice Response (IVR) Systems dar, das sich mit folgender Problematik beschäftigt: Um für die Spracherkennung benötigte Sprachdaten zu sammeln, werden web-basierte Audio Recorder verwendet, damit das dafür nötige Equipment nicht in die Länder geschickt werden muss, in denen die Zielsprache gesprochen wird. Jedoch existiert in einigen Teilen der Erde keine Internet-Infrastruktur oder eine schlecht ausgebaute. Wenn diese Gebiete jedoch ein gut ausgebautes Telefonnetz besitzen, ist eine Sprachdatensammlung über das Telefon eine Lösung. Durch Verwendung existierender Open-Source-Software-Tools und einiger klassischer Konzepte für die Umsetzung von Dialogsystem, gehen wir durch den gesamten Prozess von der Konzeption bis zur Umsetzung eines kostengünstigen Telefon-Systems für die gestellte Aufgabe. Unseres edgültiges System besteht aus zwei Modulen für die Sammlung von gelesener und spontaner Sprache. Die beiden Module wurden anhand einer "Wizard of Oz" Simulation getestet. Eine zweite Auswertung des implementierten "read speech" Moduls wurde ausgeführt. Die Ergebnisse von der zweiten Auswertung zeigen, dass die Test-Anwender ihre Aufgaben leicht ausführen könnten und generell zufrieden mit dem System sind. Das laufende System kann via das Telefonnetzwerk auf +4972118030681 erreicht werden.

Contents

1. Introduction

In the following work, we describe the complete process of building a telephone based dialog system with the specific task of collecting audio data for speech recognition. The reader will be guided through this process from the birth of the idea and all necessary terms and concepts, to the system design and final implementation. In the following chapters, we are going to discuss not only the solutions of all arising problems, but also their possible alternatives.

Designing a dialog system generally involves taking a lot of decisions which should make it intuitive, robust and effective. We try to review all high and low level design decisions connected with this topic as well as their possible states. Based on these, our intention is to find the right software tools in order to realize our ideas. Due to the nature of the required communication, we can not skip explaining the fundamentals of the classical and Internet telephony. The audio data which we intend to collect is used to develop speech processing systems.

The motivation for a telephone-based dialog system for speech data collection is to find inexpensive ways of getting audio data required for the process of training speech recognition engines at Cognitive Systems Lab (CSL)[1]. It is an alternative interface to the web-based audio recorder used in our Rapid Language Adaptation Toolkit (RLAT)[2] for participants preferring to use telephone connection instead of Internet or those who do not have an Internet connectivity. In Section 1.2 we explain the purpose and meaning of rapid language adaptation.

1.1 Goals of this study

The main goal of this study is the implementation of a cost-effective and easy to use interactive system with the specific task to collect audio data used for speech recognition purposes. The only requirement for successful accomplishment of this task should be any kind of phone connection.[1] There are a lot of other requirements for the system itself to improve its stability and user friendly appliance, but these are secondary constraints, evolving from the main system goal which are detailed discussed in Section 3.1.

[1] We do not count the willingness of the people to participate in our project, of course.

1.2 Rapid Language Adaptation

The Rapid Language Adaptation Toolkit (RLAT) extends the SPICE system developed at the Carnegie Mellon University (CMU). SPICE stands for Speech Processing - Interactive Creation and Evaluation Toolkit for new Languages [3] and is based on the existing projects GlobalPhone [4] and FestVox [5]. Since the building of speech processing systems for new languages involves a significant amount of time and effort, the main purpose of SPICE is to reduce those.

SPICE is a web-based system and its tools can be used for collecting text from the web. The system is designed to be utilized by a large range of users - from novices to speech technologies experts. A powerful concept in SPICE's architecture is the sharing of data which happens on two levels: across languages and across system components.

Building a speech processing system for a new language with SPICE happens in nine steps such as "Text and prompt selection", "Audio collection", "Phoneme selection", etc. and it takes approximately 60 days for the development of a two-way speech translation system [6]. To find additional details about the project, please visit the RLAT webpage [2]. Using the system is completely free of charge.

RLAT offers a web-based recording tool. For portability reasons, it is implemented in Java. If the user has already prepared audio files in wave format, then the easiest way is to upload them.

For more information related to speech recognition and synthesis technologies, please refer to Section 2.1.2.

1.3 Structure

The second chapter introduces all terms and concepts which are important to understand this work. Furthermore, we will give an overview of approaches which are either similar or connected to our study.

Chapter 3 defines requirements which should be respected during the process of building such a system. These requirements outline the desired qualities of the conversational system. In the same chapter we describe related and existing solutions. Some of these solutions and ideas are useful and others are interesting to be explored for the purpose of possible innovations.

The two following chapters are the most important part of the work as they describe precisely every single step of the realization of our project. In the process of designing our system a variety of decisions is considered to select the most appropriate ones. From analyzing users and their tasks to choosing very fine technical features, all these necessary phases influence the final implementation of the system.

The implementation process itself consists of three parts - choosing the system's tools, implementing the already planned and partially tested design, and finally integrating each part into one complete system. In the world of voice over IP (VoIP) there are a lot of possibilities for the different components, which, unfortunately, are rather partial solutions. Following one standard is not a trivial task since in this technological area there is no definitely dominating standard. This makes the

compatibility of all system components an issue. We tested some commercial and open source solutions which are also described in Chapter 5.

In Chapter 6 we evaluate the system by two tests, one in the design phase and one after the implementation.

Finally, we conclude with a review of our solution, problems, possible improvements as well as directions for future developments.

1.4 Summary

The goal of this work is to design and implement a telephone-based interface for speech data collection. This interface complements an existing web interface in the RLAT system of the CSL at the Karlsruhe Institute of Technology.

The benefits of a telephone-based speech data collection system are that such systems can be used in the field where no or slow Internet connectivity is available. Such systems are comfortable, as they rely on technologies that majority of population is equipped with. As drawbacks we mansion the lower quality of the signal due to the lower sampling rate of the telephone network and the less structured collective process.

We aim to use an open source software due to the requirement to build a low cost solution. As another requirement, our system must be robust and easy to use from both experienced and unexperienced users.

2. Basics

This section introduces into the terms and concepts of telephony, conversational systems and speech technologies in general.

2.1 Important terms

Recently, we observe many products which are result from combining different kinds of technologies. Examples of such fusions are, for example, car navigation systems, entertainment electronics with Internet interfaces and bio-sensors systems. The implementation of a modern IVR system makes no exception. It involves knowledge from many areas, more precisely from the field of telephony, available speech tools and existing web technologies.

2.1.1 Telephony principles

A telephone similar to the electronic devices that we have nowadays, came into existence in the 1870s. The idea of connecting two distant places with each other is very old. However, the first successful attempt for voice signal transmission dates from 1876, when Alexander Graham Bell was able to send a voice message carried by electricity, which was reproduced on the Thomas A. Watson's receiving set [7].

The first telephones were connected directly with each other by cable. As this technology continued evolving, the necessity of finding a more scalable solution arose. One centralized system uses n instead of $n * (n - 1)$ connections between n participants.[1] This enormous reduction led to the invention of an electronic circuit system called **telephone exchange**[2] . Such a switched system began to play the role of the connecting part between two or more telephony end points. Later it was called the central office. Every telephone set is connected to the central office by a pair of cables, defining this way the so called local loop. At the beginning the destination number was sent by dial pulses as a result of measuring how far the rotary dial is, before being released. Nowadays, this method is replaced by sending a sequence of touchtones, known as **dual-tone multifrequency (DTMF)**. There are sixteen

telephone exchange

DTMF

[1] Complete centralization is, however, also not desirable due to technical difficulties.

[2] Also known as telephone switch.

keys (numbers in the range from 0-9, the letters A, B, C, D and the symbols $*$ and #) on the telephone keypad. Pressed, each of them produces two tones, which are sent to the central office as audio signal. The keys are ordered in a 4×4 matrix. Each row of this matrix represents low and each column high frequency. For example, if you press 5, two frequencies will be sent to the central office to determine which key is associated with these frequencies. For more information about this topic, refer to [7].

For simplification purposes, the architecture of a phone system adopted later concepts like trunking which can be explained as a tree structure with a main transmission channel (the trunk) beginning from the switch center and branching at the opposite end into smaller lines. The amount of all possible telephone exchanges forms the so called **Public Switched Telephone Network (PSTN)**. This network has a hierarchical structure roughly divided into classes for long distance switching and those for connection to the end subscribers. There are three kinds of networks defined: local network, exchange area network and long-haul network.

PSTN

The granularity of the telephone structure can be further refined with the introduction of so called **Private Branch Exchange (PBX)**. It plays the connecting part between a private telephone network of an organization and the PSTN. With the extremely fast development of the Internet in the 1990's, this concept evolved to VoIP PBX (Voice over Internet Protocol). VoIP describes a technology that allows delivery of voice signals through a packet-switched network (usually Internet) able to communicate with other networks through the Internet Protocol (IP). VoIP is often referred to as Internet telephony or voice over broadband (VoBB).

PBX

Later it was obvious that not every small company has the resources and competence to manage its own PBX. Therefore the telephone service providers offered to host and manage PBXes, delivering all features and calls via the Internet. As a consequence of this evolution, nowadays there are four types of PBX, visualized in Table 2.1.

	circuit switched	packet switched
private	traditional PBX	IP PBX (VoIP PBX)
hosted	virtual PBX	virtual IP

Table 2.1: Types of PBX

The hosted solutions offer flexibility as they allow merging between IP telephony and PSTN without the need of specialized telephony hardware. The connection between VoIP solutions and the traditional telephone infrastructure could be implemented in two ways: Either by a hosted (virtual) PBX service or by a VoIP Gateway between the telephony network and the Internet. Using virtual IPs is meaningful for easy

softphone

tests of the user friendliness and other features of the dialog system. In this case, we can use different types of IP phones (also called **softphones**) to call a conversational application running on remote computer.

To summarize, Figure 2.1 gives an overview of a telephone-based dialog system architecture.

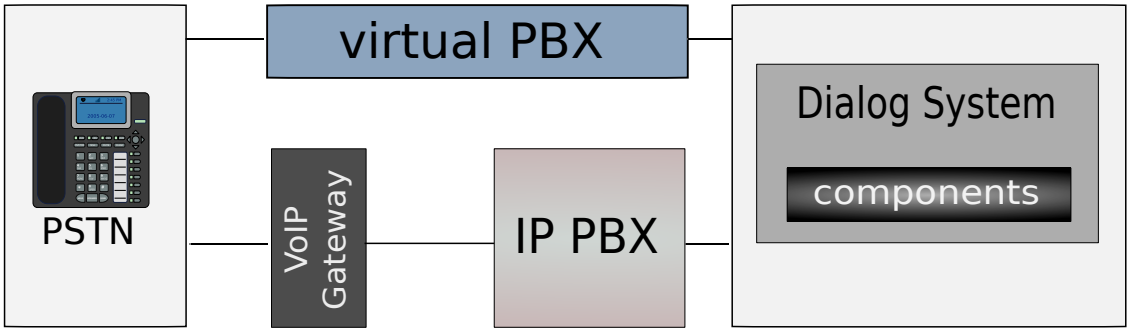


Figure 2.1: The big picture of a telephone-based dialog system

2.1.2 Speech technologies

Before the clarification of the main Interactive Voice Response (IVR) components, we give a proper definition of what IVR exactly is. According to the PC Magazine [8], IVR refers to an automated telephone information system that speaks to the caller with a combination of fixed voice menus and data extracted from databases in real time. The caller responds by pressing digits on the telephone or speaking words or short phrases. IVR

In order to implement an IVR system, components capable of interacting with the people are necessary. The basic system modules are an input component (either recognition of a spoken language or DTMF control signal) and an output component (the synthesis of speech) [9].

The **text-to-speech (TTS)** unit is responsible for transforming an input text into an audio file. TTS engines are often referred to as speech synthesizers. There are different ways for synthesizing speech [10]. Some systems use an unit selection principle, while others produce completely synthesized speech by replacing the formants with pure tones. An example of unit selection speech synthesis is when phones, diphones, syllables etc. are segmented from a large database with recorded utterances and then concatenated with each other in the process of synthesis. TTS engines built by unit selection sound more natural than parametric ones but they rely on gigabytes of recorded speech which could be a problem. A different approach is the parametric speech synthesis which does not use human speech but artificially produced output by configuring parameters such as fundamental frequency, voicing and noise levels of a waveform. TTS

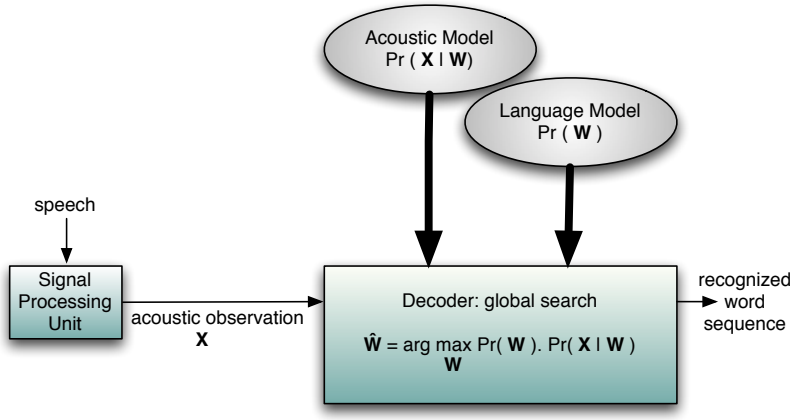


Figure 2.2: Architecture of ASR system

In telephony IVR systems, such engines are used only for reproducing dynamic content like the current time, or some other issues that are either unpredictable or have too many possible values. Such content is hard to be recorded. The reason why prerecorded prompts are generally preferred in dialog systems is due to their natural sounding which supports the human-machine communication.

There are two kinds of inputs used in telephone-based IVR systems. Either the dialog flow[3] is interactively controlled by DMTF signals or we can get information for the user's choices from the speech recognizer. Both methods have certain advantages and disadvantages: We can itemize to the advantages of the speech recognition approach as the natural and simple way of interacting, while DTMF offers more complicated and not always intuitive interaction. On the other hand, using DTMF input guarantees the complete lack of speech recognition errors, while the other technology still may suffer from significant error rates[4].

ASR

Speech recognition technology is referred to as **Automatic Speech Recognition (ASR)** and generally means how the human speech is converted to text through a machine. In telephone-based dialog systems, this technology is often implemented as a part of a speech server and it is responsible for the recognition of the user's input. The quality of the dialog system using ASR depends on the quality of this component implementation since the user's input controls the dialog flow. Every recognition error could possibly lead to a false state in the call flow. Most important characteristics of the different ASR systems are word error rate (WER) and real time factor, or simply accuracy and speed. But factors as format handling, usability and command success rate are also interesting for performance measurements [9].

