IDIAP COMMUNICATION

IDIAP

Martigny - Valais - Suisse

VoicePhone: An Interactive Vocal Server for Telephone Numbers

Hans JONGEBLOED *+

IDIAP–Com 96-04

December 96

Dalle Molle Institute for Perceptive Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 – 27 – 721 77 11
fax +41 – 27 – 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

* IDIAP
\+ Cognitive Science and Engineering, University of Groningen

Abstract

VoicePhone is a dialogue system, designed to recognize telephone numbers over the public telephone network. This report describes the basics needed for this program, including theory on speech recognition, human-computer interaction and user friendly dialogue design. Next, the design of the application is described, including the set-up of a Wizard of Oz environment. This environment is used during a first user test to evaluate the first version of the dialogue system.

Samenvatting

VoicePhone is een dialoog systeem, ontworpen om telefoon nummers te herkennen over de telefoon. Dit verslag beschrijft de basis voor dit programma, inclusief theorie over spraakherkenning, cognitieve ergonomie en het ontwerp van gebruikersvriendelijke dialoog systemen. Ook wordt het ontwerp traject van het systeem beschreven, inclusief het opzetten van een Wizard of Oz module. Deze module is gebruikt tijdens een eerst gebruikers test, om de eerst versie van het dialoog systeem te evalueren.

Résumé

VoicePhone est un système de dialogue, construit pour la reconnaissance des numéros de téléphone via la téléphone. Ce rapport décrit la base théoretique pour ce programme, y compris les relatives a la reconnaissance du parole, a l'interaction homme-machine et a la construction de dialogues conviviaux. Ensuite, le rapport decrit la construction de l'application, y compris un environnement Wizard of Oz. Cet environnement a été utilisé pendant un premier test avec des utilisateurs pour evaluer la premiere version du système de dialogue.

Contents

Chapter 1

Introduction

This is the final part of two different projects for the diploma work in the curriculum of Cognitive Science and Engineering, performed at IDIAP in Switzerland under supervision of Jean-Luc Cochard. The first part dealt with the implementation of a '2-dimensional signal comparison' program, using an Evolutionary Algorithm (see [Jongebloed-96.1]).

Cognitive Science and Engineering is a four-year interdisciplinary program at the University of Groningen, with roots in psychology, philosophy, computer science, physics and linguistics. In addition to courses from several disciplines, the program also contains a number of interdisciplinary courses. These are only included in the program insofar as they are concerned with one or more aspects of (human or artificial) intelligence. Within the program of Cognitive Science and Engineering, the emphasis is both on the theoretical aspects of cognitive science and on its applications.

In the project, a number of Cognitive Science and Engineering research topics are encountered, like speech recognition, natural language interfacing and cognitive ergonomics. The general aim of the final part of Cognitive Science is to design (part of) a system imitating or modeling cognitive capabilities or to create a system interacting with humans taking their cognitive capabilities into account. Both are applicable here, the aim of a Interactive Vocal Server being twofold: allowing users to interact with a computer in a friendly manner and providing them with a system they can communicate with.

1.1 IDIAP: Institut Dalle Molle d'Intelligence Artificielle Perceptive

The Dalle Molle Institute for Perceptive Artificial Intelligence Perceptive (IDIAP) is situated in Martigny, Switzerland. The research institute is one of three, all founded by Angelo Dalle Molle. They are situated in Genève (ISSCO, 1973), Lugano (IDSIA, 1987) and in Martigny (1991). The research is both fundamental and application oriented in different fields of Cognitive Science (resp. Language, Knowledge and Perception). The aim of the institutes is threefold: fundamental research, training of researchers and engineers, and the participation to the economic development of the region in which they are situated. At the moment, IDIAP is involved in international projects (CAVE, SpeechDAT, M2VTS), with partners all over Europe, both industrial (KPN research (NL), UBILAB (CH), PTT Telecom (CH)), and academic (EPFL Lausanne (CH), KUN Nijmegen (NL)) and in international research groups (ESA).

The research activities at IDIAP are all related to aspects of perception. The three main themes are:

- Speech Recognition & Processing
- Neural Networks
- Image Recognition

The Speech group performs research on different aspects of Speech Recognition & Processing, such as Speaker Verification, Dialogue System (Interactive Vocal Servers) and new directions in Speech Recognition (e.g. Multi-agent systems). Furthermore there are different projects dealing with the setup of speech databases.

1.2 Context of the Project

At IDIAP, various interactive vocal servers have been built, to demonstrate the capabilities of real time speech recognition. The latest project deals with the implementation of a dialogue system that replaces the current 'reversed bill' call systems. In such a system callers having a special card for this service, can enter their card number and a personal PIN code, before placing the call. The costs for the call will then be reversed to his own telephone account, so he doesn't have to insert money. This service is offered by the Swiss Telecom, under the name 'Telecom Card'. To use this system, the telephone has to be equipped with a DTMF[1] keypad, to enter the card and PIN code. The project here at IDIAP, called 'Voice Telecom Card' offers a voice driven version of this system (described in [Genoud-96]). In this application, the card number and PIN code are spoken by the caller and recognized by the system. This information is used to identify the caller, as well as to verify his identity by voice. After the acceptance the caller is asked to enter the desired telephone number on the DTMF keypad. This project offers the elegant feature of being able to verify a callers identity, but still the telephone from which the call is made has to be equipped with a DTMF keypad. At the moment a lot of telephones still use the rotary variant, so it might be useful to also make the last part of the application voice driven; the entering of the telephone number.

For this, a database containing spoken telephone numbers was used to create a speech recognizer for this task. The first tests with this recognizer showed a low performance for recognizing a complete number. To encounter this problem, my project was set up to design a dialogue system performing the task of recognizing a complete telephone number.

Swiss spoken telephone numbers are grouped in numerals (e.g. 'twenty one'), when communicated between humans (as well as in other countries, e.g. France [Gagnoulet-89] and Norway [Kvale-96]). This behaviour makes the task much more difficult, compared to the tasks of recognizing a digit string. The system has to deal with more possible words as well as taking care of ambiguously spelled numbers (e.g. 'twenty four' = '20 4' or '24'). The system should be easy to use (user-friendly), which implies that the option of asking a caller to speel tthe number digit by digit can not be used. During this research the following research questions will be handled:

- How does a user “dictate” a telephone number?
- Design a dialogue between the system and the user, that enables the system to recognize the number.
- Do this in such a way that the user can easily deal with the system (as user-friendly as possible).
- Find a compromise for the above two items.

Because of the time remaining for this project, the goal is not to build a final optimized system, but rather to implement a first dialogue system, which can be tested on users. Preferably these tests will be done using a Wizard of Oz environment that simulates the dialogue system, thus allowing more flexible tests. The implementation of such an environment will be of interest for all future dialogue systems at IDIAP. Ideas obtained during the design and evaluation can then be used to improve the system accordingly. Conclusions and ideas from this project will be used afterwards at IDIAP for the further development of dialogue systems.

[1]DTMF = Dual Tone Multi Frequency, is used for coding the keys on the keypad over the telephone line.

1.3 Contents of the Project

The initial basis for the project was to do research, using Hidden Markov models for speech recognition. More specifically, this should be done using the Hidden Markov Toolkit (HTK) to get experience with the most widely used way speech recognition is done. To start with, the usage of HTK for speech recognition is studied.

To get a better overview of the field of dialogue systems, a literature study is performed. This resulted in a number of issues, that should be included during the design of a dialogue system. Special attention is paid to the user-friendliness of dialogue systems.

The implementation of a user-friendly dialogue system, requires knowledge on the user behaviour. Therefore a pre-recorded database of spoken telephone numbers is used to extract typical behaviour. This is also used to determine a grammar for the speech recognition of a complete telephone number.

The implementation itself, called VoicePhone is based on a previously designed dialogue system, of which the complete dialogue had to be changed (i.e. new flow and a different speech recognition task). Also the creation of feedback to the user and more sophisticated turn taking were included. This development was enhanced by testing it many times on a reference user, who was able to give general comments and ideas. Both the initial existing program and the final program are completely documented in two program descriptions/manuals.

The design strategy of dialogue systems, includes performing user tests as soon as a first dialogue flow is implemented. For this, a technique called Wizard of Oz is used, in which the speech recognition is replaced by a human operator (the Wizard) who listens to the user utterances. This technique was implemented in a separate Wizard of Oz module that was attached to VoicePhone. The module provides a graphical interface through which the Wizard can enter his description of the user utterance. This description is then used in VoicePhone, instead of the outcome of the speech recognition.

Finally, a Wizard of Oz experiment was set-up to perform a user test with a first version of VoicePhone. The used dialogue was not yet a highly sophisticated one, but rather a simple initial version to test user behaviour in front of such a system. Based on this user behaviour, the outcome of a questionnaire, and earlier obtained ideas, a number of directions are proposed to improve the current dialogue. In the continuation of this research a more elaborate usertest (i.e. a large number of users from the general public) is needed.

Chapter 2

Speech Recognition, Using Hidden Markov Models

This chapter will review the basic theory on hidden Markov models, and their usage in speech recognition. It starts with an overview of Markov models, before the transition to hidden Markov models is made. It is very well possible to skip this chapter for those readers who already know the basic theory of speech recognition. More details can be found in [Rabiner-93] and [HTK-93].

2.1 Usage of Markov Models

Markov models are a very useful tool, to model sequential processes that have some (unknown) underlying generation mechanism. They consist of:

- a number of states: $(S_1, S_2, \ldots, S_N)$
- an initial state probability distribution: $\Pi = (\pi_1, \pi_2, \ldots, \pi_N)$.
- a matrix of transition probabilities between states:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix}$$

So in Markov models a_{ij} denotes the probability of a transition from state S_i to state S_j. In Markov models each state represents an observation in the real world. For example imagine a Markov model for the following semi-random[1] choice process. From a bag with a red (R), a blue (B) and a yellow (Y) ball, a certain ball is chosen. The colour of the ball can be represented by a state in the model. An observation sequence $O(1), \ldots, O(T)$ in the real world is represented by a certain state sequence in the Markov model. For example From the viewpoint of the Markov model, the model *emits* a symbol (R,B,Y), being a colour in the real world a colour. Note that in the simple Markov model each colour is bounded to a specific state. There are two important assumptions that form the basis of these Markov models:

- The transition probability of going from one state to another is independent of the previous state.

[1] In a semi-random process, the probabilities are not equal. There might be an underlying stochastic process, which is unknown.

- The observation of $O(t)$ is also independent of previous states. Note, that in the Markov models as described here this relation is one-to-one, so that this is trivial. The importance of this statement will become more clear when dealing with hidden Markov models.

To work with Markov models, some algorithms are needed for training and recognition procedures. The following subsections will explain these algorithms.

Forward Procedure

Now an interesting question arises: Given a Markov model, what will be the probability being in a certain state S_i at time t. This can be calculated be using the **forward procedure**. This is a recursive procedure, which means that the probability of being in state S_j at time t is calculated based on the probability of being in one of the states S_i at time $t-1$, multiplied by the transition probability between the two states. The final step in this recursion is calculating the initial state probability Π. For this procedure we define $\alpha_j(t)$ as the probability of being in state j at time t. This gives:

$$\alpha_j(t) \ = \sum_{i=1}^{N} \alpha_i(t-1) \cdot a_{ij}, \tag{2.1}$$

with the initial condition:

$$\alpha_j(1) \ = \pi_j.$$

Viterbi Algorithm

Here a more complex question is handled: "Which observation sequence, ending in state S_i at time t has the highest probability?" This is the same as regarding, which state sequence ending in state S_j (which belongs to $O(t)$) has the highest probability. This can be done by using the **Viterbi algorithm**. It resembles the forward procedure, only now, at each step, a maximization instead of a summation over all possible states is done.

For this procedure, define $\delta_j(t)$ as the most probable sequence leading to state S_j at time t. Also a pointer $\Psi_j(t)$ giving the state S_j as the "best" state at $t-1$ is defined to store the previous state (notice that $\Psi_j(t)$ will be an element of $(1 \cdots N)$):

$$\delta_j(t) \ = \overset{max}{1 \leq j \leq N} \ (\delta_i(t-1) \cdot a_{ij}), \tag{2.2}$$

with the initial condition:

$$\delta_j(1) \ = \pi_j.$$

And the pointer to the previous state:

$$\Psi_j(t) \ = \overset{argmax}{1 \leq j \leq N} \ (\delta_i(t-1) \cdot a_{ij}) \tag{2.3}$$

The complete Viterbi algorithm then consists of calculating $\delta_j(t)$ and backtracking with $\Psi_j(t)$ to find the 'most likely path'. The probability of the most likely state sequence is then found as $\overset{max}{1 \leq j \leq N} \ (\delta_j(T))$

Recognition with Markov Models

In a recognition task, there is a given observation sequence and a number of classes. During recognition, one of the classes is assigned to the observation sequence. Each of the classes is represented by a Markov model. The problem is then to find the model M for which $P(O|M)$ (the probability of observing the state sequence, corresponding to observation sequence O, generated by the Markov model M) is maximal. In simple Markov models this is just the multiplication of all state transition probabilities (a_{ij}) between the consecutive states in the sequence.

Training of Markov Models

For recognition tasks, a group of Markov models, at least one per class is needed. (It might be necessary to create two different Markov models for one class, when the variations within a class are larger then between classes, this is called *clustering*). The problem is thus to find these Markov models, all trained for a special class of sequences. The algorithm can be described as follows:

1. count the number of transitions $S_i \rightarrow S_j$; this gives F_{ij}: the likelihood of going to state S_j, given that the current state is S_i.
2. count the number of transitions from S_i; this gives F_i, the likelihood of being in state S_i.
3. dividing these two gives $\tilde{a}_{ij}$.
4. normalizing the rows gives the matrix A, which defines the Markov model.
5. when using multiple training sequences, use the $O(1)$'s to calculate the initial state probability π.

In this way multiple training sequences per class of stochastic process are used, to train the different Markov models.

2.2 Usage of Hidden Markov Models

In simple, small domains Markov models will be very useful for constructing a model of a process and performing a recognition task to determine which process was observed. But when it comes to more complex domains, there will be many possible states and thus many state transitions to be calculated. When using M states, there will be M-1 $\times$ M-1 transitions to be calculated. To decrease this number of transitions, all possible observations are distributed over the states, so that basically every state can generate each observation symbol $O(t)$. Now the number of (hidden) states can be made as big as we wish (say N), and for each state, we have to calculate the probability of emitting $O(t)$ (there are M possible observations). So, there are only N-1 $\times$ N-1 transitions and M $\times$ N probabilities to be computed (learned), which is definitely less for N << M.

Each state S_i gets a probability distribution $b_i(O(t))$ which denotes the probability of emitting O(t), while being in state S_i. The observation sequence O is shown, but the state sequence remains hidden, because any state could emit any observation symbol. A hidden Markov model can thus be characterized as a simple Markov model with an additional **observation probability matrix** B**:**

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1M} \\ b_{21} & b_{22} & \cdots & b_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \cdots & b_{NM} \end{pmatrix}$$

An observation sequence is then generated by beginning at a certain state S_i (probability given by Π) and emitting an observation symbol $O(1)$ (generated by the probability $b_i(O(t))$). Next go to a next state S_j (based on the probability a_{ij}) and so on.

Recognition in HMM's

There are two ways to do recognition in HMM's, the first being more accurate[2] the second easier (faster) to calculate.

