IDIAP COMMUNICATION

IDIAP

Martigny - Valais - Suisse

Speech Recognition Engine for Interactive Voice Response application on Windows

Haiyan Wang

IDIAP-Com 01-10

DECEMBER 2001

Dalle Molle Institute for Perceptual 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

1. Introduction

This report presents a part work of the *InfoVOX* project. The work was to implement a Speech Recognition Engine (SRE) on Windows with state-of-the-art Speech Recognition (SR) technologies, and integrate SRE within an Interactive Voice Response (IVR) system.

With the increased use of cellular telephones and more and more government regulation of how and when you can use your cell phone, Speech Recognition is a powerful tool for many of today's automated telephone services. The recent emergence of accurate large vocabulary recognition at acceptable cost has increased the range of practical applications. Research in this field has given us a deeper insight into the nature of speech (Articulation, Acoustic, Linguistic and Perceptual Descriptions) and the way we use it to communicate. The related fields of Computational Linguistics and Natural Language Processing have also contributed to this endeavor.

The main scientific goal of the *InfoVOX* project is to perform further research and development in the field of interactive voice servers, with applications in the key area of call centers for computer telephony applications. Moreover, the generic goal of the *InfoVOX* project is to improve the state-of-the-art automatic speech recognition and natural language processing (dialog) technologies to access large and complex databases, and to integrate this technology in a well defined computer telephony application prototype.

More precisely, *InfoVOX* is developing a prototype system to provide access to large and complex (possibly distributed) information databases through which should enable anybody to have access to the restaurants of Martigny from a telephone, as well as from a computer. From a business perspective, the *InfoVOX* project covers a much broader concept of virtual-commerce, which is about adding the transaction capability to information services. The ideal of virtual-commerce is to let customers enjoy e-commerce transactions any time, anywhere, and anyhow by choosing her/his preferred communication channel such as the Web, voice or Wap (Wireless Application Protocol).

The *InfoVOX* system will be a network-based application allowing for vocal interactions. Speech input is achieved via the recognizer (which may be supported by DTMF or dial-pulse input, the signaling system used on push button phones), and speech output is by means of synthesizer (which can be based on stored concatenated speech or may be a fully fledged text to speech system). Note that a text-to-speech synthesizer is currently not implemented for the *InfoVOX* project.

Figure 1 illustrates the structure and data flow of a typical interaction with such an information server, which illustrates the main processing steps and their dependencies.

