IDIAP
Martigny - Valais - Suisse

Hes·so
Haute Ecole Spécialisée
de Suisse Occidentale

IDIAP COMMUNICATION

IDIAP
Martigny - Valais - Suisse

Personal Voice Dialing over PC

Frédéric Bressoud
Haiyan Wang

IDIAP-Com-00-05

August, 2000

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. Project Introduction 4
1.1 Developing Environment 4
1.2 Technology Requirements 4
2. Compatibility between Hardware and APIs 5
2.1 Tests 5
2.2 Result 6
3. Application's description 7
3.1 Dialog box 7
3.1.1 View a position 8
3.1.2 Help 8
3.1.3 Refresh the PIN list 8
3.1.4 Delete button 9
3.1.5 Modify button 9
3.1.6 Test call button 9
3.2 General use 10
3.3 Add to the database 10
3.4 Delete from the database 11
3.5 List the database 12
3.6 Call someone 12
3.7 List key words 13
4. Modelisation 14
4.1 What is UML 14
4.1.1 Importance of modeling 14
4.1.2 Goals of UML 14
4.2 Classes and objects 15
4.2.1 Use case diagram 15
4.2.2 Class diagram 15
4.2.3 Sequence diagram 16
4.2.4 Activity diagram 16
4.3 Voice Dialing 's modelisation 18
4.3.1 Class diagram 18
4.3.2 Sequence diagrams 19
5. Coding 24
5.1 Class CCommunication 24
5.2 Class CAudio 26
5.2.1 Uses modification 26
5.2.2 What is WAV 27
5.2.3 Message from the application to the user (TellMessage & CloseWaveOut) 28
5.2.4 Data from the user to the application (SetUserData & SaveUserData) 29
5.2.5 Window message principle 30
5.3 Class Cannuaire 31

5.4 Class CRecognition 32
5.4.1 Create keywords Model 33
5.4.2 Recognition 34
5.5 Classes CapplicationDlg, CDeleteDlg and CModifyDlg 34
6. Tests and validation 35
7. Project's status 37
8. Project future steps 37
9. Conclusion 38
9.1 Project progress 38

1. Project Introduction

As an important representation form of human beings, speech recognition is widely used in our daily life. During the last ten years, automatic speech recognition has been applied in real applications like vocal notebooks, vocal message control and so on.

Now many software companies can design, develop and distribute speech recognition software products, which enable users to interact with and instruct a wide variety of electronic devices through speech.

With a program using Speech Recognition, you can ask, "What time is it?" and hear if you're late for dinner. You can open your spreadsheet by saying, "Open the February forecast." Or use voice command to control your starship in a game. The possibilities are as limitless as the human imagination.

A voice dialling system over PC is a typical application of speech recognition. It is implemented using speaker dependent speech recognition based on phone-like units as models.

1.1 Developing Environment

Software - "*Windows NT*", the "*Microsoft*" APIs (TAPI[1]) and C/C++ language under "*Microsoft Visual Studio*" are chosen to develop this project.

Hardware – PC; D/41ESC Global SCSA 4-Port Voice Processing Board; telephone machine.

1.2 Technology Requirements

Automatic Speech Recognition - This project applies speaker dependent recognition over the telephone. It conceives a system of personal voice dialing via phone interface. During speech recognition, it use Dynamic Time Warping (DTW); Pattern assignment module which done the generation of templates from quantified vectors; Acoustic analyses is to extract features parameters from input speech.

Microsoft" APIs(TAPI)- The Microsoft Telephony API, provides telephony-related services for Win32 applications.

[1] TAPI : Telephony API

2. Compatibility between Hardware and APIs

The first task is the verification of the compatibility between the "*Dialogic*" phone card and the "*Microsoft*" APIs. If they are compatible, there is no problem, but if they are not, the card, or the interfaces, must be changed. This is the most important part, because all the study depends on this compatibility. The more compatible, the easier coded.

2.1 Tests

To realize these tests, a simple code, using TAPI, is made. This code, which is in annex A, must test the most usual functions of the phone card to verify compatibility of the card with the library. All existing codes, over Internet or on the MSDN[2] library, are too complex and too dense. For this reason a code that uses simple and basic functions of TAPI is created. It takes longer to build but the advantage is that it can serve as a basis during the coding part.

First, the capabilities to match a call are tested. This code contains a class that supports the basic functionality needed to use TAPI. This class is the "*CTapiSimple*", which contains basic functions of TAPI based on the MSDN library. The code can make an outgoing call; it goes through.

Secondly, the capabilities to receive a call are also tested. This code is the same as the first one but functions to make an incoming call are added. The function "*lineCallbackFunc*" receives all events sent by TAPI. This function, in my code, does not work, because it needs an application that uses a message loop mechanism, and tests cannot be made. If necessary, this function will be examined later during the realisation. Other ways exist to test an incoming call.

"*Dialogic*" adds with its card some examples. The "*talker32*" uses TAPI functions (*Start, Programs, Dialogic System Software, Dialogic Sample Programs, and TAPI*). With this example, the user can make outgoing and incoming calls. This example proves that the hardware is compatible with the software, but these results cannot be shown.

[2] MSDN: Microsoft Software Developer Network

The last part of the test must show that the hardware is really compatible with the software. For this, the "*Microsoft® Windows(TM) TAPI Browser*" can prove with printable results that the card fits. These results are call-back messages that are received from the line. (See annex B).

2.2 Result

The TAPI browser results demonstrate that the "*Dialogic*" phone card is compatible and can work with the "*Microsoft*" API libraries. For the remaining part of this project these components will be used.

Nevertheless, a note could be added: the PBX[3] type must be known by the application. The PBX sends signalling tones to the application. These tones are analysed and the application executes the appropriate treatment. If tones are unknown, or different than the awaited one, the application cannot run correctly, e.g. the application receives a connected tones, which is normally sends when the caller picks-up the phone, whereas the caller does not have picks-up.

[3] PBX : Private Branch Exchange

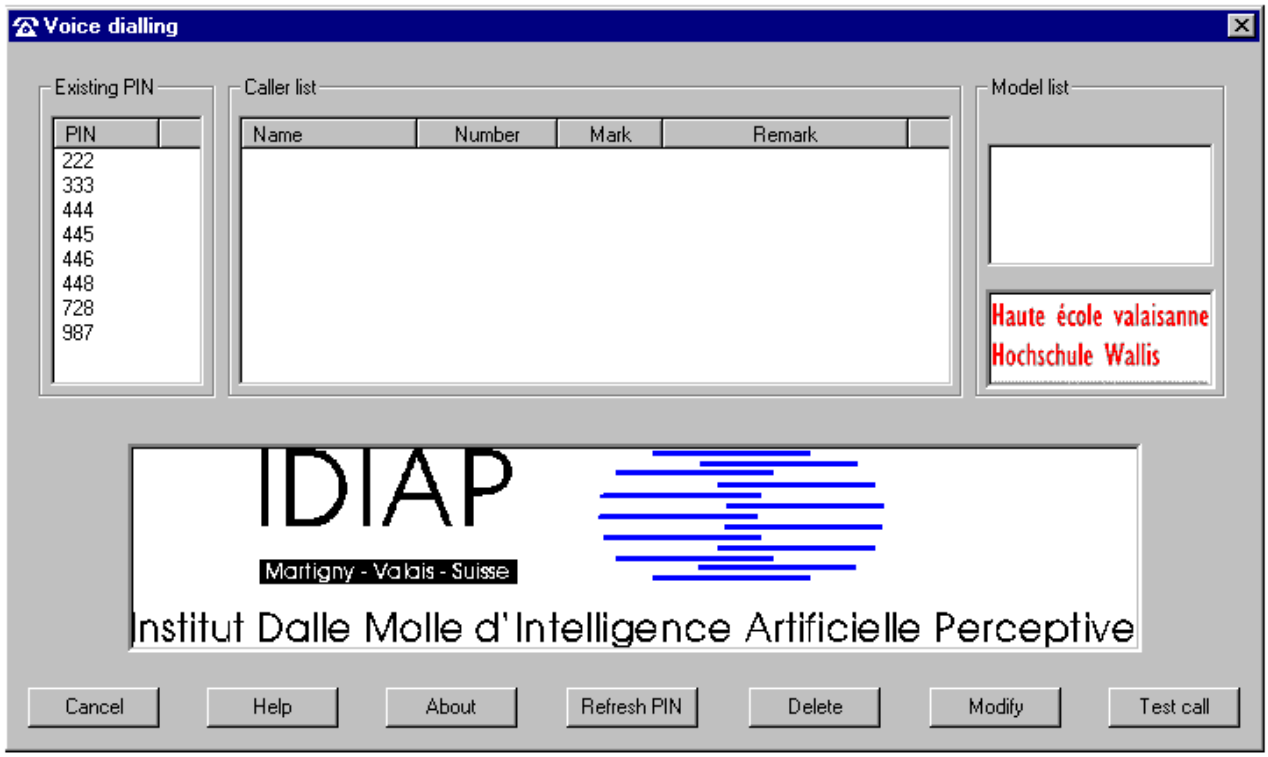
3. Application's description

Before modelling and coding, the application must be carefully defined. The more defined it is, the easier the modelisation and later the coding. With this definition, objects to use and to create can be correctly conceived.

Furthermore, this description could be used as specifications so that, during the coding, functions can't be added to the mandate.

3.1 Dialog box

The user can view the application on his PC[4] (central). With this dialog box, the user can call a part of functions that are normally called with the phone. This dialog box is necessary to initialize and shutdown TAPI, when the user opens or closes the box. Open this window is the first action to do if the user wants to use the application.

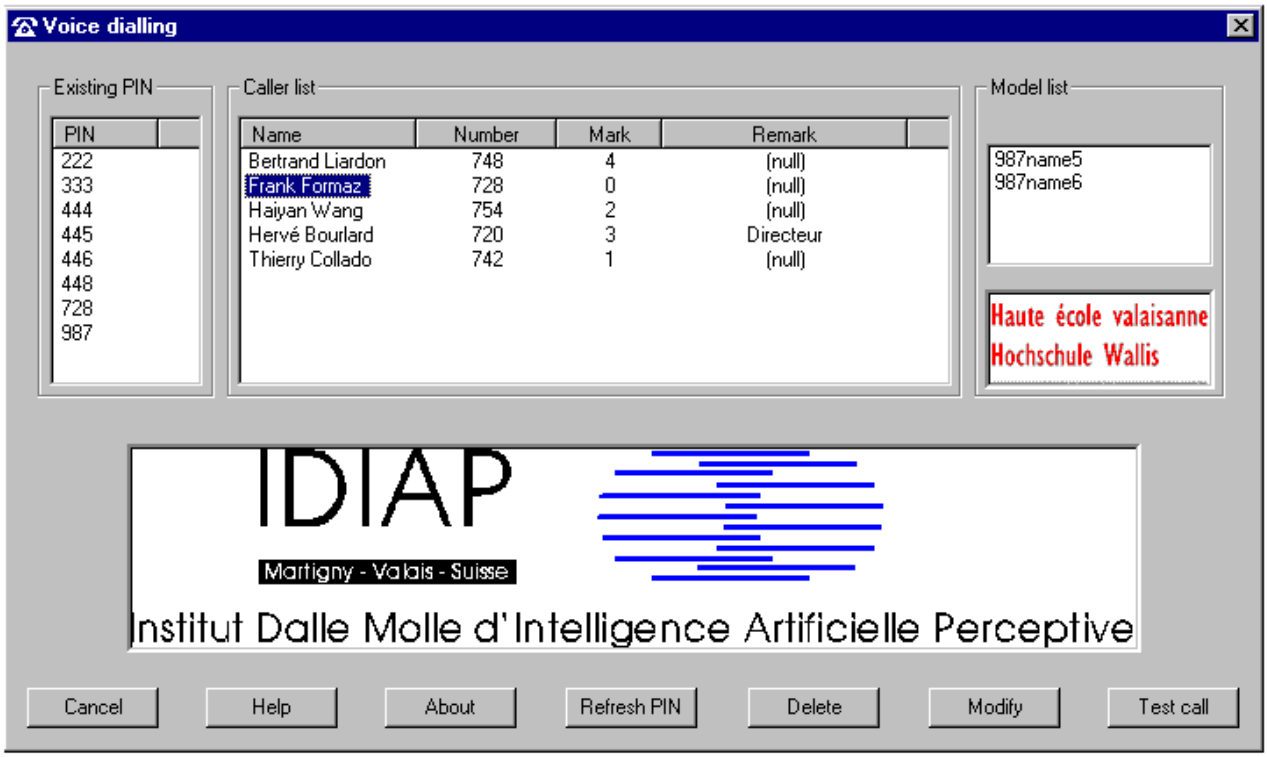


In this window, the user can see all PIN[5] codes that are already used. Each PIN code corresponds to a specified user. These PIN codes are not changed in the future because they are just a solution to differentiate all sound files and to assign a user table (database) to each user.