[2] This is because it computes the exact probability given in the model using al possible state sequences, whereas the second gives an estimation of this, using only the most probable state sequence.

- For each HMM, calculate the total probability of emitting the given observation sequence. The model with the highest probability, gives the class. This is done with the **forward procedure**.
- For each HMM, calculate the most likely state sequence, which generates the given observation, including the probability. The model with the highest probability, gives the class. This is done with the **Viterbi algorithm**.

The calculations needed for these algorithms can easily be deduced from the case of simple Markov models. Only now the whole observation sequence $O(1 \dots T)$ is given and the probability is calculated with the use of $b_j(O(t))$, the probability of emitting observation symbol $O(t)$, given that the current state is S_j. The formulas used for the calculations are as follows:

- For the forward procedure:

$$\alpha_j(t) \ = \sum_{i=1}^{N} (\alpha_i(t-1) \cdot a_{ij}) \cdot b_i(O(t-1)) \tag{2.4}$$

  with the initial condition:

$$\alpha_j(1) \ = \pi_j \cdot b_j(O(1))$$

  The probability of the observation sequence is then given as:

$$P(O|M) \ = \sum_{j=1}^{N} \alpha_j(T)$$

- For the Viterbi algorithm:

$$\delta_j(t) \ = \max_{1 \leq i \leq N} (\delta_i(t-1) \cdot a_{ij}) \cdot b_i(O(t-1)) \tag{2.5}$$

  Also with the initial condition:

$$\delta_j(1) \ = \pi_j \cdot b_j(O(1))$$

  And the pointer:

$$\Psi_j(t) \ = \operatorname*{argmax}_{1 \leq i \leq N} (\delta_i(t-1) \cdot a_{ij}) \cdot b_i(O(t-1)) \tag{2.6}$$

  The complete Viterbi algorithm then consists of calculating $\delta_j(t)$ and backtracking with $\Psi_j(t)$ to find the 'most likely path'. The probability of the most likely path is then given as:

$$P(O|M) \ = \max_{1 \leq j \leq N} (\delta_j(T))$$

Note, that the procedure used for recognition tasks in simple Markov models is the same as when using the forward procedure, with matrix B being the unity matrix (because each state gives exactly one observation symbol with probability one).

Training of HMM's

As with simple Markov models the HMM's have to be trained, before doing some recognition task. The procedure that will be used is iterative, that is, one starts with an initial set of parameter values and calculates the new values from the old ones and the training data. The process terminates, if the new parameters don't differ much from the old ones. The algorithm is known as the **Baum-Welch re-estimation formulas**. The condition of termination shouldn't be too tight, because the danger arises that the Markov model will only be able to recognize the training data. The initial parameters can be generated in a number of ways, e.g.. normalized random choices or a uniform distribution with random distortions. The basis is of course, much alike the procedure used with simple Markov models:

$$\tilde{a}_{ij} = \frac{probability\ of\ making\ transistion\ S_i \rightarrow S_j\ given\ we\ are\ in\ S_i}{probability\ of\ being\ in\ S_i} \tag{2.7}$$

$$\tilde{b}_{i(O(t))} = \frac{probability\ of\ observing\ symbol\ O(t)\ given\ we\ are\ in\ S_i}{probability\ of\ being\ in\ S_i}$$

$$\tilde{\pi} = probability\ of\ being\ in\ state\ S_i\ at\ t = 1$$

These probabilities are calculated from the parameters of the previous model and the training data $O(1 \ldots T)$. Before calculating them, a tool is needed to calculate the probability of being in state S_i at time t given the observation sequence $O(1 \ldots t \ldots T)$. This can be done in a way similar to calculating the probability of being in state S_i at time t given the observation sequence $O(1 \ldots t)$, using the forward procedure, though now in reversed direction. Hence, it is called the **backward procedure** (formulas given below). The probability of being in state S_i at time t given the rest of the observation sequence $O(t \ldots T)$ is then simply the combination of these two probabilities. This tool is called the **forward-backward algorithm**. So we have in formulas:

- For the forward procedure:

$$\alpha_j(t) = \sum_{i=1}^{N} (\alpha_i(t-1) \cdot a_{ij}) \cdot b_j(O(t)) \tag{2.8}$$

  with the initial condition:

$$\alpha_j(1) = \pi_j \cdot b_j(O(1))$$

- For the backward procedure:

$$\beta_i(t) = \sum_{j=1}^{N} (\beta_j(t+1) \cdot a_{ij}) \cdot b_j(O(t+1)) \tag{2.9}$$

  with the initial condition:

$$\beta_j(1) = \pi_j \cdot b_j(O(1))$$

Now the combination has to be made. Remembering, that we are looking for $L_j(t) = P(S_j(t)|O, M)$, the likelyhood of being in state S_j at time t, given the observation sequence O(1 ... T) and model M. The forward-backward procedure gives $P(O, S_j(t)|M)$, the probability of being in state S_j at time t during the observation sequence O(1 ... T), given model M. The relation between them is given by

$$P(S_j(t)|O, M) = \frac{P(O, S_j(t)|M)}{P(O|M)}$$

So to get the desired likelihood, simply divide by $P(O|M)$, the probability of observing O(1 ... T), given model M. Thus we have:

$$L_j(t) = \frac{1}{P}\alpha_j(t) \cdot \beta_j(t)$$

The next tool is one for calculating the probability of being in a certain state and making a transition to another state. This is done by calculating the joint event of being in state S_i at time t and being in state S_j at time $t+1$, thus making transition a_{ij} and emitting $O(t+1)$. This is given in the formula:

$$\xi_{ij} = \frac{1}{P}\alpha_i(t) \cdot \beta_j(t+1) \cdot a_{ij} \cdot b_j(O(t+1))$$

Given the tools for calculating the probabilities, the calculation of the new parameters, based on the training data, must be done. Therefore an average of the calculated probabilities over all times t and all (independent) training observation sequences is needed This gives:

$$
\begin{aligned}
\tilde{a}_{ij} &= \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} L_i(t)} \\
\tilde{b}_j(O(t)) &= \frac{\sum_{\substack{t=1 \\ O(t)}}^{T-1} L_j(t)}{\sum_{t=1}^{T-1} L_j(t)} \\
\tilde{\pi}_i &= L_i(1)
\end{aligned}
\tag{2.10}
$$

Extra Elements

There are some more elements that can implemented in HMM's.

Null States

Null states are non-emitting states. As they won't emit an observation symbol, they also won't consume time.

Multiple Codebooks

To deal with more than one feature per observation symbol, multiple codebooks can be used. For each of the possible representations of the observations a different codebook is created and used in the emission of a symbol.

Left-to-right HMM's

Till now it was assumed, that each state of the (hidden Markov models could be reached from each other state (fully connected HMM). In practice it can be useful to introduce a direction in the model which can be seen as the some time in the real world. In this case, the states can implement some real-world physical state. Most widely used in speech recognition is a model with all states being arranged from left to right.

Continuous versus Discrete HMM's

In the previous sections we assumed **discrete** HMM's. For each observation symbol the probability was exactly defined. In order to do this, all possible observations in the real world (greater than the number of observation symbols in the model) have to be represented by a symbol from a finite alphabet. This is done by **vector quantization**, where a set of prototype vectors is identified from all possible observations in the real world.

In most practical applications where continuous data is modeled, it can be useful to have continuous output distributions. This is can, for example, be done with Gaussian distributions. It prevents problems with quantization errors and the number of parameters are reduced, because the set of target vectors (i.e. a parameter value) for quantization is replaced by the two parameters (μ and Σ) to represent the Gaussian distribution.

HMM's for Speech Modeling

Hidden Markov models have found to be a robust and reasonably good performing way to do speech recognition. The basic assumption is that speech can be divided into short (typical around 25 ms., time step 10 ms.) frames, in which the signal is stationary. The speech signal can than be described by some typical parametric representations, such as LPC and MFCC. These so called speech vectors are then used as the emitting outputs of the HMM's. The recognition of speech is based on Bayes' Rule:

$$P(w_i|O) = \frac{P(O|w_i)P(w_i)}{P(O)}$$

To maximize this equation, it is sufficient (given $P(w_i)$) to maximize $P(O|w_i)$. This is nothing more than estimating the parameters of the HMM used for representing the possible utterances.

The HMM's models can be used at different levels to model speech. To do isolated word recognition, it is sufficient to use one HMM for every word. When dealing with a lot of words (large vocabular), this approach needs a lot of training data. Also for applications in continuous speech recognition, this approach is not appropriate because of coarticulation effects between words (the first phoneme of a word is influenced by the last phoneme of the previous word and v.v.). In these cases, phoneme or even di- or triphone models (to incorporate coarticulation effects between phones) offer a better result.

2.3 Using HTK; the Hidden Markov Toolkit

To use hidden Markov models for speech recognition, the Hidden Markov Toolkit (HTK), created by Entropic Research Lab, Cambridge is used (for more details, see the V1.5 manual [HTK-93]). HTK uses typically left-to-right HMM's in which the first and the last state are null-states. This is done to be able to concatenate the HMM's . The number of states can be 5-20 states for word-models (i.e. average of 3 states per phoneme) and 5 states (so 3 emitting states) for sub-word-models (e.g. phonemes). More details can be found in the HTK manual [HTK-93]).

In section 2.2, the output generated by the HMM was given by $b_j(O(t))$. In HTK speech is coded in multi dimensional Gaussian distributions, so that there are continuous HMM's instead of discrete HMM's (though HTK can deal with discrete HMM's as well). These multi dimensional Gaussian distributions are in the form of **Mixture Gaussian output distributions**. In these distributions all non-independent Gaussian curves are divided in independent mixture components (2-5), so that every mixture component has a diagonal covariance matrix. Per state, each mixture is then characterized by its mean μ_{jm} and its covariance matrix Σ_{jm}.

Another generalization is made by dividing the observation vector $O(t)$ in S independent data streams $O(t)_s$. In HTK maximal four streams can be used: the basic parameter vector, the first and second differential and the logarithmic energies of them. (this is more or less analogue to the multiple codebooks discussed in the previous chapter). The following formula for $b_j(O(t))$ is used:

$$b_j(O(t)) = \prod_{s=1}^{S} \left[\sum_{m=1}^{M_s} c_{jsm} \mathcal{N}(O(t)_s; \mu_{jsm}, \Sigma_{jsm}) \right]^{\gamma_s} \tag{2.11}$$

where:

- M_s is the number of mixtures in stream s.
- c_{jsm} is the weight of the m'th mixture.
- γ_s is the weight of the s'th stream (most of the time assumed to be 1).
- $\mathcal{N}(\cdot; \mu, \Sigma)$ is a multivariate Gaussian.

Initialisation in HTK

In case of isolated word recognition, it suffices to use HInit to initialize the parameters of the initial HMM's. This can be done in multiple ways

- use general speech data to give all HMM's the same initial values.

- use the *segmental k-means procedure* to create the different HMM's for each possible utterance (i.e. each phoneme, word)
- the observations will be equally divided over the three states.

After this, the procedure HRest can be used to perform the training of the models (explained in the next section). Each time HRest is called, it performs one training cycle. For the case of continuous speech recognition, one would need a training set of continuous speech, with a complete (label-file) description of the division in sub-word units. Not only will this cost a lot of work, the boundaries might not be optimal from the viewpoint of the HMM. Therefore HERest can be used to do so called *embedded training*, which choses its own boundaries within the different concatenated HMM's used to model the training data. The algorithm is as follows:

- allocate and zero accumulators for all parameters of all HMM's
- get the next training utterance
- construct a joint HMM consisting of all models corresponding to the symbol transcription of the utterance
- calculate the forward en backward probabilities for these HMM's
- use them to compute the likelihoods of state occupation at each time frame and update the accumulators
- repeat from step 2 for all training utterances
- use the accumulators to calculate new parameter estimates for all HMM's

Training in HTK

The only thing that differs in HTK from the HMM's in the previous chapter is the implementation of $b_j(O(t))$. The different streams are independent, so they won't alter much. The different mixtures can be viewed as sub-state in a state, so that our formula for the likelihood of being in mixture m of state S_j becomes:

$$L_{jm} = \frac{1}{P}\alpha_j(t)\beta_j(t)c_{jm} \tag{2.12}$$

where c_{jm} denotes the probability of being in mixture m, given the system is in state S_j.
We're now able to give the re-estimation formulas for the mixture weights:

$$\tilde{c} = \frac{\sum_{t=1}^{T} L_{jm}(t)}{\sum_{t=1}^{T} L_j(t)} \tag{2.13}$$

Now we can have a look at a HMM in which the output distributions in a state are single mixture Gaussians (this is a simplified form, in which there's only one single mixture Gaussians per state, instead of, say, 39). The output distribution is then given as:

$$b_j(O(t)) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} e^{-\frac{1}{2}(O(t)-\mu_j)'\Sigma_j^{-1}(O(t)-\mu_j)} \tag{2.14}$$

The parameter estimation formulas for $\tilde{\mu}_j$ and $\tilde{\Sigma}_j$ are then given as:

$$\tilde{\mu}_j = \frac{\sum_{t=1}^{T} L_j(t)O(t)}{\sum_{t=1}^{T} L_j(t)} \tag{2.15}$$

$$\tilde{\Sigma}_j = \frac{\sum_{t=1}^{T} L_j(t)(O(t)-\mu_j)(O(t)-\mu_j)'}{\sum_{t=1}^{T} L_j(t)}$$

which look a little like their discrete counterparts. The rest of the re-estimation formulas are then the same as in the previous chapter.

Recognition in HTK

The basics of the recognition task, is recognizing an utterance modeled by the best matching HMM:

$$P(O|w_i) = P(O|M_i)$$

In HTK this done with the approximation of using the most likely state sequence, given the observation sequence, calculated by the Viterbi algorithm. There is no direct tool for doing this, but instead it is incorporated in the HVite tool, which is the general recognition tool of HTK.

Till now, only the recognition of a single speech utterance, modeled by a HMM, is explained. This is sufficient for doing simple isolated word recognition, but when dealing with longer utterances these models have to be concatenated. This done by using the non-emitting first and last state of the HMM's as the 'glue' between the models. There are two ways to approach this *continuous speech recognition*.

- connected word recognition, based on whole word HMM's
- continuous speech recognition, based on sub word HMM's

The first one is mainly suited to do recognition tasks, where only a small vocabulary is used (then the training won't cost very much recorded data). However, for larger vocabulary applications or more flexible vocabularies it is preferable to create sub-word HMM's based on all training data.

To do the recognition task, the tool HVite performs the Viterbi algorithm on the HMM's. It is possible to include some constraints on the possible utterance by some kind of (finite state) grammar or a bi-gram model. Bi-gram models (or even N-gram) give the probability that a (sequence of) HMM model(s) is followed by certain next HMM. Grammar can be used to allow only a certain sequence of HMM's. More details on the use of grammars is given in the next chapter.

To evaluate the recognition performance of the created set of HMM's, the tool HResults analyses the recognition. It compares the output label files generated by HVite with the original reference label files of the test data. The following formulas are then used to display the results:

$$\text{Percentage Correct} = \frac{N-D-S}{N} \times 100\%$$

$$\text{Percentage Accuracy} = \frac{N-D-S-I}{N} \times 100\%$$

where N is the total number of labels, D is the number of deletions, S gives the Substitutions and I the Insertions.