The basic architecture of an ASR system is presented in Figure 2.2. As described in [11], the speaker produces speech with his or her vocal apparatus. The signal processing unit captures this signal and then through acoustic analysis the observed acoustic signal is parametrized into a sequence of vectors $\mathbf{X} = X_1X_2...X_n$. The next step is to estimate the most probable word sequence $\hat{\mathbf{W}} = W_1W_2..W_m$ according to our acoustic observation $\mathbf{X}$ and the acoustic and linguistic models we use. In

[3]The terms "dialog flow" and "call flow" have the same meaning in this work.

[4]Strongly dependent on the language, vocabulary size, perplexity and speaker-dependance, the acceptable error rates are usually at 10%.

other words, we try to find the maximal $\Pr(\mathbf{W}|\mathbf{X})$. Applying the Bayes' rule we reformulate the problem to: Find the word sequence

$$\hat{\mathbf{W}} = \arg\max_{W} \Pr(\mathbf{W}) \Pr(\mathbf{X}|\mathbf{W}).$$

Acoustic models represent the knowledge about acoustics, phonetics, gender and dialect differences among speakers, etc. Since there is a large number of words, we try to decompose them into a sub-word units when creating an acoustic model. This procedure is very closely related with the phonetic modeling. Therefore we try to find the probability that a sub-unit sequence occur if we have already observed $\mathbf{X}$.

The language model works one level higher and observes the relations between words. Since some words are more likely to co-occur with others and also the occurrance sequence plays a role, all this additional information is present in the language model. Generally, the language model captures the linguistic properties of the language and provides the a-priori probability $\Pr(\mathbf{W})$ of a word sequence. The decoder uses both acoustic and language models and generates the most probable word sequence as result.

VoiceXML

The **Voice Extensible Markup Language (VoiceXML)** is an XML-based markup language specially designed for usage in dialog applications. It aims to standardize the way in which IVR applications take advantage of existing web-based technologies. VoiceXML documents mark dialog features such as synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed initiative conversations. They do not perform these actions but control the information flow between the system's components. Similar to the HTML pages, VoiceXML pages are used in the voice browser, introduced in Section 2.1.3, to expose the speech content and wrap the logic of the conversion. For the full specification of the VoiceXML standard in its current version 2.1, please refer to [12].

CCXML

Call Control eXtensible Markup Language (CCXML) is another standard developed by W3C [13] which aims to provide call control functions to dialog systems. CCXML standard differs from VoiceXML, as it is designed to support mainly telephony applications. Although CCXML and VoiceXML are separated languages, the CCXML is not created with the idea to be an alternative for VoiceXML standard. Both standards work together since CCXML is designed to complement and integrate with VoiceXML interpreters. For more details about the current CCXML version, please refer to [14].

SCXML

Since some developers of open source voice browsers[5] have announced that they will implement an SCXML-capable module, we are also interested to present this standard. **State Chart XML (SCXML)** is an general-purpose event-based state machine language defined by the W3C. It is a powerful abstraction that combines concepts from CCXML and Harel State Tables [15] and is able to describe complex state-machines. It can be used in many ways. For example, as a high-level language controlling VoiceXML application modules. SCXML is also useful as a metalanguage for voice applications to control not only the VoiceXML and CCXML functions but also the database access and business logic modules. It can describe notations such as sub-states, parallel states, synchronization, or concurrency. To see a list of the

[5]For example JVoiceXML presented in Section5.1.4

SCXML application possibilities and the specification of this standard, please visit [16].

SRGS

The speed of the speech recognition is of great importance in real-time voice applications. To speed-up the voice recognition process, a common practice is the usage of speech recognition grammars. In the grammar files, developers can specify the words and patterns of words to be recognized by a speech recognizer. The syntax of the grammar format is specified by the **Speech Recognition Grammar Specification (SRGS)** standard which can have two forms: Augmented Backus-Naur Form (ABNF) or XML-based syntax. Both forms are semantically mappable which allows the automatic transformation between the two forms. Since ABNF has a plain-text representation, it is easily readable by humans. The speech grammars are used by the grammar processors which are not necessary speech recognizers. Another class of grammar processors is DTMF detectors. In the grammar files used by an DTMF detector, the developer specifies the expected DTMF inputs. To read more about the SRGS, please refer to [17].

MRCP

Media Resource Control Protocol (MRCP) is a communication protocol for controlling media service resources [18]. The MRCP client sends requests to an MRCP server over a network and this way it controls the media processing resource, for example speech recognizer or synthesizer. MRCP Version 2 uses the **Session Initiation Protocol (SIP)** as a control protocol. The message format for MRCP is text based and carries data like recognition grammars, recognition results as well as synthesizer speech markup between the client and the server applications.

SIP

SIP is a text-based signaling protocol, often used to control the multimedia communication session such as VoIP [19]. This protocol is exclusively involved in the signaling portion of the communication session. Therefore it works together with several other protocols such as the **Real-Time Transport Protocol (RTP)** responsible for the voice and video transfer and the **Session Description Protocol (SDP)** which carries the multimedia session parameters (like port number, audio codecs, etc). In other words, SIP capable devices send messages to each other, containing invitation for conversation, cancellation of the current session etc., or receive response if the session initiation was successful or interrupted by errors.

2.1.3 Web technologies

speech server

Commercial IVR solutions introduce the **speech server** component. Although IT companies have different implementations of this module, the generic speech server consists of a speech recognizer, TTS engine and optionally a DTMF unit. The term is also used to describe some open source speech solutions. We will give several speech server examples in Section 5.

voice browser

The second term to be introduced is **voice browser**. "Voice browser" is a very abstract term as there are lots of voice browsers built from different components and working on different principles. A voice browser is defined by its functions rather than by its structure and implementation. Most people have an idea what a plain Internet browser looks like as it is visible. But in fact, what is important, are the functions of this program. We know that by clicking on the back button we are able to open a previous visited page. The voice browser is similar to the Internet browser with the difference that you can control its functions by voice commands. The controls are invisible for the user but still accessible through his or her speech.

An **application server (AS)** is a dedicated program responsible for managing and running software applications accessible from a remote machine or another application (the client). A web server is part of the application server usually storing web application and documents accessible through the Internet. The difference between a web and an application server is that the web server handles HTTP requests while the application server offers greater flexibility by dealing with a wider range of protocols. Examples of application servers are the IBM Websphere AS, Microsoft IS as well as JBoss for Java deployments. The most well known web server is the Apache Tomcat developed by the Apache Software Foundation [20]. application server

In the next chapter, we set the requirements for our system and introduce some work related to our project.

3. Analysis

In the following section, we discuss and outline the requirements of our system.

3.1 Requirements

There are two types of properties that could be defined for the implementation of a system - desirable and obligatory. The desirable properties are qualities, which make our system more user friendly and easy to interact with. The obligatory properties are requirements, which must be satisfied in the implementation of our system. We start with the requirements. Derived from our motivation to find an inexpensive way for audio data collection, our system should also be developed with a minimum of financial cost. Although we take a look at some commercial solutions, our end system uses exclusively open source tools and principles. As a second requirement we define the user friendliness of our interactive system. We conclude that user friendliness is in very strong relationship with simplicity. A complex system with a lot of "fancy" features such as user registration and identification is not suitable for our purposes. Moreover, such a system can cause confusion among the participants in our project. As a consequence of this requirement, the system should offer a clear navigation.

As desirable qualities of the system, we additionally mention intuitiveness and minimal demands on user's memory and cognitive processes. By intuitiveness, we mean no prior knowledge and training for the usage.

3.2 Related work

The idea of collecting speech data through a telephone interface is not a new one. At the beginning of the 1990's, many researchers have tried to find an inexpensive way to collect audio data. Unfortunately in these times, it was not easy since telephone companies offered proprietary solutions at very high prices. For example, the T1-based speech data collection platform described in [21] faces the same challenge as our work, but they put the emphasis on the hardware connection between a telephone line and a computer, rather than on comfortability issues.

Another project, SmartWeb [22], born at the University of Munich researched the collection of spontaneous speech data. Their main challenges were to elicit representative utterances from the participants that are as natural as possible, in real situation and in a cost efficient way. Similar to our test approach, they also conducted the "Wizard of Oz" experiment, described in Section 6.1, and divided the prompts into topics. Each prompt in their system starts with a "beep" signal and had a maximal recording time of 12 seconds or when the silence detection signals the end of the speaker's turn.

Although the SmartWeb project seems to have a lot of similarities with our project, there are some differences. First, SmartWeb is made only for German language. Therefore it uses a German TTS engine of AT&T. It covers only cell phones, no landline telephones as the acoustic signal is transferred over Universal Mobile Telecommunications System (UMTS) [23] and Bluetooth, which is not supported by the landline. The biggest difference is that users were prepared for the recording. They were asked to prepare some topics a few days before the recording. Therefore the prompts are also individualized for each user. Finally, the researchers from the University of Munich use signal post-processing to extract the signal from the noise and the client-server application WebTranscribe for transliteration purposes. More information about SmartWeb can be found in [22].

A lot of projects aim to collect speech data for a specific domain. For example the project MoTiV [24] intends to collect speech data for voice control in car environments. Other projects do exactly the opposite. They invent methods for more general domains, like collecting natural speech from non-native speakers without a telephone [25].

3.3 Summary

To summarize the most important conclusions from this chapter, the critical system requirements we agreed on are as follows:

- Low budget
- Simplicity
- Intuitiveness
- Minimal demands on user's memory

Within the next chapter, we try to divide the problem into smaller pieces and consider finer details for the system's design.

4. Design

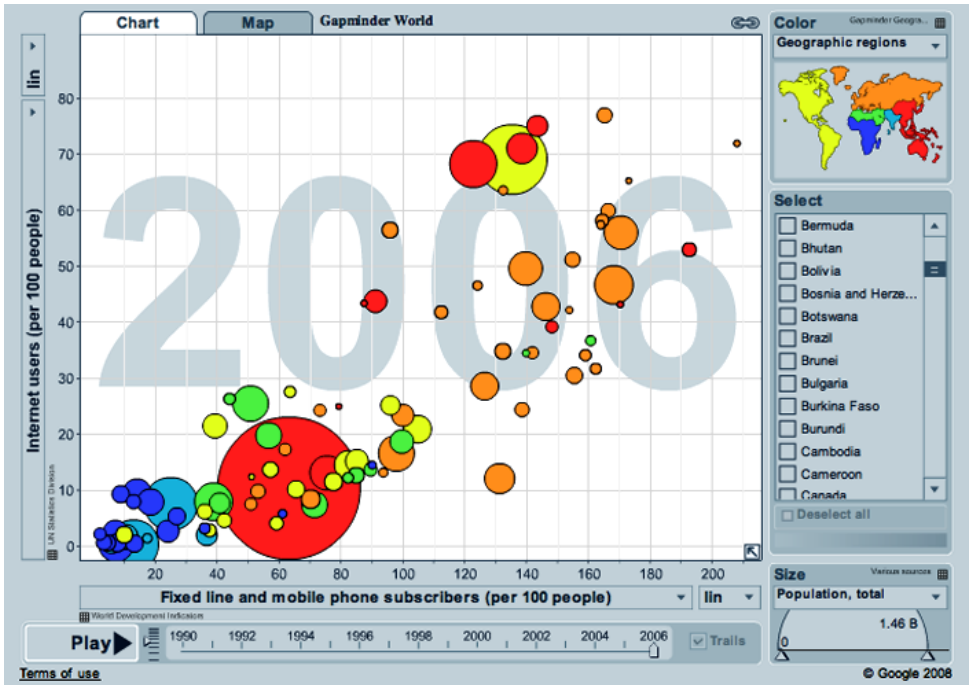
The design process of phone-based interaction systems passes some obligatory phases. First, the user group is regarded, followed by the consideration of motivation, usage frequency, available phone connection and personalization. In the second phase, a visual presentation of the interactions between the system and users is needed to describe the main tasks of the system. After having a rough overview of how the dialog system is going to look like, the design is further refined. Examples of such decisions are barge-in style, audio formatting, synthetic speech or recorded prompts, global commands, human operators accessibility, speaker dependent or speaker independent type of the speech recognition[1], natural language or directed dialog. After these high level design decisions, finest design issues such as prompt design, error recovery, sound and feel, word choice for the prompts are considered. Based on the high and low design decisions, an information flow scheme is drafted to support the following tests and improvements. As a result of the end phase, one dialog script is presented.

4.1 Analyzing users

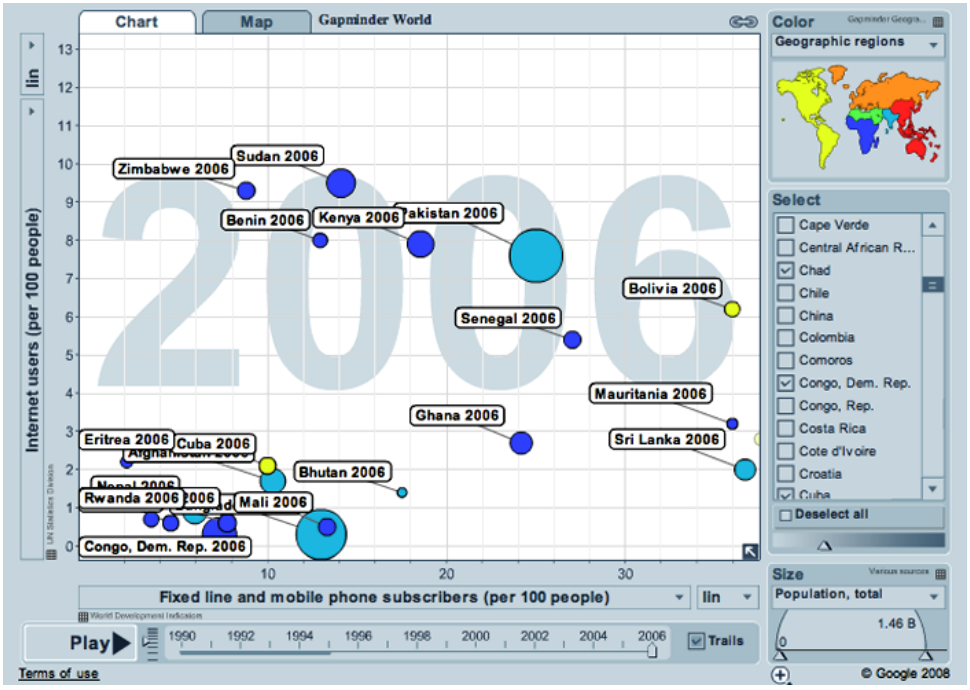
The main purpose of building a conversational system for collecting speech data is the fact, that a lot of countries have a better phone infrastructure than an Internet structure. Such countries are basically the developing countries in Africa and some countries in Asia [26]. Figure 4.1 plots simple statistics about the telephone and Internet usage worldwide. All this information is available online [27]. The first figure proves that worldwide at least twice as many people are using phone connections than Internet ones. Obviously there are countries which have 20% of the population with phone connection, but less than 5% of the population in these countries has Internet access. The blue circles represent some countries mainly located in Africa. To visualize a detailed distribution of these countries, the second figure shows this information using a different scaling.

Our intention is to collect speech data in multiple languages from native speakers. For the communication with the interface, we decided to use English in order to

[1] Only in case we use ASR technology.