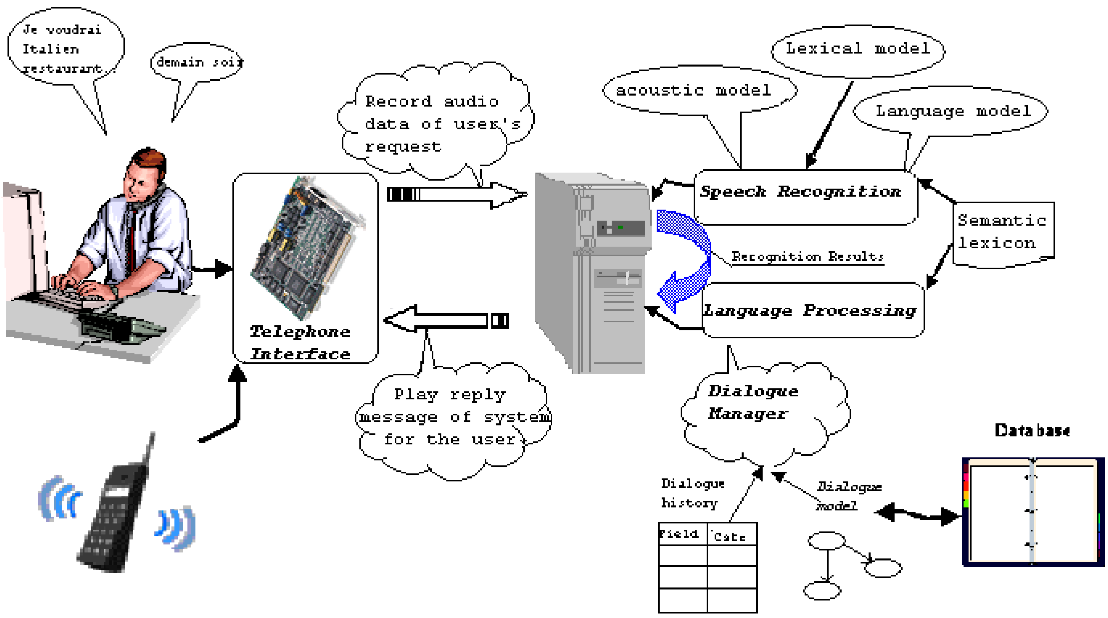


Figure 1 System architecture

From this figure of *InfoVOX* system, it shows that the user could call to the server via a telephone or mobile, and query any information about the restaurants in Martigny. The Telephone Interface will record user's voice into audio data, then the speech recognition engine takes the raw audio data and process it into interpretable sentence. The sentences are upon output to language processing block (dialog models). The language processing block will extract meaningful information from the sentence, and select a reply message from the database, afterwards sent this message name to Telephone Interface, the Interface will play this message to the user. The user could continue to ask other things or answer system's questions. The system will run till the interaction over. Appendix A gives a sample interaction with the *InfoVOX* system.

2. Speech Recognition Engine

ASR (Automatic Speech Recognition) technology enables us to directly communicate with computer systems in the most natural fashion - using speech itself. In the *InfoVOX* project, the speech recognition engine employs the automatic speech recognition technology developed at IDIAP to extract sentences from this raw audio data. Most ASR algorithms and techniques are developed on Solaris for research at IDIAP. However, The mature PC market requires IVR systems to support Windows.

2.1 Development Environment

Software - "*Windows NT*", C/C++ language under "*Microsoft Visual Studio*"

Hardware – PC, D/21H or D/41H telephone card. The Dialogic cards are high-performance 2- or 4-port DSP-based voice boards with on-board analog telephone interface.

2.2 Technology Requirements

In the *InfoVOX* project, Automatic Speech Recognition (ASR) is based on a medium vocabulary continuous speech recognition system, adapted to Swiss-French, and running on a Windows NT platform. The recognizer is based on a state-of-the-art **hybrid HMM/ANN system**, using Hidden Markov Models (HMM) and Artificial Neural Networks (ANN) to model phonetic units. In a hybrid HMM/ANN speech recognition system available at IDIAP, the ANN takes preprocessed acoustic data (features) as input and estimates posterior probabilities for different classes (e.g. phonemes) as output. A particular form of ANN referred to as Multi-Layer Perception (MLP) was used to compute local emission probabilities of HMMs.

While yielding similar or better recognition performance than other state-of-the-art systems, this approach has indeed been shown to have several additional advantage which are especially interesting in the framework of the present research, such as:

- A small set of HMM/ANN context-independent phone models is yielding similar performance to a large set of HMM content dependent phone models.

- Given the above, development of new tasks (involving different lexicon) is easier. It has also been reported that generalization across tasks (training and test set containing different words) was more robust.

- Consequently, hybrid HMM/ANN systems are usually easier to implement and modify, which allows us to focus our research on those most interesting and promising aspects.

- Less memory and CPU are required.

The parameters of this recognizer have been trained on the Swiss-French polyphone database, recorded over the Swiss national telephone network (analog and digital) and containing a large amount of speakers pronouncing phonetically balanced sentences and simulating 111 service queries.

2.2.1 Technology Software

There are several technology software used in the SRE of the *InfoVOX* system, they are originally from ICSI, IDIAP and CCITT software.

- **CONVERT**
- **RASTA**
- **MLP**
- **DECODER**

CONVERT preprocessing must be done before RASTA processing in the *InfoVOX* system. The input for RASTA audio data is in PCM format, whereas the Telephone Interface records audio data in a-law or u-law format. In order to match data formats, the audio data must be converted before RASTA processing. In the current SRE application, CONVERT processing has been integrated into the RASTA module.

RASTA processing extracts feature vectors from digitized acoustic data. This is a speech analysis technique, and the noise robustness of RASTA features is important for telephone line recording conditions in the *InfoVOX* system. In general, each frame of RASTA output, representing 30 ms of speech, is typically encoded by 12 RASTA coefficients, 12 delta-RASTA coefficients, 12 delta delta-RASTA coefficients, and an energy term, a delta-energy term and a delta delta-energy term. The width of the window is 30ms of which the center shifts every 10ms.