Moreover, it is possible to generate a confusion matrix that shows the number of times a certain model was used to model (i.e. recognize) a certain phone. This can be used to see which HMM's have to be improved. An example of such a confusion matrix is given in appendix A.

Improving the Recognition

Once a (simple) speech recognizer has been build (i.e. a single mixture, mono-phone recognizer), a lot of options arise to improve the recognition.

- creating di- or triphones from the mono-phones. This are models that model a certain phone in the context of another (set of) phone(s). This will decrease enormously the number of models (e.g. the mono-phone model for /a/ will be replaced by di-phone models like /a/+/n/, /a/+/d/, etc. or by the tri-phone models /l/-/a/+/m/, /p/-/a/+/m/, etc.).

- introducing more mixture components in the Gaussians.
- parameter tying at different levels (e.g. useful when certain di-/tri-phone models are undertrained).
- clustering of the training data.

Chapter 3

Towards a Grammar for Spoken Telephone Numbers

To gain some insight in how people pronounce telephone numbers, a number of different grammars for complete telephone numbers are reviewed here, tested on 1000 annotated spoken telephone numbers. This will also give some ideas for the minimal grammar needed to describe a complete telephone number. More theory on natural language parsing with grammars can be found in [Covington-94]. The results concerning possible dictations should not be used in future set-ups, due to the fact that the telephone numbers were changed in November 1996. The Swiss Telecom (like others in Europe) made the transition to 10 digits, changing the situation drastically for possible ways of dictating it.

3.1 Usage of Grammars in HTK

As stated in the previous chapter, it is possible to define a finite state grammar in HTK to control the allowed sequences of HMM's (see [HTK-93]). Lets start with a simple example: For a sequence of one or more (French) digits, with an optional begin/end silence (`[sil]`), a grammar would look like:

```
(
 [sil]  < zero | un | deux | trois | quatre | cinq |
  six | sept | huit | neuf > [sil]
)
```

In this grammar ( ) denotes an expression, <.> denotes one or more times, | denotes a choice and [.] denotes 'optional'. More complex grammars can be build, consisting of sub-grammars. For continuous speech recognition, also the words have to be subdivided in their sub-word models. This also allows the definition of a word with different pronunciations (e.g. due to dialects). As an example consider the same grammar, rewritten for the continuous case using phoneme models and using one sub-grammar to represent all digits at once:

```
$N0 = WD_BEGIN%zéro zz ei rr (oo|au) WD_END%zéro;
...
$N0to9 = $N0 | $N1 | ... | SN9;

( [sil] | < $N0to9 > | [sil] )
```

HTK provides a tool, named HSPar, that allows to test a designed grammar on a set of sentences, stored in a file. Sentences that can't be parsed are displayed indicating the error point. As well it is possible to do the reverse direction; the tool HSGen generates a set of random sentences conform given grammar.

3.2 Tests Using Telephone Number Grammars

The HSPar tool is used in combination with a number of grammars, to reveal the user behaviour for dictating telephone numbers. The typical users behaviours that do not fit in the grammar are detected when the grammar is 'tested' on a set of user utterances. 1000 example items taken from the Swiss French Polyphone database, created at the IDIAP [Chollet-96]. This database consists of a number of responses to different prompts, including a prompt to dictate spontaneously a telephone number. The word descriptions[1] of these reactions were used as test items for a grammar for spoken telephone numbers, for example:

```
zero vingt six vingt et un septante sept trente et un.
```

Basic Test

In the basic test, a minimal vocabulary was tested. The vocabulary is based on Swiss French, the language spoken in this region of Switzerland. This leads to some extra words to be included in the grammar, because of the differences given below:

numeral	Swiss French	French
70	*septante*	*soixante dix*
73	septante trois	soixante treize
80	*huitante*	*quatre vingt*
85	huitante cinq	quatre vingt cinq
90	*nonante*	*quatre vingt dix*
97	nonante sept	quatre vingt dix sept

table. Differences between Swiss French and French.

In French the numerals 71-79 and 91-99 are formed by using the corresponding numeral from 11-19 as shown in the table. Their Swiss counterparts are formed regularly. The following grammar consists of all possible words in a telephone number:

```
$digit  = zero | un |  deux | trois | quatre  | cinq | six |
          sept | huit | neuf | dix | onze | douze | treize |
          quatorze  | quinze  |  seize | vingt  |  trente  |
          quarante  |  cinquante  | soixante  |  septante  |
          huitante  | nonante | cent | et;

( $digit )
```

This simple grammar reveals many typical speech recognition problems:

- change of order: 0.8% (different possibilities), for example:

```
025-63 16 16:
soixante trois seize seize au zero vingt cinq.
                              ^*** Error
```

[1]No information from time labels/pauses was incorporated.

- le/au in front of the number: 0.7%
- unintelligible/strange pronunciation: 0.6%
- Hesitation errors: 0.5%
- missing data: 0.4%
- non-numbers: 0.2%
- self-corrections: 0.1%, for example:

```
026-46 19 28:
quarante six, zero vingt six, pardon, quarante six dix neuf vingt huit.
                            ^*** Error
```

- verbal separation of area code and the rest: 0.1%, for example:

```
021-881 37 00:
zero vingt et un indicatif donc
                 ^*** Error

huit cent quatre vingt un trente sept zero zero.
```

Some of these problems can be solved by choices in the dialogue, for example, non-number models (le/au/euh (1.5%) etc.), explicit questionning for the area code (1%). The error sentences (3.6%) are not used testing the other grammars. In a dialogue application, these inputs will still give problems. It might therefore be an idea to create a grammar to capture false inputs and ask the caller to rephrase the telephone number, giving some extra information on the requested format. (this so-called directive error feedback will be discussed later).

Telephone Number Grammar

To be able to write a grammar for telephone numbers, an initial look at the database contents is needed to get an idea of the vocal lay-out of a telephone number. Here a number of database entries are given, together with the corresponding telephone numbers (manually determined based on the database entries):

```
793 02 37          - sept cent nonante trois zero deux trente sept
43 13 34           - quarante trois treize trente quatre
617 77 70          - six cent dix sept septante sept septante
625 26 87          - six cent vingt cinq vingt six huitante sept
022 - 784 26 29    - zero vingt deux sept cent huitante quatre vingt six vingt neuf
837 43             - huit cent trente sept quarante trois
41 13 55           - quarante et un treize cinquante cinq
33 32 31           - trente trois trente deux trente et un
01 - 850 02 49     - zero un huit cent cinquante zero deux quarante neuf
6 27 85            - six vingt sept huitante cinq
328 25 09          - trois cent vingt huit vingt cinq zero neuf
039 - 31 29 21     - zero trente neuf trente et un vingt neuf vingt et un
021 - 799 36 47    - zero vingt et un sept cent nonante neuf trente six quarante sept
0033-1-34 89 16 62 - zero zero trente trois un trente quatre quatre vingt neuf seize
                     soixante deux
```

It is observed, that people tend to speak telephone numbers in groups of digits (so-called 'numerals'). Also groups like '02' exist. Here a first problem arises. Based only on these transcriptions some numbers are ambiguous (the most logical telephone number is given). Moreover some international numbers are found (like the last one, which is actually a number in Paris). Based on these observations a more natural grammar, for Swiss telephone numbers is created, based on three numerals and an optional area code:

```
$dizaine = $N1 | .. | $N9;
$bloc = $N0to99 | $N0 $dizaine;

$nombre = [$N0 $N0to99] $N1to999  $bloc $bloc;
```

This also showed some outliers (6.2%):

- other grouping of numerals (2.7%).
- partly spoken digit by digit (1.1%).
- foreign numbers (1.1%).
- area code last (1.2%).
- non-number (0.6%).

Two numbers where found to be ambiguous and some users used another arranging, which facilitated reminding the number:

```
33 141 33
> trente trois cent quarante et un trente trois.
               ^*** Error
```

Some of these errors might be corrected, by allowing more combinations in the grammar. However, more ambiguous cases may arise from this. Choosing for a dialogue in which the groups (max. three digits) are separated, might also help. The found non-numbers were removed (0.6%). This grammar seems good enough to parse Swiss telephone numbers in general, therefore it was also used in the HMM test, described in 3.3).

Final Test: Explicit Grammars

After the above mentioned removals, the remaining 958 numbers were classified into different classes by using explicit grammars. This gives some insight in the way people divide the numbers into groups (only for 6 and 7 digits the counting was explicitly done. For 5 and 8 digits, only a small number of examples were found and the subdivision was not reviewed). The following definitions were used:

```
$dizaine = $N0 | .. | $N9;
$bloc3 = [$N2to9] $N100 [$N1to99];
$bloc2 = $N10to99 | $N0 $dizaine;
```

This resulted in the following classification of the telephone numbers:

- 11 International numbers, using

  ```
  $nombreINT = $F0 $F0 $N1to99 < $N0to999 >;
  ```

- 35 numbers with only 5 digits, using

```
$nombre5 = [$N0 $N1to99] $bloc3 $bloc2 |
           [$N0 $N1to99] $bloc2 $bloc3 |
           [$N0 $N1to99] $N1to9 $bloc2 $bloc2 |
           [$N0 $N1to99] $bloc2 $N1to9 $bloc2 |
           [$N0 $N1to99] $bloc2 $bloc2 $N1to9  ;
```

- 442 numbers with only 6 digits, using

```
$nombre6 = [$N0 $N1to99] $bloc3 $bloc3 |          (10)
           [$N0 $N1to99] $bloc2 $bloc2 $bloc2;   (432)
```

- 458 numbers with 7 digits

```
$nombre7 = [$N0 $N1to99] $bloc3 $bloc2 $bloc2 |        (448)
           [$N0 $N1to99] $N1to9 $bloc2 $bloc2 $bloc2 | (226!)
           [$N0 $N1to99] $bloc2 $bloc3 $bloc2 |          (1)
           [$N0 $N1to99] $bloc2 $bloc2 $bloc3;           (2)
```

- 7 numbers with 8 digits, using

```
$nombre8 = [$N0 $N1to9]  $bloc2 $bloc2 $bloc2 $bloc2 |
           [$N0 $N1to9]  $bloc3 $bloc3 $bloc2 |
           [$N0 $N1to9]  $bloc2 $bloc3 $bloc3 |
           [$N0 $N1to9]  $bloc3 $bloc2 $bloc3;
```

- 6 numbers were spoken digit by digit
- 16 numbers with other problems as shown earlier
- 2 non-numbers where still detected (due to more explicit grammar).
- 977 numbers in total instead of 958!

These results show some trouble with this approach for grammar tests. The total number of telephone numbers, that can be parsed by a set of grammars is larger than the number of telephone numbers that can be parsed by a disjunction of those grammars. This is found within a group (like for the 7 digits case), or from different groups resulting in a total of 977 telephone numbers, whereas 958 were used. This indicates, that some phone numbers are ambiguous (i.e. they can be parsed by more than one grammar). This quite logical in French, caused by the use of composite words for 70, 80 and 90 as shown earlier and the possible separation of the decade and the units (`trente quatre` is 34 or 30 4). Even a combination is possible as shown by:

`trente quatre vingt dix = 34 20 10 or 30 80 10 or even 30 90 !!`

A way to resolve this problems is found by looking at human dialogues. A person dictating a telephone number would like to avoid being ambiguous. Inserting clear pauses between the numerals or even waiting for a reaction of the other party will help to increase the intelligibility. This observation should also be used to parse from recognized numerals (still written in words) to the digit by digit format, used to make the desired connection.

3.3 Used Speech Recognition Techniques in VoicePhone

Based on the findings in the previous section, an initial grammar was chosen for VoicePhone. The speech recognition task performed consists of recognizing (parts of) telephone numbers. A first attempt to built a set of models for this, used phoneme based word models. The used phoneme models where obtained from a general set of training data. This gives to rather poor results: < 20% correct on a set of complete telephone numbers. To improve this, HMM's modeling a complete word (e.g. 'six') were created, using telephone number entries from the Polyphone database ([Chollet-96]). In these models the following parameterisation is used for the MFCC coding: (p. 27 HTK USER-manual and p. 19 HTK Ref. Manual)

```
setHCOERCE=MFCC_E_D_A
HCode -e -m -k 0.96 -h -t -f 10.0 -w 25.6 -q 2 -l 16 -n 12 -p 24
```

The used HMM models were tested with a grammar for complete Swiss telephone numbers obtained in the previous section (only the highest level is given):

```
$dizaine =  $N0 | $N1 | .. | $N9;
$bloc = sil_mil%% ($N10to99 | zéro $dizaine);
$xdizaine= un | (vingt | trente | quarante | cinquante | soixante |
                 septante | huitante | nonante) ($et1 | $N2to9);
$nombre = [zéro $xdizaine sil_mil%%] $N0to999 $bloc $bloc;

( sil_av%% $nombre sil_ap%% )
```

In this grammar `$N0to999` represents all possible numbers between 0 and 999 in both French and Swiss French. For the test, a set of 995 telephone numbers from the database was used. The output of HResult (the complete confusion matrix can be found in appendix A):

```
------------------------ Overall Results ---------------------------
PHRASE: %Correct=69.05 [H=687, S=308, N=995]
PHONE:  %Corr=93.56, Acc=92.23 [H=7457, D=143, S=370, I=106, N=7970]
--------------------------------------------------------------------
```

This output shows some typical connected word recognition problems. The recognition rates on PHONE's (in this case, these are single words, e.g. 'six') are acceptable, but this rate decreases rapidly, if they are combined to PHRASES (e.g. a complete telephone number). In general a formula can be written for a combination of N PHONE's to a PHRASE:

$$\texttt{\%Correct}\text{ on PHRASE} = (\texttt{\%Corr}\text{ on PHONE})^N$$

This general formula of course only holds if at each place in the PHRASE, all PHONE's are allowed. The grammar can decrease this number of possible PHONE's and therefore increase the recognition rate.

Moreover this chapter showed that the used training data also contains entries that are not allowed in the grammar (and can't be recognized anyway!). After leaving out these training data the recognition rate on telephone numbers increased to 75%.

Real-time Speech Recognition in VoicePhone

In a dialogue system a fast speech recognition algorithm is necessary. VoicePhone is therefore equipped with a special implementation of the Viterbi algorithm, that starts as soon as the first speech data are recorded. At that time the HMM's are already in place, due to starting of the speech recognition before the prompt is played. This allows a nearly real-time speech recognition when performed on a fast computer. More information on the project for this real-time feature extraction and recognition (FERRT) can be found in [Bornet-95].

During the test with VoicePhone, the program worked on a somewhat slower machine that was not cleaned from other processes. This caused a worse performance on recognition time, because the recognition processes had to 'fight' to get all computing time (although it had 'top priority'). It is therefore not exactly known how fast the system can work, but calculations in [Bornet-95] show that real-time recognition should be possible.

Proposed Improvements

A number of changes can be considered to improve the recognition rate on complete telephone numbers.

- use di-/tri-phone models which make it possible to model different pronunciations of the same word. Care should be taken to use data from spoken telephone numbers, leading to models closer to the input of the system. However, this is of course limited by the amount of training data available[2]
- It should be possible to store all possible area codes in the grammar, thus only allowing valid area codes (around 21 for entire Switzerland).
- In parallel to this grammar, some other grammar could detect for non-valid strings of numerals (e.g. an incomplete phone number). This can filter out false recognitions, due to insertions, by replacing them with an error signal like 'not a telephone number'.
- As well the grammar should be able to parse digit strings for people who are used to dictate telephone numbers digit by digit.
- Moreover the effects of the transition to 10 digits should be studied and, if necessary, incorporated into the grammar.
- The above mentioned grammar is only able to parse Swiss telephone numbers. For international numbers or information numbers, a separate grammar should be developed.