[4] PC : Personal Computer
[5] PIN : Personal Identification Number

3.1.1 View a position

The user chooses in the "*Existing PIN*" box his PIN code and double click over it. This action will be showing him his complete caller list from the database. Double click over a name to view the both corresponding models in the "*Model list*" box. These models are used for the recognition.



3.1.2 Help

If the user clicks over the "*Help*" button, the "*User's Guide*", which is also in annex E, will be opened. He normally can, with this document, use correctly the application.

3.1.3 Refresh the PIN list

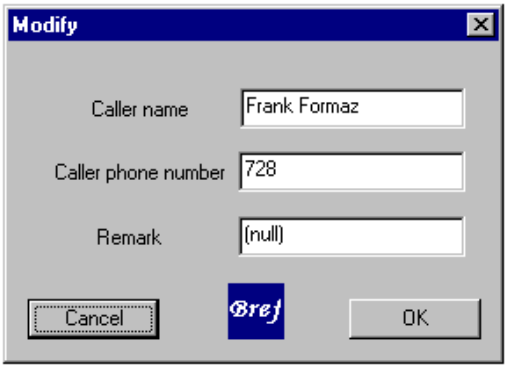
When a new user record his PIN code (with the phone), the application doesn't refresh the "*Existing PIN*" list automatically. To view the new PIN code and its corresponding "*Caller list*" click on the "*Refresh PIN*" button.

3.1.4 Delete button

Click on the "*Delete*" button to delete the selected or underlined position from the database and thus a new window comes. This window asks the user if he really want delete the selected name. If he clicks on the "*Yes*" button, the application deletes this position from the current database and also all sound files and models that are associated with the deleted name. The "*Caller list*" box is updated and shows the modified database, without the name that you have just deleted.

3.1.5 Modify button

To modify the caller name, the caller phone or the remark click on the desired name, in the "*Caller list*" box and thus a new window comes. The user can change each parameter that appears in the different fields and save them in the database with the "*OK*" button, then the "*Caller list*" box is updated. If he clicks on "*Cancel*", no change is saved.



3.1.6 Test call button

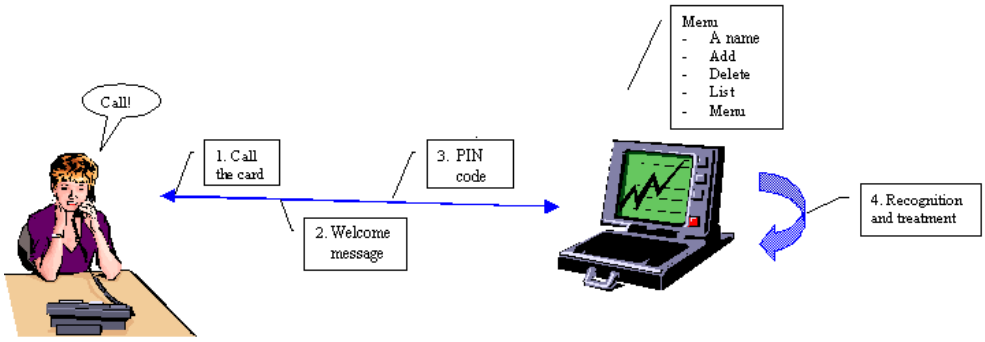
This button is to test if the application can generate a call. Since the dialog box cannot establish a call, because nobody is on the phone, there is a "*Test call*" button, which only call the number that corresponds to the selected or underlined name and hangs up after one ringing.

3.2 General use

The voice dialling wants to be a demonstrator that facilitates the phone numbering. The application plays first a welcome message. Then the user must enter his PIN code to open the great database and use the correct models to do the recognition. If the user is a new one, he must enter the "*000*" PIN code. After that, the application asks him to choose a valid PIN code, unused and different than "*000*". The application creates a new user table in the database with the given PIN code. Then it asks him to enter each key word and the two responses, yes and no, to create the models.

After this, the new user is like all other users. The application can ask him about his use wishes. Give one of the menu key words or direct a name that the user already has recorded to call it. The application recognise the given word and then, depends to the key word, the application do the appropriate actions. If this is not a call, the application comes back to the awaiting key state and that until the phone is hung-up.

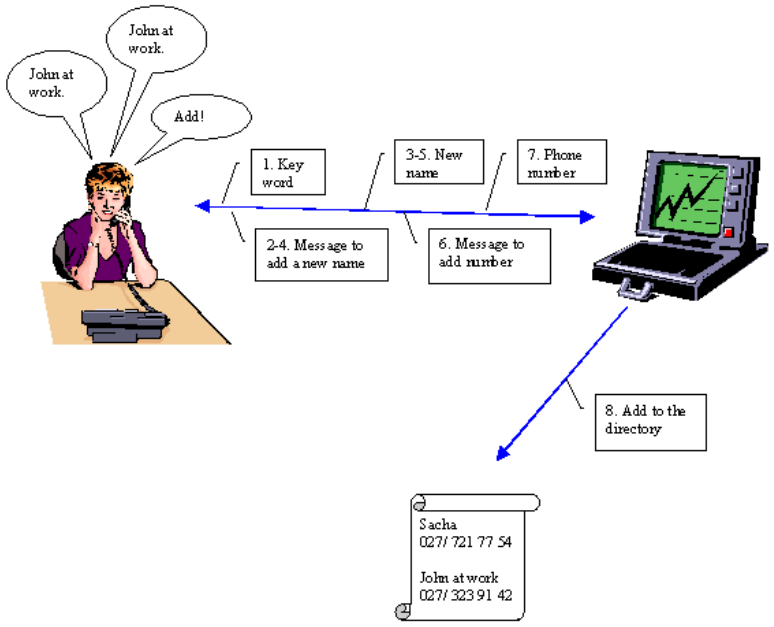
If the user pronounces the key "*call*", then the application makes the desired call, the user speaks with his caller and the process stops with the "*hang-up*" user's action.



See the flowchart in annex C.1 and C.2.

3.3 Add to the database

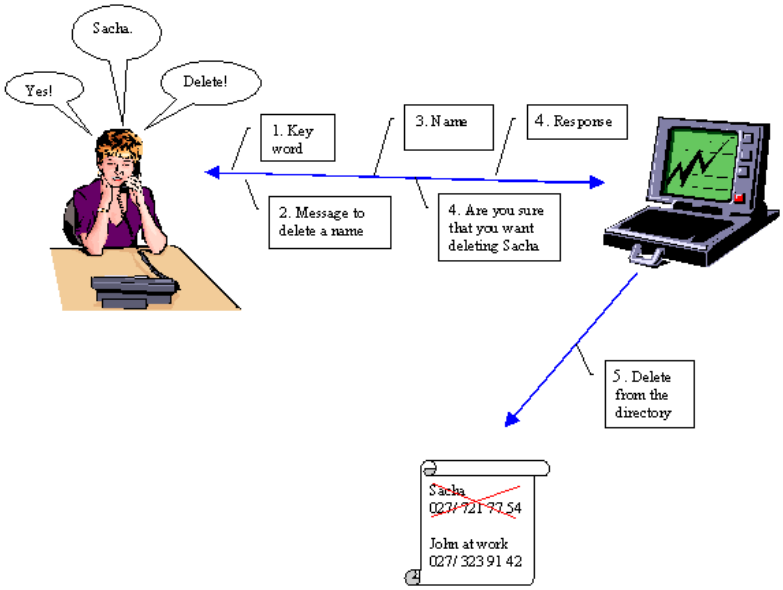
The user pronounces the key "*add*". He gives twice the new name that the application records and saves in the database. He also gives the number with the digital phone's keyboard. This number comes in a buffer and the application stops itself when this buffer is full. Then it saves the number in the database. Finally the application goes back to the awaiting key state.



See the flowchart in annex C.3.

3.4 Delete from the database

The user pronounces the key "*delete*". He gives the name that he wants be deleted. The application repeats the name, which is recognised, to be sure that it is the right one and, if the user confirms with a "*yes*", deletes it. This action also deletes all associated sound files and models. If he answers "*no*", the application doesn't made something and return in the awaiting key word state.



See the flowchart in annex C.4.

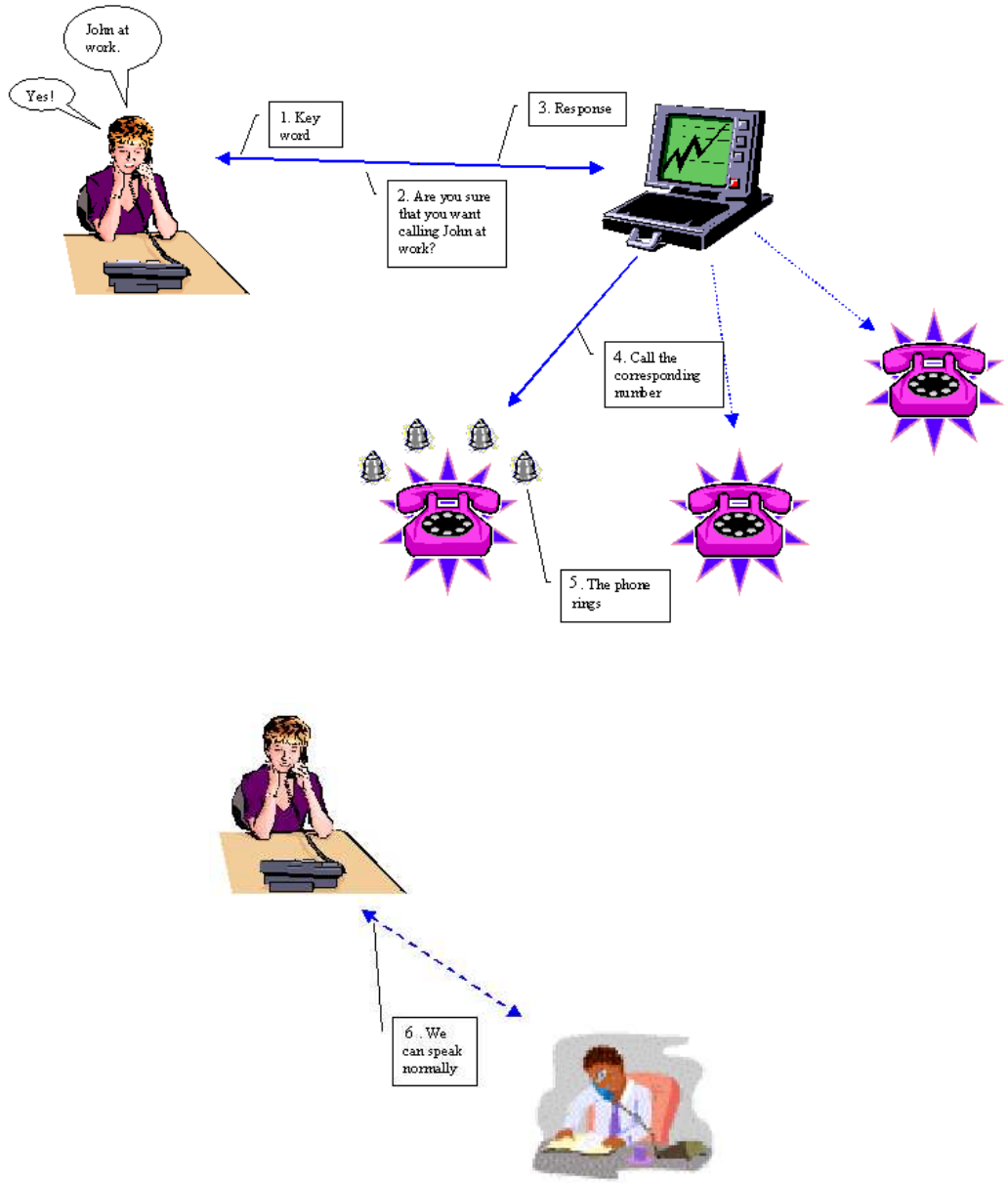
3.5 List the database

The user pronounces the key "*list*". The application enounces all names that are contained in the user table and goes back in the awaiting key word state.