(a) Ratio of phone usage compared to Internet usage worldwide. For the visualization linear scaling is used.



(b) Closer view to the ratio of phone usage compared to Internet usage for some of the developing countries. For the visualization linear scaling is used.

Figure 4.1: Ratio of phone usage compared to Internet usage in 2006. Source: http://www.gapminder.org/

reduce the development work. As a consequence, most of the callers will be non-native in English. This fact implies the requirement of building a system which does not require complex answers from the users and has low requirements on listening. The simpler the answers, the more steps are needed to finish a given task. However, we do not count this to be a problem due to the low usage frequency. By "simple" for example answers with "Yes" or "No" are meant as well as answers involving no longer explanations. Additionally, we have the advantage of implementing our system in a telephony platform which gives us the possibility of using DTMF inputs and this way having a guarantee of 100% error free answer recognition. Consider the following example: Using the question "Tell us which languages you speak and at what level?". This question could work well in some situations, but we cannot be sure that it always works[2]. Furthermore the question is not clear enough as the user has no given set of possible answers. He or she could react unpredictably and give answers not handled or not written in the grammar. Two kinds of problems derive from this: First the user may become confused and second our system's stability is in danger. The practice shows that such questions could be easily broken into smaller and more efficient pieces. Doing this is the right approach despite of the greater time consumption.[3] We can reformulate the example question depending on the context either as "What is your native language?"[4] or just make two smaller and simpler questions from it.

As previously mentioned, the majority of the users are supposed to be non-native English speakers. The conversation could be divided into fragments from two languages: The first serves as introduction and guides through the menu of the dialog system. The second represents the collected speech data. There are two languages used: English for the navigation through the system and the native language of the speaker giving an amount of audio data. The collected data should be finally transcribed.

The dialog system should be built stable enough for working in any kind of surrounding area - from noisy streets to home environment. From this requirement derives the need of well-considered global commands and a strategy for making the users familiar with those. A global command can be defined as a word or expression which could be used in every moment of the dialog, stands separately from the local prompt grammar and leads to a specified frame of the dialog flow. Examples of such commands could be the "end of conversation" or "repeat" - commands in cases the user wants to be immediately disconnected from the system or respectively, he or she was not able to understand the previous prompt. The VoiceXML standard, discussed in the Section 5.1.1, offers options for defining such commands.

The phone connection varies from land-line to cellular or even cordless. The bandwidth of the frequencies transmitted in a telephone network is between 300 Hz and $3,500$ Hz. In comparison, the frequency range produced by speaking is between 250 and $8,000$ Hz and the range of the sounds perceivable by a human from 20 to $20,000$ Hz. For detailed information, please refer to [28]. We can conclude that using a telephony interface for such a purpose is not the best solution since we loose

[2]Even worse, we are sure it will not work in most of the cases as such questions are confusing for the system and for the users.

[3]Having the work properly done is always with greater priority than the time and resources put into it.

[4]As it is in our case.

part of the information but it is still an acceptable tradeoff between cost efficiency and work done. The bandwidth where speech can be transmitted is a limiting factor for telephony based speech recognition, however. The standard land-line telephone has a bandwidth of 64 kbit/s at a sampling rate of 8 kHz and 8 bits per sample. Therefore, to integrate a speech recognition engine to work with a telephony application involves training acoustic models with 8 kHz/8 bit speech audio files. For desktop ASR, the current standard are acoustic models trained with speech audio data recorded at sampling rates of 16 kHz/16 bits per sample.

The majority of the system's users interact with the dialog system from one to a couple of times, so that no personalization in terms of login data (username, password or PIN) is required. One more reason for enabling free access to the system is that no personal information, for example bank account or ID card numbers, is needed for accomplishing the users' tasks. Therefore no personalization modules are implemented.

By summarizing the results from the phase, where we analyzed the users, the following design rules derived:

- Simple questions, implying simple answers
- The low usage frequency allows the freedom of more navigational steps
- English used for the navigation
- Introduction for basic global commands
- No personalization

4.2 Analyzing user tasks

The main task of the user is giving speech data. There are a lot of possibilities for doing that per telephone. For example repeating a given text, or reading a given message. The last option appears to be difficult and inappropriate in our case because all possible ways of doing it, for example via letters, SMS, or books are either to slow, or too expensive.

The most natural way of donating speech data seems to be the spontaneous speech in contrast to read speech. This approach brings some advantages like diversity in the data collection or natural intonation. The biggest disadvantage of spontaneous speech is the need for transcription. There is also another disadvantage - the spontaneous speech contains disfluencies and is therefore much more difficult to recognize than speech read from script. Due to the fact that the spontaneous speech is much more spread in everyday life than the read one, it is from importance. Spontaneous speech is a challenge for the speech recognition engines. Furthermore such audio data can be used for improving strategies able to solve problems, which occur only in spontaneous speech but not in read one. This is very important since many systems should be robust enough for dealing with these speech phenomena, for example filled pauses, false starts, hesitations and ungrammatical constructions. Interesting articles about state of the art in human language technology can be found at [29].

We decided to implement two modules with slightly different tasks. The following simple scenarios help us to analyze both possibilities and to visualize how the final system is going to look like.

- In the first scenario read speech is collected. The text to read must be available to the users without using the Internet. Sending short messages to a mobile phone, containing the text to be read seems to be faster. The cost issue, however, remains and the message lengths are limited which means a small speech data amount for each message. Another disadvantage is the limitation of using exclusively a mobile phone connection. If we use the system as a remote recording tool, then the collection of read speech is no problem. We can send the text to be read in a variety of ways including via Internet. We can also point to some paragraphs from a selected book and expect the user to read those. This approach is very comfortable when our team works together with a research or university group in the field. The most important advantage is that we get the transcription of the read text easily.
- The second scenario is based on the idea of recording spontaneous speech. The dialog system offers a list of topics, prompting the user to choose one specific topic from the list. After the selection, the user is asked to answer some questions. The answers are recorded in the database as audio files. Only the topic of the conversation and the meaning of the separate questions is known. What needs to be done next, which is actually the biggest disadvantage of such a method is transcribing all audio files. This step can be partly automated by methods such as unsupervised training [30]. Therefore the success of the whole phase still depends on a human transcriber. It can be partly automated by running a basic speech recognizer to transcribe parts of the spoken texts to make the transcriber's work easier. This approach has the disadvantage of missing transcriptions, but on the other hand, it offers natural sounding speech data.

Table 4.1 describes the use cases of the dialog system.

Having an idea of the main features of our IVR system, we specify the order of the interactions now. The following UML sequence diagram in Figure 4.2 represents the desirable scenario for the user's interactions with the system:

Summarizing the second part of the design process leads to the following decisions:

- The system should be originally designed to collect audio data from spontaneous speech. As the collection of read speech is technically easier, one additional module can be implemented especially for projects. Without appropriate equipment the participants in these projects can use their phones and read a given text.
- The main goal of our telephone-based dialog system is to record speech data on the phone.
- The speaker should have the possibility to choose at least the topic of the dialog. Because there is a finite number of pre-recorded questions, the topics are also predefined.

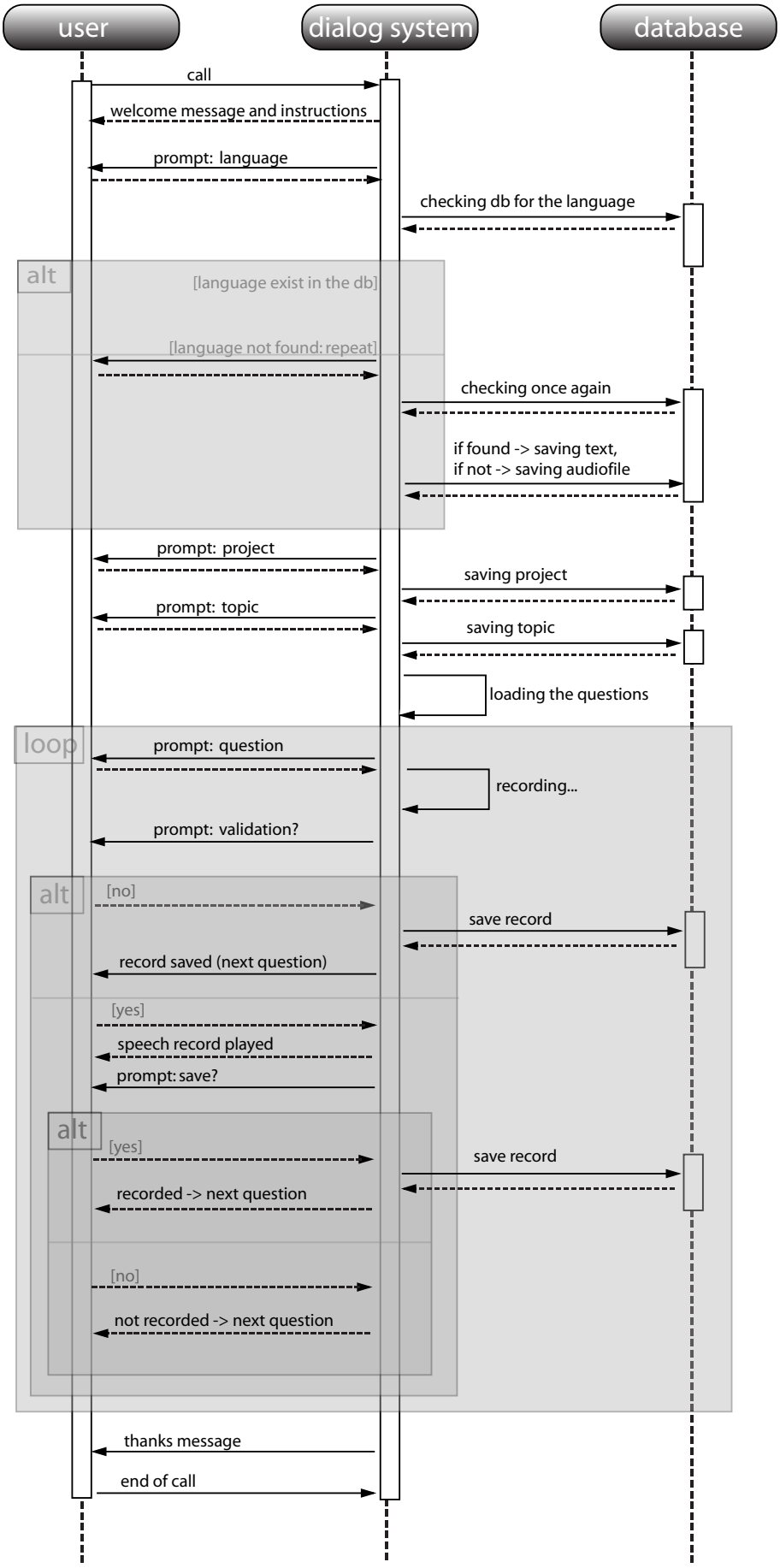


Figure 4.2: Sequence diagram representing the desirable interactions between the dialog system and the user.

4.3 High level design decisions

The first decision to be taken is applicable exclusively in systems with a speech recognition unit and depends strongly on the nature of the user task. It is very important to define the **dialog initiative** which means to answer the following question: Who will lead the conversation and how? There are three possible types of dialog initiatives. In the case of a system-directed dialog, the system controls the dialog flow and despite of the strong advantage of being stable, this way is too restrictive, inflexible and requires a lot of questions, which immediately implies longer and sometimes annoying conversations. On the other hand, the opposite approach of building a user-directed system would not be appropriate due to the characteristics of the user group consisting of non-native English speakers and more importantly, due to the nature of the task: the system wants speech data from the user. The golden middle is offered by a dialog system with a mixed initiative where the advantages of both approaches are used. Instead of simply processing answers, the system should be well-designed for giving feedback to users' questions as well. In systems taking advantage of DTMF the interaction is always led by the system.

dialog initiative

In conclusion, the design of this conversational system should not enable a user initiative. It should be based on a system-directed dialog. There are two reasons for that: First, the non-native speakers target group and second the simple and exactly defined user task for which the system's stability is from great importance.

The second high-level decision is connected with the **barge-in style**. The barge-in in context of speech recognition means the ability for the user to start speaking before system output has ended. This concept is largely used in conversational systems with a mixed initiative. It offers comfort and time saving when applied to frequently used systems. For building a robust dialog system which collects speech data no barge-in is needed. It could even cause problems. Enabled barge-in has the following disadvantage: Background noise may interrupt the conversation. Then the recovery of the system to the current state is not a trivial task. An intelligent solution to similar problems in mixed initiative dialog systems is presented by [31].

barge-in

USER	SYSTEM
can make different choices (for example topic of the conversation)	gives instructions to the user how to use the system
can accept or reject the record	records the speech data
	the user listens to his or her recorded file and decides if it should be deleted or kept (optional)

Table 4.1: Simple use case table

In our case, however, there is no benefit of enabling barge-in. Therefore the task is simplified by disabling this feature.

human operator

The next question involves the decision if a transfer to a **human operator** should be possible. In principle, human operators are very helpful if the task to be solved is critical and should succeed in any case. Such an option is an obligation in dialog systems within health and citizen institutions, fire brigades, bank services and support centers. The biggest disadvantage is higher costs as hiring human operators is expensive. For a non-commercial and not critical conversational system, implementing this possibility is out of question.

recorded prompts

The discussion if synthetic speech or **recorded prompts** should be used is related to the issue of the system's "sound and feel". On the one hand recorded speech sounds more natural and friendly, on the other hand not every prompt could be predicted and respectively recorded. For example, if the system should repeat a part of the user's answer such as the current time, name of a city or another input variable. It is not impossible but complex or time-consuming in most of the cases. There is another disadvantage of prerecorded prompts and it is a flexibility issue. We can set perfect recordings with professional speakers but in the future, if we want to extend our system with additional prompts, we should record everything again[5]. Mixing both alternatives is possible. Therefore the predictable prompts can be recorded and for other parts a TTS synthesizer can be used. We have decided to experiment with both a TTS engine and a set of prerecorded prompts.

speaker dependency

There is one more issue related to systems with a speech recognizer. In case of a **speaker dependent system** the user must provide samples of his or her speech to calibrate it before usage. Such systems are very accurate but not flexible as the recognition rates are very bad for other speakers. Another solution is proposed by speaker independent or adaptive systems where no calibration is needed or is respectively automated. They have the advantage of being appropriate for any speaker and consequently flexible. However, a speaker-independent model is more general but less accurate [11]. Since our system requires minimal ASR capabilities, we do not have to consider a speaker dependent system.

global commands

As mentioned in Chapter 4.2, a decision about all **global accessible commands** should be taken. In our case the specification of these commands is very important since it allows a flexible conversation and an elegant solution for a lot of problems like misunderstanding or noisy environment. One important global command to be implemented is the repetition of a prompt. The possibility of hearing something wrong or misunderstanding the previous prompt is very high, especially because the users are not supposed to be native English speakers. Additionally, conversations in noisy environment require some mechanisms which allow to step back. Thus the first global command to be implemented is a command for repeating the previous prompt. This must be valid especially for cases where instructions are given to the user.