[2]di- and tri-phones risk to be undertrained, because of the lack of speech data for certain not often occuring ones.

Chapter 4

Design of User-Friendly Dialogue Systems

During the last few years, the ability of recognizing speech by computers has led to a growing interest in developing systems using this new way of Human-Computer communication. To be able to create systems that can actually be used by the general public some extra steps should be taken. This chapter will start with a general overview of the design of user-friendly interfaces, followed by some first considerations on dialogue systems. The last part of the chapter gives some basic principles that should be followed during the design of a dialogue system.

4.1 User-Friendly Design

The use of computers by non-experts has led to research on Human-Computer Interaction. This highly interdisciplinary field, with contributions from psychology, computing science and cognitive science in general, can be very useful for setting up applications which are to be used by the general public (i.e. naive users). Additional info on the field of Human-Computer Interaction can be found in [Dix-93] and [Waern-89].

In this field it was found, that it is not possible to give an exact strategy for designing systems that are to be used by humans. Rather a set of general design principles is given that should be included during the design process:

- Learnability
- Flexibility
- Robustness

These design principles will be illustrated and sub-divided in the next sub-sections. Special attention is given to the usage of them in the design of dialogue systems.

Learnability

Every system has to be used for the first time by the user. A first impression can matter a lot in the further usage of the system. The *learnability* of the system deals with the question how much effort the user has to spent to get familiar with the system. One can think of *consistency* in the system, as well as the *predictability* (is it easy to predict the effect of a future action?) More important during the first time is the *familiarity* with the system (is it possible for the user to use knowledge and experience interacting with the new system?). For the special case of speech recognition applications, users will have much experience using speech in conversations and interaction with other people. Knowledge from human-human voice communication can be of great help in setting up an easy-to-handle dialogue.

Flexibility

Human-Computer interaction is based on the exchange of information. The *flexibility* of the system is related to the variety of ways in which this can be done. In the current vocal server application, this can mostly be related to the dialogue itself. Can it be initiated by the user, or has the system itself the highest influence? Some advanced dialogue systems, for example for travel information, can greatly benefit by allowing the user to decide the sequence of his input to the system. However, other applications (like telephone number recognition) might benefit more from steering the user.

Robustness

The main goal of any computer program should be to allow the user to accomplish his tasks as successful as possible. This is certainly true for voice driven systems. Most users will base their impression of the system by this principle. Robustness can be achieved by giving the system some of the following features, which will also be encountered in the more detailed sections on designing a vocal server: *Observability* allows the user to evaluate the internal state of the system. This can give the user important information how far he is from reaching his goal. Using the *recoverability* of the system, he should be able to correct false information in that state. In speech recognition system this will consist of providing feedback to the user of what has been understood and allowing a sub-dialogue to correct possible errors. Related to this last feature is the *responsiveness* of the system, which indicates how the user perceives the rate of communication. Especially telephone applications have the danger of giving the user the feeling of 'trying to communicate with a stone'.

4.2 User-Friendly Dialogue Systems

Dialogue systems should enable people to interact with a computer in a familiar way (i.e. using speech). This has led to a lot of difficulties in the design of dialogue systems. This is mainly due to differences between the speech currently understood by speech recognition techniques and the way people tend to use speech. This section will give three important points of view on a vocal server system: A task model, a user model and a model of the capabilities of the system. These points of view will be exemplified by approaches in other dialogue applications and problems encountered with them. Special attention will be given to hints useful for the design of a telephone recognition application. This section will review the most important issues encountered in articles read for this research. Very useful were [Kamm-94] and [Karis-91], which give a general overview of the field of dialogue design.

Task Model

Dialogue

The main purpose of an interactive vocal server is to help the user to achieve his goal as efficiently as possible. The amount of information needed to accomplish this task, can form a basis for the dialogue. This dialogue can be a simple yes/no dialogue, but this will neither be satisfying to the user nor sufficient for many applications (e.g. a train travel information system). A variety of more or less complex dialogues could be used, based on the task model. The simplest approach would be some kind of fill-in-form, to obtain all data needed:

system:	Which month will you leave?
user:	*November.*
system:	Which day in November?
user:	*15th.*
system:	So, you want to leave on November the 15th?
user:	*Yes.*

Dialogue 1. Fill-in form based (note the implicit feedback and approval of the recognized month).

This is however very time consuming in some applications. It would be much more human-like to leave the information order up to the user, as well as the amount of information given in one sentence. Missing information can be prompted for by the system:

system:	Hello, how can I help you?
user:	*I want to make a journey by train. I want to travel from Martigny to Groningen on Friday the 20th September.*
system:	What time do you want to leave Martigny on the 20th September?
user:	*Around eight o'clock.*
system:	At eight o'clock in the morning?
user:	*Yes.*
system:	I'll look up the connection for you. Wait a moment, please.

Dialogue 2. A more natural way; the order of information is chosen by the user.

Depending on other factors the dialogue can thus be either highly structured and guided by the system or be more flexible in which the user and the system change roles from directive to passive participant in the dialogue.

Two Other Dialogue Approaches

The study of human-human dialogues can give ideas about which information flow humans will be used to. Care should be taken not to model exactly the human-human dialogue in a system. It has been observed, that humans interact quite differently with such a system. Still, some useful information can be obtained about the syntax of possible answers, suggestions for prompts and the set-up of the vocabulary to be recognized.

Starting from the system, one could build a system based on an existing automated service based on key-strokes. In these kind of systems, typically a menu of items is presented in which each item is related to a digit. By pressing this digit, choices can be made. In a spoken version of this system, it would be more easy for the user to chose by keywords, rather than the digits. Keywords are much easier to remember because of their correlation to the user's task knowledge. The vocabulary might increase, but during each user response only a subset of it has to be recognized.

Multi Modality

Another important issue for the information exchange is the multi modality of the system. For some speech applications, more than one modality is available for the interaction, be it keystrokes, visual feedback, etc.. Some part of the information exchange might even be performed more efficiently in another modality (as would be the case for telephone numbers). In the application under consideration, this usage of the keypad is not available, so that it also works for rotary telephones. In general, a telephone application should always allow the user to switch back to a human operator.

Errors and Costs

A last point related to the task, are the costs of errors and elaborate dialogues/recognition tasks. These can be direct (e.g. increased holding time in a telephone application) or indirect (e.g. negative promotion for the system). The costs certainly should not outnumber the savings obtained by providing the service. Moreover, this favours the use of fast speech recognition.

User Model

For a service to be user friendly it should at least be robust to all kinds of input. It is therefore necessary to know how the user will react to the system. The user model[1] should give information on the expectations and expertise of the user.

Novice vs. Expert Users

People using the system for the first time, won't know what is expected from them and they might want some extra help and introduction to the system. Expert users should however be able to skip this to be able to move more efficiently through the dialogue.

Speech Behaviours

The evolution of language as an instrument for conversations has resulted in the allowance of very complex dialogues. Many of the human speech behaviours take place unconsciously and are over-learned, so that it is not easy to overcome them.

First of all, people tend to speak in a *continuous manner*, the borders between words only exist in the listeners mind. Humans perform very well in separating the flood of phonemes into words, but this isn't noticed. (When listening to a unfamiliar language this is clear; most European listeners perceive Japanese as one fluent stream of phonemes.) Analysis of the speech signal shows definitely, that no silences between words exist. This will also cause so-called coarticulation effects between words: The last phoneme(s) effect the pronunciation of the first phoneme(s) of the next word. This is one of the major issues in the field of speech recognition, when dealing with continuous speech. In earlier applications for number recognition, 35% of the users did not speak the digits in an isolated manner although they were instructed to do so. This shows the difficulty people have to speak in a non-natural way.

Secondly, people utter many non-speech sounds that could easily be interpreted as a word from the vocabulary. Sounds like breathing, lips opening etc. are unavoidable and even amplified in telephone systems. Besides these sounds also hesitations (like 'uhh' or 'um') are very common in human dialogues, but could easily be recognized as 'un' (i.e. 'one' in French). Moreover an utterance like 'uhh' could be considered by the system as a complete dialogue turn. In [Strik-96] it was found that modeling of these sounds was needed in the implementation of a railway time table system. Another way to avoid these problems is the use of 'keyword spotting' techniques. In that case only keywords (based on the application) from the input sentence are recognized and used. An example:

user:	*I want to make a trip from Groningen to Martigny.*
system:	
(recognized:)	(from Groningen) (to Martigny)
(knowledge:)	`departure(groningen,Time,Date).`
	`arrive(martigny,Time,Date).`
(response:)	When do you want to travel from Groningen to Martigny?

Dialogue 3. An example of 'keyword spotting'.

[1] It should be noted, that the term 'user model' does NOT refer to computational models of the user as used in advanced design of user interfaces.

Besides these hesitation problems, people also perform self-corrections during an utterance. This might be catched by modeling the words used for correction (e.g. 'pardon'). Other physical problems with dialogue input are background noise, third person interference, etc..

Dialogue Behaviours

Not only at the level of speech, but also at the level of dialogues people tend to use experience from human-human dialogues. This can also cause serious problems.

People are used to conversation partners, who are able to understand a new utterance, although they haven't finished their own sentence yet. This is often used to answer a question as soon as it is understood (redundancy of language). However this 'talk-over' behaviour might not work in a computer based dialogue, which is running on a serial computing device without an ability to do speech synthesis and recognition at the same time (this can be partly captured in a multi-task environment). In [Karis-91] this behaviour was observed in 14.4 % of the cases in a field test. A suggestion could be to use of the known 'wait for the beep' used in answering machines. However, in other applications this has proven not to decrease (rather increase) talk-over behaviour.

If one doesn't interrupt during a sentence, still some implicit turn taking signs exist, such as a pause. System designs should therefore comprise complete dialogue turns, without pauses in them, as well as being formulated short and clear.

System Model

The third model, which is important for the successful design of a dialogue system, should give an idea of the system capabilities. The current state of speech recognition techniques does not allow the variety of dialogues and utterances as encountered in human conversations. This is partly due to the processing techniques of natural language input and partly to the incorrectness of this input, due to speech recognition errors. A model of the system can be used to design a dialogue within the abilities of speech recognition.

The most important issue is the response time. As mentioned above, a system that needs too much time may irritate the users, as well as cost to much. This disables the possibility of elaborate recognition algorithms. Moreover, it is not alway necessary to have elaborate algorithms, this rather depends on the recognition task. A number of items concerning the different speech recognition tasks are listed here:

- Speaker dependent/independent: Speaker dependent recognition is much easier and performs better. However, this can't be used for a dialogue system serving for general usage by the public.
- Real-time speech recognition: The current speech recognition techniques are besides accurate also fast. If the appropriate computers are used, then the most 'simple' recognition tasks can be done in nearly real-time. This is of course highly favourable in a dialogue system.
- Isolated word: This is the best performed task, almost 100% recognition rate can be obtained on yes/no or single digits 0..9 (in isolated digit tasks, also strings of words may occur separated by pauses).
- Connected words: This refers to strings of connected words, however it is also used to denote the technique used in the speech recognizer itself (viz. connecting word models to perform recognition). In case of word strings, this task can be performed reasonably well, especially if a strict grammar can be used (decreasing the number of possible strings).
- Continuous speech: To recognize more elaborate sentences with a large number of possible words, it is no longer wanted to train the system for each word. Therefore sub-word units are recognized (phonemes or syllables) and combined to words in a grammar. This kind of recognition task is however very computational intensive.

- Keyword spotting: Another way to recognize elaborate sentences, is to recognize only keywords in the sentence that represent the query. This works very well in a dialogue system where the questions direct the user towards a certain type of answer. This technique needs an elaborate user study to include as many types of sentences as possible (an example of this was given earlier in this chapter).

Based on the user behaviour as described in chapter 3, the best option for a telephone number recognition (vocabulary of about 30 words), seems to be the connected words solution. The isolated word recognition would take too much time for the user and would moreover be boring (leading to human errors, instead of system errors). The exact implementation of this was described in section 3.3.

Another issue concerning the system capabilities are the dialogue prompts. Current speech synthesis techniques allow reasonable human-like speech, but the set-up of such systems (including prosody, intonation etc.) costs much time and effort. The alternative consists of using pre-recorded messages. This will certainly give a human-like voice (preferably a female voice, which is easier to understand), although problems may arise for feedback prompts. These prompts serve to give the user information on the outcome of the recognition and may therefore be different for each dialogue. In this case either a concatenation of pre-recorded words (i.e. the ones that are in the vocabulary of the speech recognizer) or a more sloppy (non-natural) speech synthesis might be considered. Both will be easier for implementation and computation but sub-optimal for the users.

4.3 Dialogue Design

The previous sections reviewed the different issues for a dialogue system. The constraints encountered, can still lead to a well performing dialogue system, when care is taken in the choices made in the set-up of the system. More details and examples of dialogue design can be found in [Kamm-94] and [Karis-91] (a theory concerning a set of principles for designing human-machine dialogues is described in [Bernsen-96]). First an example is given, what kind of dialogues could result from just implementing some arbitrary dialogue:

system:	Please say the telephone number.
user:	*Zero twenty six [pause] seven hundred twen..*
system:	Please repeat.
user:	(louder) *Zero twenty six seven hundred twenty one seventy seven thirty two.*
system:	Zero forty seven, please continue.
user:	What's going on here? [hangs up]

Dialogue 4. An example of an unsuccessful dialogue interaction.

This test was performed for one of the first versions of the VoicePhone system, which was actually only done to test the system itself (not the dialogue). The instructions for this test should have been: "say a telephone number, divided in parts; after each part the system will reply what it has understood". Even if the instructions were given, people didn't read them or forgot them once they were interacting with the system. The system simply interrupts the input (after detecting a short pause after the first part), doesn't understand it, and asks to repeat. The user has no idea how to improve the recognition and simply speaks louder. This time something is (wrongly) recognized and the system asks the user to continue, which he doesn't understand at all and hangs up.

This can be improved by creating prompts that give the user better information on what kind of input is expected. Moreover, in case of an recognition error, there should be some way to recover from that. The second confirmation is even more confusing, because it does not at all resemble the input. A better dialogue is proposed in dialogue 5:

system:	Hello, please say the telephone number part by part, starting with the area code.
user:	*Zero twenty six.*
system:	I understood 'zero twenty six'. Continue if this is correct.
user:	*Seven hundred twenty one.*
system:	I understood 'seven hundred thirty one'. Continue if this is correct.
user:	*No, it is not.*
system:	Please repeat the second part.
user:	*Seven hundred twenty one.*
system:	[Even if again 'thirty one' is recognized, this option is skipped]
	I understood 'seven hundred twenty one'. Continue if this is correct.

Dialogue 5. A more comprehensible dialogue.

The most important issue in the dialogue flow is to maintain the usability of the system, therefore the following usability principles should always be kept in mind:

- simple/natural dialogue
- speak the users language
- be consistent during the dialogue
- minimize the user's memory load
- provide feedback
- provide clearly marked exits
- provide good error messages
- prevent errors

During the following subsections, a number of issues will be stated relating to their part in the dialogue.