See the flowchart in annex C.5.

3.6 Call someone

The user gives direct the name to call instead of a key word. This name is recognised by the application. The application asks the user if he really wants call this caller. If the user answers "*yes*", the application calls it and establishes the communication.



See the flowchart in annex C.6.

3.7 List key words

The user pronounces the key "*menu*". The application gives the key words that the user can say and goes back in the awaiting key word state.

See the flowchart in annex C.7.

4. Modelisation

4.1 What is UML

The Unified Modelling Language (UML) is a language for specifying, visualising, constructing, and documenting the artefacts of software systems, as well as for business modelling and other non-software systems. The UML represents a collection of the best engineering practices that have proven successful in the modelling of large and complex systems.

4.1.1 Importance of modeling

Developing a model for an industrial-strength software system prior to its construction or renovation is as essential as having a blueprint for a building. Good models are essential for communication among project teams and to assure architectural soundness. As the complexity of systems increases, so does the importance of good modelling techniques. There are many additional factors of a project's success, but having a rigorous modelling language standard is essential.

4.1.2 Goals of UML

The primary goals in the design of the UML were as follows:

1) Provide users with a ready-to-use, expressive visual modelling language so they can develop and exchange meaningful models.

2) Provide extensibility and specialisation mechanisms to extend the core concepts.

3) Be independent of particular programming languages and development processes.

4) Provide a formal basis for understanding the modelling language.

5) Encourage the growth of the OO[6] tools market.

6) Support higher-level development concepts such as collaborations, frameworks, patterns and components.

7) Integrate best practices.

[6] OO : Oriented Object

4.2 Classes and objects

All diagrams made for this application can be found in annex D. To interpret these diagrams, they are given a brief description hereafter.

4.2.1 Use case diagram

The first stage is to define which user interacts with the application and what is he capable of doing. All actions, which the user can make, are indexed in a use case

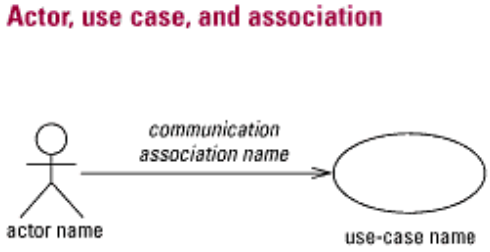
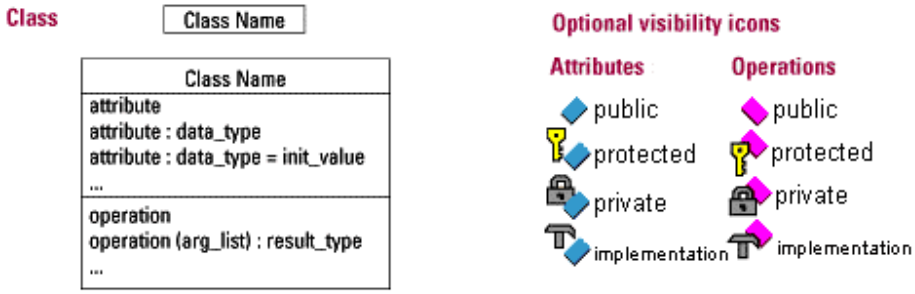


diagram.

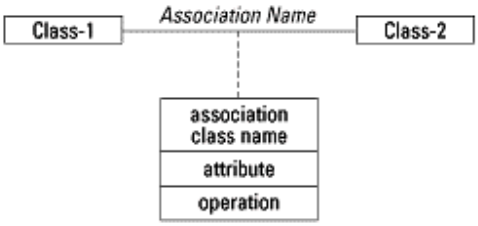
4.2.2 Class diagram

In a class diagram, there are many classes that the application uses. A class contains properties/attributes and methods/operations. For example, a class "*car*" has properties like wheel, bodywork or motor. It has also methods like start, turn, accelerate or brake. An object belongs to a class. For example, my car, which is an Audi A3, is a car.

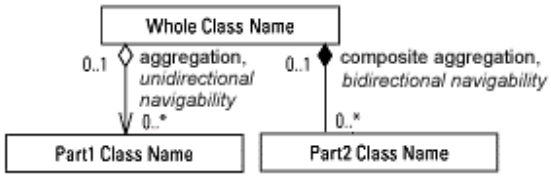


In UML, the application can be modelled with a class diagram. It contains all relationships that exist between classes. In this application the classes could be an association, i.e. a first class uses or knows a second class, or an aggregation, i.e. the first class has a second class's object. For example, my Audi A3 has a TDI motor, from the motor class. The black lozenge would show this relation. Another example is that a person could have no, one or many car(s), the white lozenge would show this relation.

Association classes



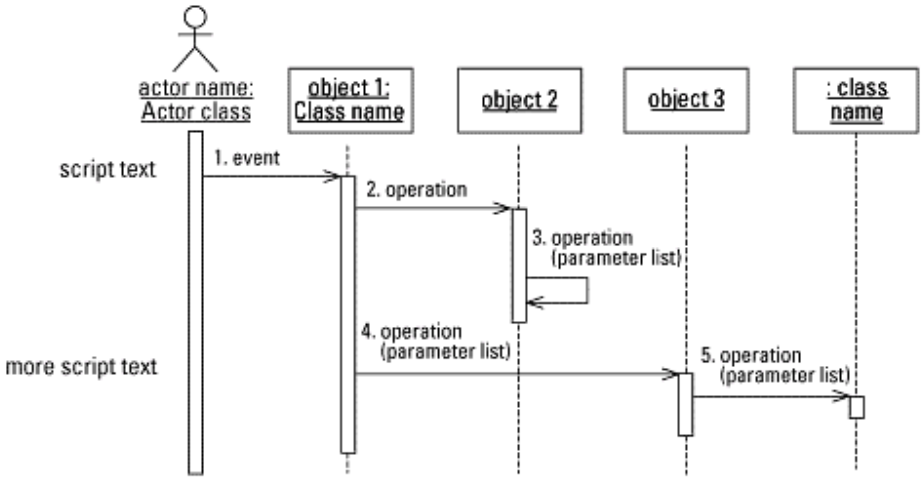
Aggregation, navigability, and multiplicity



4.2.3 Sequence diagram

The sequence diagram describes for each use case which objects are used and how they interact, and which methods they call. The arrow points over the object in which the used method is defined.

Sequence diagram

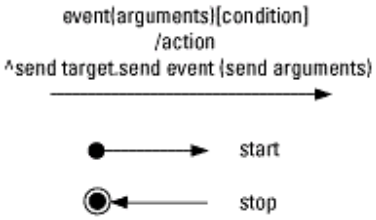


4.2.4 Activity diagram

The activity diagram shows the activities of a given context and indicates which class is designated to do each activity.

State icon

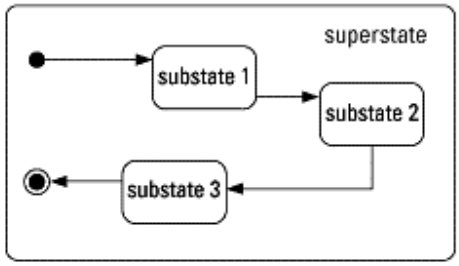
State transitions



History

Ⓗ

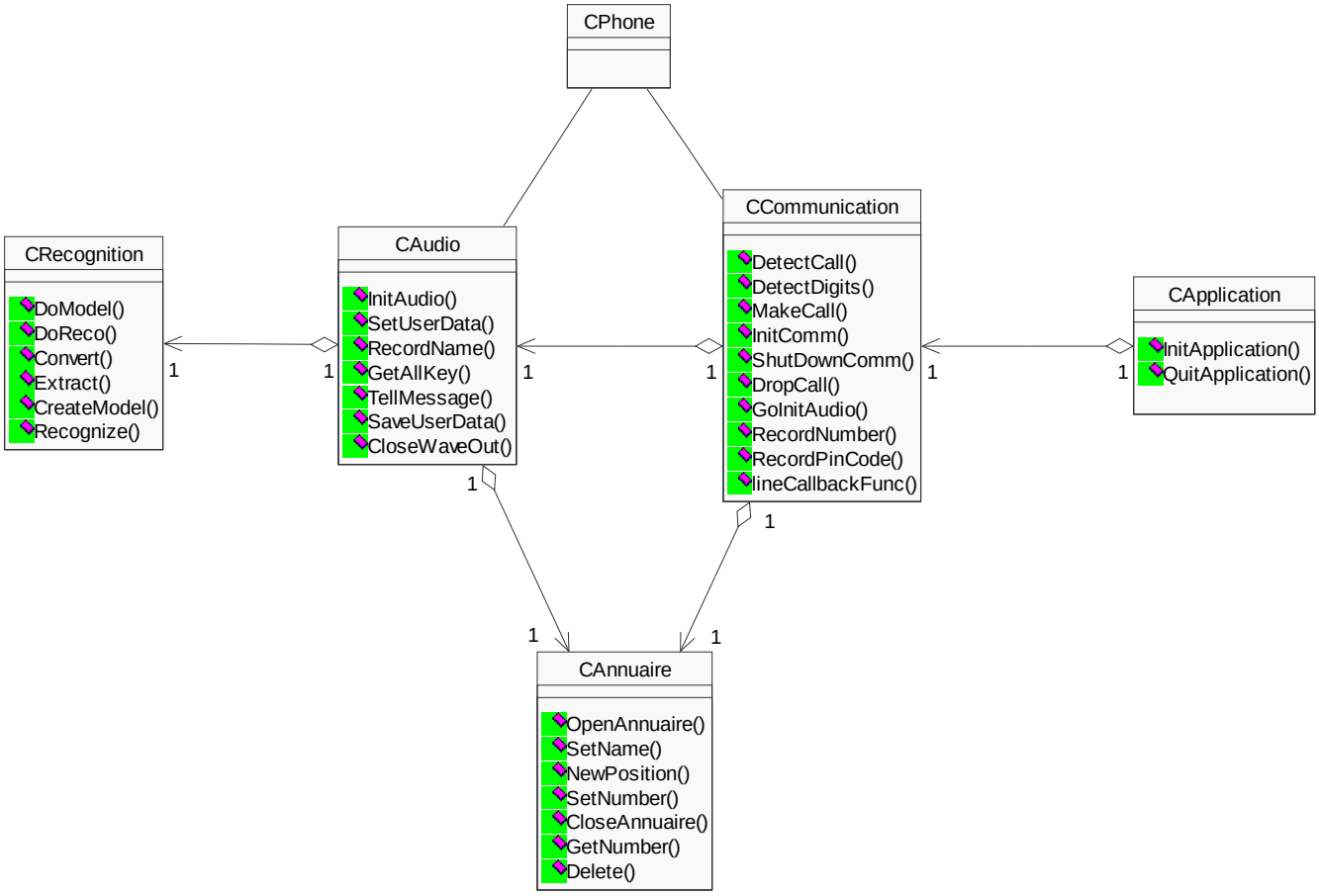
Nesting



4.3 Voice Dialing 's modelisation

4.3.1 Class diagram

Six classes are created and linked in this project but without the MFC [7]. "*CCommunication*" is a TAPI wrap class. "*CAudio*" is the SAPI wrap class. "*CAnnuaire*" is the phone book or the database. "CRecognition" is the recognition wrap class that wrap the IDIAP code. "*CApplication*" represents the screen interface and will be used by "*CApplicationApp*", "*CApplicationDlg*", "*CDeleteDlg*" and "*CModifyDlg*" that are created with "*Visual C*$^{++}$". And finally, "*CPhone*" is unused but represents all possible changes between the user voice and the transmitted data.