The next issue to be implemented as a global command is a method for providing help information to a user. Another possible solution is to specify a help messages after a given time period without input or with a false input by the user. Such feature is easy to implement in a VoiceXML-based application.

[5]Except for the case when we are able to find the same speaker.

The last command which has to be globally accessible is ”end of conversation”. It redirects the user to the end message. The other possibility to quit the conversation is just quitting.

At the end of this section one more topic remains. We have to decide which **audio format** our systems is going to use. A great variety of audio formats is available. They all use different compression algorithms, rates and have different quality and popularity. RLAT can work with the wav and adc data formats. That is why, the recorded files must be wav or adc decoded. The format we are going to use for the prompts itself does not matter as long as it is compatible with the system and compressed enough to be transmitted in real time through the phone.

audio formatting

Here is a short summary of our high level design decisions:

- System directed conversation
- Barge-In disabled
- No option for transfer to a human operator given
- Recorded prompts, TTS only if required
- Global command for ”help”

4.4 Low level design decisions

The next design step involves taking decisions of how our interactive system should roughly look like.

The **”sound and feel”** of an interactive telephony application depends on the style of the prompts' construction, on the specific words chosen and on the speaker herself. The best choice for us is avoiding dialectic and slang phrases and let the system speak freely with a friendly and calm intonation.

"sound and feel"

In addition to the previously mentioned limitations, the **word choice** in our case has one more limitation. Knowing the target user group consisting of non-native English speakers, it is a good decision avoiding more sophisticated English words. Often they can be easily replaced with frequently used and well known synonyms.

word choice

A very good impression remains when a **consistent prompt design** is followed. This means every prompt must be spoken from the same person with the same volume level, using similar sentence constructions and carefully chosen phrases, compatible in speech style.

consistency

4.5 Dialog flow

The interaction flow of our application is a result of a constant loop through the following actions: designing - testing - improving. More details are described in Section 5.2.4. The information we want to get from the user in order to classify his or her audio data properly is:

- Spoken language - the native language of the caller.

- Project participation (optional) - if the caller participates in some specific project where our university, a partner university or an organization is involved in. In this case, the caller is aware of his or her task and knows the name of the project. This option makes the sorting of the recordings easier and is especially designed to improve the cooperation with another research groups.
- Topic for answering questions - the domain where the asked question should derive from. This information plays an important role since it controls the dialog flow. It also allows the user to choose a topic such that he or she would like to talk about.

4.6 Summary

In this chapter we took a lot of important decisions. These support the following implementation. After the design phase, we should be able to answer how we imagine the main functionality of our system, what technique we intend to use and how these techniques help us to achieve our goals.

5. Implementation

5.1 Existing dialog system software tools

In the following subsections, we present an overview of some existing software solutions for dialog systems, beginning with the commercial ones and concluding with open source projects. Most of them are parts of bigger solutions. However, it is important to take a look at them and to point out their advantages and disadvantages as they offer very different implementations for the same problem. Section 5.2.1 explains the reasons for our choice of software tools. In Section 5.2 the development process of our telephone-based dialog system is described.

5.1.1 The IBM speech architecture

The solution offered by IBM is an almost complete set of software tools for building dialog systems. This means that the developer's task is to integrate all components of the system and to design the conversation flow. The components are independent from each other but designed to interact easily with each other. In the tested configuration, we used Websphere Voice Server 5.1.3.3. with speech recognition and synthesis for German language.

Figure 5.1.1 shows the architecture of the IBM dialog system solution.

The user connects to the voice browser via SIP compatible phones. An SIP phone is a hardware or software based SIP client which provides call functions such as dial, answer, reject or hold. A softphone is a good example for a client software. The main job of a softphone is making calls over Internet which are usually free of charge. If somebody intends to establish a PC-to-phone or phone-to-PC conversation[1], it is not free anymore, but this topic is discussed in Chapter 5.1.4.

The IBM voice browser is called Voice Enabler. It accesses a voiceXML file which defines the control flow of the dialog. In the voiceXML document, we design the controls of our interface between the dialog system and the user. On the other hand the Voice Enabler serves like a communicator between the softphone (via SIP)

[1] By term "phone" we mean the classical PSTN-connected device.

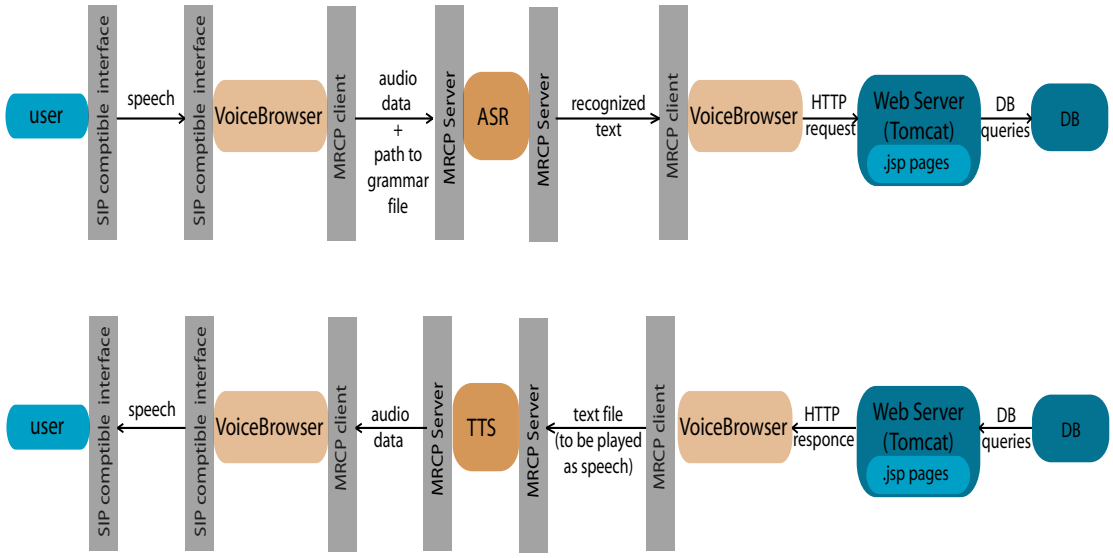


Figure 5.1: Message flow between components in the IBM dialog system solution.

and the speech recognizer or speech synthesizer. To configure the Voice Enabler, we modify our MRCP settings for the IBM recognizer and synthesizer which are accessible under the following URIs:

```
rtsp://< host-ip-address>/media/recognizer
rtsp://< host-ip-address>/media/synthesizer
```

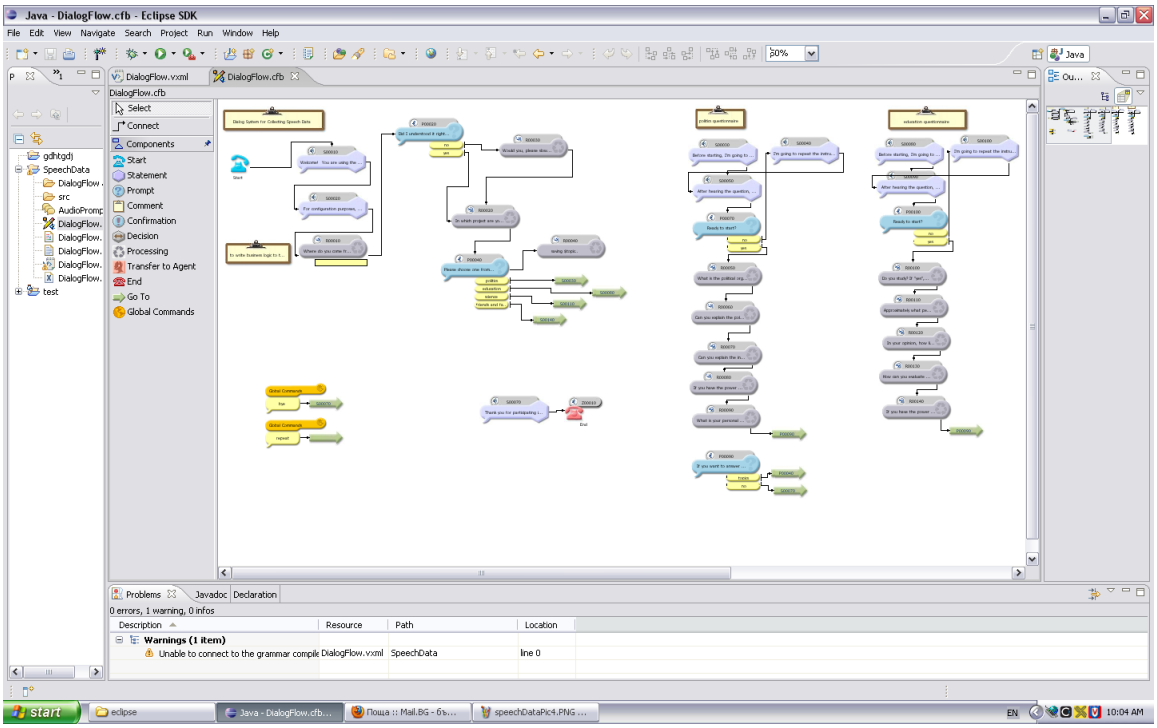
The IBM Voice Enabler is a J2EE[2] component, packaged as an enterprise archive (EAR) and hosted on the Websphere Application Server.

One part of the IBM solution is the Websphere Voice Server. It contains a speech recognizer and a synthesizer for different languages, for example English and German. The Voice Server can be installed on Linux or Windows platforms. We tested one version of the server, embedded in Cent OS, running as a virtual machine on a Windows XP host platform. This version is easier to configure than the version not embedded in a Cent OS and contains the Websphere Application Server installed on the guest Linux platform. The properties of both servers can be reviewed and changed in a browser. They are accessible after configuring the network connection between the guest and the host systems.

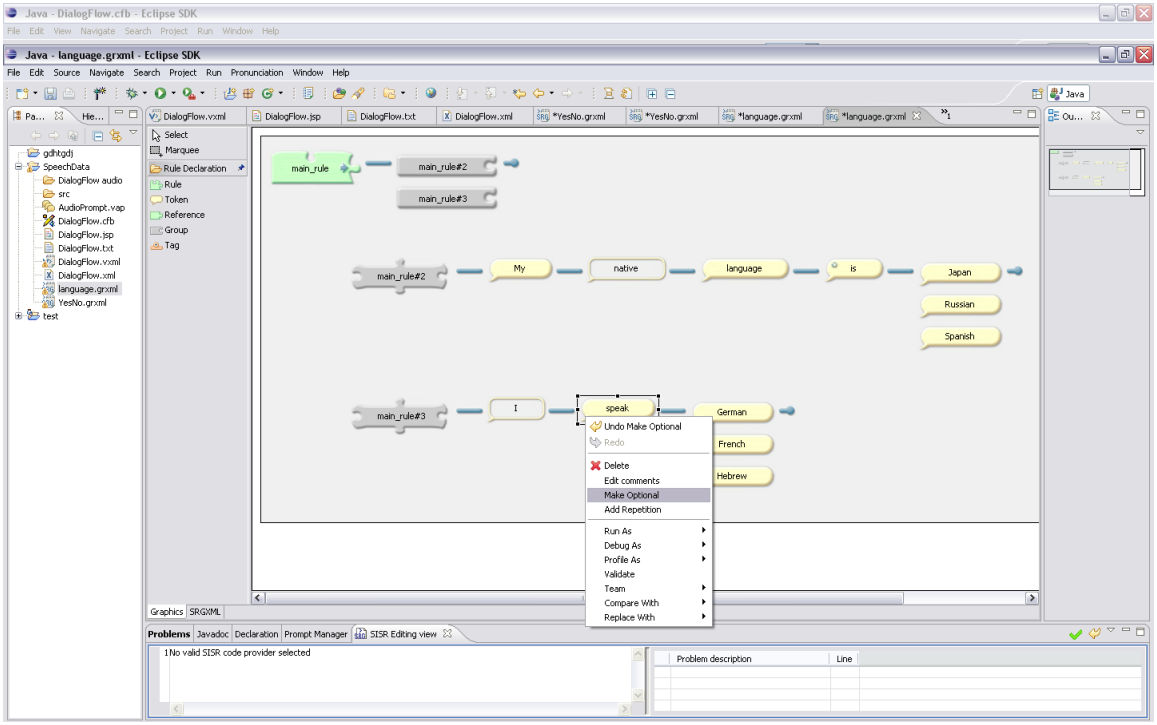
There are many free tools for designing voiceXML documents. IBM obtain their own tool which is a plugin for Eclipse called Websphere Voice Toolkit. It offers a graphical user interface (GUI) for the design of voiceXML documents and tools for designing grammars. The GUI editors for voiceXML call flows and for creating XML-based SRGS files are shown in Figure 5.2.

Unfortunately, users have to install the whole Websphere Eclipse SDK since the plugin is embedded in the version of Eclipse modified by IBM. Furthermore, it is

[2]J2EE refers to Java Platform, Enterprise Edition, widely used server platform for multi-tier Java applications.



(a) The GUI editor for voiceXML-based call flows and dialog simulation.



(b) The GUI editor for creating XML-based SRGS grammar files.

Figure 5.2: Overview of the IBM Websphere Voice Toolkit plug-in for Eclipse. The developer can edit XML-files either through the GUI or direct in the XML-format.

exclusively available for Windows platforms. Developers can not rely on the Voice Toolkit to design the control flow of the application completely as the GUI of the toolkit does not cover all available voiceXML functions. Finally additional handling of the XML file is necessary. However, the tool is a good starting point, especially for complex applications with a lot of dialog branches.

This solution is simple but not free (except for the Voice Toolkit plug-in for Eclipse). Buying a license from IBM is an option for those who want to use the offered speech framework. The license type depends on the requested configuration. Support is included.

5.1.2 Microsoft Speech Tools

The second commercial solution is the Microsoft Speech Application SDK 1.1 [32]. It requires installing the .NET framework and Microsoft .NET Speech Server (2004 R2). For the test, we downloaded the software from the university's MSDNAA (Microsoft Developer Network Academic Alliance) system. The role of a voice browser is played by the Internet Explorer 6.0 tuned with the free Speech Add-in. After creating the speech application it is hosted on the Microsoft application server known as Internet Information Services (IIS). Instead of using voiceXML documents, the Microsoft solution works with SALT (Speech Application Language Tags). For the SALT specification, please refer to [33]. Designing a dialog flow with SALT is easy since the Microsoft Speech Application SDK offers GUI editors for SALT documents as well as a lot of built-in grammars for currency, dates, phone numbers, etc. The Microsoft speech framework includes own speech recognizers and synthesizers.

5.1.3 The CMU OpenVXI voice browser

Another solution for a voice browser is the open source voiceXML interpreter from Carnegie Mellon University (CMU) [34]. This speech browser uses third party open source libraries such as Apache Xerces XML parser, OpenSSL and the Mozilla SpiderMonkey ECMAScript interpreter. Although it is only one component of a complete voiceXML platform, it is a huge timesaver for developers in comparison to coding the components integrating platform from scratch.