Dialogue Flow

One of the items already mentioned is the habit of people, not to wait till the end of a prompt ('talk through'). This might be detected by the fact that the first word is not comprehensible as well as the existence of a high energy at the start of the input. The dialogue might then use a sub-dialogue asking to repeat the input *after* the end of the prompt.

Dialogues should be directive and clear. Even a novice user should understand what the system expects as an input. Care should be taken to keep the prompts short. This will avoid 'talk through' behaviour of expert users. Moreover, long prompts take more time. The combination of directive, clear and short is not an obvious combination. In most cases some way in between should be chosen (e.g. see [Marcus-96]).

If possible it should be allowed for the user to take over the dialogue steering. When more than one input item is needed in the program (e.g. a Time and Day of departure), he should be allowed to decide the order of those item himself. Missing items will than be detected by the system and asked for at a later point.

Instructions

It might be useful to have some build-in sub-dialogue to allow users to ask for help. (e.g. by using the keyword 'help'). Also a list of options could serve both novice users as well as expert users who didn't remember all possible items at some point in the dialogue. An example is given below:

- Help Text: *The following commands are available; 'Help' gives this text, 'List' gives the possible options, 'Quit' ends the service, ...*
- List Text: *Available commands are: Help,Quit,...*

If the sentence order is reversed, so that the keyword comes at the end of the help text items, the user will find it easier to obtain the keyword representing the desired request (e.g. *To quit, say 'Quit'*). A nice option is to automatically give a short hint if the user doesn't give an input because he doesn't know what to do.

Feedback and Confirmation

It is very important to provide the user with the information, already obtained by the system from the user. This allows users to feel at ease with the system (confidence in its recognition capabilities), as well to have the ability to correct this information if needed. In case of an error, the system should have the possibility to do a backtracking to a stable point in the dialogue. Care should however be taken that the user is not overloaded by confirmation messages. Rather the use of implicit confirmations is preferred (i.e. the recognized utterance is used in the phrasing of the next prompt).

In case of ambiguities it is better to ask for a direct confirmation of one of the options, rather than reprompting for the input. Care has to be taken, not to give ambiguous feedback! Consider this example from VoicePhone.

user:	*Twenty three hundred forty.*
	(= 23 140 or 20 340)
system:	I understood 'twenty three hundred forty'.

Dialogue 6. Feedback that doesn't solve an ambiguity.

This could be improved, by inserting pauses in the feedback between the separate numerals.

user:	*Twenty three hundred forty.*
system:	I understood 'twenty three [pause] one hundred forty'.

Dialogue 7. Feedback that does solve an ambiguity.

When asking for confirmation, the user might give immediately a repetition of the input, after saying 'no'. Either the system should interrupt the user and reprompt for the input or perform directly a recognition task on this version of the input. In [Niimi-95] a model is presented to evaluate the use of three different confirmation techniques; direct confirmation, indirect conformation and a prompt to speak again (if the recognition is below a certain threshold).

Error Correction

Besides the error correction mentioned above, some other issues can be mentioned. It has been found in [Marcus-96], that users tend to adapt their behaviour. This cooperation can benefit enormously from providing feedback on the probable cause of the error (e.g. "please speak louder"). Moreover, the system should provide an emergency exit, which connects users to an human operator.

4.4 Iterative Design

This chapter showed many user related problems in dialogue systems, as well as a variety of ways to attack these problems. The main issue during the design of dialogue system is to find an optimum, comprising all these strategies. This can't be done in the first version of a system, so an iterative design process is proposed in [Karis-91], which verifies the system as early as possible, using user experiments.

1. Consider the human-human variant of the dialogue. How do they communicate? This won't give the exact dialogue to be modeled (users adapt for speaking to computers, moreover a human-like dialogue is yet to difficult for a computer), but it does point out some possible problems and an idea of the necessary information flow.

2. Design a first dialogue and test this, using a "Wizard of Oz" system. This technique replaces the speech recognition algorithm by a human listener (the Wizard), who enters the recognized dialogue turns in the dialogue system. The user (i.e. caller) will still have the impression that he talks to a computer (note that *only* the recognition is done by a human). This first test can easily be done in the laboratory environment, saving a lot of time.

3. Based on the findings of this first experiment, update the system and perform a Wizard of Oz experiment with naive users (i.e. users from outside the lab, being representative for the future users of the system). During this phase also the speech recognition errors should be simulated, showing how well users succeed to correct these errors. Moreover these tests should give a first impression on how people perceive the system.

4. Use the reactions and observed behaviours from the previous tests to chose the final dialogue, including error correcting sub-dialogues. Also the speech recognizer should be included. To train the speech recognizer it might be handy to use the data obtained during the experiments. They should be considered to contain the same kind of input as encountered later (e.g. including background noise).

5. Test the final version of this system, including the speech recognition. As in the previous test, it should still be possible to intervene manually if something goes wrong (e.g. when nothing can be understood correctly).

During all these five stages, the reactions of the people should be well monitored. Moreover, it is very important to ask the test persons to describe their feelings with the system. This can for example be done, by arranging call-back interviews (see [Dybkjaer-93]), or fill-in forms. Especially, care has to be taken for the difference between lab conditions and real world conditions[2] (although better noise models might help, see [Kuroiwa-95]). The system should be evaluated using the responses from the test users, rather than only taking into account the recognition rate of the system.

[2]The speech recognition in VoicePhone is even trained using real world data (ie. a database created from phone calls). This should avoid most problems.

Chapter 5

VoicePhone, an Interactive Vocal Server

In the previous chapters, the basic background for the development of dialogue systems has been presented. This chapter describes the actual design of a dialogue system that enables the recognition of a spoken telephone number over the public telephone net. The designed system is implemented in a program called VoicePhone. More details on the implementation are given in [Jongebloed-96.2]. An additional module was included to perform Wizard of Oz experiments, as described in the next chapter. The structure of the flow charts explaining the different dialogue structures was developed at IDIAP and described in [Bornet-97]: A brief description is given in appendix B.

5.1 VoiceCard a First Approach

The basic ingredients of the dialogue system were found in the interactive vocal server `InfoMartigny`, that was previously developed here at IDIAP. This program was meant as a demonstration of real-time speech recognition techniques (more information on the project for real time speech recognition can be found in [Bornet-95]). The program enables the users to ask information about spare time topics in Martigny (e.g. cinemas, museums). For the speech recognition, the 'word spotting' technique was used. The dialogue consisted of an infinite loop, asking for topics. For some topics, extra choices were built in (e.g. two different cinemas). It was also possible to leave a spoken message and ask for a list of possible topics at each point in the dialogue. More information on InfoMartigny can be found in [Bornet-96].

Design Considerations

For a first implementation of a dialogue system for recognizing telephone numbers, a program called VoiceCard[1] was written. This first implementation mainly served to test the different features of the program, especially the duration of the different processes in the program (e.g. the speech recognition). A very simple dialogue was chosen, which asks the user to say a telephone number piece by piece, starting with the area code, as shown in figure 5.1. For each turn the speech recognition consisted of recognizing a numeral between 01 and 999. More information on the speech recognition can be found in section 3.3.

The idea behind the dialogue is based on the observations during the tests of different grammars, that people are used to dictate telephone numbers grouped in numerals. The first tests of the speech recognition on complete telephone numbers turned out to give very bad results ($< 20\%$!, see section 3.3). This would suggest to perform recognition on shorter numeral strings, for which the recognition rates would be higher. This is based on the assumption that the probability of correct recognition of the

[1] The name VoiceCard was chosen, referring to the application of the Swiss PTT's Voice Telephone Card. In a later version this name was changed.

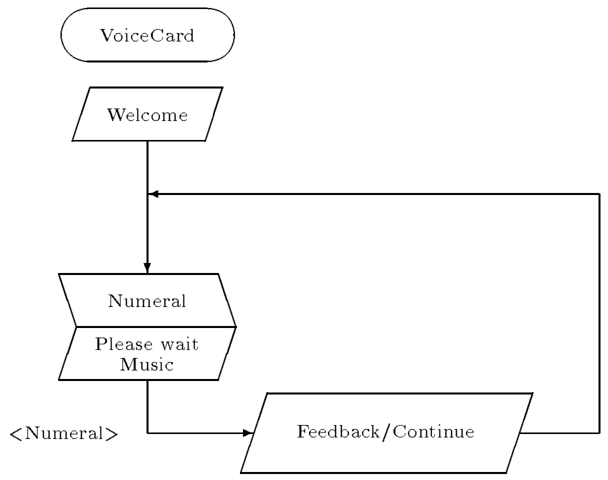


Figure 5.1: Dialogue flow of VoiceCard.

different numerals is independent, so that the probability for correct recognition decreases when more numerals are considered. Due to improvements in the speech recognizer, the program could eventually be implemented with an average recognition rate of 95% per single word, leading to 75% correct recognition for a complete telephone number (on average consisting of ten words). Another reason for the separation of the telephone number over multiple dialogue turns, is the facilitation of error correction. After each numeral obtained from the caller, the system gives a feedback of what has been understood. The user can then immediately correct an occured error by repeating the last numeral instead of a complete telephone number. However, this error correction sub-dialogue is not implemented in VoiceCard.

Details on the Dialogue

An example dialogue with the system would look as follows[2]:

system:	Bonjour, bienvenue, ici la téléphoniste automatique. Vous pouvez dicter le numéro de téléphone désiré, morceau par morceau. Commencez s'il-vous-plaît avec l'indicatif.
user:	*Zéro vingt six.*
system:	Un instant s'il-vous-plaît. "`Music`".
system:	J'ai compris "zéro vingt six". Continuez s'il-vous-plaît.
user:	*Sept cent vingt et un.*
system:	Un instant s'il-vous-plaît. "`Music`".
system:	J'ai compris "sept cent vingt et un". Continuez s'il-vous-plaît.
:	:

Dialogue 1. An example dialogue for VoiceCard, asking to spell the telephone number piece by piece.

The prompts, used during the dialogue (as well as the feedback), were recorded over the telephone, using a very simple dialogue system called `TextRecorder` (see appendix E). For this first version I used my own voice, which was considered to be good enough. In a later version, however, the voice of a native speaker should of course be taken. Preferably, this should be a female voice, because it is more comprehensible over the telephone.

The dialogue turn of the user is ended, using a silence detection; if a silence in the input is detected which is longer than a certain threshold, the system continues with the next prompt. This threshold was

[2]The dialogues in this section will be annotated in (Swiss) French, the original language of the program. A translation of the dialogues can be found in appendix C.

estimated based on some field tests. For numerals the silence period could be taken very short because people are used to speak each numeral as one 'word' without any pauses. Longer periods caused people to continue because the system didn't "take his turn". This shows the working of turn taking in human dialogues as discussed in section 4.3.

The dialogue continues with the message "Un instant, s'il vous plait.", to notify the user, that the speech recognition is in progress. During the time the speech recognition is being performed, the caller listens to a music fragment. This gives the user the feeling the system is still working and reduces the perceived waiting time.

When the speech recognition is finished, the recognized numeral is fed back to the user. This system was not yet equipped with some error sub-dialogue, but should be initiated here to repair the error as soon as possible. The user could be assumed to correct only if necessary. This gives an implicit approval of the recognition outcome. The feedback to the user is created by combining a sequence of audio files, each containing one word from the list of words that were included in the speech recognition vocabulary. This was perceived to be better than synthesized speech, without any prosodic information. The straightforward concatenation of the audio files lead to responses, that couldn't be understood by the callers. Most problems occured in the case of a recognition error. It turned out that people were very capable of understanding word strings building a valid numeral, especially if it was the correct numeral. Word strings on the other hand, that were not a numeral (e.g. 'six eight seven'), led to problems[3]. Some manually inserted pauses already showed an increase in comprehensibility.

General Impression

The general impression of the system turned out to be less positive than expected. The time, necessary to recognize a single numeral turned out to be rather long, which is undesired for telephone applications. Moreover, the danger arises that people continue with the wrong numeral. This might be because they 'hear' the telephone number continue in their head, thus skipping a numeral, or even because they completely forget the last spoken numeral. This last case is especially unfortunate if the speech recognizer makes an error, causing complete confusion for the user. As stated in section 3.3, the system has not yet been tested on maximum performance (i.e. running on its own on a more powerful computer). It might turn out, that a very fast system could avoid the problems with the long duration.

Another problem however with the speech recognition, is the recognition performance. Due to the relatively large influence of noise, before and after the numeral, the speech recognition algorithm showed insertions of models that are close to the silence models (e.g. 'sept cent quarante cinq' (745) instead of 'quarante cinq' (45), or 'trente six' (36) instead of 'trente' (30)). This might be improved by better background noise models.

A dialogue flow problem would be the end-of-number detection. In fact, the dialogue should be finished as soon as a complete telephone number is composed from the correctly recognized digits. For Swiss telephone numbers, this can be easily estimated by counting the number of digits in the number, which should be ten (the Swiss Telecom changed all telephone numbers to ten digits, but during half a year the old numbers still exist). However, this is not possible for international numbers. In that case, an special grammar or some kind of user signal should be included (e.g. "ready"). Information numbers form a third type of telephone numbers. They can be treated by storing them in a database, allowing to search if a number is already finished, because only a limited number of them have less than the standard 10 digits (e.g. the weather forecast is '162').

5.2 VoicePhone

The problems with the first approach, led to a new dialogue in which a complete telephone number is requested. This 'allows' relatively longer recognition times; people take for granted that a complete

[3]Of course the response should only be allowed to contain numerals, because of the speech recognition grammar. Only in an early version the most general grammar was chosen (i.e. all possible sequences of words were allowed).

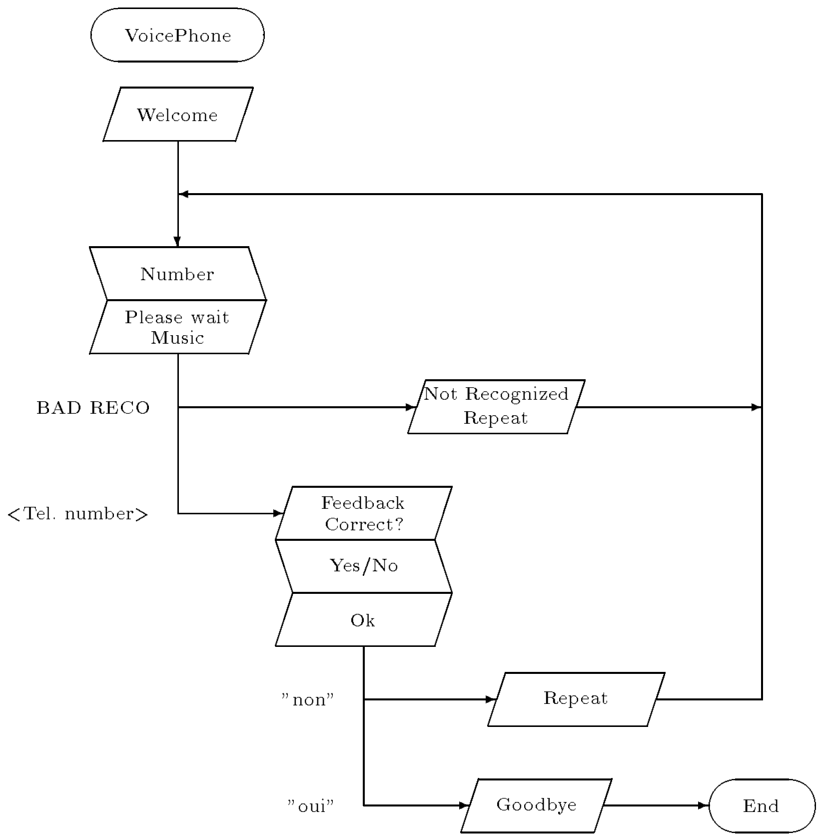


Figure 5.2: Dialogue flow of VoicePhone.

telephone number could cost some time to be understood by a computer, rather than expecting a numeral to cost a lot of time. To provide a more intelligible dialogue, the voice of a female native speaker was used for the dialogue prompts. Only the semi-synthesized feedback was left intact. An overview of the new dialogue is given in figure 5.2. An example dialogue with the system would look as follows:

system:	Bonjour, bienvenue, ici la téléphoniste automatique. Vous pouvez dicter le numéro de téléphone désiré. Commencez s'il-vous-plaît avec l'indicatif.
user:	*zéro vingt six [pause] vingt et un [pause] septante sept [pause] trente deux.*
system:	Un instant s'il-vous-plaît. "`Music`".
system:	J'ai compris "zéro vingt six [pause] vingt et un [pause] septante sept [pause] trente deux". Est-ce correct?
user:	*Oui.*
system:	Merci pour votre appel. Désolé ce système est seulement un test et ne peut pas établir le contact. Vous pouvez maintenant raccrocher.