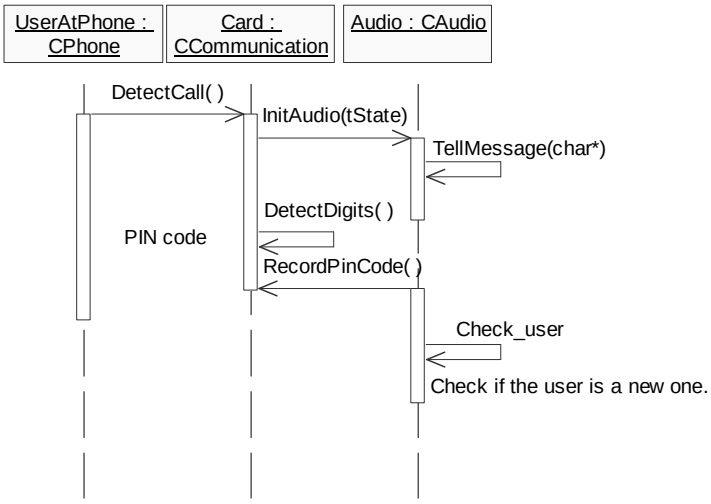


[7] MFC : Microsoft Foundation Classes

4.3.2 Sequence diagrams

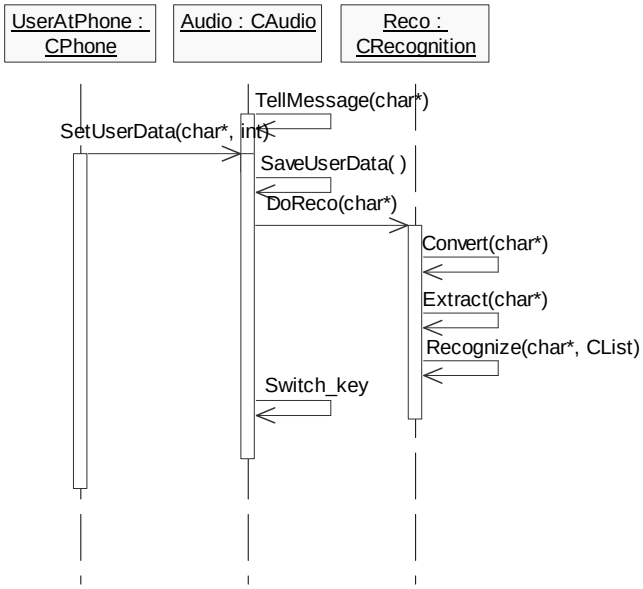
4.3.2.1 General use

When the user composes the card number, the card detects that a call is coming. The application asks the user to enter his PIN code. It checks if he is a new user.



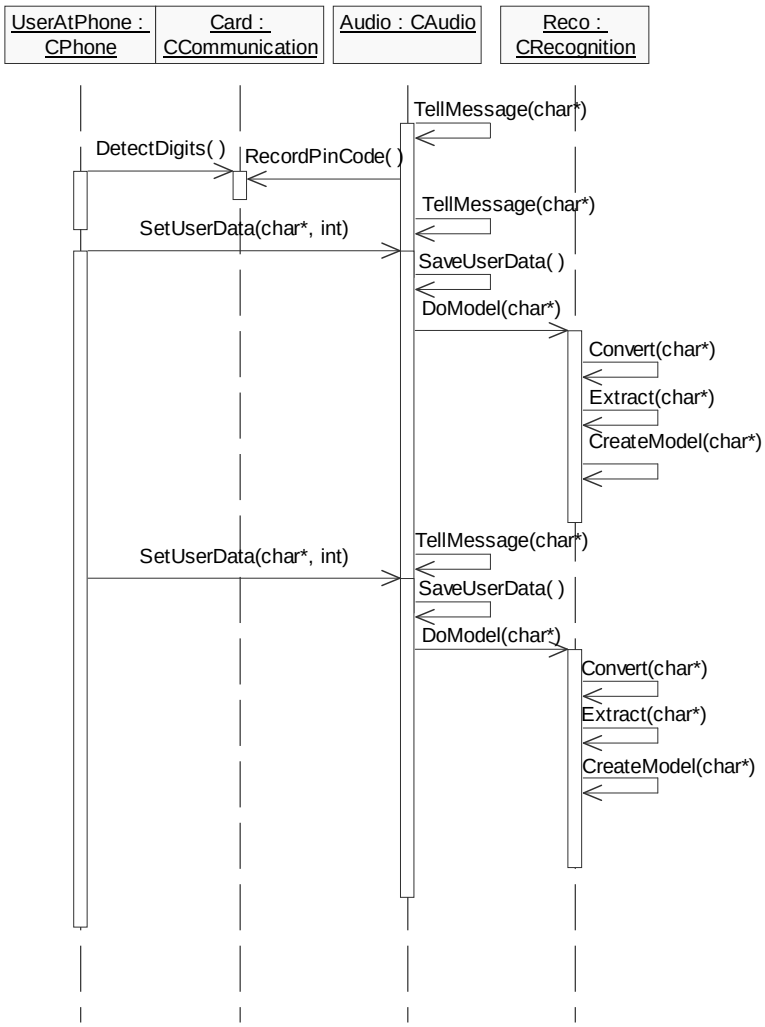
4.3.2.2 Awaiting key

The application tells the user that it waits about a key word. This key word is recorded and recognised. Then the application does the appropriate treatment.



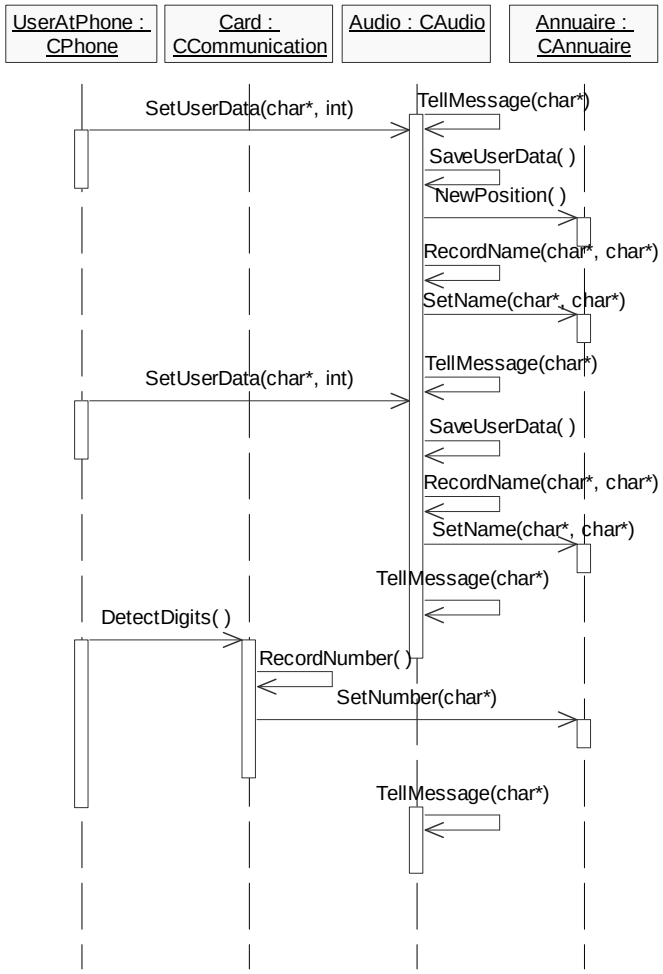
4.3.2.3 Create user table and models

If the PIN code indicates that the user is a new one the application creates his user table and records twice each key word and response.



4.3.2.4 Add to the database

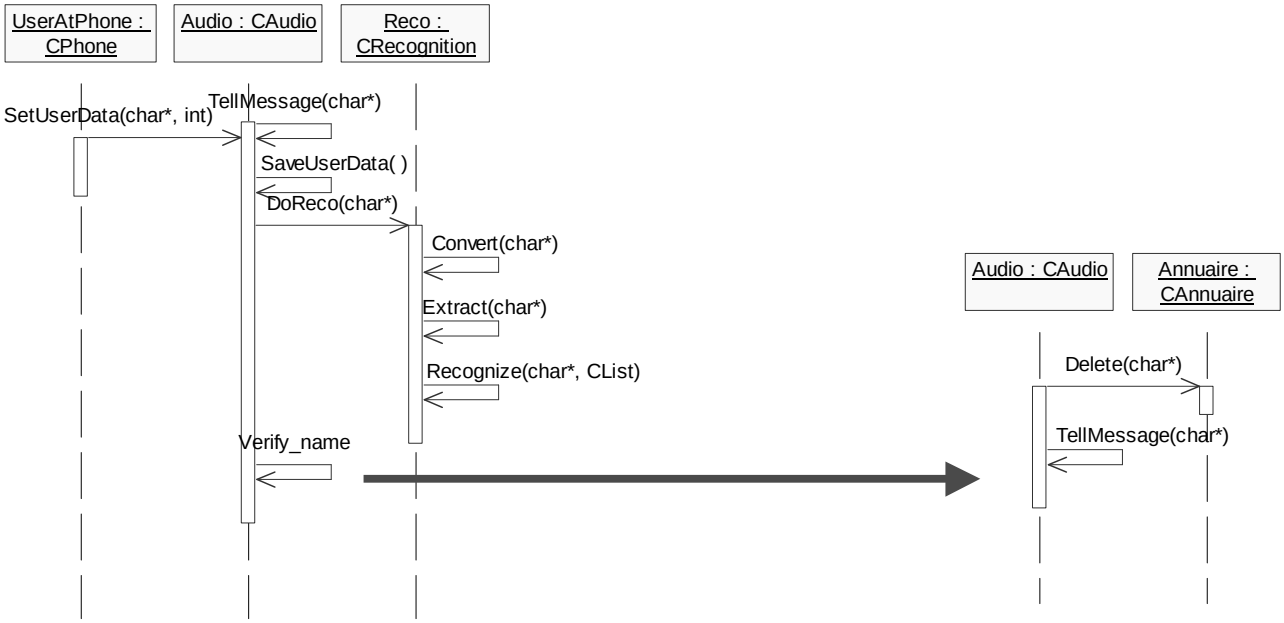
After the message, the user must pronounce the new name. This operation is twice made for a best recognition. He also adds the phone number that the application saves in the



database.

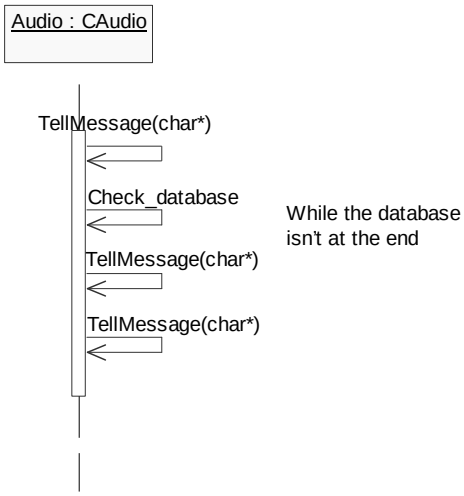
4.3.2.5 Delete from the database

The user gives the name to delete after the application message. The application repeats it to be sure that it is the right one and, if the user answers yes, deletes it.