Unfortunately, the OpenVXI browser has a lot of disadvantages. The last version of the browser (OpenVXI 3.0) dates from 2003 which means that there are no upgrades since 2003. The third party libraries necessary for compiling the browser's code have to be in their old versions as well. Trying to compile the browser with the latest versions of Apache Xerces 3.0.1 and the Mozilla SpiderMonkey 1.7.0 failed as some functions used in OpenVXI have already been deprecated.

Compiling OpenVXI with old versions of the third party software succeeded but future problems with browser's usage could be expected.

Another drawback of this browser is the documentation. The Platform Integration manual provided with the open browser refers to version 3.0. It seems very detailed. Unfortunately, unpacking the downloaded archive with the browser showed that most of the described examples and tests are missing. They are present in the archive of the previous browser version (version 2.0.1) but some other issues such as interface names differ.

The open speech browser is shipped without speech recognizer. It includes simulated speech recognition support, prompt and TTS capabilities as well as telephony functions. The users of the framework are responsible for providing the integration to the actual components. However, there are interfaces specified to support the integration of speech engines and telephony devices.

The browser is assembled from C and C++ components. The VXIrec API provides functions for integrating a speech recognizer. VXItel API is responsible for the telephony integration. Similarly the VXInet API provides access to application documents via HTTP. There is also a logging API helpful in the development phase for debugging.

5.1.4 The JVoiceXML voice browser

The next tested solution is a system based on the open source voice browser called JVoiceXML which is a Java based engine able to interpret voiceXML 2.0 standard documents [35]. There are many advantages offered by this solution. The latest project's version is from August 2009 which makes the browser up to date with the recent technologies. Also, there are developers in the JVoiceXML project interested in completing the missing browser's functions. On the other hand, similar to every newly developed product, the JVoiceXML browser is not very mature and continuously transformed in the process of improving and bug-fixing.

JVoiceXML is written in Java which makes it consistent and platform independent. In practice, the platform independency is valid only for operating systems supporting Java 6 runtime environment (JRE) since the browser implementation uses some structures from JRE 6. The latest Linux, Windows and Mac OS platforms can compile the source code but for older operating systems the user should use an older version of the browser.[3]

The JVoiceXML browser comes with third party libraries packed as java archives and added to the classpath. This way the browser includes the open source speech recognizer Sphinx 4 and the FreeTTS engine [36]. Both engines communicate with the browser through Java Speech API (JSAPI) which is a cross platform API used for controlling and supporting speech recognizers, speech synthesizers and dictation systems [37]. This API consists of three packages: The `javax.speech` package represents the generalization of the main concepts. Examples for such concepts are the classes `Engine` as a superinterface for the `Recognizer` and `Synthesizer` interfaces or `SpeechEvent` as a superclass of `AudioEvent`, `EngineEvent`, `GrammarEvent`, `ResultEvent`, `SpeakableEvent` etc. The `javax.speech.recognition` package supports the usage of recognition grammars and speech recognizer integration. Finally, the `javax.speech.sythesis` package contains all classes necessary for successful speech synthesizer integration.

Similarly to the architecture of the IBM solution, JVoiceXML uses two libraries supporting the MRCP communication. `Cairo-client.jar` is an open source speech client library, which uses MRCPv2 to communicate with an MRCPv2 compliant speech resource server. The second library `cairo-sip.jar` provides a simple Java API for supporting SIP/SDP message communication between MRCPv2 clients and servers.

[3] Unless you use some tricks to get JRE 6 running on your older platform.

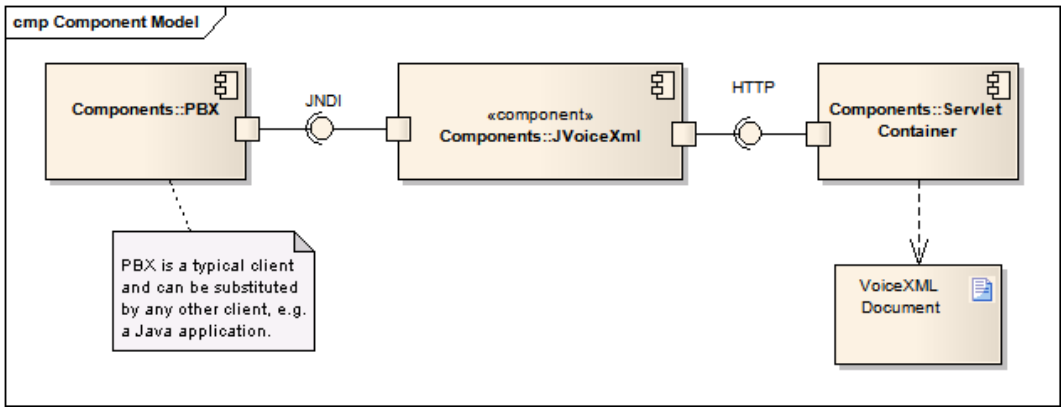


Figure 5.3: Architecture of the JVoiceXML speech browser (excerpt from the JVoiceXML User's Guide [38])

To understand the interaction between the JVoiceXML components, please refer to Figure 5.3. The client application accesses the JVoiceXML browser through Java Naming and Directory Interface (JNDI). JNDI is an API allowing Java software clients to discover and search data referring to it by name. It is used by the Java Remote Method Invocation (RMI) API to invoke remote procedure calls. The above mentioned APIs allow us to have access to the JVoiceXML browser functions from a client applications. A typical client is PBX or any Java client application equipped to make the connection with JVoiceXML.

JVoiceXML runs as a standalone server and retrieves documents via HTTP from the servlet container, for example from Apache Tomcat. Usually voiceXML documents and grammar files are requested that way.

To get a better idea of the big picture of one telephone-based IVR system, see Figure 2.1. The switching between the PSTN and the VoIP network can happen in several ways: Either by using an own hardware device (VoIP gateway) in combination with IP PBX or by using Virtual PBX which means paying for a service from a telephony service provider. In the first case, we have to connect the physical device with an IP PBX like Asterisk [39] or 3CX [40]. Asterisk IP PBX runs on Linux platforms and can be used as a switch for IP or hybrid PBXes, or as a media gateway bridging the PSTN with IP telephony working together. The analogue software of Asterisk for a Windows platform is called 3CX. The 3CX Phone System supports calls via existing phone lines using VoIP gateways. The VoIP gateways transform audio data from phone calls to digital data. Therefore they are used to convert incoming PSTN lines to VoIP. There are two types of VoIP gateways - analog and digital.

The JVoiceXML user guide is not very detailed so far, even if there is a large documentation describing the API functions. However, there is a good community support in the developers' forum.

5.1.5 Zanzibar Open IVR

Now we will have a look at Zanzibar Open IVR [41]. Zanzibar Open IVR v0.1 uses JVoiceXML as a built-in interpreter. The structure of Zanzibar Open IVR is

visualized in Figure 5.4. It consists of two parts communicating with each other: The `MRCPv2 server` (Cairo) and the `Speech application server` (Zanzibar). The `MRCPv2 server` itself is a platform combining three components. The first one called `Resource server` manages the client connections with the speech recognition and synthesis functions of Zanzibar. For example, supposing our client is an Asterisk application, then Asterisk builds a connection with the `Resource server` through SIP. After successful established SIP connection, the audio streaming is regulated by RTP. Then the audio data is processed by the `Receiver resource` which is actually the speech recognition module. Zanzibar uses Sphinx4 ASR to transcribe audio data into text. In the other direction we have the `Transmitter resource` responsible for the audio data generation which should be streamed back to the user. For this purpose Zanzibar uses the FreeTTS speech synthesizer. Finally, to run the Cairo server, we should run these three components separately beginning with the `Resource server` (rserver.sh).

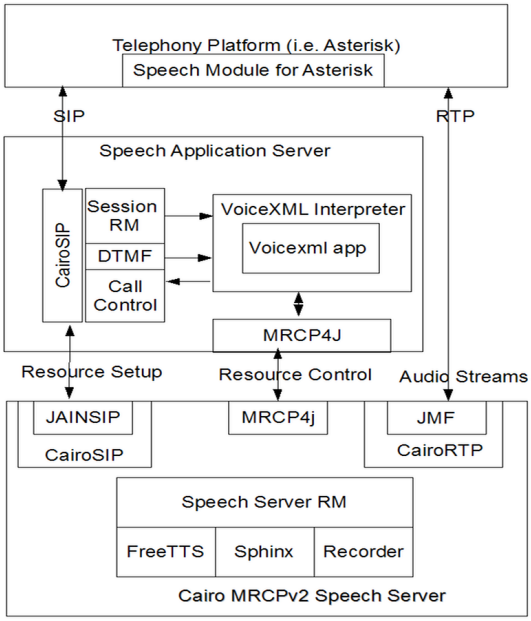


Figure 5.4: Architecture of the Zanzibar Open IVR engine (excerpt from the Zanzibar project's website [41])

The second component of Zanzibar Open IVR is the `Speech Application server` known as Zanzibar. The core tasks of this server are initializing sessions and running applications for the caller. Part of this server is the voiceXML interpreter, discussed in Section 5.1.4. As the call control function is executed by a voiceXML document stored on the application server, we may regard the application server as a module in this framework. The `Speech Application server` is configurable through a Spring configuration file where different application functions are declared as Java beans.

We find the integrity of Zanzibar with Asterisk important, relevant for our work and offering a lot of possibilities for the system development and improvement for the future. With a running Asterisk server, users can experiment with Zanzibar Open IVR.

5.1.6 Summary

In conclusion we show a compact view of the properties offered by the different systems we tested. In Table 5.1 we present the most important systems' features.

	IBM Speech engine	Microsoft Speech SDK	OpenVXI	JVoiceXML	Zanzibar	Asterisk
free of charge	no	no	yes	yes	yes	yes
open source	no	no	yes	yes	yes	yes
platform	linux,win	win	all	all	all	linux
TTS	✓	✓	✗	✓	✓	✓
ASR	✓	✓	✗	✓	✓	✗
voiceXML interpreter	✓	✗	✓	✓	✓	✗
dialog logic language	VXML	SALT	VXML	VXML	VXML	AEL
application server	✓	✓	✗	✗	✗	✗
SIP compatible	yes	yes	yes	yes	yes	yes
MRCP interface	yes	no	yes	yes	yes	no
integration easiness	⊕⊕	⊕⊕	⊖⊖	⊖⊖	⊖	⊕
programming language	no info	.NET	C, C++	Java	Java	C
documentation	⊕⊕	⊕⊕	⊖⊖	⊖	⊖	⊕
community support	close source	close source	⊖	⊕⊕	⊖	⊕⊕
timestamp	new	new	old	new	new	new

Table 5.1: Comparison of the tested solutions.

5.2 Our approach

After analyzing different solutions for our problem, we focused on the Asterisk open source PBX. The reasons for choosing exactly this tool are given in the following subsection. We describe all software tools used for the final implementation as well as the underlying hardware configuration of the test system. Then we continue with the configuration and actual dialog flow implementation of our telephone-based IVR system.

5.2.1 Asterisk PBX

Asterisk is an open source telephony project developed by Mark Spencer at *Digium*. The application comes as a result of the rapidly growing demand for a tool, which connects the proprietary telephony systems with the powerful concept of VoIP. At the beginning of the 21^{st} century, the Internet technologies were mature enough to offer more functions and flexibility than the existing telephony hardware. The interested reader can find more about the history and development of Asterisk in the book "Asterisk: the future of telephony" [28].

Asterisk can perform a lot of functions: It can be used as a switch for IP or hybrid PBXes allowing purely IP based communication. It can also play the role of a gateway bridging the PSTN with IP telephony which is actually the main reason Asterisk is used for. Additionally, Asterisk can perform as a media server giving us the possibility to implement a lot of IVR functions. Due to its open source nature and high popularity, this application can be easily integrated with almost every application including the Zanzibar Open IVR engine.

Our reasons for choosing Asterisk as a backbone of our application are many: First of all, the tool is open source. There are no chances for Asterisk to change its way of development and continue as a commercial application. There are commercial versions of Asterisk but they are not of interest to us since this would be in a conflict with the requirements we defined.

Another reason for choosing Asterisk is that this engine runs on UNIX platforms. A free tool running on a free platform which implies consistency with the system requirements and no costs for the system itself. Besides, our RLAT engine is also installed on an Open Suse which makes Asterisk consistent and easy to integrate.

Our choice was also motivated by the fact that Asterisk offers a large amount of possibilities for using pre-installed functions as well as adding new ones. It supports the integration with a lot of other open source applications like databases (Postgre, MySQL, etc.). Also those applications which do not have a built-in integration support can be easily connected via scripts in the programming language preferred by the user. One important issue for us is the fact that Asterisk does not need additional software for audio recording. All software needed is present in the shipped version.

Asterisk has a large community active in the VoIP forums where Asterisk-related problems are discussed. If the answer is still missing, there are some helpful books and online tutorials such as [28] and [42].

Asterisk strengthened its position as the most popular PBX software in the world of telephony. It became a standard and invented new standards in the field of VoIP[4]. Asterisk is compatible to all wide-spread audio formats, telephony interfaces and protocols. The developer can choose among them according to the requirements and his or her personal preferences.

Other pros are reliability and simplicity of usage, maintainability and applications development. According to [39], Asterisk is used by a lot of big corporations for their internal office telephone structure. It is applied in banks, online stores and other customer services programs which proves Asterisk's reliability.

5.2.2 Installation

Hardware requirements for Asterisk vary depending on the main purpose of the target system. For small office systems with up to 10 channels, they are set to 1GHz x86 CPU with 512 MB RAM. To install Asterisk, no dedicated system is required. Our test system uses a PC with Intel Core 2 Duo CPU at 2.33 GHz which is more than Asterisk needs.

The installation of Asterisk is straight-forward. We installed version 1.4 as it is stable and wide spread. Due to the backward compatibility of Asterisk, an upgrade to version to 1.6 is no problem.

There are only two issues to be careful with when installing Asterisk. First, we have to decide which modules are important for us and select them from the `menuselect` list in the bash. For example, if we plan to use a database, then we should select `crd_pgsql` (for PostgreSQL) or `cdr_sqlite` (for SQLite) options in the "Call Detail Recording" menu[5]. Careful inspection of all other items from the `menuselect` can save time and reinstallation in the future. The second issue is that root rights for the underlying Linux system are required. This is necessary not only for the installation but also later when the system is configured and managed. If you do not have root rights for the underlying operating system, you should have admin privileges for all folders and files used by Asterisk. Unfortunately, those are not stored only in one place but spread over the whole system. Thus, it is helpful to contact a network and system administrator. In [28] you can find a list with all folders and items used by Asterisk.

5.2.3 Configuration

As discussed, there are two ways of bringing the PSTN and VoIP together: Either through dedicated hardware or by using the services of a VoIP provider. Both methods have advantages and disadvantages described in the following paragraphs.

We use the services of www.sipgate.de [43]. After the registration we were able to choose among different telephone numbers. The created account is now associated with the telephone number 18030681 valid in the city of Karlsruhe. To access our

[4]However, this statement does not mean that Asterisk tries to introduce new standards to achieve monopoly. These new standards offer better performance and simplifications for the Asterisk's community.