MLP is a simple three-layer feed-forward neural network trained with the back-propagation algorithm. Data to the input layer consists of 9 frames of input feature vectors. Each frame directly takes one frame from RASTA output, but leave out 12 delta delta-RASTA coefficients, and an energy term. Typically, we use 600 hidden units. The output layer has 36 units, one for each phoneme in the *InfoVOX* project. The MLP is trained on the phonetically Swiss-French polyphone database to produce the initial state alignments. Emission probabilities of HMM phoneme states are finally generated at the MLP outputs.

DECODER is a start-synchronous decoder designed for medium vocabulary continuous speech recognition, using long-span language models. In the current implementation, the decoder takes phoneme posterior probability vectors as input (typically produced by a neural network) and produces a recognized word string as output. Currently only n-gram language models have been implemented (n = 2,3) using a backoff approach. In addition to specifying a language model, a pronunciation dictionary and basic set of phone models must also be specified. Since this decoder was designed for use within hybrid ANN/HMM systems that do not use cross-word context-dependent models, so support for cross-word phoneme models has not been implemented.

At present, IDIAP has implemented four approaches in the DECODER: standard Viterbi Decoder; Beam Viterbi(or pruned Viterbi) Decoder; NO_LM Decoder; Forced Viterbi Decoder. As a compromise between system costs and recognition accuracy, we employ the Beam Viterbi Decoder in the *InfoVOX* system

Figure 2 represents the data flow of the speech recognition engine. SRE takes raw audio signal data, **CONVERT** the data from a-law or u-law format into PCM format; **RASTA** extract speech feature vectors from the signal data; **MLP** cope with the feature vectors and acoustic model, estimate phonemes probabilities; **DECODER** address the phonemes probabilities, use lexical model and language model, finally output the speech sentence.

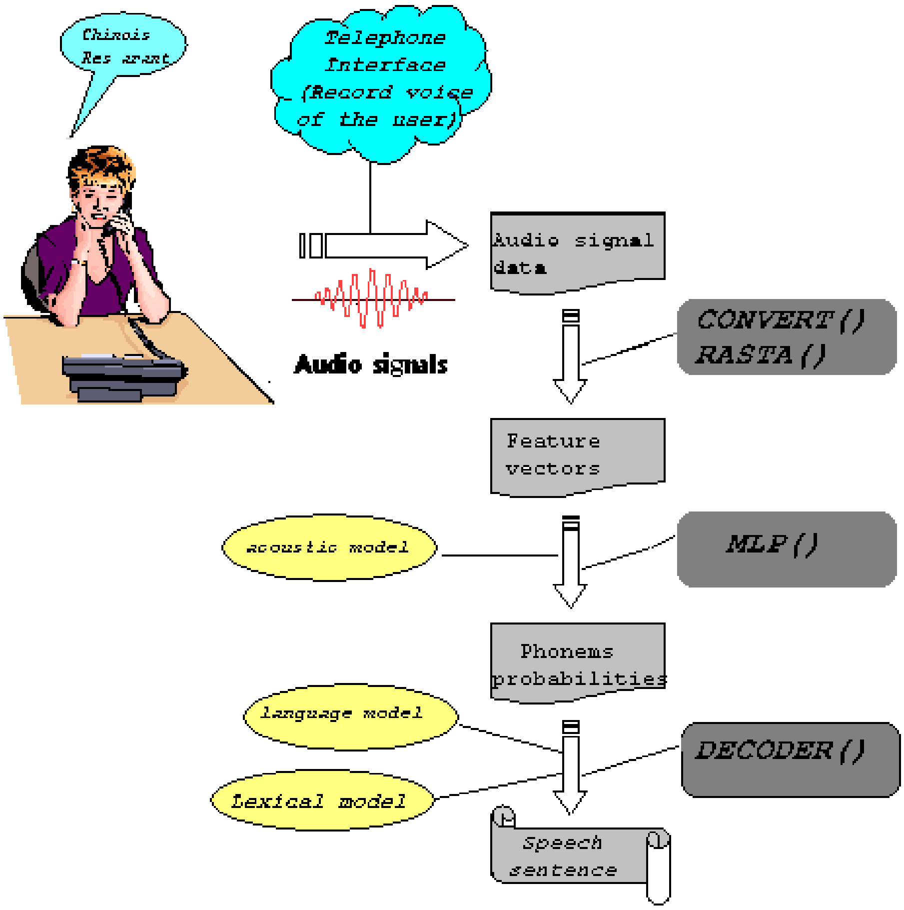


Figure 2: Speech Recognition Flow

2.2.2 Technology Models

In our hybrid system of the *InfoVOX* project, we built an acoustic model, a lexical model and a language model for the speech recognition engine. Hereafter we will describe the technologies in the models.