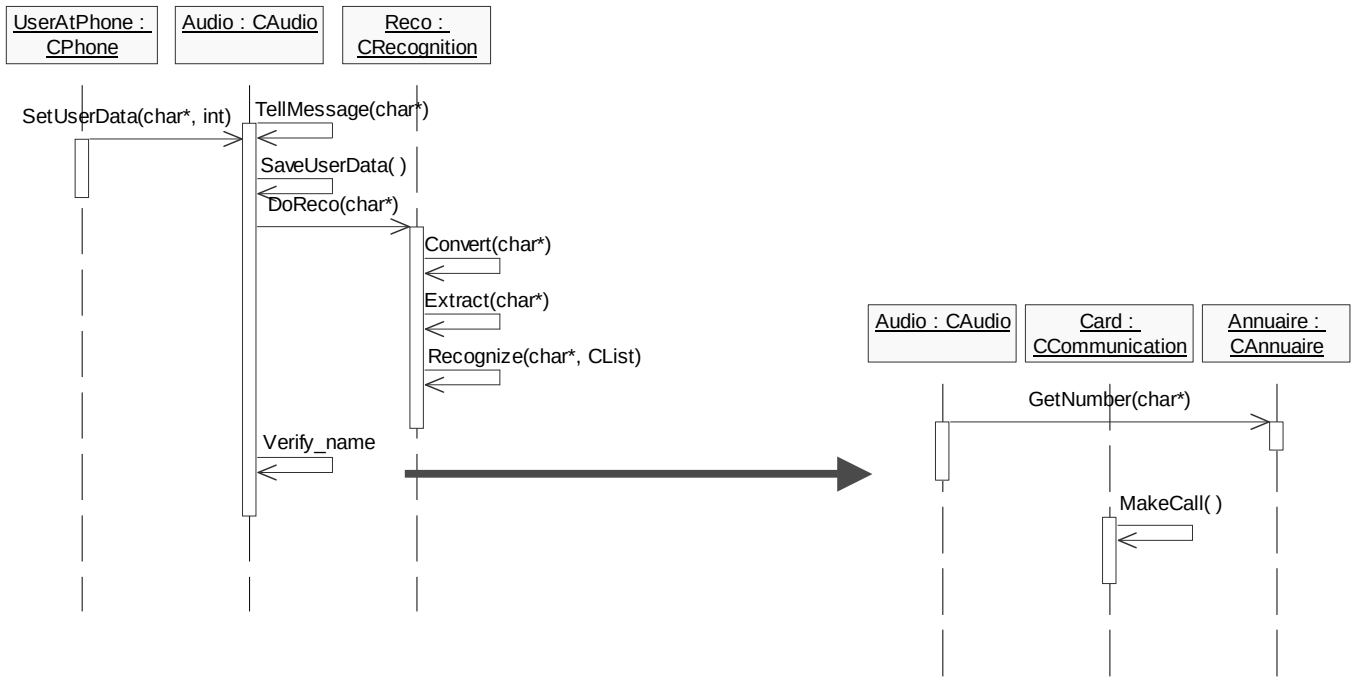
4.3.2.6 List the database

The application lists all names that are in the database.



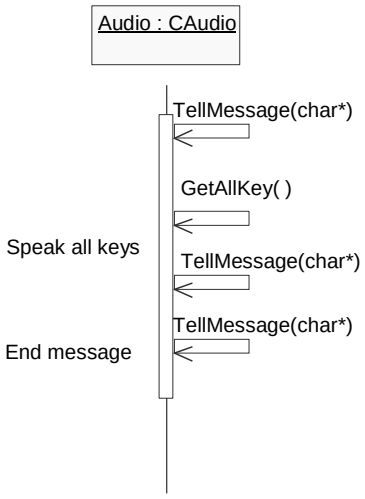
4.3.2.7 Call someone

The user gives the name to call instead of a key word. The application repeats it to be sure that it is the right one and, if the user answers yes, calls it.



4.3.2.8 List key words

The application enounces all possible key words and a message to inform him that he can also pronounce direct a caller name.



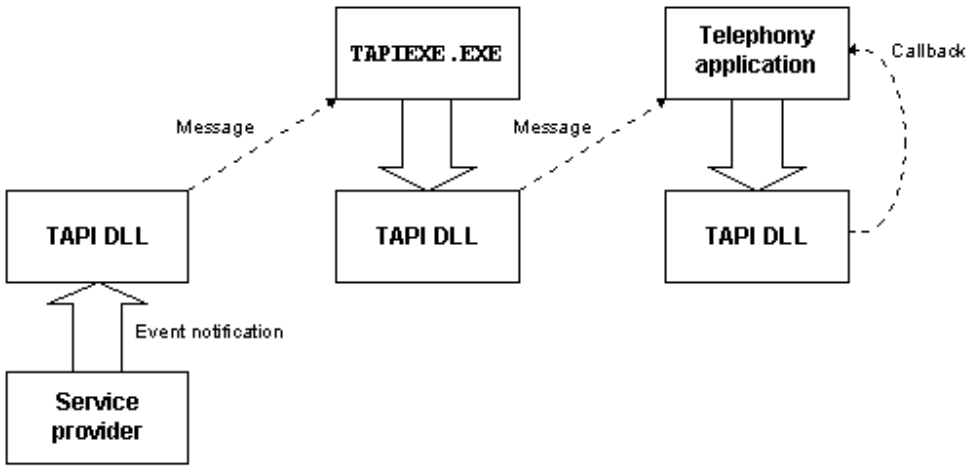
5. Coding

To realize this application, a new workspace must be first created with "*Visual C*$^{++}$" and at the same time a new project. This project must be a MFC one because all graphical objects like windows are managed by the MFC. The second stage is to create all the windows the application needs. The last stage before the code is to update the MFC project with the class model from UML. Finally the application coding could be start and if necessary the round engineering is applied.

Following paragraphs explain special functions or uses of application classes. In annex F, there is a code exemplary where all classes are detailed with comments.

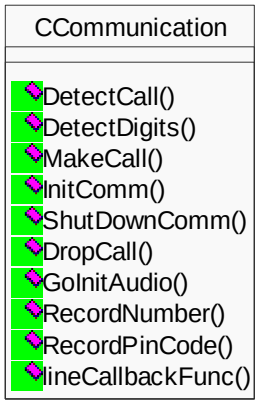
5.1 Class CCommunication

The most important part of this class is the call-back function. This function establishes a way to communicate with TAPI. It is used to notify application of changes in calls or lines device statuses. TAPI uses the call-back function to send messages to the application, via the TAPI DLL [8]. The following graph shows how the call-back mechanism works.



[8] DLL : Data Link Library

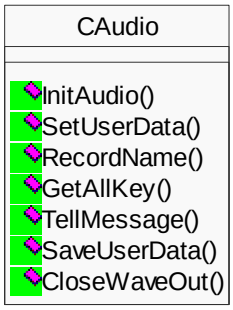
Almost all other functions are calls to the TAPI functions. One or two of these used a buffer that must be prepared before the function call. Information about these TAPI functions can be found in the MSDN library.



The last function, "*GoInitAudio*", is the audio class call. When the program comes back in this function, it is to detect digits, to make a call or to go in the waiting state. If a call wants be made, this function updates the phone number to call.

GoInitAudio() is to call the audio initialisation
InitComm() is to initialize TAPI, negotiate API, get capabilities and open a line
ShutDownComm() is to close the line and shutdown TAPI when the application is exited
MakeCall() is to establish the call with the given number
DropCall() is to close the line when the user hangs up the call
DetectCall() is to answers the specified offering call
DetectDigits() is to detect and buffer the incoming digits
RecordNumber() is to call the annul part to record the phone number
RecordPinCode() is to save the pin code for the audio part
LineCallbackFunc() is to send messages to the application

5.2 Class CAudio



InitAudio() is to synchronise all message from and to the card
SetUserData() is to record the user data in the given file
RecordName() is to record the new name in the directory
GetAllKey() is to give to the user all key that he can use
TellMessage() is to play the given file to the user
SaveUserData() is to save the given data into a file
CloseWaveOut() is to close all created objects and wave out device

5.2.1 Uses modification

SAPI is a complex and big API, which contains a lot of interfaces like "*ITTSAttributes*[9]" or "*ISREnum*[10]". There are also a lot of object interfaces like "*IAudio*" or "*IDirectAudio*", a total of 43 interfaces. Only after a through good study of each interface the best one can be taken.

When the great interfaces are chosen, they must be understand and correctly used. The functions of all interfaces are well defined, but how to link different objects (interfaces) together is vague. Moreover, classes or interfaces, "*CSTTTSQueue*" or "*ITelInfo*" for example, used for telephony application are wrap classes and, therefore, functions are not

[9] TTS : Text To Speech
[10] SR : Speech Recognition

so well defined. In this project, time doesn't permit allow thorough investigation, perhaps in a next project where COM[11] objects will be study.

Another important point is that the IDIAP develops recognition algorithms, which are of the latest technology. It is absurd to use algorithms worse than theirs. The project goal is to test an algorithm with any phone card, card independent.

As TAPI is tidy, all functions do what they must do. The first solution is to use "*Dialogic*" functions to record and play wave files. However, the two drivers are not compatible and the program will don't be independent of phone cards.

The second solution is to use only TAPI functions, include functions that permit to record and play wave files like "*waveIn*" and "*waveOut*" functions. Here is the program independent of phone cards.

This last solution will be hold to develop the application.

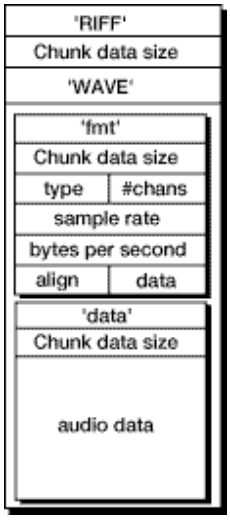
5.2.2 What is WAV[12]

Almost every browser has built-in WAV playback support. The RIFF[13] WAVE format was developed jointly by "*Microsoft*" and "*IBM*" as a method of saving high-quality sound. The default Windows WAV format is PCM [14], which is basically just uncompressed sound data, and these files tend to be rather large. However, many variations of compressed WAV files are possible (μ-Law and A-Law).

A WAV file contains digitized, sampled, sound data. A WAV file also contains information about the sound data format, such as the number of bits per sample and the number of audio data channels, mono vs. stereo. The various kinds of data in a WAV file are isolated from one another using an architecture based on chunks. A chunk is simply a block of data together with a chunk header, which specifies both the type of the chunk and the size of the chunk's data. A WAV file always contains at least two chunks, a data chunk that contains the sampled sound data, and a format chunk that contains information about the format of the sound data. These two chunks are packaged together inside another chunk, called a container chunk or a RIFF chunk. Like any chunk, the RIFF chunk has a header, whose chunk type is 'RIFF', and some chunk data. For RIFF chunks, the chunk data begins with a data format specifier followed by all the other chunks in the file. The format specifier for a WAV file is 'WAVE'.

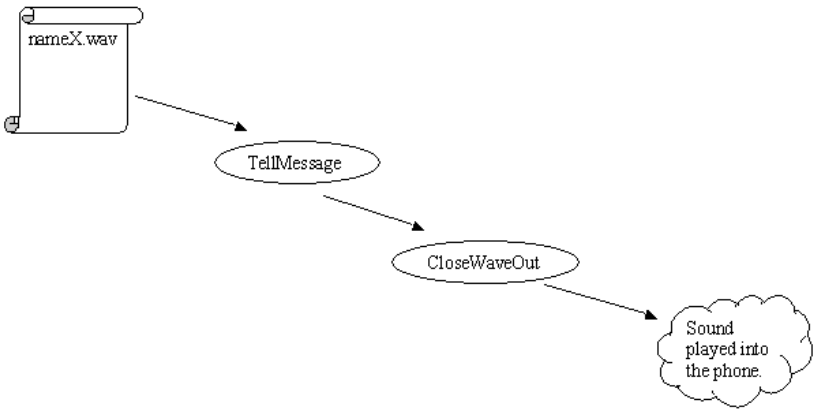
[11] COM : Component Object Model
[12] WAV : WAVe
[13] RIFF : Resource Interchange File Format
[14] PCM : Pulse Code Modulation

The general structure of any WAV file is as follows :



5.2.3 Message from the application to the user (TellMessage & CloseWaveOut)

The "*TellMessage*" function plays the given file through an opened WAV device. The WAV format must be prepared. It is a PCM one, with an 8 kHz frequency, 8 bits per sample and mono peculiar to telephony. The WAV buffer that receives sound data must be also prepared. In this function, the CFile class can be used to read the sound file, to define the buffer length and to put data into. When the WAV device is opened, a message window procedure must be created. This procedure calls the "*CloseWaveOut*" function, which just closes the WAV device and deletes the created objects used in this part, when the data are played. Then a synchronisation flag is set and the application can continue in



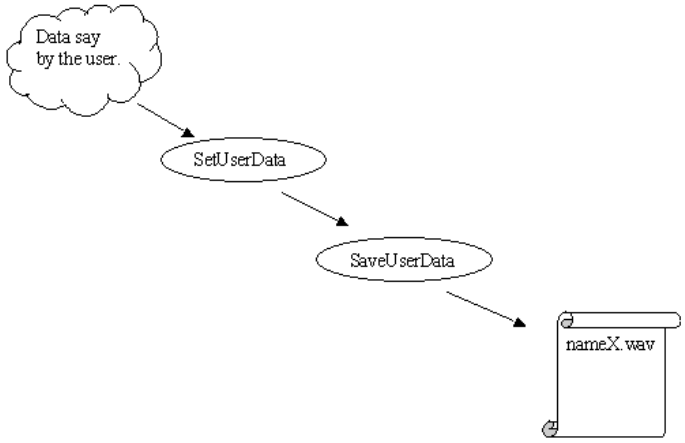
the audio sequence part.