[5]Although Asterisk comes equipped with a simple database called AstDB, if our application needs more sophisticated database functions, we should install these packages. They are not selected by default for installing.

IVR system, the user dials the telephone code of Germany (+49) and Karlsruhe (721). The complete number is: +4972118030681. The costs per minute vary for the different countries. You can find the complete list with the call tariffs on [44].

The Asterisk configuration file responsible for the management of the SIP channels is called `sip.conf`. In Figure 5.5 we show the relevant code snippet of this file. There all connections using SIP to communicate with the Asterisk server have to be defined. In our case, there are two channels used for receiving and sending messages to Sipgate.

Each configuration file in Asterisk consists of different sections which are called "contexts". There are three contexts defined in `sip.conf`: The [general] context sets some global values for all channels. We receive telephone calls through the channel [3576378] and send a call invitation through the channel [sipgate-out]. USER and PASSWORD are placeholders for the real username and the password given to us by Sipgate. Since our interactive system does not make calls, we are not interested in the [sipgate-out] context but define it for possible usage in the future.

We discuss only the fields which are important for our case. The $dtmfmode$ variable is set to $rfc2833$ which means that out of band DTMF is used. This configuration is specific and the decision what kind of DTMF signaling must be used depends on the telephony service provider or the utilized hardware. The *context* variable points to a context in `extensions.conf`. All incoming calls on this channel will be redirected to this context. The *nat* field helps to get through the NAT firewall of the local network. If set to *yes*, Asterisk assumes that the peer may be behind a NAT. In two paragraphs, we return to this point to explain the possible difficulties and how to cope with them.

Using a telephony service provider is financially and technically easier, but there are some difficulties connected with this approach. The firewall problems are common issues for a lot of Asterisk users. Our university network is protected by a firewall. This is the second firewall that the voice packets from and to Asterisk have to overcome as the first one is our local network firewall at the Cognitive Systems Lab (CSL). We overcome the CSL firewall by defining $nat = yes$ in the `sip.conf file`. The university firewall, however, caused some problems: At the beginning we were neither able to receive audio packages nor to hear the played prerecorded prompts. We were able to send SIP invitations and to get package exchange confirmation but all transmitted audio packages had a size of 0 bytes. Although Asterisk offers a lot of debugging functions, it took time to figure out that all audio packages were stopped by the university firewall. A similar problem is described on [45].

To explain what the problem with the second firewall is, we will first explain more details about the SIP session. SIP uses other protocols to transmit media files between the SIP peers. The RTP protocol mentioned in Section 2.1.2 takes care of the media files transmission. To solve the problem with the missing audio packages, we opened the ports where these packages are expected. The default RTP addresses lie between 5,000 and 31,000 but opening so many ports may arise security problems. To solve the problem, we had to open some ports in the university firewall and use exclusively ports between 10,000 and 10,010. This is illustrated in the `rtp.conf` file:

```
[general]
srvlookup = yes
port = 5060
bindaddr = 0.0.0.0
register => USER:PASSWORD@sipgate.de/USER
context = speechData
qualify = no
disallow = all
allow = gsm
allow = alaw
allow = ulaw
allow = g729
srvlookup = yes
dtmfmode = rfc2833

[sipgate-out]
type = friend
insecure= invite
nat = yes
username = USER
fromuser = USER
fromdomain = sipgate.de
secret = PASSWORD
host = sipgate.de
qualify = yes
canreinvite = no
dtmfmode = rfc2833
context = from-sipgate

[3576378]
type = friend
context = speechData
fromuser = USER
username = USER
authuser = USER
secret = PASSWORD
host = sipgate.de
fromdomain = sipgate.de
insecure = port,invite
qualify = yes
canreinvite = no
nat = yes
dtmfmode = rfc2833
```

Figure 5.5: The SIP configuration file: sip.conf

```
[general]

rtpstart=10001
rtpend=10010
```

Figure 5.6: The RTP configuration file: rtp.conf

The reader may ask why we would not have these problems when using a private VoIP gateway. The answer: One VoIP gateway bridges the PSTN with VoIP directly, which means, we get the signal from the PSTN transformed to a VoIP signal in our network. By using a hosted solution, this transformation happens at the VoIP service provider (Sipgate in our case). The connection between the provider and our system goes completely via Internet and the route of all packages includes more networks.

The module loader configuration file `modules.conf` shown in Figure 5.7 is used to define which modules should be loaded and which not. An interesting line in this configuration file is *load => chan_alsa.so*. We can load either ALSA (Advanced Linux Sound Architecture) sound card drivers or OSS (Open Sound System) drivers but not both. Exclusively one of them should be loaded, the other must be defined with *noload*.

```
[modules]

autoload = yes
noload => pbx_gtkconsole.so
load => res_musiconhold.so
load => chan_alsa.so
noload => chan_oss.so
```

Figure 5.7: The module loader configuration file: modules.conf

The last configuration file which we explain is `festival.conf`. It is used to configure the open source TTS engine Festival which uses parametric speech synthesis as described in [46]. Festival is not a part of Asterisk but it can be easily integrated with the open source PBX engine as described in the following lines:

```
[general]

host = localhost
port = 1314
festivalcommand=(tts_textasterisk "%s" 'file)(quit)\n
```

Figure 5.8: The Festival configuration file: festival.conf

The Festival server should be installed separately. If we intend to use speech synthesis, we should start the server this way first:

```
festival --server
```

There is a special function *Festival*() available for use in the dialplan defined by the `extensions.conf` file. The term ”dialplan” has the same meaning as ”dialog flow” but it is used exclusively in Asterisk. We discuss this issue the next section.

Having a configured and running Asterisk system, we take a look at the implementation of the IVR module. Figure 5.9 visualizes and summarizes everything which has been done so far.

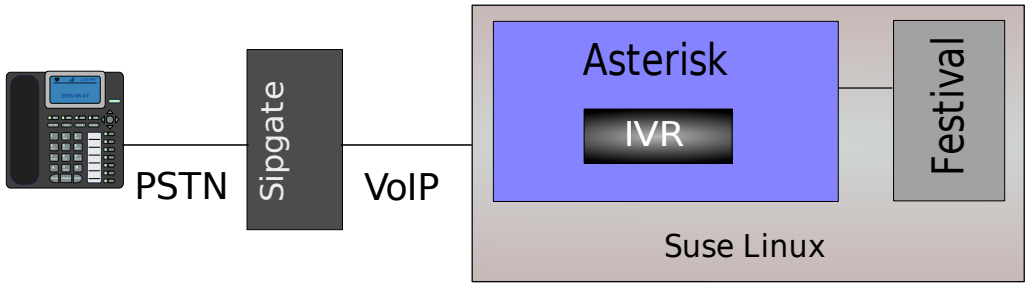


Figure 5.9: Overview of our system.

5.2.4 Dialplan implementation

Asterisk runs as a server which can be started and stopped in the Linux terminal with root rights. To start the Asterisk server, type in the terminal:

```
sudo asterisk
```

To get access to the Asterisk console known as Asterisk CLI (Command Line Interface), the user must specify the grade of verbosity. For example:

```
asterisk -vvvvr
```

This command means that the verbosity is set to 4 (4 times letter ”v”) and the letter ”r” starts the CLI console connected to an Asterisk instance already running as a background daemon on the machine.

With the Asterisk console you can check the configuration files, see the output from debugging or monitor the call. After some changes in the dialplan or in the SIP configuration, a reload of the modified components is needed. The dialplan can be refreshed with the following command:

```
dialplan reload
```

If you are interested in more CLI commands, get more information about the possible actions with the *help* command in the Asterisk CLI.

In the `extensions.conf` file, we define the actions to be taken after a call is placed. This file controls the call flow. Normally all configuration files are located in the directory *\etc\asterisk*. The content of the `extension.conf` is divided into different contexts. As declared in Section 4.2, we may offer two possibilities of collecting speech data: One for recording read speech and one for spontaneous speech. These possibilities can be separated into two different contexts: [recordInterfaceTTS] and [speechDataTTS]. Once a user calls, Asterisk gives the user the possibility to choose either to read something or to answer our questions. The breakdown of the dialog flow into two main parts can be traced in Figure 5.10. The numbers present DTMF keys.

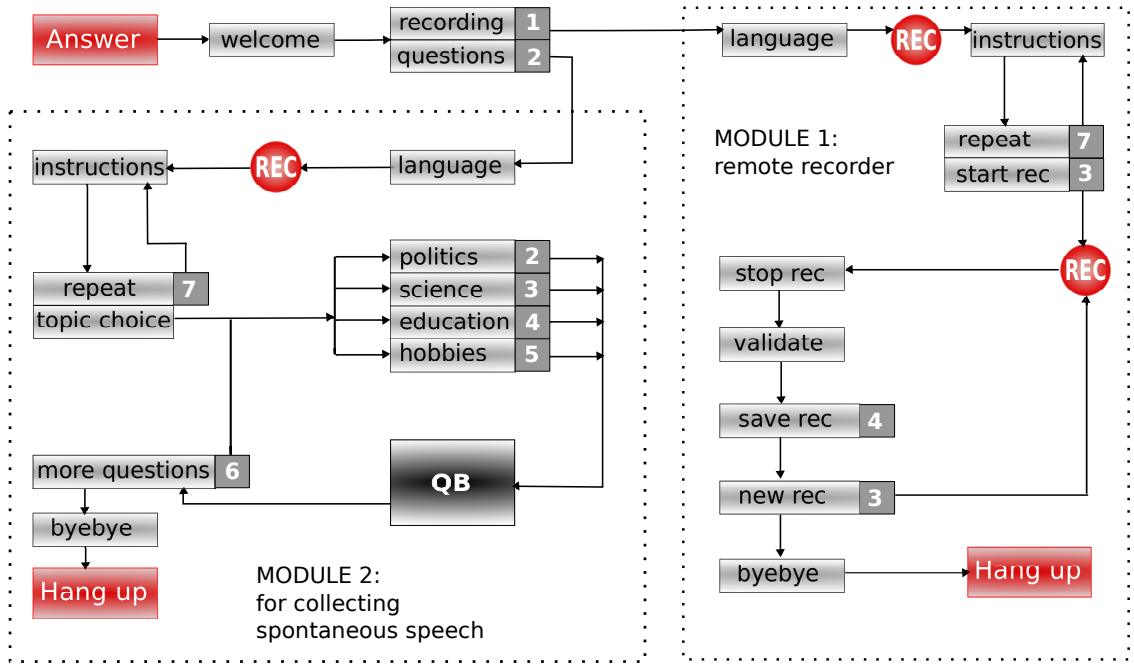


Figure 5.10: Dialog flow outlines.

For our IVR system, we defined three contexts in which we specify the dialog flow. The context which is executed as entry point when Asterisk receives a call through the [3576378] channel is [menuTTS]. Here we present the two options to the user, either to read a prepared text or to answer some questions. If the user chooses to read a text, then he or she has to press 1. The other possibility can be reached by pressing the DTMF key 2.

The first module we implemented is a plain recording tool for remote collection of read speech. Its implementation is easy as just a recorder is needed and it is important to know which language the user speaks in order to easily classify the recorded texts. For the prompts, we used the voice of Daniel[6] from the Nuance RealSpeak TTS engine [47] as it sounded more natural to us.

[6]British English

All files recorded in a single call are saved to a specially created directory. To guarantee a unique name for the newly created folder, we used the Asterisk's variable ${UNIQUEID}. To explain what happens, we refer to Figure 5.11. Each call gets a unique identifier assigned from Asterisk. The ${UNIQUEID} variable contains the unique identifier of the current call. Therefore, we use it to create a new folder at the beginning of the call and then we collect all recordings made in this call there.

```
exten => 1,n,System(/bin/mkdir
/var/lib/asterisk/sounds/SpeechData/records/speaker${UNIQUEID})
exten => 1,n,Playback(danielLanguage)
exten => 1,n,Record(myLanguage:wav,2,10)
exten => 1,n,Playback(danielYourLanguage)
exten => 1,n,Playback(myLanguage)
exten => 1,n,System(/bin/mv /var/lib/asterisk/sounds/myLanguage.wav
/var/lib/asterisk/sounds/SpeechData/records/speaker${UNIQUEID}/
myLanguage.wav)
```

Figure 5.11: A code snippet from the [recordInterfaceTTS] context in extensions.conf

Our system allows the recording of multiple audio files containing read speech by taking advantage of the ${STRFTIME(${EPOCH},,%Y%m%d-%H%M%S)} construct as shown in Figure 5.12. First the user is prompted to make a record which the system saves as `record.wav` in the Asterisk's folder for sounds `/var/lib/asterisk/sounds`. The maximal length of the record is set to 1,200 seconds. If the system detects silence for more than 4 seconds, the recording is automatically stopped. We observed that a timeout of 4 seconds works well for the experienced and unexperienced users. In the second step, the user is given the possibility to hear what he or she has recorded. If the result is satisfactory, the user can save the record by pressing 4. This action copies the recorded audio file in the previously created user's folder and changes the file's name by adding the timestamp.

```
exten => 3,n,Playback(danielReadAfterSignal)
exten => 3,n,Playback(danielStopInstructions)
exten => 3,n,Record(record:wav,4,1200)
exten => 3,n,Playback(danielYourRecord)
exten => 3,n,Playback(record)
exten => 3,n,Background(danielSaveRecord)
exten => 3,n,Background(silence/5)

exten => 4,1,System(/bin/mv /var/lib/asterisk/sounds/record.wav
/var/lib/asterisk/sounds/SpeechData/records/speaker${UNIQUEID}/
record${STRFTIME(${EPOCH},,\%Y\%m\%d-\%H\%M\%S)}.wav)
exten => 4,n,Playback(danielSuccessfullySaved)
```

Figure 5.12: A code snippet from the [recordInterfaceTTS] context in extensions.conf

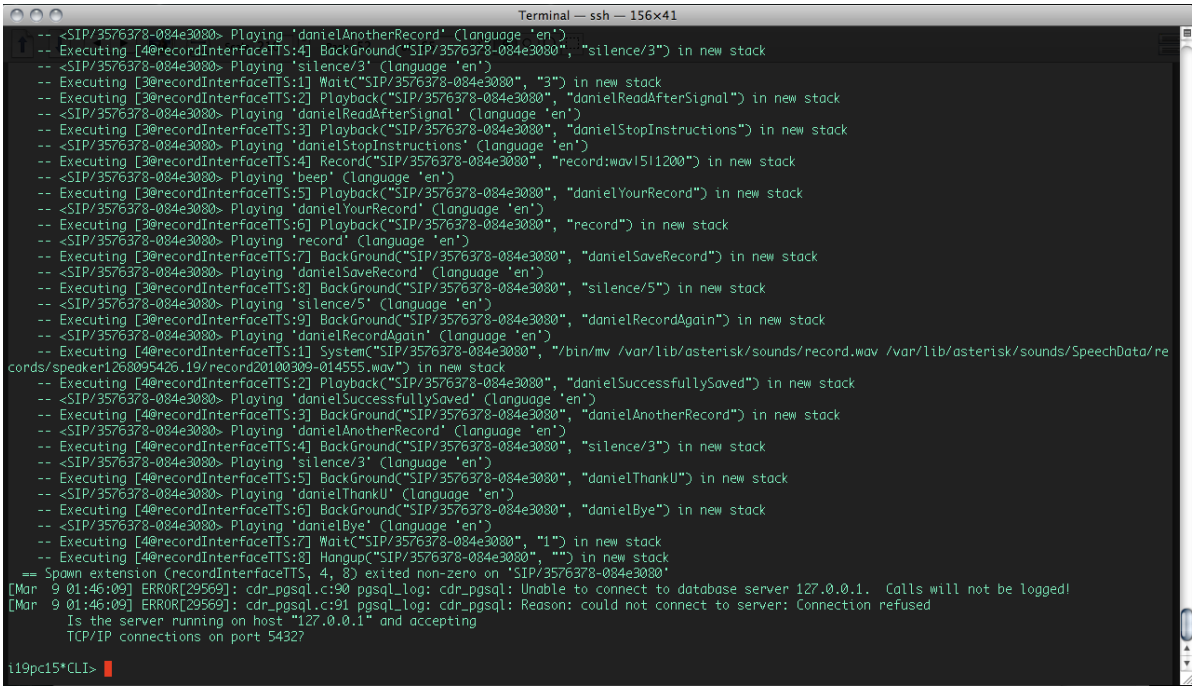
If the user does not want to save the record permanently, the temporary audio file `record.wav` is overridden the next time, when a user makes a record. The same procedure is repeated if the user wants to continue with recording audio.