Dialogue 2. An example dialogue for VoicePhone, asking for a complete telephone number at once.

As shown in figure 5.2, two different error backtrack loops are included: First of all, if the speech recognition does not succeed in recognizing the input, the system reprompts immediately for the telephone number. The other case occurs when the user does not agree on the outcome on the speech recognition. This also leads to a prompt to rephrase the complete number. An example of both backtrack loops if given in dialogue 3:

user:	*Zéro vingt six [*`long pause`*] vingt...*
system:	Un instant s'il-vous-plaît. "`Music`".
system:	Je n'ai pas compris votre demande. Répétez s'il-vous-plaît le numero de téléphone désiré.
user:	*Zéro vingt six [pause] vingt et un [pause] septante sept [pause] trente deux.*
system:	Un instant s'il-vous-plaît. "`Music`".
system:	J'ai compris "zéro trente six [pause] vingt deux [pause] septante sept [pause] trente deux". Est-ce correct?
user:	*Non.*
system:	Répétez s'il-vous-plaît le numéro de téléphone désiré.

Dialogue 3. Error correction in VoicePhone.

General Impression

This dialogue appeared much easier to handle for users. A surprising result was the rather small decrease in recognition performance. This may be caused by the relatively smaller influence of background noise surrounding the utterance. Moreover the grammar was very well able to handle small pauses between the different numerals. Another surprising result was the relatively short time (compared to VoiceCard) needed for the speech recognition. This is explained by the fact that people will spend relatively less time on pronouncing the complete telephone number, compared to the summation of separate numerals. It is also caused by the dialogue turn consists mostly of speech (instead of silence). However, it should be tested how long the pauses between the numerals are. People might even be encouraged to speak rather fast, with smaller pauses, should this lead to better recognition rates. To perform this test, a Wizard of Oz environment was created for the system, which is described in the next chapter. The recorded dialogue turns, obtained during the experiments could be used to estimate the variance in the input. One of the other interesting issues in these experiments will be the way people adapt, when they are asked to repeat the number in case of a recognition error.

Chapter 6

Wizard of Oz (WoZ)

After a first test with VoiceCard it was decided, that the time needed to perform the speech recognition allowed the usage of a Wizard of Oz module. This chapter describes the design of a simple Wizard of Oz interface needed to perform the experiments.

6.1 Wizard of Oz Experiments

To test a dialogue system, the best approach is to perform the first user tests as soon as a first dialogue is created. Most of the time the speech recognition needed for the dialogue is not yet available, so it is replaced by a human operator. This human operator is called the Wizard; people using the system will think they operate with a real speech recognizing system but actually it is a human (conform the story, in which the Wizard of Oz creates an fake appearance for the people, as described in [Baum-00]). This is accomplished by using as much as possible the envisaged dialogue system, including prompts, and feedback procedures. The output of the Wizard of Oz tests will be threefold; first of all, it allows the observation of human behaviour in front of the system. Secondly, the test user should have the opportunity to give comments on the system, which gives information on the feelings of the user when working with the system. And last but not least, the incoming speech can be recorded and used for speech recognizer training/testing.

During the Wizard of Oz experiments, some different aspects of the dialogue system can be tested:

- **prompts** Are they clear? Does the user react as expected? Even different prompts can be tested in parallel during the same experiment (i.e. by dividing the test users in groups).
- **feedback** Is it clear? Does the user appreciate this feedback? How does he react on this feedback (e.g. by trying to correct a recognition error).
- **errors** Is the user able to correct an error? Do the error correction sub-dialogues suite the user?
- **time** How long does it take to perform the complete task? How does the user perceive the periods used by the system to perform the speech recognition?
- **dialogue flow** Is the dialogue in general useful for the user, does he agree with the order of the prompts?
- **user turns** Does the user behave according to the envisaged grammar/dialogue structure?
- **general** How does the user evaluate talking to a computer?

The actual set up of the experiments done with VoicePhone is shown in the next chapter, and deals with most of these questions.

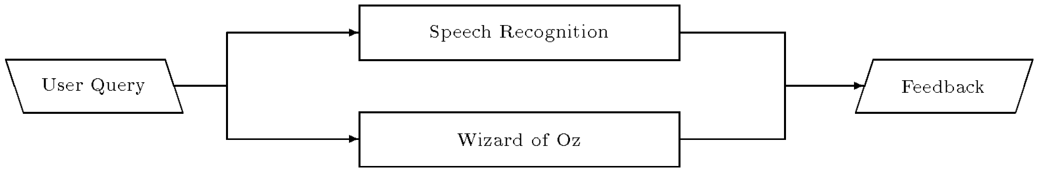


Figure 6.1: Wizard of Oz module and speech recognition in parallel.

6.2 WoZ Interface Design

As well as with dialogue systems, even a 'simple' interface should be user-friendly. During the Wizard of Oz experiments, the Wizard has to be able to perform his task easily and relaxed. This leads to the usage of the general interface design principles, as discussed in section 4.1:

- Learnability
- Flexibility
- Robustness

To be able to separate the interface as much from the dialogue system as possible, it was decided to create a module that works in parallel to the speech recognition process. It only communicates with the program in the same way as the speech recognition (i.e. using a number of files containing input and output). An overview of this is given in figure 6.1. This set up allows the *re-usability* of the interface for other dialogue systems. In some Wizard of Oz implementations, the Wizard was able to chose the dialogue flow. The current set-up only allows the input of dialogue turns, to reduce the mental load of the Wizard (i.e. he has not to worry about the handling of the dialogue). The remaining task to perform consists thus of feeding the system with a list of recognized words (i.e. identical to the file required by the system from the speech recognition algorithm). The next subsections will show the actual implementation of the Wizard of Oz module. More details on the implementation can be found in appendix [Jongebloed-96.2].

Finite-State Transducer

The heart of the interface consists of a Finite-State transducer, that allows the Wizard to enter digits and numerals, instead of entering complete words. This Finite-State transducer takes each part of the entered sentence and converts it into their corresponding words. Some examples are shown in table 6.2. More information on Finite-State transducers can be found in [Covington-94]. The table shows as well some other facilities, built into the transducer; for example allowing mixed forms and abbreviations for often used words that can't be represented by a digit.

Wizard input	Output string
zero vingt six	zero vingt six
0 30 2	zero trente deux
021	zero vingt et_un
230	deux cent trente
f 85	quatre vingt cinq
ch 93	nonante trois

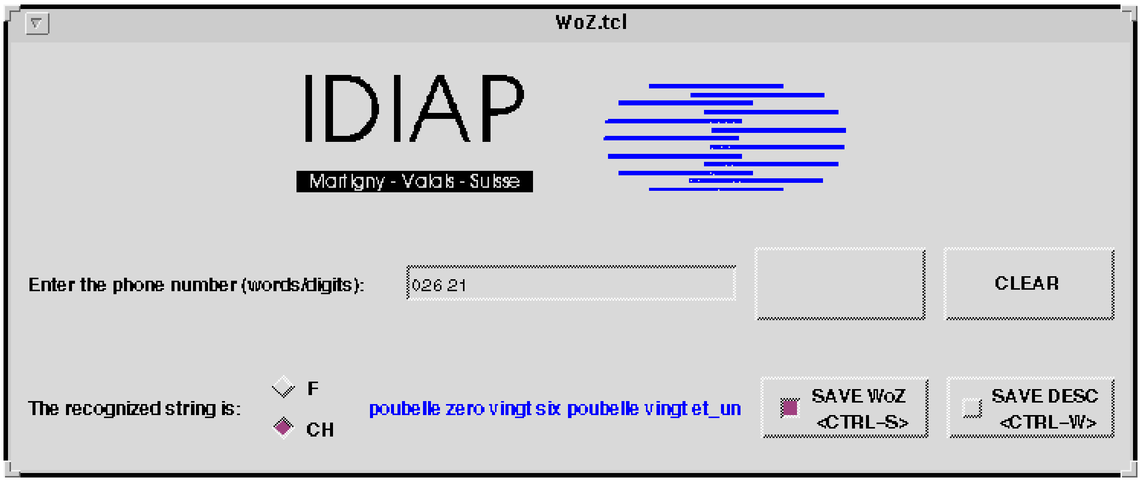
Table 6.1: Allowing different Wizard inputs in the WoZ interface.

To deal with the differences between Swiss French and official French, the parser uses a language variable to represent the language that should be used during the conversion. The last example in table

6.2 shows the usage of this variable: "f" sets the usage of French and "ch" sets Swiss French (per default, Swiss French is used). As shown in chapter 3, the French versions lead to highly ambiguous sentences, so care should be taken to keep the numerals 'together'. To accomplish this, the transducer inserts an extra 'poubelle' (this is the used garbage model in the grammar) before each separate numeral[1]. These extra silences are not shown in table 6.2.

User-Friendly Graphical Interface

The Wizard should have a easy-to-handle interface, therefore a graphical interface is preferred, rather than a textual interface: A graphical interface allows the definition of buttons and easy-to-correct fill-in lines. Care should be taken that everything can still be handled, only using the keyboard. This prevents the swapping between mouse usage and keyboard usages which costs extra time. The complete interface is shown below:



(The WoZ graphical interface)

It consists of a fill-in line and three buttons "CLEAR", "SAVE WoZ", and "SAVE DESC". As well two text strings are given to clarify the fill-in line and the second line (at the moment of starting both are still empty). The Wizard can enter the recognized speech in the fill-in line and hit "enter" on the keyboard. The fill-in line can be cleared with the "CLEAR" button, but also the usual keyboard functions 'backspace', 'del' and 'ins' might be used. The entered string will be translated by the transducer as shown above, and the result will be printed in the empty line (in the figure, the example '026 21' is shown). The language of the string can be changed by the two language buttons ('ch' and 'f'), as far as no language settings are done in the input string. If the Wizard agrees with this string, he can hit "SAVE WoZ"; the string will then be saved to the file that will be used in the dialogue system. Of course, the Wizard has the possibility to simulate recognition errors by simply typing a different query than the one he heard. To keep the information of what was really said during the user turn the corrected string can be saved in an other file using "SAVE DESC" (i.e. 'save description'). As stated, all actions should have a keyboard counterpart, therefore two `ctrl`-key sequences are defined for the buttons. The actual sequences are also shown on the button itself (self explanation of the system) and can be defined according the preferences of the Wizard. The two "SAVE" buttons are equipped with a 'signal light' that changes colour as soon as the action has been done. This serves as a feedback to the Wizard to know the actions still to be done. The interface is quitted automatically after the "SAVE DESC" button has been pushed, assuming the Wizard to be finished.

[1]table 6.2 shows the usage of 0 30 2 instead of 032, this does won't work currently due to these 'poubelles'!

Events Flow During a Wizard Dialogue Turn

Due to the (telephone) hardware and implementation of the vocal server, it is not possible to take the incoming dialogue directly from the telephone line. The incoming dialogue turn is saved to a file that is used for the real time parameter conversion and speech recognition. This happens in parallel, so that the conversion and recognition can be started as soon as the first data comes in (more details can be found in [Bornet-95]). A similar technique is used to create the auditive input to the Wizard, leading to more or less 'live' listening. As stated, the Wizard's dialogue turn should take at maximum the simulated recognition time, allowing a small delay in this auditive Wizard input.

The interface is started before the actual play back of the dialogue turn takes place. This 'popping up' of the window alerts the Wizard that some dialogue turn is coming in. This gives the Wizard the time to activate the interface window, adjust his hands to the keyboard, and shift his attention to the speaker. Such a procedure has an large advantage over a continuously present interface window, causing the Wizard to be surprised when the speaker starts playing the dialogue turn and not immediately able to handle interface. In that case there is even a risk of 'loosing' the first part of the dialogue turn.

Some first tests showed that it was not always possible to listen to the dialogue turn and typing it in at once. If the incoming speech was rather fast, the Wizard was not able to keep the complete sentence in mind. To help the Wizard overcome a "REPLAY" button was created, to replay the audio file. This function could not be integrated in the dialogue interface and was therefore implemented in a toolkit, as described below.

The Wizard's turn is finished after he hits the "SAVE DESC" button or as soon as the simulated recognition time is ended, causing an empty string to be returned as being recognized.

VoicePhone Toolkit

During some first experiments to test the WoZ set-up, it appeared, that some scripts were needed from time to time in the case that some small problem arose (not during the final experiment, but the idea still holds). A toolkit with different buttons is created that can be placed on the screen during the experiments. The toolkit can also kill some no longer needed processes or even blocking processes (e.g. audio processes). As well as it can replay the dialogue turn, during WoZ experiments.

(The VoicePhone graphical toolkit)

Moreover, during the experiments, it is useful to have the parameters that are to be tested at hand, so the Wizard can act conform them (e.g. the number of recognition errors to be simulated). The "NEXT" button should be used to mark the transition to a next incoming call.

6.3 User Test of the Wizard Interface

During the WoZ experiments described in the next chapter the interface was also evaluated. Some problems were detected:

- The Wizard has to enter the description rather quickly during the experiments and therefore some problems arose with entering also the real descriptions of the user utterance. This lead to either empty descriptions or descriptions equal to the WoZ description (which was false half of the time). The large distance between the keyboard key 'Return' and the pre-defined `ctrl`-keys also decreased the efficiency of this function. Therefore, it is proposed in the next chapter to include a function that generates the Wizard's description (i.e. including simulated recognition errors) automatically, based on the correct version of the user utterance.

- the replay option was not often needed, due to experience in listening to incoming messages, knowledge of the incoming phone numbers, and rather slow speaking of participants. As well this option might lead to problems when simulating short recognition times.

Chapter 7

WoZ User Test

The last part of this research consisted of a first WoZ test with users. In the first section the set-up is given as it was created *before* the experiments. This set-up enables the review of the experiments which is done in the last section. After the presentation of the set-up, the results are given and discussed.

7.1 Set-Up of Experiments

Things that will be tested during the first experiments:

- What does the user input look like?
- Problems with the dialogue.
- How does the user adapt to recognition errors?
- Familiar vs. non-familiar telephone numbers.
- Influence of the speech recognition waiting time.
- Evaluation by the user (each dialogue).