5.2.4 Data from the user to the application (SetUserData & SaveUserData)

The "*SetUserData*" function records data into a WAV buffer. The WAV format, which is the same as previously, must be also prepared. When the WAV device is opened, a message window procedure must be created, this is the same as the WAV out. This procedure receives an end message when the buffer is full and calls the "*SaveUserData*" function.

The "*SaveUserData*" function save buffered data into a file, which is given to the function. To realize this work, mmio[15] functions must be used. These functions permit to create and to write a WAV chunk. Then WAV device can be closed and objects deleted.

The data file must have a unique file name to differentiate each recorded name. To realize this unique name, a counter is used. The name will be as follows: nameX.wav, where X is the counter value. This value must be saved in a file (mark.brf) to retrieve all files in a next use and updated it with the application opening. If this counter is always initialised to zero, data are written over existent ones that are loosen.

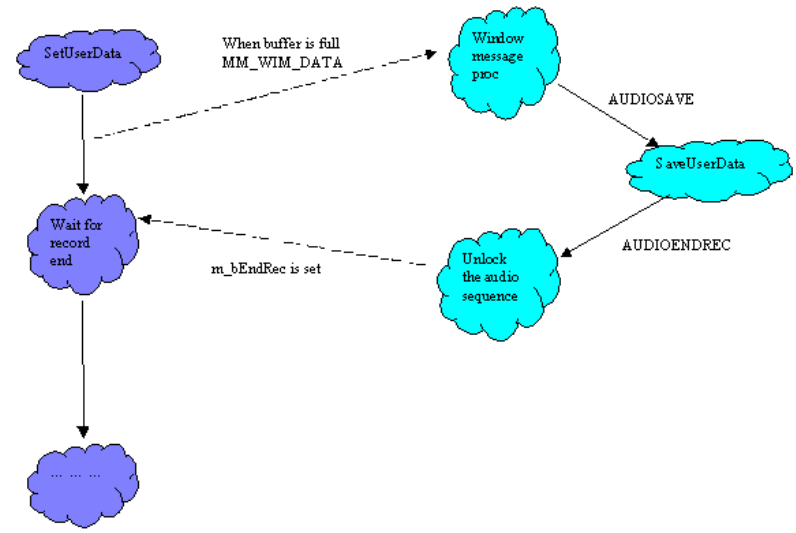


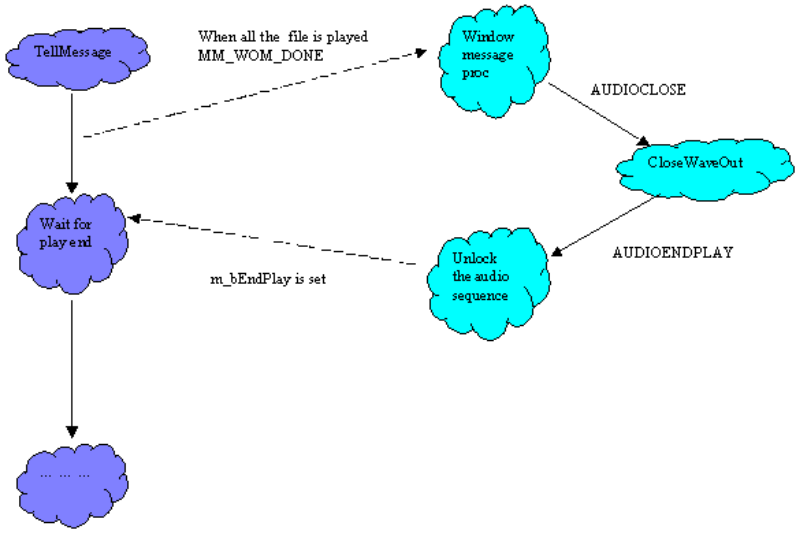
Finally, a synchronisation flag is set and the application can continue in the audio sequence part.

[15] MMIO: MultiMedia Input Output

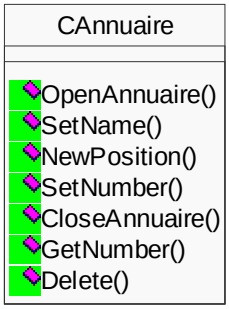
5.2.5 Window message principle

The message window is fast the same as the call-back function. This window receives message from other classes, WAV in and WAV out, and calls the appropriate function. This principle is being chosen because the call-back does not work correctly, after a buffer full message it cannot close the WAV in device. This method permits to quit the audio class to close WAV in, or WAV out device.





5.3 Class Cannuaire



In this class, a database is created to index all callers. The most appropriate database file to use will be an "*Access*" one, SQL[16] requests must be known.

A database is a file where you can index information. This information could be composed of many fields. For example, in this application the fields are the followings: identifier, phone number, caller name, model1, model2 and remark.

A database could be like the following one:

User987 : Table

	Id	PhoneNumber	CallerName	Model1	Model2	Remark
▶	5	728	Frank Formaz	987name5	987name6	Project chef
	6	742	Thierry Collado	987name7	987name8	
	7	754	Haiyan Wang	987name9	987name10	
	8	720	Hervé Bourlard	987name11	987name12	Director
	9	748	Bertrand Liardon	987name13	987name14	
*	(AutoNumber)					

Record: 1 of 5

In this case, the application must recognize what the user said. For example, when he says "*add*", the application must recognize this key word and execute the appropriate function. Each person has neither the same voice nor the same pronunciation. Then the application must used models, which correspond to the user, to do the recognition. For this reason the PIN code is used to build all file names. When the user 123 tell something, only the 123 model names are used to do the recognition.

All model are created as follows:

Pin code Model name Counter

123name15

123add1

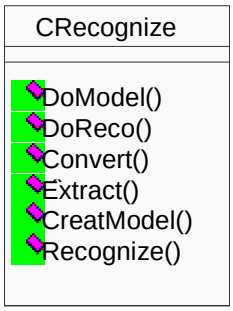
[16] SQL : Structured Query Language

To each user it corresponds a table. To create all table names, the application also uses the PIN code.

SetName() is to put the given file in the directory
NewPosition() is to create a nem Pin-code table in the database
GetNumber()is to get the phone number of the associated file
SetNumber() is to record the number into the two last recorded name
Delete() is to delete the specified file

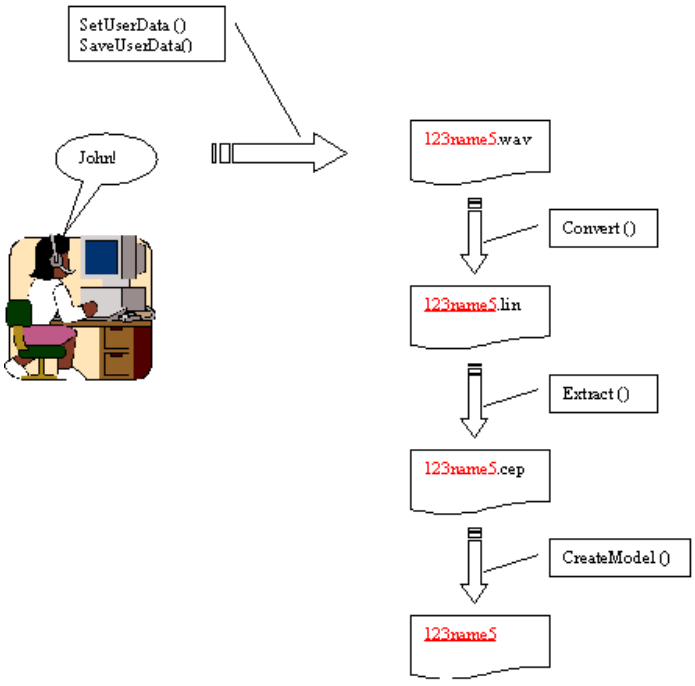
5.4 Class CRecognition

This class is to do speech recognition. It receives a wave file which recorded by "SetUserData" and "SaveUserData" functions of CAudio class. Then the mainly work is divided into two parts: create a model file and do recognition.



5.4.1 Create keywords Model

The speaker dependent speech recognition, which we employ in this application, requires that the keywords models are created for each user. Activated by input the command code "000" through the telephone keyboard, the modelling process, first, asks the user to input his pin code, then records his voices of the indicated keywords. A small specially developed programme called "Convert" is used to decode and encode the recorded Microsoft WAV data into the well used speech processing data format - NIST sphere. After extracting the 13-dimensional vectors, the keywords are spotted into the proper models with k-means clustering. At last, the created model names are recorded in the database while are returned to the application. The processing flow chart is as follows:



Function "Convert()" transforms a wave file to a NIST sphere file, because the “SaveUserData” function records user’s voice as a Microsoft WAV file but not a NIST sphere file, which can be well processed by our feature extraction and keywords spotting software.

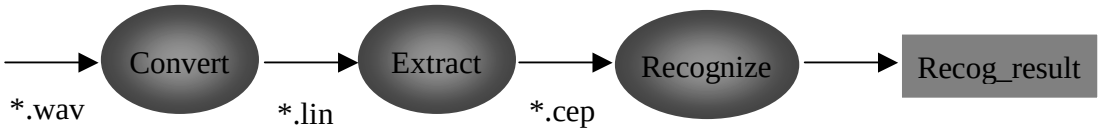
Function "Extract()" creates an acoustic vectors file. The acoustic vectors are generated by Acoustic Front End (a part of ANALYSE). 13-dimensional vectors are generated every 10 ms 13, which consist of 12 static cepstral components and one log-energy component.

Function "CreateModel()" generates a template for each keyword with a vector quantization algorithm, which will be used as the model to do future recognition. This

function will return the created model's name to let the application store it in the database.

5.4.2 Recognition

After a user has created his keywords models, the names of these models will be returned as the recognition results when the user repeats these keyword again later. If the system refuses to recognise some keywords, it returns the string "Nothing recognised". The processing flow chart as follows:

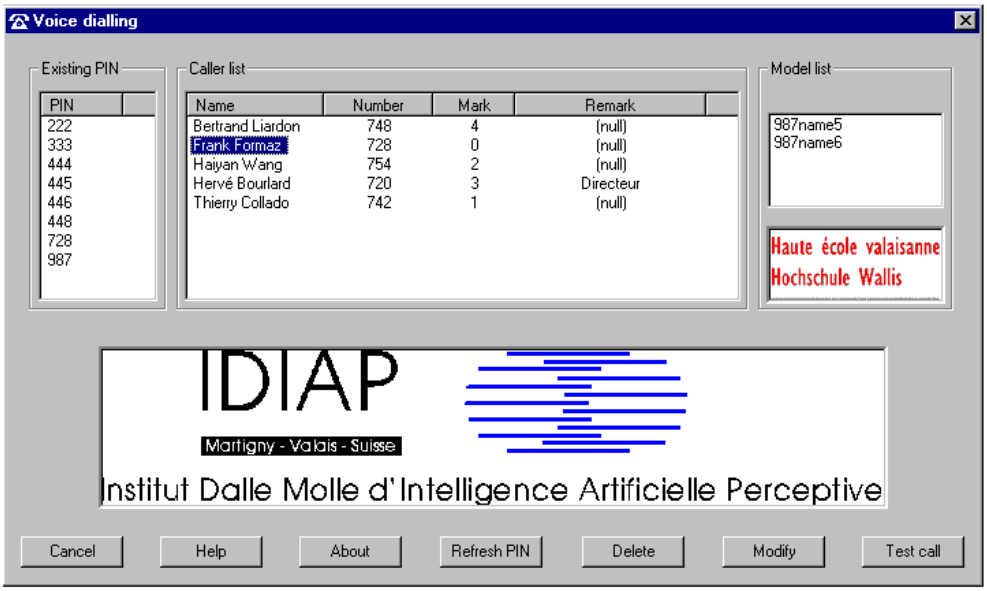


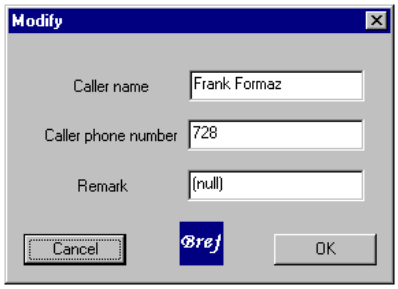
Recognize() applies Dynamic Time Warping technology. DTW compare the utterance with all the models for the closest one (or recognising as garbage). The Recognize() function returns the matching mode name or "Nothing recognised".