As shown in Figure 5.13 (b), all records are saved with different names thanks to the ${STRFTIME(${EPOCH},„%Y%m%d-%H%M%S)} function, which returns the formatted current time [48].

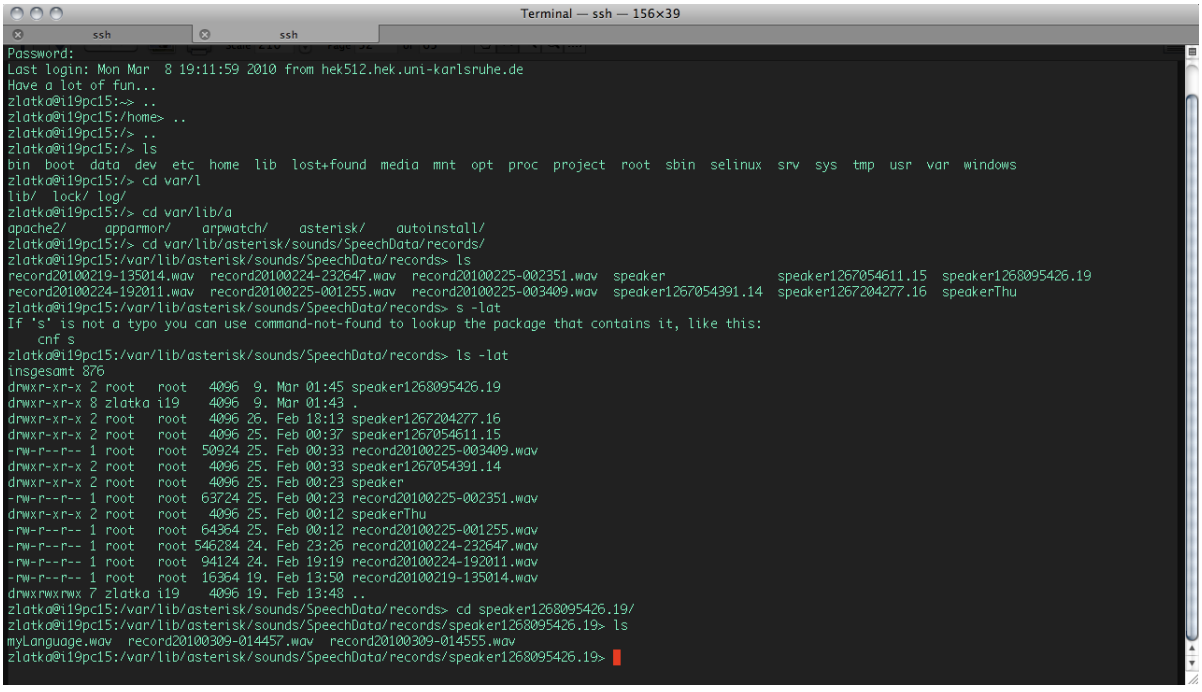
In the second module, the goal is to record answers from the user. We surely have to know what the native language of the speaker is and what topic he or she likes to discuss. Till now, we have implemented four possible topics for questions which are ”politics”, ”education”, ”science” and ”hobbies”. The ”question block” (QB) consists of five questions to the selected topic. They search for a personal opinion rather than having right and wrong answers. The goal is to provoke longer answers by the caller to record many words and respectively many phonemes. In Table 5.2, we give the questions from the blocks sorted by topics.

	POLITICS
1	What is your personal opinion about war?
2	Which party builds the major part of the government in your country?
3	What ideas supports the leading party in your country?
4	What is the political organization in your home country and how is it structured?
5	If you had the power to change the political direction of your home country, what would you start with and why?
	SCIENCE
1	What particular science subject are you passionate about?
2	Can you describe the most interesting scientific question for you?
3	Can you give an example of some famous scientists from your country?
4	Does the government in your country invest enough resources in research?
5	Why did you choose this topic? Are you a scientist?
	EDUCATION
1	Share your thoughts about the educational system in your country?
2	Is there a relationship between success and education, and if yes, how are they related?
3	Tell me more about your education.
4	What is more important for a successful educational system - hard work or creativity? Why?
5	How do you imagine the perfect educational system?
	HOBBIES
1	What is your hobby and why are you passionate about it?
2	Do you have other hobbies and if yes which?
3	How often do you practice your hobby?
4	Does your hobby bring you benefits in your professional career?
5	Can you imagine your hobby as your job?

Table 5.2: The questions used in the module for collecting spontaneous speech sorted by topics.



(a) The developer can observe the execution of the dialplan via the CLI.



(b) Overview of the saved records.

Figure 5.13: Managing the system in the terminal.

For the spontaneous speech module we use Festival. All Asterisk configuration files can be found in the directory `/etc/asterisk/` of pc15 at CSL as it is located at the student's pool and has the right hardware requirements. As clear from the code snippet in Figure 5.12, the folders with recorded audio files are accessible in the `/var/lib/asterisk/sounds/SpeechData/records` directory on the same machine.

5.3 Summary

In this chapter, we tested different solutions for the implementation of our idea and finally implemented the interactive system by using the open source PBX software Asterisk. The explored possibilities are based on different technologies and have a set of different components. The first analyzed commercial solutions are easy to use and more powerful by default. However, they are close source and expensive. An open source software for an IVR application has the disadvantage of being difficult to configure and lacks of standardization most often. With the open source engine Asterisk, we found a good compromise between available functions and costs.

6. Evaluation

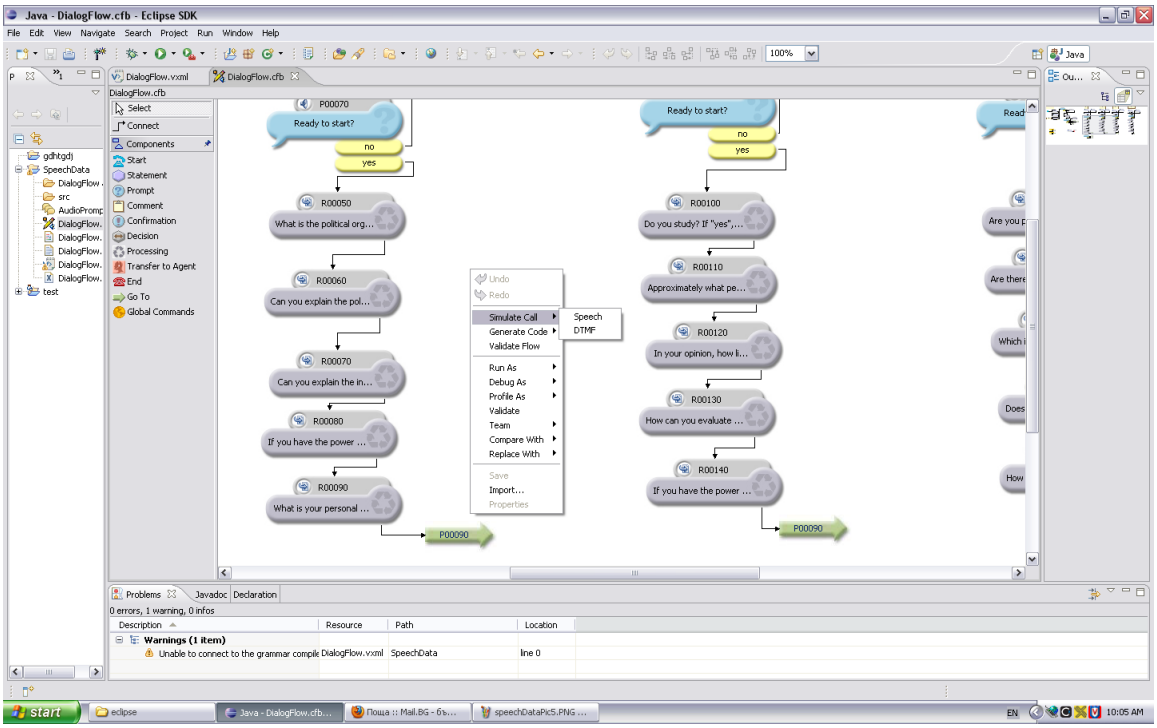
In Section 4.4, we mentioned that the process of developing an IVR system is a loop through the phases ”design” - ”test” - ”improve”. In this chapter, we summarize important results from the two tests we have performed. First a ”Wizard of Oz” experiment was done before the implementation of the system. The second test uses the already implemented system and a short questionnaire, filled by volunteers after completing tasks with the system. Both tests are different by nature. Even if they have the same goal to improve the usability of the IVR system, the ”Wizard of Oz” experiment works on the conceptual level, while the questionnaire test works on implementational level.

6.1 Wizard of Oz test

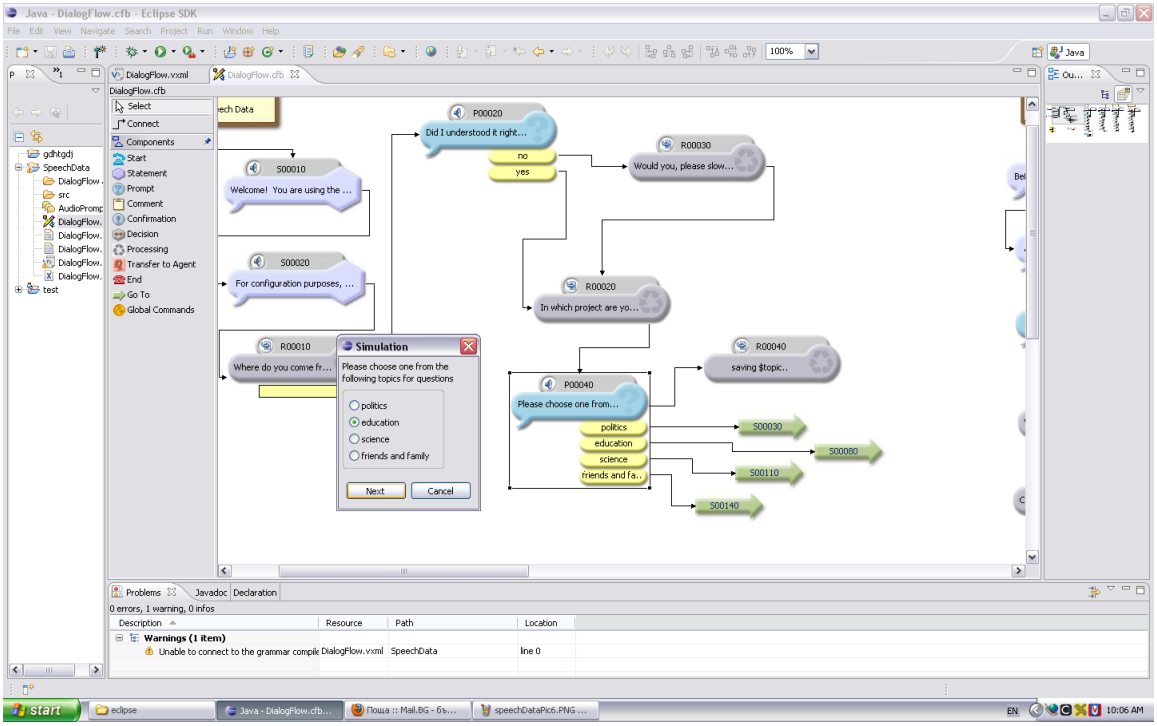
The ”Wizard of Oz” test is an experiment where a person plays the role of the missing system to record and analyze the reactions of the users. The system is at least partly simulated by a human, the wizard [49]. Why is this helpful? Before we spend a lot of energy in implementing a piece of software which does not work properly or relies on a false assumed user's reaction, it is always a better idea to see first how people would interact with such a system. The earlier we find an error the more work we would save. Sometimes an idea that seems to work perfectly in theory, appears to be completely inappropriate when realized. We have observed this phenomena in the conceptual phase: The idea of controlling the recording process with voice commands such as ”start” and ”stop” seems intuitive, but the experiment showed that the users forget to say these commands. By simulating an interaction between the user and the system, we are able to enhance the system stepwise before we implement it.

Our initial configuration for the ”Wizard of Oz” test was a voiceXML file created with the help of Eclipse and the IBM voice tools plug-in which we described in Section 5.1.1. We simulated the interaction with the dialog system in Eclipse as shown in Figure 6.1.

Each test conversation was recorded and then analyzed. Six subjects took part in the test, all of them non-native English speakers. The results of this analysis are given in Table 6.1 as well as their possible solutions.



(a) Call simulation based on a voiceXML file.



(b) The answers in the simulation can be given as text or DTMF input.

Figure 6.1: Call simulation with the IBM Websphere Voice Toolkit plug-in for Eclipse.

PROBLEMS	SOLUTIONS
START/STOP: The idea of marking the begin and the end of the speech data to be recorded looks theoretically very suitable, but practically it came out that this idea is not good since it is not natural and people forget it. They begin to say "start" and "stop" after a lot of training which in our case is not suitable.	The solution to this problem is to implement an intelligent method to capture the whole data to be recorded. We considered carefully timeout values, help prompts and remarks.
LONG SILENCE INTERVALS: We observed longer silence intervals, typical for spontaneous speech.	Timeouts have to be set carefully and tested.
MULTIPLE LANGUAGES MENTIONED: The speaker speaks more languages and mentions all of them in the language prompt.	The question must be changed from "What language do you speak?" to "What is your native language?"
"FRIENDS AND FAMILY" TOPIC ASSUMED AS DIFFERENT TOPICS: Most participants assume that the topic "Friends and Family" are two different topics which we regarded one topic.	Either these topics should be separated or another name must be chosen. For example: "Hobby"
CONFUSION WITH ENGLISH ANSWERS: It happens that the users are confused by the questions asked in English and in such case the answer comes in English while he/she has to answer in his/her native language.	The first question by all topics can involve answer with native words. For example: "Tell the name of a popular native game in your country? Explain the rules of the game." or "Tell the name of a popular native dish in your country and explain how is it made and from which ingredients?"
TOO SHORT ANSWERS TO SOME QUESTIONS: some questions are answered very shortly.	We reformulated some questions.