For efficient testing, test persons are asked one by one to call three times. The test persons will be chosen here at IDIAP, thus saving a lot of time. As stated in the report (see section 4.4), it is not necessary to perform the first experiments outside the lab.

Each of the test persons is equipped with a fill-in form providing information on the experiment (e.g. which number to call for the system), for each call the telephone number that has to be dictated and some questions to be filled in after each call (an example of the form is given in appendix D).

Simulated Recognition Time

It is not always possible to perform the speech recognition task in real time. Therefore the time needed by the speech recognition is simulated during a wait state. The length of this wait state represent the extra time that is needed (with respect to real-time recognition). For this, two different parameters are used: `long` for telephone number recognition and `short` for 'oui/non' (an overview of the dialogue can be found in figure 5.2). 4 persons per value (values are not changed within the tests of one person). The values for the parameters will be as follows (in seconds, added is the relation to the really needed recognition time[1]):

[1] It is not exactly known how much time the recognition would cost in an optimal environment, as explained in section 3.3

- 25 | 10 slow recognition.
- 20 | 10 average recognition time.
- 15 | 10 currently possible recognition time.
- 10 | 10 currently possible on a fast machine.

Simulated Recognition Rate

Speech recognition is not perfect and therefore errors have to be simulated. Currently 70% correct (1 or more digits incorrect for 3 out of 10 numbers) is possible. This will be simulated by entering 0,1 or 2 times per dialogue an incorrect recognition result, using a pre-generated list of random 0,1,2's. An incorrect recognition result consists of one or two changed words (e.g. 'trente' is entered by the wizard instead of 'quarante').

Telephone Numbers

Also the difference between known and unfamiliar telephone numbers will be measured. Two times a known number will be used to give the user the chance to get used to the system. The third time an unknown number has to be dictated that is selected at random, but the same for each test user, to measure differences in the grouping of the numerals. Which telephone number a test person has to pronounce is written on the question form:

- First/second call: own number at IDIAP.
- Third call: random number = 050-5418846[2]

The old number was chosen to be the *old* number of everybody's personal telephone. The change to 10 digit isn't yet that familiar, so the old numbers would better represent behaviour with familiar numbers. However, one of the test users who has arrived only a short time ago will be instructed to dictate the new (10 digit) number, because she isn't familiar with the old number.

Out of Grammar Input

Users will of course try to dictate a telephone number but still it is possible that this isn't captured in the dialogue turn. Therefore some rules are set up on what to do with out of grammar input (BAD RECO represents the state that the system (i.e. speech recognizer) is not able to understand the input, causing it to ask the user to repeat the telephone number):

Telephone number:

- Garbage before and after telephone number: represent by `poubelle`
- Garbage during telephone number : BAD RECO.
- Incomplete number : BAD RECO.
- No area code: BAD RECO.

Oui/non:

- Semantically correct: either oui or non according to input.
- Semantically incorrect: either oui or non according to the correctness of the last feedback.

[2]this is to be considered as a random Swiss number although the number itself does not exist. Actually, it is my own number in the Netherlands, where also the transition to ten digits was made.

User Evaluation

The test users are asked to fill in a form. This form contains for each call the following questions:

- Are the questions comprehensible?
- Are the reactions of the system clear?
- How do you evaluated the waiting times? short / average / long / too long.
- What do you feel, when talking to the system?
- What was the difference with previous calls?
- Ideas for rectifications.
- Remarks.

7.2 Results of the Experiments

All persons that were asked participated in the test, giving a group of 14 test users. Not everybody did fill out the form between the experiments, this was rather done afterwards. Although a presentation was given, a week in advance, about the envisaged experiments (serving as a interim meeting on the project), so that almost all test users could have known that the system was 'fake', they turned out to believe they were working with a real speech recognizer. It should be noted that the test users are not to be considered as general public because of their experience with (speech) recognition techniques. This section will describe the outcomes of the experiments, sub-divided in the following issues:

- User behaviour.
- Perception of recognition time.
- General perception of the system.
- Ideas for improving the system.

User Behaviour

Different items concerning user behaviours were observed:

- 2 test users thought that the system wanted the telephone number divided in parts. This was due to wrongly interpreting "Please, start with the area code". They corrected this behaviour in reaction to the BAD-RECO warning.
- 3 users dictated the telephone number digit by digit. This was mainly observed with non-native speakers who were not used to dividing the number in numerals (especially not in French).
- A major problem arose with the feedback from the system. This led to users who didn't understand the feedback and said "I'm not sure here" instead of "yes/no". Also some test users said "yes" although a recognition error was introduced. One user even reacted (after recognizing '47' instead of '46' in the last numeral):

  I'm not sure... No, I don't think so, it was forty six actually.

- Most users adapted their input after a (simulated) speech recognition error. The differences with the original input were however small. It consisted mainly of speaking a little bit slower (especially in the part of the recognition error). Another observed behaviour after a speech recognition error was a rephrase of the question. One test user switched from French to Swiss French (although the recognition error occured at a different point), whereas another user regrouped the numerals of the unfamiliar telephone number.
- 3 test users encountered the result of speaking too slow, being cut off by the system during the dialogue turn (although the parameter for the silence detection was chosen to enable rather large silences). They corrected this behaviour in the next turn, when the system announced BAD-RECO.
- A number of users tended to react more elaborate on the yes/no question. This can be catched using keyword spotting or elaborating the grammar:

  "yes, it was correct."
- For some calls the typical non-speech/background noise was observed. Two test users coughed during a telephone number, one user had music in the background and even talking to a third party was observed:

  "Ciao, Miguel!"
- The division of the familiar number in pieces didn't show any variety. This is due to the way it is normally written and pronounced here at IDIAP.
- The unfamiliar telephone number gave more problems in the dictation, although only for a part of the test users. The division of this number in numerals was most of the time done in the same way, although some users preferred another division. Both possibilities should be included in a grammar:

  050-5418846 = 050 541 88 46 or 050 54 18 846

  Moreover, it should be allowed to dictate the telephone numbers digit-by-digit.

Perception of Recognition Time

During the experiments, different recognition times were simulated. The users showed a completely different evaluation of the recognition time as would be expected from the parameter setting. One test user evaluated even the shortest recognition time as 'too long', whereas another test user evaluated the longest recognition time as 'short'. One possible reason for the large variance in perception might be the experience of the test persons with speech recognition (half of the test persons work themselves with speech recognition). It is also related to the outcome of the recognition; a user accepts more easily a long time, if the outcome is correct. Too many false outcomes lead to impatience. Most test users chose the option 'average'.

In general, the response time for yes/no questions was found to be too long. With the current speech recognition techniques, it should be very well possible to perform this task much faster than the simulated time. Though the 10 seconds were needed to handle the wizard's task.

General Perception

Overall, the test users were satisfied with the system taking into account that it was a first version. The usage of a female voice was perceived as good; it made the questions clear. The music was perceived too loud, partly because of the intensity difference between the voice and the music.

Ideas for Improvements

Most test persons had some ideas for improving the system. The most important ones are listed here:

- Enlarge time between "wait a moment, please" and the music.
- A faster reaction in case of BAD-RECO should be possible for real system.
- Some gaps (silences) in the dialogue are to be filled (e.g. between the music and the feedback).
- More variety in the wait music is desired. A longer piece of music, more adapted to the waiting time should be chosen (currently, the wait music is repeated until the recognition is finished).
- The wait state necessary to do the simulate recognition was found irritating by one user that actually had tested the system in earlier pre-tests. It would therefore be of major priority to do some research on the really needed recognition time and to keep it as short as possible.

7.3 Review of the Experiments

The Wizard of Oz experiment gave some important review of the first version of the dialogue system. Moreover, the experiment gave also important information on the usage of such experimental methods in general. In this section the wizard's experiences with the system are described. This should help to improve future experiments. The specific review of the interface was also described in section 6.3. The following issues were encountered during and after the experiments:

- Some advantages were taken in these experiments, that are not obvious in large scale tests; the test users were asked one by one, avoiding mixing up of calls and allowing parameter changes between users. Moreover, it allowed the identification of the caller in advance, as well as a better timing of the incoming call.
- During the experiments, it turned out to be useful to make some quick comments on the call (e.g. report different way of saying 'oui'). This was done in a separate window saving them for each call in a file.
- A problem arose with the creation of correct descriptions of the incoming dialogue turns. When simulating very short recognition times on the telephone numbers (10-15 seconds) it was not possible to enter both the false (simulating a recognition error) and the correct description (needed for the automatic database and the review of the dialogues). This had to be corrected afterwards, taking some time.
- It is very useful to do some test runs before starting the experiments. This enables the wizard to get used to interface and the other actions to be done during a call. This helps the wizard to work rather relaxed during the experiments. (Even during one of the first experiments some wizard errors were made; "yes" instead of "no").
- It is not easy to decide the kind of recognition errors during the experiments. Moreover this has to be done in parallel to understanding the user dialogue turn. If in some future experiments the effects of different speech recognitions will be tested, it would be handy to have some kind of automatic error simulation build in the WoZ interface. This could work more or less the same as the current Finite-State transducer only now with errors. Moreover, it could be based on some random process so that the system does not always make the same errors. Moreover, this would prevent the problems as mentioned above with entering the description of what was really said.
- In general, the time needed to perform the necessary wizard actions (listening, entering the string and saving the descriptions) is around 10-15 seconds minimal. This leads to problems if one wants to simulate real-time recognition during an experiment. It is therefore useful to perform these experiments semi-automated (as stated above) and by an experienced wizard. This would enable more realistic experiments (assuming the possibility of real-time speech recognition in the final application).

Database

The performed experiments were also used to automatically generate a database containing all user dialogue turns. This set-up has succeeded, leading to a database containing about 100 entries for spoken telephone numbers (including partial telephone numbers that occured due to speaking too slow). A database containing about 100 entries for answers to a “yes/no” question was generated, including some 10 entries other than “yes/no”.

The complete database consists of three directories, one with the sampled speech data, one with the description of the utterance and one with the MFCC coded data, coded according to the parameters used in the current grammar(as described in section 3.3). These files can be used to do retraining of the speech recognizer or to do tests on the performance of different grammars/recognizers.

Chapter 8

Conclusions & Continuation

During this project a first version of a dialogue system for recognizing telephone numbers was build, called VoicePhone. Designing such a system is not straightforward, and therefore an iterative design process was followed. This iterative process was executed until the first user evaluation. The user experiments were done, using the Wizard of Oz technique in which the speech recognition is performed by a human operator. The Wizard's interface was also developed, providing a useful tool to evaluate dialogue set-ups in an early stage of the dialogue design. The user experiments gave the following items, concerning the improvement of VoicePhone:

- The speech recognition has to be more accurate. This might be achieved by other models and a better grammar. It can be tested on the database created during the first experiments, as well on the Polyphone database. The time necessary to do the speech recognition is not yet tested. This has to be done in advance to estimate the feasibility of the application. Nearly real-time recognition should be possible viewed from the side of the system and is moreover wanted from the side of the users.
- The feedback to the user has to be more comprehensible. This can be done by either re-recording of the words concatenated with more pauses between or using speech synthesis.
- The speech recognition necessary for the "oui/non" sub-dialogue is not yet implemented. This can be done using the keyword spotting strategy (being more robust).
- For the Wizard of Oz environment, problems arise when the simulated speech recognition time should be very short (e.g. to simulate real-time speech recognition). A more automated version of the interface will help the Wizard, including a function to automatically generate a 'false recognized' user utterance.

Continuation

The improvements mentioned above have to be implemented and tested in a larger user test. This user test must still be done with a Wizard of Oz environment and can also comprise more advanced variants of the dialogue. Moreover, some more elaborate tests can be done on the effects of different kinds of speech recognition errors, and the use of implicit feedback. In the advanced versions of the dialogue, the following hints should be considered:

- The Swiss Telecom has changed the telephone numbers to 10 digits. This leads to strange user behaviours, when constructing a new variant of the old (familiar) telephone number (i.e. reconstructing it from the old one). This effect will last for some time and should be considered in a future test. Special attention has thus to be given to possible new forms of dividing the telephone number into numerals.

- An error correction sub-dialogue, using the separation in numerals to repair the error, has to be inserted in the dialogue. Care has to be taken that the interaction might be very time consuming!
- The (speech recognition) error correction can also be done using the second-best from the speech recognition output.
- The error correction might as well be based on the differences between two user inputs (e.g. prosody or talking speed).
- An emergency exit has to included, to contact a human operator (this might as well be to the Wizard) in case of severe problems.
- The feedback to the user about the probable cause of an error in the case of BAD RECO (e.g. spoken digit by digit, too slow etc.) has to added, guiding the user in adapting his behaviour.

Thanks to...

- IDIAP[1] for giving me the opportunity to work in the beautiful mountain region of Switzerland.
- Jean-Luc Cochard, for the opportunity to work here on an interesting project.
- Perry Moerland (PP), for his help in the first contact with IDIAP, as well as all his comments and Dutch food.
- Robbert Visscher, for joining me during our internships in Switzerland.
- Olivier Bornet, for his help during the implementation of VoicePhone.
- Sylvie Millius, for her vocal assistance in the creation of VoicePhone.
- The rest of the IDIAP for their participation during hike's, "rando's" and the VoicePhone user test.
- Tjeerd Andringa, for his supervision from Groningen.

[1] from now aka. Intelligent DIalogue APplications

Bibliography

[Baum-00] L. F. Baum. The Wonderful Wizard of Oz. April, 1900.

[Bernsen-96] N. Bernsen, H. Dybkjaer and L. Dybkjaer. Principles for the Design of Cooperative Spoken Human-Machine Dialogue. In *Proc. ICSLP 96*, Philidelphia, 1996.

[Bornet-95] O. Bornet. Analyse Spectrale Temps Réel et Viterbi Séquentiel ou FERRT Feature Extraction and Recognition in Real Time. Report for the Swiss Telecom, IDIAP, CP 592, CH-1920 Martigny, 1995.

[Bornet-96] O. Bornet, G. Chollet, J. -L. Cochard, A. Constantinescu and D. Genoud. Secured vocal access to telephone servers. In *Proc. ICSLP 96, International Conference on Speech and Language Processing*, Philadelphia, USA, October 1996.

[Bornet-97] O. Bornet and J. -L. Cochard. Graphical formalism for IVS dialogue description. Communication 97-??, IDIAP, PO Box 592, CH-1920 Martigny, to appear in 1997.

[Chollet-96] G. Chollet, J. -L. Cochard, A. Constantinescu, C. Jaboulet and Ph. Langlais. Swiss french Polyphone and PolyVar: Telephone speech databases to model inter- and intra-speaker variability. Research Report 96-01, IDIAP, PO Box 592, CH-1920 Martigny, 1996.

[Covington-94] M. Covington. Natural Language Processing for Prolog programmers. Prentice Hall, 1994.

[Dix-93] A. Dix, J. Finlay, G. Abowd and R. Beale. Human-Computer Interaction. Prentice Hall, 1993.

[Dybkjaer-93] H. Dybkjaer, N. Bernsen and L. Dybkjaer. Wizard-of-Oz and the trade-off between naturalness and recogniser constraints. In *EUROSPEECH 93*, Berlin, pages 947-950,1993.