ACOUSTIC MODELS

The initial acoustic models have been trained on the Swiss-French polyphone database, using 10 phonetically rich sentences read by each of 400 speakers. Acoustic features are first extracted in the preprocessing of the speech signal based on the relative spectral analysis (RASTA). RASTA features are particularly robust to convolutive and additive noise, so they are well suited for telephone speech. STRUT (Speech Training and Recognition Unified Tool) software was used to train acoustic models.

LEXICAL MODEL

The lexical models are usually a priori defined on the basis of a dictionary containing all possible words with their phonetic transcription.

The **phonemes** used in the phonetic transcriptions were those of the SAMPA phoneme set for French. Their models were constructed building upon 35 basic phonetic units and 1 silence state corresponding to the 36 MLP outputs.

The **phonetic transcriptions** of the general vocabulary words were obtained from the BDLex-50000 dictionary. For these words, different phonetic transcriptions were introduced in the lexicon, to take into account the phenomenon of “liaison”, very common in French, and to enrich the lexical modeling with pronunciation variants. The proper words were transcribed by a rule-based system followed by a manual correction.

LANGUAGE MODEL.

The language model is usually trained on a large amount of text corpora reflecting at best the conditions of use of the recognizer. The corpora used to train language model was obtained in the WoZ experiment of the *InfoVOX* project. The CMU (Carnegie Mellon University) Toolkit is a set of utilities for creating bi-gram and tri-gram language models from large text corpora. In the *InfoVOX* project, the text corresponding to the training part of speaker request orthographic transcriptions was processed into various back-off n-gram language models. The language model and lexical model can be implemented with the DECODER.

.

3. Development and implementation

The development of a speech recognition engine consists of two stages: transporting the speech recognition algorithms from Unix to Windows NT; building the speech recognition engine and optimizing code on Windows NT.

3.1 Porting from Unix to PC

Most of the available speech processing algorithms at hand, such as RASTA, MLP and DECODER, are implemented on Unix system for research proposes. Unlike the Unix operating system, Windows is a multi-thread message based operating system. In Windows, data is stored and transferred in a different way as in Unix. Processes and threads are managed differently under the new multi-thread scheme. Therefore, the only successful way to transport these SR algorithms from Unix to Windows is to work on the source code level. The source code of the algorithms need to be first read and understood and then re-coded and debugged in the Windows development environment.

Porting code from UNIX to Windows NT may be as simple as recompiling, or may require significant redesign and recoding. Good code design makes porting easier between different UNIX implementations, as well as between UNIX and Windows NT. Usually, the amount of porting work depends on how portable the application is and the intended use of the application. Is it for in-house use, or for the commercial market? The porting effort is also highly dependent on whether base operating system services are required. Since it is not possible to list all the possible recoding cases in this porting task, it is assumed that all the source code are well designed, although that is not always true in the *InfoVOX* system. In a task of transporting a well-designed source code from Unix to Windows, basically you need to pay attention to six kinds of factors:

- Data formats
- Static constants name
- Function name
- Calling method
- Missing functions
- Old bugs.

3.1.1 Data formats

Binary data is stored differently on Unix and Windows operating systems, whether in memory or disk. Unix stores binary data in so called big endian format, in which the byte order of data is organized from high byte to low byte. But Windows keeps binary data in little endian format, where the byte order is arranged from low byte to high byte. For example, an integer on Unix is:

```
(Big endian integer)
[03 02 01 00]
```

in big endian format while the same integer on Windows would be:

```
(Little endian integer)
[00 01 02 03]
```

in little endian format. Whenever before reusing any binary data produced on Unix, you must remember to reverse the order of bytes.

3.1.2 Static constants name

Some data type constants and static constants have different names between Unix and Windows. For example, the Maximum float value, is named “MAXFLOAT” on Unix whereas “FLT_MAX” on Windows. “pi” is named M_PI on Unix, but is only defined in Math object on Windows. So a pre-definition must be added in the source code.

```
#define M_PI         3.14159
#define MAXFLOAT     FLT_MAX
```

3.1.3 Function name

Some common library functions do not share the same names on Unix and windows, such as popen(), pclose(), rindex() and so on. Their names on windows are individually _popen(), _pclose(), strrchr(). Because there is no such list that can tell you the matching name of each function on Windows, you must be care for such difference.