5.5 Classes CapplicationDlg, CDeleteDlg and CModifyDlg

Both dialog boxes are designed and functions are implemented. These dialog boxes are implemented to give to the user a best comfort. If the user want just delete a caller name he can without all phone steps.

In these three classes, there is no relevant speciality to explain.





6. Tests and validation

During the realisation, each part is developed and tested separately before its integration in the application code. All major problems are resolved before the part integration. Thus primary tests are regularly made.

- ✔ TAPI realises outgoing calls and answers to the incoming ones (not tested in the same time).
- ✔ WAV files can be played and recorded.
  - ✗ Recording without silence detection.
- ✔ A database can index all caller names.
- ✔ Dialog boxes show desired information.

The important part, now, is to test if the application is correctly sequenced and if all defined functions are implemented. Firstly, we can test if the description flowcharts are respected.

- ✔ Add to the database.
- ✔ Delete from the database.
- ✔ List the database.
- ✔ Call someone.
  - ✗ Test call is OK but when another call is already answering the application cannot call another phone.
- ✔ List key words.

Secondly, when the recognition files and the PBX will be correct, some special uses that a user can do can be tested.

- ✔ Not enough digits or no digits when the application wait on, these errors are detected.
- ✔ Unknown keys, responses or names are treat.
- ✔ Hang-up the phone when the application is doing something is detected.

In annex E is the "*User's Guide*" applied to the version using the recognition. To use this demonstration version, the user can follow this guide.

In this version the user can add twice the new caller name, add the corresponding phone number and reheard all names that are in the directory. All screen interface functions describe in the user's guide are implemented and usable.

7. Project's status

The project is not at its end, because it cannot execute all functions that are describe in paragraph 5. The reason is that the recognition files are coded on a "*Unix*" environment and in this project a "*Windows*" one is used. A goal summary can be as follows:

- ☺ Application: run correctly until the calling
- ☺ UML: model is created but no round engineering is made
- ☺ Visual C^{++}: code is generated
- ☺ Dialogic: the D/ 21H card is functional and used
- ☺ TAPI: this part is the most successful one
- ☺ Screen interface: made all desired functions except the add one because we can't record the both model
- ☺ Database: an Access one is used and a table for each user is created
- ☹ SAPI: it is replace by WAV functions to record or play files and IDIAP algorithm to recognize user data
- ☺ Recognition files, which are given by the IDIAP, run on "*Windows*"

8. Project future steps

Since this project is not ended, not perfect, here are the future improvements to bring.

- Firstly, the application must generate call, also if a call is answering, and may be the conference functions will be used to establish the link between the two phones.
- Secondly, the data record will be made with a silence detection and not with a defined buffer length.
- Thirdly, the application will be tested with other cards from other manufacturers to know if TAPI can control all cards, card independence.
- Fourthly, SAPI could be another project that requires COM objects and recognition principle study.

9. Conclusion

9.1 Project progress

A bibliographical study of different APIs and technologies was made during the semester project. This knowledge will be used while integrating voice recognition in a voice dialling application via phone interface. "*Microsoft's*" APIs, which are TAPI, with a "*Dialogic*" phone card, will be used. After that, the compatibility between Hardware and APIs will be demonstrated. Then before the code realisation, the application will be modelled with UML[17]. Finally the documentation and the report will be written.

Knowledge has been applied in this "*Voice Dialling*" application and has been broaden. Sound files, sound headers, recognition principles and databases have been used thoroughly.

The report writing has been also an interesting and useful work that will be bring me practice in future job. With this report, English bases are revised and improve so much in the writing than in the reading.

Thanks to all people that have collaborate to this project and particularly to the IDIAP and to the EIV teachers for theirs knowledge.

[17] UML : Unified Modelling Language

Voice Dialling 1.2

User's guide

10. Introduction

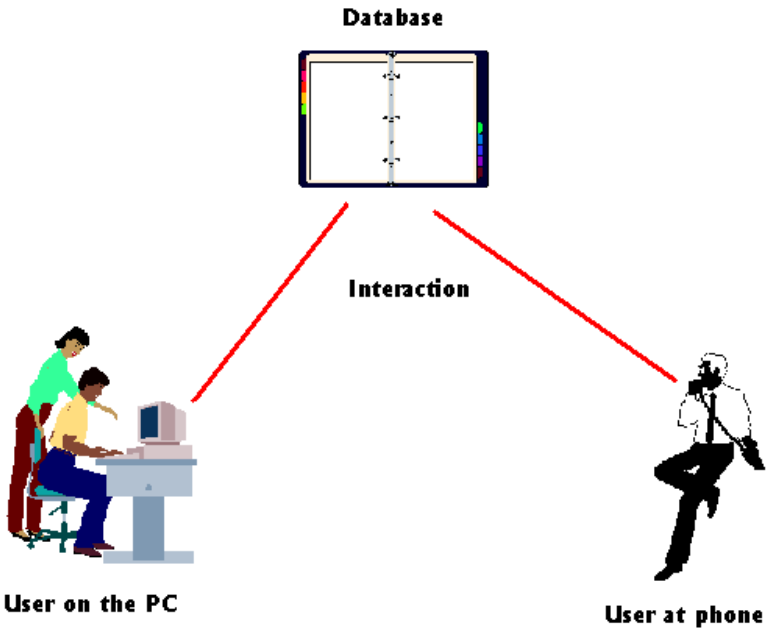
This application is based on a database. A database is a file where you can index information. This information could be composed of many fields. For example, in this application the fields are the followings: identifier, phone number, caller name, model1, model2 and remark. A database could be like the following one:

User987 : Table

Id	PhoneNumber	CallerName	Model1	Model2	Remark
5	728	Frank Formaz	987name5	987name6	Project chef
6	742	Thierry Collado	987name7	987name8	
7	754	Haiyan Wang	987name9	987name10	
8	720	Hervé Bourlard	987name11	987name12	Director
9	748	Bertrand Liardon	987name13	987name14	
(AutoNumber)					

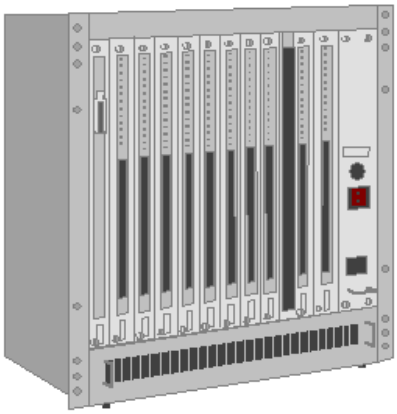
Record: 1 of 5

This database is like a phone diary but over a computer. The user also can add a phone code of his friend, delete it or modify something but without pencil.
To execute these actions, the user can use a screen or a phone interface. How do run these? It is explained later.

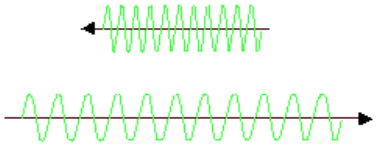


11. Private Branch Exchange

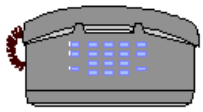
All actions that the user does with his phone are transmitted with DTMF[18] signals. For example, when the user picks up the phone, a 420 Hz signal is send to the central to inform it that the user wants call someone. And when the communication is connected, the central returns another signal to inform the phone card or the phone that the communication could be establishes.



PBX, phone central



Signals DTMF



User phone

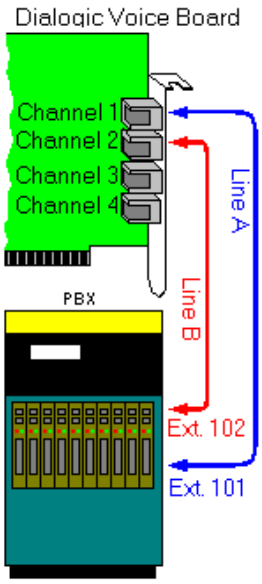
For this reason, control that the PBX[19] (central) type is the same as the installed one. *Start, Settings, Control Panel, Telephony, Telephony Drivers, Configure, Advanced and PBX strings*. The card must receive known signals to run correctly, e.g.: it must answer when the user calls the card and not when he hangs up the phone...

11.1 Dialogic card

If this type is not the correct one, use the "*PBXpert*" program to detect signalling tones. *Start, Programs and Dialogic System Software*. This software asks you the PBX manufacturer and model. After this, you just must make the correct connections, like shown on your screen, and the program tests all used tones, which are written in an appropriate file.

[18] DTMF : Dual Tone Multi Frequency
[19] PBX : Private Branch Exchange

Caution, to detect these signals, the application uses two phone lines.



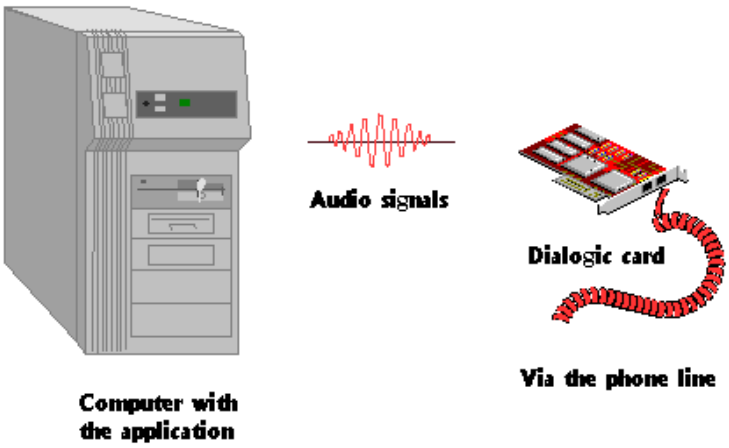
Then change the PBX type and put the created one. *Start, Settings, Control Panel, Telephony, Telephony Drivers, Configure, Advanced and PBX strings*.

11.2 Other cards

Look in your card menu if there is a program that can detect PBX signalling tones. If there is not, control if your card can detect itself different PBX signal.

12. Canal

This application plays sound files, to the user, to inform him what he must do, and records, from the user, others ones that are choices or names. For these reasons, the card canals must appear in the audio devices.



The card canal to use is, in principle, the 1 one. You must connect the phone line with this one. This canal must appear in second position in the playback and recording preferred device. *Start, Settings, Control Panel, Multimedia, and Telephony Audio*.

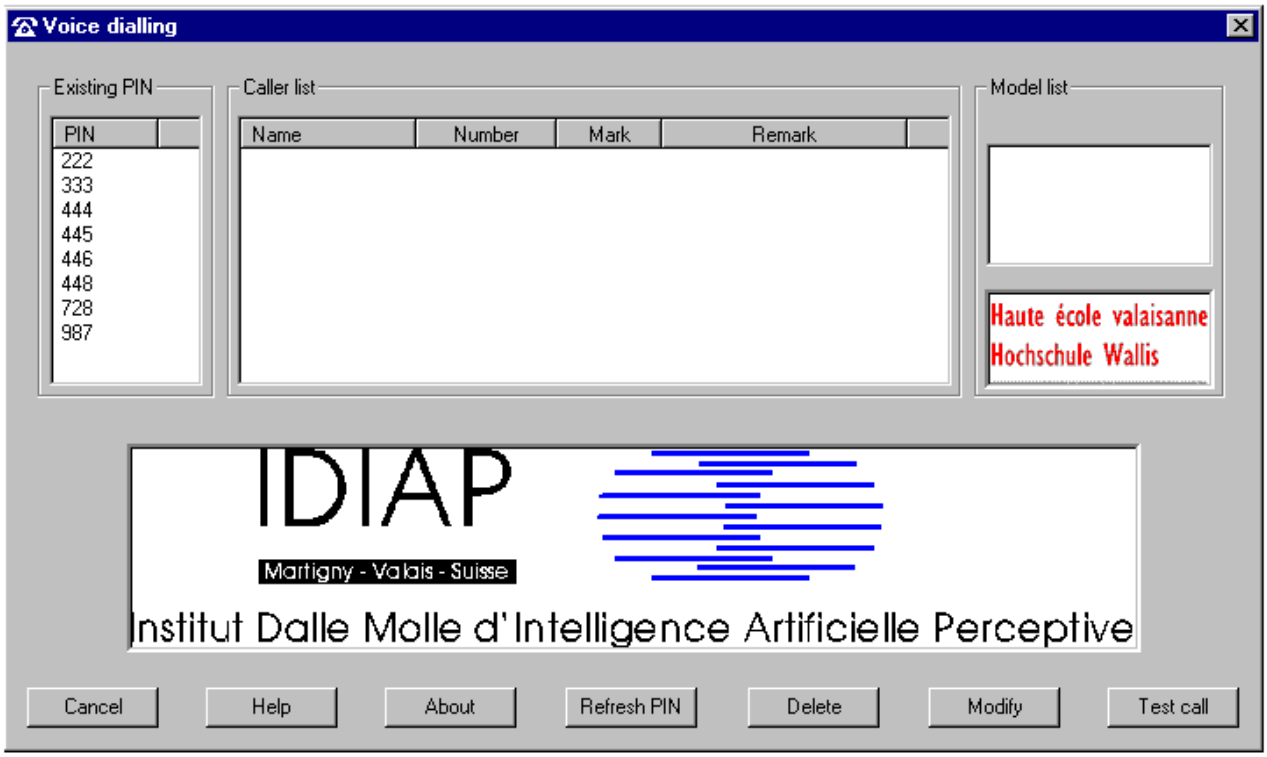
If you have no sound card and the both canals appear in the two first places, then use the canal 2.