Table 6.1: Results of the first "Wizard of Oz" test

During the development of a dialog system, the developers make difference between two kinds of system outputs: prompts and messages. While messages are used to give an information to users, prompts are there to get an information from the user. According to some notes made during the test, users answered not only to prompts, but also to messages. They were not able to distinguish between both components. This may have an important implication to systems with mixed and user-directed initiatives.

The "Wizard of Oz" test helped us finding some problems in the early phase of the system's development. Some tendencies of the user's behavior are difficult to predict. Finding these issues improves the quality of the system and saves developer's time.

6.2 Questionnaire

The second evaluation we performed is a simple usability test for the read speech module. The volunteers operated with the real system, which we built on the results of the "Wizard of Oz" test. This time they had to share their experience with us through a short questionnaire. Based on their improvement proposals, we reconstructed parts of the system and made it more user friendly and stable.

Since the user's experience is a subjective issue, we used questions from the Telecommunication Standardization Sector of International Telecommunication Union (ITU-T P.851) recommendation [50] for evaluating spoken dialog systems according to de-facto standards [51].

As proposed in [50], we used two methods for extracting information: One through the users' questionnaire and the second through the observation of the users' interactions with the system. The questionnaire consists of four parts: User's background, user's expectations, a section with questions about the TTS quality and a section about the overall impression from the system. In [52] two points of view are discussed: System developers are concerned with the performance issues such as successfully completed tasks percentage. The other point of view is the user's satisfaction, which is a subjective measure. Both objective and subjective measures do not have direct link with each other. We adopted the questionnaire as a basis for our subjective measurement of the system's qualities and we also made records of some performance factors such as average duration of the calls, how many people succeeded in fulfilling the task, how often the user needed help from the system to complete the task, etc.

Nine subjects of age between 22 and 30 took part in our test and filled out the questionnaire after the interaction with the system. All of them were non-native English speakers from six different countries, speaking six different languages: Arabic, Bulgarian, French, German, Polish and Vietnamese. The majority of the subjects speak English fluently, about one third of the participants positioned their knowledge of the English language as intermediate. 44% of them had no previous experience with telephone-based interactive systems. All subjects in the test think that no prior learning is necessary to interact with the system and also categorized their task as "easy".

We used a five point scale to extract the subjective opinion of the participants: From 1 meaning "I disagree" to 5 meaning "I agree". We calculated the average grade $\bar{x}$

for every question as the arithmetic mean using: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ where n is the number of the subjects participated in our test[1] and x_i is the integer mark in the interval $[1, 5]$ they selected. The results are shown in Table 6.2.

QUESTIONS	RESULTS
I had to concentrate to acoustically understand the system.	2,2
The system voice sounded natural.	4,1
The system did not always do what I wanted.	2,4
The information provided by the system was clear.	4,3
The provided information was incomplete.	1,9
The system is unreliable.	1,4
The system reacted too slowly.	2,6
The system is friendly.	4,4
The system made many errors.	1,7
I got easily lost in the dialogue flow.	1,7
The dialogue quickly led to the desired aim.	4,4
The interaction with the system was pleasant.	4,2
It is easy to learn to use the system.	4,3
I prefer to record speech data in a different way.	2,2
I managed to complete my task.	4,9
Overall, I am satisfied with the system.	4,6

Table 6.2: Average results for the questions in the questionnaire, measured on the five-point scale with 1 meaning "I disagree" and 5 meaning "I agree".

The average call duration was 3 minutes 32 seconds, while the overall duration of the collected audio files was 15 minutes 58 seconds. The majority of the users chose reading one to four paragraphs' articles, chosen from a news website. 90% of the subjects successfully completed the task. The participant who did not manage to complete the task succeeded in a second try. First try was not successful, because many people around were speaking loud and the system did not stop the recording since it did not detect intervals without human speech. Nobody requested help by pressing the DTMF number of the help extension.

From the questionnaire, we got suggestions how to increase the quality of our system and after the test we improved it by solving the problems which occurred during the test as well as implementing some of the users' suggestions. We improved timeouts according to the users' suggestions and values from the test. We also updated the systems with additional help options and made some changes in the dialog flow. The best result from Table 6.2 is for the statement "I managed to complete my task.". Also in the same table it is visible that the majority of the subjects are satisfied with the overall performance of the system and they categorize the system as reliable. The most negative result with a score of 2.6 was for the statement "The system reacted too slowly.". After calculating the results and paying attention to the users' suggestions, we improved the following system's issues: We made the

[1] In our case, $n = 9$.

silence intervals between the prompts shorter to speed up the interactions. We also included help extension for users who need help and we added messages with additional information about what happens with the recording if the user does not want to save it . To collect more data, we made it possible to record multiple audio files with speech data within a single call.

6.3 Summary

Improving human-machine communication is a goal of many science studies. Most people still prefer talking to real people not to computer programs. Paying enough attention to the usability is an investment for the future. Therefore, experiments with real persons and learning from their experience and opinion is vital for systems interacting with human beings. An interactive system that will perform perfectly in every situation is a challenge. People are different, therefore not completely predictable. However, if with the help of simple tests, we manage to discover issues which are source of errors, this will result in a significant improvement.

7. Summary and perspectives

Implementing a telephone-based interactive system is a complex task, primary due to the lack of standardization. Decades of using proprietary telephone software tools and hardware devices led to a big amount of standards, most of them bound to the usage of a given small set of products. However, with the rapid development of VoIP came also many opportunities available to the wide public. In Chapter 2, we mentioned the technologies and standards which build the fundamentals for our work. These can be categorized in three different fields: telephony, speech processing and web.

In Chapter 4, we considered different high- and low-level system design issues to outline how our system should look like. This early model was used in the "Wizard of Oz" test described in Chapter 6 to identify major design problems and remove them before the implementation of the system.

Most important part of our work was to find the appropriate software tools and to bring our idea into existence. We analyzed some commercial and open source solutions, described detailed in Chapter 5. After identifying their advantages and drawbacks, we decided to implement our IVR system using the open source PBX software Asterisk.

The final system we presented is a product of continuous test-improvement cycle. It consists of two modules: One for collecting read text audio data and the other for collecting spontaneous speech by answering questions on different topics. Till the present moment, there are four topics that our system can offer: Politics, science, education and hobbies. Due to the flexibility of Asterisk, adding more topics to the system is not a problem.

The module aimed to collect read speech is simple but brings a lot of benefits. It serves as a remote recording tool and is especially useful in the situations when appropriate recording equipment is not available to the speakers. It may save travel costs and equipment costs.

Although overall all test users were satisfied with the the IVR system we implemented, it is a good base to be improved and extended. In the following paragraphs we try to define some perspectives and guidelines for future improvements.

Choosing Asterisk gives us a lot of possibilities to extend our IVR system with additional modules and new functions or even to exchange some of the system's components such as databases, speech processing parts, etc.

Since our system relies on DTMF inputs, the next step will be to integrate a speech recognition engine or even a whole speech server like the Zanzibar Open IVR, for example. The majority of Asterisk's users utilize commercial speech recognition and speech synthesis engines such as LumenVox [53], others prefer the open source concept by integrating speech recognition software such as Sphinx. For the successful integration of a speech recognizer, however, special acoustic models trained with audio data sampled at 8 kHz are necessary. If this precondition is satisfied, the Janus speech recognizer used at CSL can be also integrated in our IVR architecture.

As our system is implemented completely in Asterisk Extension Language (AEL), we can integrate a voiceXML interpreter and this way take advantage of the voiceXML-standard's flexibility and popularity. We can also integrate Asterisk with a whole speech server. This opportunity includes voiceXML parser, web server and other useful technologies. Some of the integration candidates are Zanzibar Open IVR, JVoiceXML and OpenVXI voice browser.

Further we have the possibility to improve technically our system by exchanging our hosted solution described in Section 5.2.3 with a private VoIP gateway. It might be more expensive than what we have now, but at least, we would not be dependent from VoIP provider and all the firewall obstacles.

A module with a semi-automatic transcription might support a human transcriber. Another idea was to make a quiz with questions and answers on the telephone to collect speech data. Games stimulate users to donate more speech data by playing.

Finally, we may pay more attention to the small details in our system. Each further test user can share interesting suggestions which will make our interactive system more stable and user-friendly.

Bibliography

[1] Cognitive Systems Lab at Karlsruhe Institute of Technology (KIT): http://csl.ira.uka.de/.

[2] Rapid Language Adaptation Toolkit: http://csl.ira.uka.de/index.php?id=29.

[3] T. Schultz, A. W. Black, S. Badaskar, M. Hornyak, and J. Kominek, SPICE: Web-based Tools for Rapid Language Adaptation in Speech Processing Systems, in *INTERSPEECH*, 2007.

[4] GlobalPhone website: http://www-2.cs.cmu.edu/%7Etanja/GlobalPhone/.

[5] FestVox: http://festvox.org/.

[6] H. Engelbrecht and T. Schultz, Rapid Development of an Afrikaans-English Speech-to-Speech Translator, in *Proceedings of International Workshop of Spoken Language Translation (IWSLT), Pittsburgh, PA*, 2005.

[7] S. J. Bigelow, J. J. Carr, and S. Winder, *Understanding Telephone Electronics*, Newnes, 4th edition, 2001.

[8] http://www.pcmag.com/encyclopedia/.

[9] IBM, *WebSphere Voice Server for Multiplatforms*, tenth edition, 2006.

[10] Speech Synthesis: http://en.wikipedia.org/wiki/Speech_synthesis.

[11] X. Huang, A. Acero, and H.-W. Hon, *Spoken language processing: a guide to theory, algorithm, and system development*, Prentice Hall, 2001.

[12] VoiceXML 2.1 specification: http://www.w3.org/TR/2007/REC-voicexml21-20070619/.

[13] W3C website: http://www.w3.org/.

[14] CCXML standard specification: http://www.w3.org/TR/ccxml/.

[15] D. Harel and M. Politi, *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*, McGraw-Hill, 1998.

[16] SCXML standard definition: http://www.w3.org/TR/2009/WD-scxml-20091029/.

[17] SRGS specification: http://www.w3.org/TR/speech-grammar/.

[18] MRCP Memo: http://tools.ietf.org/html/rfc4463.

[19] SIP Memo: http://tools.ietf.org/html/rfc3261.

[20] Apache Tomcat Webserver: http://tomcat.apache.org/.

[21] K. Paul, A T1-Based Speech Data Collection Platform, Institute for Signal and Information Processing, Mississippi State University, 1991.

[22] *SmartWeb UMTS Speech Data Collection*, LREC 2006, Genova, Italy, 2006.

[23] UMTS: http://www.phonearena.com/htmls/terms.php?define=UMTS.

[24] *CSDC - The MoTiV Car Speech Data Collection*, ICLRE 98, Granada, Spain, 1998.

[25] L. M. Tomokiyo and S. Burger, Eliciting Natural Speech from Non-Native Users Collecting Speech Data for LVCSR, in *Proceedings ACL-IALL Joint Workshop on Computer Mediated Language Assessment and Evaluation in NLP*, 1999.

[26] J. Badenhorst, C. van Heerden, M. Davel, and E. Barnard, Collecting and evaluating speech recognition corpora for nine Southern Bantu languages, in *First Workshop on Language Technologies for African Languages*, 2009.

[27] Statistics Website: http://www.gapminder.org/.

[28] J. V. Meggelen, J. R. Smith, and L. Madsen, *Asterisk: the future of telephony*, OReilly, 2007.

[29] Human language technologies: http://cslu.cse.ogi.edu/HLTsurvey/.

[30] T. Kemp and A. Waibel, Unsupervised Training Of A Speech Recognizer: Recent Experiments, in *in Proc. EUROSPEECH*, 1999.

[31] N. Ström and S. Seneff, Intelligent barge-in in conversational systems, MIT Laboratory for Computer Science, 2000.

[32] Microsoft Speech Application SDK: http://www.microsoft.com/downloads/details.aspx?FamilyId=1194ED95-7A23-46A0-BBBC-06EF009C053A&displaylang=en.

[33] SALT specification: http://msdn.microsoft.com/en-us/library/ms994629.aspx.

[34] Open Source Speech Software from CMU: www.speech.cs.cmu.edu.

[35] JVoiceXML: http://jvoicexml.sourceforge.net/.

[36] FreeTTS engine: http://freetts.sourceforge.net/docs/index.php.

[37] Java Speech API: http://java.sun.com/products/java-media/speech/.

[38] JVoiceXML Guide: http://prdownloads.sourceforge.net/jvoicexml/jvxml-userguide-0.7.2.pdf.

[39] The open source telephony project Asterisk: http://www.asterisk.org/.

[40] The software based PBX for Windows 3CX: http://www.3cx.de/.

[41] Zanzibar Open IVR: http://www.spokentech.org/openivr/aik.html.

[42] http://www.asteriskguru.com/.

[43] The VoIP provider Sipgate: http://www.sipgate.de/.

[44] Sipgate call tariffs: http://www.sipgate.de/user/tariffs.php.

[45] http://www.fridu.org/asterisk-pbx-faqs-tips-78/36-4-asterisk-sip-signalling-nat.

[46] A. W. Black and T. Schultz, Speaker clustering for multilingual synthesis, in *MultiLing*, 2006.

[47] Nuance RealSpeak: http://212.8.184.250/tts/demo_last.jsp.

[48] STRFTIME function parameters: http://www.voip-info.org/wiki/view/ Asterisk+func+strftime.

[49] M. Hajdinjak and F. Mihelič, Wizard of Oz Experiments, 2004.

[50] S. Möller, Neue ITU-T-Empfehlungen zur Evaluierung telefonbasierter Sprachdialogdienste.

[51] S. Möller, P. Smeele, H. Boland, and J. Krebber, Computer Speech and Language 21, pp. 26-53, in *Evaluating spoken dialogue systems according to de-facto standards: A case study*, 2005.

[52] S. Ondáš, J. Juhár, and A. Cižmár, Evaluation of the Slovak Spoken Dialogue System Based on ITU-T, Springer-Verlag Berlin Heidelberg, 2008.

[53] LumenVox speech recognizer: http://www.lumenvox.com/.

[54] *Wizard Of Oz Studies - Why And How*, International Conference on Intelligent User Interfaces, 1993.

[55] S. Wintermeyer, *Asterisk 1.4 + 1.6 : Installation, Programmierung und Betrieb*, Addison Wesley in Pearson Education Deutschland, 2009.

[56] I. Rogina and A. Waibel, The JANUS Speech Recognizer, in *ARPA SLT Workshop*, pp. 166–169, Morgan Kaufmann, 1995.

Index