3.1.4 Calling method

Unix and windows employ different default calling parameters in some commonly used function. For example, the "fopen" function is to open a file. Without an explicit declaration, the "fopen" function will open a binary file on Unix, whereas opens a text file on Windows.

For instance: the following statement is to open a binary file, and running correctly on Unix. When transporting it from Unix to Windows, you must specify the mode of access requested for the file.

```
stream  = fopen( "data", "r" );           --on Unix
```

```
stream  = fopen( "data", "rb" ))          --on Windows
```

Otherwise, some running error will occur unpredictably, or you will get quite strange results. Consequently, you should pay attention to operating some stream I/O Routines during porting.

3.1.5 Missing functions

Not all functions under Unix have corresponding functions under windows. This especially applies to Unix system functions. Some general functions, such as rint(), fork() and getopt(), have no equivalents on Windows. You have to restructure such functions yourself. Certainly you could get inspiration from few Unix paradigms to program code on Windows NT.

Moreover, there are even differences in coding style and appearance. Most UNIX applications follow `fork()` with `exec()`, which together are equivalent to the Windows `CreateProcess()` API. Applications that depend on the cloning ability of `fork()` need to be redesigned to use threads, or use a UNIX compatible library product such as NuTCRACKER. In most cases, you have to implement these missing functions yourself.

For example: function rint() is to compute round-to-nearest integral value of an input value. The following code shows a brief code implementation on Windows:

```
 eg.  rint(1.3) = 1.0;  rint(-1.8) = 2.0;

      double rint(double f)
      {
            if(f>=0.0)
                  return(ceil(f));
            else
                  return(floor(f));
      }
```

3.1.6 Old bugs

Unix does not check exactly the same compile or run-time errors and warnings as Windows, such as for redefinitions. Some bugs in the original code cause run-time errors under Windows, which are not detected under Unix, and may even run normally on Unix. More specially, you must strictly obey all regulation of definitions if you also need to port C to C++ code.

3.2 Integrating and Optimising code on Windows NT

The performance in terms of run-time is very important. Generally speaking, some Windows NT advanced features may improve the performance or marketability of the application; however, their effective use may require complete redesign of source code. According to the real-time requirement of *InfoVOX*, there follows a description of some efforts we made in order to integrate and optimize the code of the speech recognition engine under Windows NT.

3.2.1 Using pipes to integrate modules

In the former *InfoVOX* system, Speech Recognition block on Unix used files to connect to the Telephonic Interface. In the present *InfoVOX* system, we create a pipe to integrate SRE into the Telephonic Interface, as well as a pipe to exchange data between MLP and DECODER.

Both Windows and Unix operating systems provide pipes. A pipe is an artificial I/O channel that a program uses to pass information to other programs. A pipe is similar to a file in that it has a file pointer, a file descriptor, or both, and can be read from or written to using the standard library's input and output functions. However, while pipes are a fundamental mechanism underlying the Unix Operating System, in DOS they are implemented using temporary files. It represents temporary storage in memory that is independent of the program's own memory and is controlled entirely by the operating system.

In particular, it should be kept in mind that a "pipe" is implemented by a temporary file. So the input must be complete before the output can go to the other end of the "pipe". This is crucial in the case of building multiple pipes in an application on Windows NT. Each process must have only one handle open on the pipe. A pipe is destroyed when all of its handles have been closed.

3.2.2 Exchange of data through memory buffer

In the former speech recognition block on Unix, the intermediate results of RASTA, MLP, and DECODER are transferred with files, then using Unix command "cat" to build pipes for communication. However on Windows, we currently created a Speech Recognition Engine, each respective module has been revised as a function in the SRE. In this case, the operations to read and write file take 100 times more than memory buffer processes. In the present SRE on Windows NT, we transfer I/O through memory buffer between RASTA and MLP. Computing the running time shows that run-time speed is enhanced by around 40%.

Interface between SRE and Telephonic application

The current telephonic application, a natural voice interface, is based on an analog Dialogic board under Windows NT. It records a user's voice and transfers the audio signal data directly to the SRE through the pipe. So far in the present *InfoVOX* system, the Telephonic interface need not generate audio or message files any more, thus greatly increasing the efficiency and performance of the system.