13. Run the program

Double click over the "*Application.exe*" icon will run the "*Voice Dialling*" application. This action will be initialize TAPI[20] and put the application on waiting for an incoming

call.

With this double-click, the following dialog box will appear. This is the application



window.

[20] TAPI : Telephony Application Programming Interface

14. Use this application

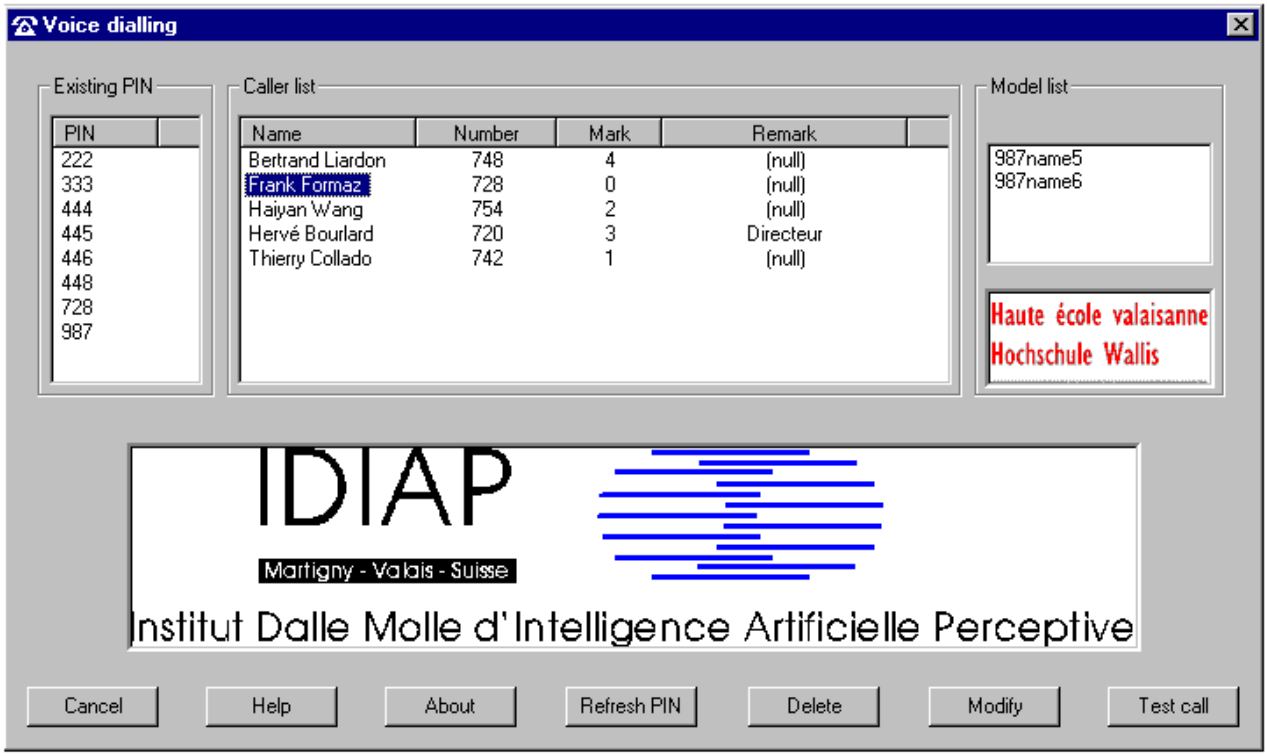
At the moment, the user can use either the screen or the phone interface to do some operations with his database.
If the user uses the screen interface, he can execute some functions with all buttons that are on the window. The screen interface is not necessary to use a phone.
If he uses the phone, he can also execute some functions but in the form of a dialog with the application. He must firstly call the telephone number of card, like he calls telephone number a friend.

15. Screen interface (user on the PC)

In this window, the user can see all PIN[21] codes that are already used. Each PIN code corresponds to a specified user. These PIN codes are not changed in the future because they are just a solution to distinguish from all sound files and to assign a user table (database) to each user.

15.1.1 View a list

The user chooses in the "*Existing PIN*" box his PIN code and double click over it. This action will show his complete *caller* list from the database. Double click over a name to view the both corresponding models in the "*Model list*" box. These models are used for the recognition, it will be explained later. If you click one PIN code in the *Existing PIN*



[21] PIN : Personal Identification Number

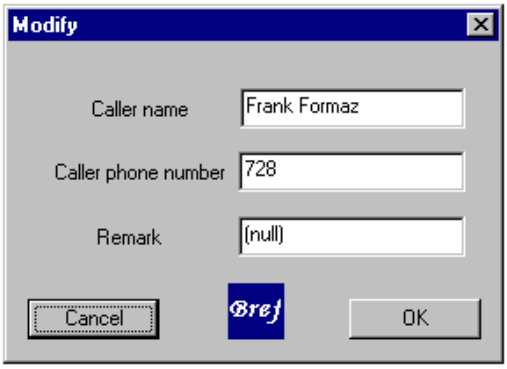
list, the all corresponding caller will appear in *Caller* list; Then if you click one caller name in *Caller* list, the corresponding models will appear in *Model list*. Moreover, if you double click one caller name in *Caller* list, a pop-up window will appear, you can edit it and save what you changed after you click "*OK*" button(this function is the same as clicking "*Modify*" botton).

15.1.2 Delete button

Click on the "*Delete*" button to delete the selected or underlined record from the database and thus a pop-up window comes. This window asks the user if he really wants to delete the selected record. If he clicks on the "*Yes*" button, the application deletes this record from the current database and also all files that are associated with the deleted name. The list box is updated and shows the modified list, afterwards it will show the all exist records without the just deleted one. If he clicks on the "*No*" button, the application does nothing. Here you can delete record in *Existing PIN* list or *Caller* list.

15.1.3 Modify button

To modify the caller name, the caller phone or the remark click on the desired name, in the "*Caller list*" box and thus a new window comes. The user can change each parameter that appears in the different fields and save them in the database with the "*OK*" button, then the "*Caller list*" box is updated. If he clicks on "*Cancel*", any change not saved.



IDIAP
Martigny - Valais - Suisse

Hes·so
Haute Ecole Spécialisée
de Suisse Occidentale

15.1.4 Test call button

This button is to test if the application can generate a call. Since the dialog box cann't establish a call, because nobody is on the phone, there is a "*Test call*" button, which only call the number that corresponds to the selected or underlined name and hangs up after one ringing.

Caution, if the user clicks over this button, he can't speak with the caller.

15.1.5 Refresh the PIN list

When a new user record his PIN code (with the phone), To view the new PIN code and its corresponding "*Caller list*" click on the "*Refresh PIN*" button.

15.1.6 Help

When a new user click "*Help*" button, "*Users guide*" document comes out.

16. Phone interface

In this case, the application must recognise what the user said. For example, when he says "*add*", the application must recognise this key word and execute the appropriate function. Each person has neither the same voice nor the same pronunciation. Then the application must use models which correspond with the user, to do speech recognition. For this reason the PIN code is used to create all file names. When the user 123 speak something, only the 123 model names are used to do the recognition.
All models are created as follows:

Pin code Model name Counter

123name15

The application always plays a message that prompt the user what he must do. After prompt and a beep, the user should speak his response directly. During answering a key word, a response (yes or no) or a name of caller, the user has three seconds.
All keywords and caller name, the application must record twice. With this method, the application has two models to do speech recognition and have more chances to recognise the latter pronunciation of user.

16.1 Call the card

Compose the card number, with any phone, to create or modify the database and use it. This number depends of the phone plug, which is on the wall. For example, the number (*027*) 721 77 97.

16.2 Dialog with the application (user at phone)

The application plays a welcome message at first. Then the user must enter his PIN code to create his table in the database and use the correct models to do the recognition. If the user is a new one, he must enter the "*000*" PIN code. Afterwards, the application asks him to input a valid PIN code, unused and different than "*000*". The application creates a new user table in the database with the given PIN code. Then the user must pronounce each key word and the two responses, "yes" and "no", to create the models. If the user input a invalid PIN code, the application will prompt for message to let user input PIN code again.

Afterwards, the new user has his own models like all other users. The application can ask him about his demand to use. Give one of the following key words: "*add*", "*delete*", "*list*" and "*menu*" or direct a name that he already has recorded to call it. Then, according to the key word, the application do the appropriate actions.

16.2.1 Unknown data

If the user pronounces an unknown word, including a silence, the application requests the user to repeat this current operation until it can recognize an awaited key, response or name.

16.2.2 Add to the directory

The user pronounces the key "*add*". He pronounces twice the new caller name that the application records and saves in the database. He also input the phone number with the digital phone keyboard. This number comes in a buffer and the application stops itself when this buffer is full. Then it saves the number in the database.

In the IDIAP[22], the buffer is composed of three numbers; this is given by the internal phone number format.

16.2.3 Delete from the directory

The user pronounces the key "*delete*". He gives the name that he wants delete. The application repeats the name, which is recognised, then ask the user if he is sure to delete this record, if the user confirms with "*yes*", deletes it. This action also deletes all associated sound files and models; If he answers "*no*", the application do nothing. Afterwards the application return in the awaiting key word state.

16.2.4 List the caller names directory

The user pronounces the key "*list*". The application plays all names that are contained in the user table and then goes back in the awaiting key word state.

16.2.5 Call someone

The user gives direct the name to call instead of a key word. This name is recognised by the application, which calls it and establishes the communication.

16.2.6 List key words

The user pronounces the key "*menu*". The application gives the key words that the user can say and goes back in the awaiting key word state.

16.2.7 Change unexpected models

If the user is not satisfied with his models (for example: the rate of recognition is low.), He can firstly delete his PIN code and record again. Anyway, If the user only wants to change *"add"* modes, he can enter the *"001"* PIN code from the beginning of Application, Afterwards, the application asks him to input a valid PIN code, Then the user must pronounce *"add"* twice to create the models.

[22] IDIAP: Institut Dalle molle d'Intelligence Artificielle Perceptive

In the same way, to change the following key words*: "delete", "list", "menu" , "yes" and "no"*, the user input *"002","003","004","005" and "006*" at the beginning of the Application, Then, with the new models, the application do appropriate actions as usual.

Attention: If the user input a invalid PIN code, the application will prompt user to input PIN code again.

16.3 Hang up the phone

When the user will end his call, he can simply hang up the phone. The application exits the running state. Anyway, He can call the card more time , which is ready for another incoming call.

17. Exit the application

Just click on the "*Cancel*" button or on the window cross, up right. This action shuts down TAPI and the application. If the Application continue not recognising user's pronunciation for 3 times, the Application will quit automatically.

It is preferable to close the application when no call is running because the wave device can stay opening. If the wave has already been open, when the user still play or record a sound, the device return an allocated error message and then he must restarts his computer to resolve this problem.