[Gagnoulet-89] C. Gagnoulet. Voice replace dial in new public phones. In *Int. Voice Syst.Rev.*, vol.1, pp. 1-29, 1989.

[Genoud-96] D. Genoud. Projet IDIAP/Telecom PTT: Voice Telecom Card. Communication 96-??, IDIAP, PO Box 592, CH-1920 Martigny, August 1996.

[HTK-93] Cambridge University Speech Group. *HTK Hidden Markov Model Toolkit.* Entropic Research Laboratories Inc., Cambridge, December 1993.

[Jack-92] M. Jack, J. Foster and F. Stentiford. Intelligent dialogues in automated telephone services. In *Proc. ICSLP'92*, pages 715-8, Alberta, Oktober 1992.

[Jongebloed-96.1] H. Jongebloed and A. Constantinescu. Towards the Genetic Fundamentals of Speech. Internal Report, IDIAP, PO Box 592, CH-1920 Martigny, December 1996.

[Jongebloed-96.2] H. Jongebloed. VoicePhone: The Programmers Guide. Program Documentation, IDIAP, PO Box 592, CH-1920 Martigny, December 1996.

[Jongebloed-97] H. Jongebloed and J. L. Cochard. User Friendly Vocal Servers. In *Proc. GRONICS'97, Gronings Information Technology Congres for Students*, 21 February 1997.

[Kamm-94] C. Kamm. User Interfaces for Voice Applications. In *Voice communication between humans and machines*. National academic of sciences, pages 422-442, 1994.

[Karis-91] D. Karis and K. Dobroth. Automating Services with Speech Recognition over the Public Switched Telephone Network: Human Factors Considerations. In *IEEE journal on selected areas in communications*, vol.9, no.4, pages 574-585, may 1991.

[Kvale-96] K. Kvale. Norwegian Numerals: A challenge to automatic speech recognition. In *Proc. ICSLP 96*, pages 2028-31, Philidelphia, 1996.

[Kuroiwa-95] S. Kuroiwa. Error analysis of field trial results of a spoken dialogue system for telecommunications applications. In *IEICE trans. Information & systems*, vol.E78-D, no.6, pages 636-641, June 1995.

[Marcus-96] S. Marcus, D. Brown, R. Goldberg, M. Schoeffler, W. Wetzel and R. Rosinski. Prompt Constrained natural language - evolving the net generation of telephony services. In *Proc. ICSLP 96*, Philidelphia, 1996.

[Markowitz-96] J. Markowitz. Using Speech Recognition. Prentice Hall, 1996.

[Niimi-95] Y. Niimi and Y. Kobayashi. Modeling dialogue control strategies to relieve speech recognition errors. In *Proc. EUROSPEECH*, volume 2, pages 1177-1180, September 1995.

[Rabiner-93] L. Rabiner and B. Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, 1993.

[Strik-96] H. Strik, A. Russel, H. v.d. Heuvel, C. Cucchiarini and L. Boves. A Spoken Dialogue System for the Public Transport Information. In: H.Strik, N.Oostdijk, C.Cucchiarini and P.A.Coppen (eds.), *Proceedings of the Departement of Language and Speech*, vol.19, pages 129-142, Nijmegen, The Netherlands, 1996.

[TCL-94] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley PC series, 1994.

[Waern-89] Y. Waern. Cognitive Aspects of Computer Supported Tasks. John Wiley & Sons, 1989.

[XTL-94] SunSoft. *XTL Appliation Programmer's Guide*. Sun Microsystems, Inc., 1994.

Appendix A

HMM Confusion Matrix

Here's the confusion matrix for the Hidden Markov Models used in the complete phone number test. First the total results are given:

```
--------------------------- Overall Results ------------------------
PHONE:  %Corr=93.56, Acc=92.23 [H=7457, D=143, S=370, I=106, N=7970]
--------------------------------------------------------------------
```

```
Confusion Matrix
        u  d  t  q  c  s  s  h  n  d  z  o  d  t  q  q  s  v  t  q  c  s  s  h  n  c  e
        n  e  r  u  i  i  e  u  e  i  e  n  o  r  u  u  e  i  r  u  i  o  e  u  o  e  t
           u  o  a  n  x  p  i  u  x  r  z  u  e  a  i  i  n  e  a  n  i  p  i  n  n
           x  i  t  q     t  t  f     o  e  z  i  t  n  z  g  n  r  q  x  t  t  a  t
              s  r                       e  z  o  z  e  t  t  a  u  a  a  a  n          Del [ %c / %e ]
   un 410  2  0  3  4  0  0  3  6  0  1  0  1  0  0  1  0  4  1  0  0  0  0  0  1  3  0    5  [93.2/ 0.4]
 deux   3329  1  2  0  0  3  3  9  1  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0   11  [93.5/ 0.3]
trois   2  0374  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0  2  0  0  1  0  0  0  4  0    4  [97.1/ 0.1]
quatr   0  0  1386  4  1  8  0  2  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  1  0    2  [95.5/ 0.2]
 cinq   0  0  0  1261  0  1  2  2  1  1  0  0  0  0  0  0  4  1  1  0  0  0  0  0  3  1    2  [93.5/ 0.2]
  six   1  3  1  0  1389  2  6  4  5  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0    6  [94.2/ 0.3]
 sept   0  1  1  6  4  2459  3  0  0  1  0  0  0  0  0  2  2  1  0  0  0  0  0  0  2  0    4  [94.8/ 0.3]
 huit   0  0  0  0  0  8  1314  2  1  0  1  0  0  0  0  0  0  0  0  0  0  0  2  0  0  0    8  [95.4/ 0.2]
 neuf   0  3  0  0  2  0  0  0293  0  0  0  0  0  0  0  0  5  0  0  0  0  0  1  1  2  0    1  [95.4/ 0.2]
  dix   0  0  0  0  0  1  0  0  0117  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0    1  [97.5/ 0.0]
 zero   0  0  0  0  0  0  0  0  0  0527  1  0  0  0  0  1  1  1  2  0  0  0  0  1  0  0    4  [98.7/ 0.1]
 onze   0  0  1  0  0  0  0  0  0  0  0 63  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0    0  [95.5/ 0.0]
douze   0  0  0  0  0  0  0  0  0  0  0  1 33  0  0  0  0  0  0  0  0  0  0  0  0  1  0    0  [94.3/ 0.0]
treiz   0  0  0  0  0  0  0  0  0  0  0  0  0 35  0  0  1  0  0  0  0  0  0  0  0  0  0    0  [97.2/ 0.0]
quato   0  0  0  0  0  0  0  0  0  0  0  0  0  0 30  2  0  0  0  0  0  0  0  0  0  0  0    0  [93.8/ 0.0]
quinz   1  0  0  1  0  0  0  0  0  0  0  0  0  0  0 29  0  0  0  2  0  0  0  0  0  0  0    0  [87.9/ 0.1]
seize   0  0  0  0  1  0  1  0  0  0  0  1  0  0  0  0 40  0  0  0  0  1  0  0  0  0  0    0  [90.9/ 0.1]
vingt   2  0  1  1  3  0  0  0  0  0  2  0  0  0  0  0  0780  7  2  2  0  1  1  4  4  0   13  [96.3/ 0.4]
trent   0  0  2  0  0  0  0  0  0  1  0  0  0  0  0  0  0  8412  3  2  0  2  0  0  2  0    7  [95.4/ 0.3]
quara   1  0  0  0  0  0  0  0  0  0  1  0  0  0  0  1  0  4  4333  3  1  0  0  2  2  0    1  [94.6/ 0.2]
cinqu   0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  2  2  1251  2  1  0  0  3  0    2  [95.1/ 0.2]
soixa   0  0  0  1  1  0  0  0  0  1  0  1  0  0  0  0  0  1  4  0  5254  4  0  0  4  0    1  [92.0/ 0.3]
septa   0  0  0  0  1  1  1  2  0  0  0  0  0  0  0  0  0  1  5  1  2  0209  2  0  2  0    1  [92.1/ 0.2]
huita   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  3106  0  0  0    0  [96.4/ 0.1]
nonan   0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  6  2  1  1  0  0  1195  0  0    1  [93.8/ 0.2]
 cent   0  0  0  0  0  0  1  1  0  0  4  1  1  0  0  0  0  1  3  0  0  6  1  2  0473  0    4  [95.7/ 0.3]
   et   0  0  1  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  1  0  1  0  0  2355   18  [98.3/ 0.1]
  Ins   3  4  2  6  7  7 10  8  1  1  3  3  0  1  0  1  0 13  3  1  1  0  1  0  0 11  1
```

The hidden Markov models listed horizontally represent the models used for recognizing a word. The models listed vertically are the correct models. The numbers in the grid give the number of times, that a certain word (vert.) was matched (recognized) to a certain model (hor.). The `Del` column at the right gives the words that were omitted (deleted) during recognition, whereas the `Ins` row contains the models that were inserted. The numbers on the diagonal give the number of correctly recognized words. They are given as a percentage of the total number of occurrences of that word in the most right column (`%c`).

Appendix B

SDL

To be able to represent dialogue systems in a nice and easy way, a *Servers Definition Language* (SDL) was developed here at IDIAP. The language consists of a set of graphical elements that can be linked together in order to build a graph defining the behaviours of the so-described dialogue system. The most important elements are shown below; `a)` play prompt `b)` record user utterance `c)` combine a/b `d)` switch recognized keywords `e)` begin procedure `f)` end procedure. More details can be found in [Bornet-97].

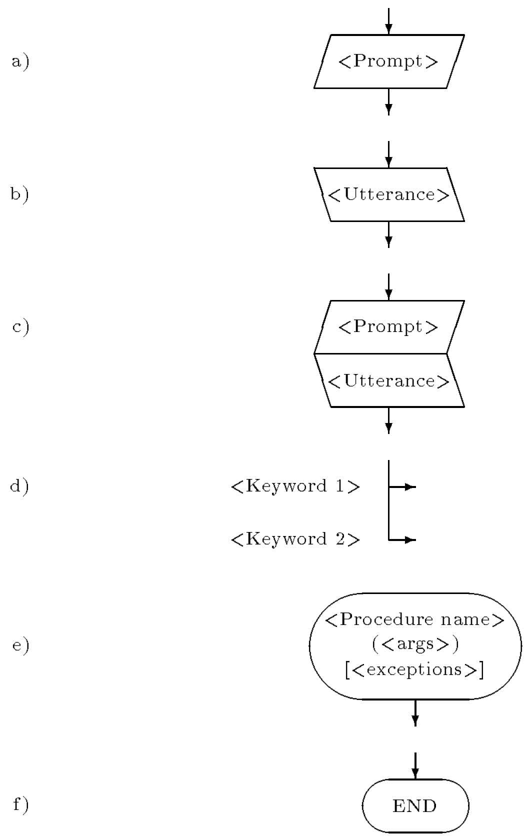


Figure B.1: Different graphical elements of SDL

Appendix C

Translation of the Dialogues

VoicePhone was designed for the Swiss French region of Switzerland. Therefor, all dialogues are in French. This chapter will repeat each dialogue, only here translated in English. It is not always possible to give an exact translation (i.e. meaning the same and still as short as the original query).

system:	Good day, welcome at the automatic operator. You can dictate a desired telephone number, part by part. Please start with the area code.
user:	*Zero twenty six.*
system:	One moment please. “Music”.
system:	I understood ’zero twenty six’. Please continue.
user:	*Seven hundred twenty one.*
system:	One moment please. “Music”.
system:	I understood ’seven hundred twenty one’. Please continue.
etc.	

Dialogue 1. An example dialogue for VoiceCard, asking to spell the telephone number piece by piece.

system:	Good day, welcome at the automatic operator. You can dictate a desired telephone number. Please start with the area code.
user:	*zero twenty six [pause] twenty one [pause] seventy seven [pause] thirty two.*
system:	One moment please. “Music”.
system:	I understood ’zero twenty six [pause] twenty one [pause] seventy seven [pause] thirty two’ Is this correct?
user:	*Yes.*
system:	Thank you for calling. Sorry, but this system is only a test and can not establish the connection. You can hang up now.

Dialogue 2. An example dialogue for VoicePhone, asking for a complete telephone number at once.

user:	*Zero twenty six [long pause] twen...*
system:	One moment please. “Music”.
system:	I didn’t understand your query. Please repeat the desired telephone number.
user:	*Zero twenty six [pause] twenty one [pause] seventy seven [pause] thirty one.*
system:	One moment please. “Music”.
system:	I understood ’zero thirty six [pause] twenty two [pause] seventy seven [pause] thirty two’ Is this correct?
user:	*No.*
system:	Please repeat the desired telephone number.

Dialogue 3. Error correction in VoicePhone.

Appendix D

WoZ evalution form

The following form was used, to do the first experiments. The questions are stated in French because it is the official language used here. Most translations of the questions can be found in section 7.1. For the experiments self, the telephone number that the test users had to dictate was added on the form.

Formulaire de requête pour VoicePhone

VoicePhone est un serveur vocal, pour la reconnaisance des numeros de telephone. Pour tester le système, je vous demande d'appeler le système **trois** fois sur le numéro interne:

767

Pour evaluer le système courant, veuillez vous remplir le questionnaire suivant, apres **chaque** appel.

Merci pour votre cooperation,

Hans Jongebloed

D.1 Appel 1

1. Les questions de système: Sont-elles comprehensibles?

2. Les reactions de système: Sont-elles claires?

3. Comment appreciez-vous le temps d'attente? court / moyen / longue / trop

4. Qu'est-ce que vous ressentez en parlant au systeme?

5. Notes pour rectifications:

D.2 Appel 2

1. Les questions de système: Sont-elles comprehensibles?

2. Les reactions de système: Sont-elles claires?

3. Comment appreciez-vous le temps d'attente? court / moyen / longue / trop

4. Qu'est-ce que vous ressentez en parlant au systeme?

5. Qu'est-ce qu'etait la difference avec votre premier appel?

6. Notes pour rectifications:

D.3 Appel 3

1. Les questions de système: Sont-elles comprehensibles?

2. Les reactions de système: Sont-elles claires?

3. Comment appreciez-vous le temps d'attente? court / moyen / longue / trop

4. Qu'est-ce que vous ressentez en parlant au systeme?

5. Qu'est-ce qu'etait la difference avec les deux premiers apppels?

D.4 Remarques générales:

Appendix E

TextRecorder

To record the prompts that are used in VoicePhone a message recorder was used, called TextRecorder. This program is the simplest version of a dialogue system. It asks for a message and saves it into a file. This is done during an infinite loop, which is ended when the user disconnects. The dialogue looks like this:

system:	Bonjour, bienvenue sur le serveur d'enregistrement des messages de l'IDIAP. Finisez chaque message avec la touche diece de votre téléphone. Rachrochez des que vous desirez plus enregistrer des messages. Donnez votre texte.
user:	*Répétez s'il vous plaît le numero de téléphone desiré.* `#`
system:	Donnez votre texte.
user:	*Continuez, s.v.p.* `#`
system:	Donnez votre texte.
user:	(hangs up)

Dialogue 1. An example dialogue in TextRecorder to record some messages for VoicePhone.

After this dialogue, two message files `message0` and `message1` can be found in a certain directory. These files can be used directly in a dialogue system application. To listen to the recorderded files, it has to be transformed into a `.lin` file, using `mic2lin`. (it is also possible to use `mic2audio_seq` to play the file. This does NOT work with large audio files and care has to taken, to kill the infinite process!).