Interface between MLP and DECODER

In future use of the *InfoVOX* system, it may be required to run MLP and DECODER on different machines or platforms, and information could be exchanged over the network under TCP/IP protocols. We create pipes for the MLP output and the DECODER input, in order to provide a standard high-level interface for passing data between two processes, whether processing local or remote pipe requests. Sockets would also work well with pipes over the network.

3.2.3 Load model file once

During the process of SRE, some software will read the data from the model files to do initialization. In the real-time IVR system, SRE will process each speech data of user's voice until the interaction is over. If SRE read the model files every time for a speech recognition process, it's redundancy. In the present *InfoVOX* system, acoustic model is only read at the first time.

3.2.4 Endeavors with Multi-thread

Windows allows building multiple concurrent execution threads running simultaneously. At the original period of developing, we've implemented a parallel control program integrated with RASTA and MLP using Multi-thread technology. A *mutex* was used to protect a shared resource from simultaneous access by multiple threads. Nevertheless, the running tests showed that the performance of the application in fact decreased. After investigation and analysis, it was concluded that it is impractical to build multi-threading between RASTA and MLP. The running speed of RASTA and MLP are quite different, so it is almost impossible to realize parallel operation. Moreover, Creating Multi-thread requires extra system expense to implement multi-threading.

However, this multi-thread application provides a framework for the *InfoVOX* system, and it may have potential usage in other applications.

4. Socket Programming

Ideally, the *InfoVOX* system should allow the remote user to access to an IVR call center, using a touch-tone telephone and interact with the system in real time under a host of applications. Sockets are chosen for network communication in the *InfoVOX* project, include Internet and Intranet. In network programming, a socket provides an end point to a connection; two sockets form a complete path. A socket works as a bidirectional pipe for incoming and outgoing data between networked computers. The network programming is currently not well designed. Hereafter describes some fundamental concepts and present work.

All network application programs (whether under Unix or Windows NT) –Client and Server – boil down to five simple steps:

- Open a socket
- Name the socket
- Associate with another socket
- Send and receive between sockets
- Close the socket

In terms of different platforms, WinSock and Berkeley sockets are applied under Windows and Unix platform respectively. The most significant difference between the two sockets is the fact that socket descriptors and file descriptors cannot be used interchangeably.

According to the essential ingredients for client or server applications, the present system has implemented a successful data transport application. In this example, the server application is built on Unix, and the client application is built on Windows NT. The data or file can be sent and received correctly. Nevertheless, the application would not work if the positions of server and client were exchanged, because the server can't be built on Windows NT workstation. The future experiment could be done after Windows NT Server is installed on PC machine.

5. Tests and Evaluations

5.1 Software Testing

In general, to ensure that the *InfoVOX* system is implemented correctly and robustly, we need to process unit testing, system integrated testing and system robust testing in most common cases under real-time environment. Unit testing aims at testing each independent module; System integrated testing focuses on whether the system can correctly process if all modules are put together; System robust testing observes the system states when performing multiple times in real-time.

The current speech recognition engine on Windows NT has been verified with the program of Unix version. Ten telephone line audio files are chosen as samples to perform this verification. The same raw audio data are fed into both speech recognition application between Unix and Windows NT. Each application employs the same running parameters. (The running parameters are listed in appendix B.) The speech recognition results are the same on different platform, which show that the porting of the speech recognition engine is successful.

After the SRE was integrated with the Telephonic interface, the whole *InfoVOX* system was executed many times in real time. Some bugs have been fixed. According to the future usage in the city of Martigny, system robust testing is still particularly required.

5.2 SRE Evaluation

The effect of speech recognition is also an important ingredient in *InfoVOX* system. Here is an evaluation performed by the SCLITE Toolkit, which score and evaluate the output sentences of the SRE.

The evaluations were operated on three different databases. “150Sen” database comprise 150 sentences, 2030 words, 10 people read each 15 same sentences. “4Spers” database include 57 sentences, 196 words, it recorded the voice of 4 French people at IDIAP when they were using the *InfoVOX* system. “Bochum” database is composed of 215 sentences, 967 words, which got from the imitated *InfoVOX* interaction.

Table 1 SRE Evaluation

Database	# Snt	# Wrd	Corr	WER
150Sen	150	2030	69.0	35.0
4Spers	57	196	48.5	58.7
Bochum	215	967	51.3	56.8

6. Conclusion & Continuation

In this report, the development of speech recognition engine in *InfoVox* system has been presented resulting in a real-time IVR application prototype. As future advanced technology applied, higher accuracies can be expected to achieve.

During the developing process, some constructive discussions and articles can contribute to the following points for the continue work:

- Efficient application management: Since most applications of *InfoVOX* are currently implemented on Windows, it is advisable to take advantage of Windows mechanisms, such as using FIFO message queue to communicate information or a request within system. Messages can be passed between the operating system and an application, different applications, threads within an application, and windows within an application. Clearly good application management will increase the performance of the IVR system.

- Code optimization: Although Visual Studio 6.0 programming software can automatically optimise code and reach maximal speed, a large amount of work is still required to reduce the computation costs in SRE. There is a fast matrix operation algorithm in the ICSI Software; probably it's worthwhile implementing similar operations in the SRE for the *InfoVOX* system. However, its input relies on a kind of compressed feature vectors, thus would require many codes to be restructured.

- Method innovation: It is feasible to make use of multi-stream ASR technology for recognition improvement. Some former research work at IDIAP has proved that combining multiple feature streams, such as Perceptual Linear Prediction (PLP) and modulation-filtered spectrogram (MSG) features would improve the recognizer.

- Provide abundant feedback information of speech recognition to help enhance the performance of other models. For example, acoustic confidence scores computed in the recognizer could possibly be used to improve the performance of the understanding module and to determine the appropriate dialog strategy in the dialog manager.

7. Acknowledgement

I wish to appreciate professor Hervé Bourlard, for his teaching and supervising in the course and report, which give me more interest and inspiration in my work.

Thanks Frank Formaz, Alex Trutnev and Thierry Collado, for providing great confidence and collaboration in our development of *InfoVOX* system, and sharing successes and failures within the process of implementation.

Thanks Andrew Morris, for his patient instruction, precious correction and suggestion, and his willingness to pioneer, which enlighten my acquaintance and enthusiasm to study.

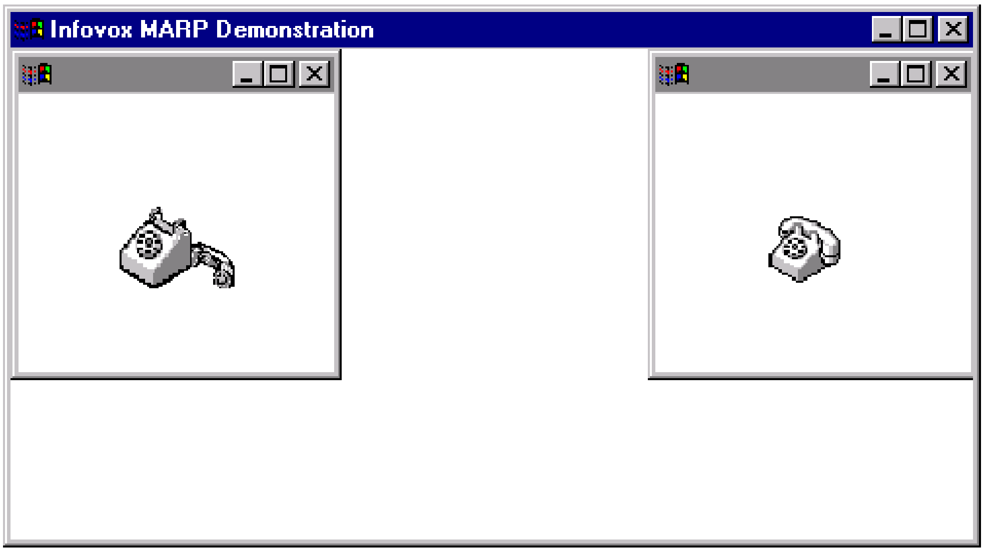
Thanks Dan Ellis, Astrid Hagen and Jitendra Ajmera, for their dedication of software explanations, advisable proposals and continuous encouragement.

8. References

- H. WANG, *“Report of Rebuilding Speech Recognition on PC—Project InfoVox”* IDIAP Communication Report. January, IDIAP-Com-01-09
- “*Interactive Voice Servers for Advanced Computer Telephony Applications*”, Executive Summary of *InfoVOX* Project. June, 2000
- “*Project InfoVOX CTI-4247.1 Documentation - interne*”, IDIAP Technical Report. May, 2000
- “*Detailed Techincal Description*”, InfoVOX project.
- *“INtegrating SPEech acoustic and linguistic Constraints: Baseline System Development”, Giulia Bernardis, Hervé Bourlard, Martin Rajman, and Jean-Cédric Chappelier*, IDIAP-RR 99-21, 1999
- Jurafsky,D., Wooters,C., Tajchman,G., Segal,J., Stolcke,A., Fosler,E., and Morgan,N., *”The Berkeley restaurant project”,* in Proc.ICSLP-94, Yokohama, Japan, 1994
- A. Potamianos, H. Kuo, C. Lee, A. Pargellis, A. Saad and Q. Zhou, “*Design principles and tools for multimodal dialog systems*” in Proc of ESCA Workshop, Germany, June 1999.
- Trutnev,A., *“Light Decoder Development”,* IDIAP Technical Report. February, 2001
- *“SWISSCOM ``AVIS'' PROJECT (No. 392) Advanced Vocal Interfaces Services”, Johan M. Andersen, Gilles Caloz, and Hervé Bourlard*, IDIAP-Com 97-06, 1997.
- *“Personal Voice Dialing over {PC}”,* Frédéric Bressoud and Haiyan Wang, IDIAP-Com-00-05
- Viktor, Toth. *“Programming Windows 98/NT Unleashed”,* Macmillan Computer Publishing, 1998
- *Digital UNIX and Windows NT Interoperability Guide*
- *The Complete Reference UNIX*
- Bob Quinn, Dave Shute, *“Windows Sockets Network Programming”,* Addison-Wesley, 1996
- Anil Bahuman *“Demonstration of Interactive Voice Response Systems: A Thesis Idea”,* December 1999
- *MSDN Library Visual Studio 6.0*

APPENDIX A

InfoVOX System Interaction (Martigny Restaurant Reservation)



***InfoVOX* :** Bonjour! Bienvenu sur Rest'Info, le serveur d'information des restaurants de la ville de Martigny. Vous pouvez faire une recherche sur le type de cuisine, le type ou le prix du repas, la localisation approximative du restaurant ou ses horaires. Quelle est votre demande?

***User* :** oui bonjour je cherche un restaurant pour manger à midi s'il vous plait.

***InfoVOX* :** Redonnez s'il vous plait, l'information sur le type du repas.

***User* :** Cusine Chinoise.

***InfoVOX* :** Restaurant Kwong-Ming: Rue du Nord 9.

Restaurants:

Le Kwong-Ming

- - - - - - - - - - - - - - -

Voulez-vous rechercher un autre restaurant?

***User* :** euh non merci

***InfoVOX* :** Merci d'avoir utilisé Rest'Info. Au revoir et excellente journée.

APPENDIX B

PARAMETRES OF the MODELS

A speech recognition application consists of following sequence models: Rasta, Mlp and Decoder. In the PC version, All environment parameters are fixed at present. Here is a short presentation of the parameter settings these three models:

#---RASTA (*extract acoustic vectors)*

-w	*30*	frame-length; analysis window size
-s	*10*	frame-shift; window step size
-S	*8000*	sample-rate; Sampling frequency
-n	*12*	number of output parameters
-q	*2*	degree of delta calculation
-c	*17*	num of critical-band-like filters
-u	*29.021531*	upper cutoff zero freq
-m	*10*	model order
-p	*0.940000*	pole position
-r	*1.000000*	for partially rasta, partially plp
-W	*0.540000*	windowing constant
-l	*0.600000*	liftering exponent
-M		adds a small constant to the power spectrum
-d		debug
-L		for log rasta; log-rasta=yes

#---MLP(*evaluate likelihood of the phonemes)*

0	sort of use(0:reco; 1:training)
../SFP.234-600-36.softmax	file of weights and biases
234	nodes num of input layer
600	nodes num of hidden layer
36	nodes num of output layer

#---DECODER(*do the recognition)*

-dictionary ../data/vocab.French

name of the file that contains vocabulary words with their transcriptions

-mlp_phonemes ../data/phonemes.French

name of the file with phonemes of MLP

-binary ../data/lm.bin

name of the file with language model in binary format

-states 1

number of states per phoneme

-silence

whether one wants to add ’#h’ (silence) as last phoneme to the vocabulary words

-alpha 0.75

scaling factor (additioner, will be logarithmed)

-beta -1

scaling factor (multiplier)

-beam 5

number of end-of-words phonemes propagated in each time frame

-temp */tmp/probabilities.temp*

temporary file where estimations are kept