User Identification Tool

- Intermediate Report -

Contents

1. General Introduction (NEEDS TO BE EXTENDED)

The need for automatic speaker authentication / identification might have appeared from the desire of having more secure accesses to the information. The existing traditional methods (password, PIN, etc) are not totally secure and can be lost or forgotten.

Automatic speaker recognition is the use of a machine to recognize a person from a spoken phrase.

Speaker recognition systems are grouped into two categories that are generally called speaker verification and identification. Speaker verification is the determination from a voice sample if a person is who he or she claims to be (requires an identity claim). On the other hand speaker identification determines which one of a group of known voices best matches the input voice sample.

Another classification for voice recognition systems takes the text dependence properties of the system into account. Text dependent systems are those for which the speech used to train and test the system is constrained to the same word or phrase. On the other hand for text independent systems training (enrolling) and testing speech are completely unconstrained. As a third choice we may talk about vocabulary dependent systems that constrain the speech to come from a limited vocabulary.

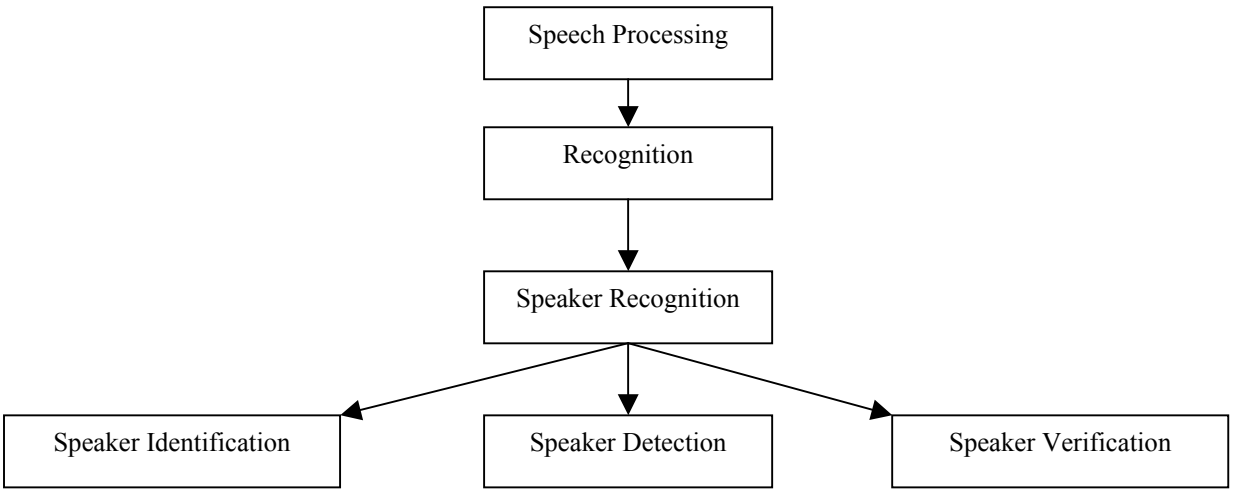

Figure 1. Speech processing (fragment from the picture in [*Campbell*])

Such systems find place for themselves in wide range of applications. For example:

- **Transaction authentication:** toll fraud prevention, telephone credit card purchases, telephone brokerage (e.g., stock trading), etc.
- **Access control:** physical facilities, computers, and data networks, etc.
- **Monitoring:** remote time and attendance logging, home parole verification, prison telephone usage, etc.
- **Information retrieval:** customer information for call centers, audio indexing (speech skimming device), etc.
- **Forensics:** voice sample matching, electronic commerce, telephone banking, friendly (customizable) electronic devices, etc.

We may refer to speaker verification as a three-state problem:

- First step is to extract speaker dependent features from spoken utterances (mel-frequency cepstrum coefficients, liner prediction cepstrum coefficients, log-cepstrum coefficients, etc.)
- Second stage is to find an algorithm that will successfully reflect the characterization of the chosen feature set. (Dynamic Time Warping, Hidden Markov Models, etc.)
- The third step which is the decision making step is where we compare he input voice to the (claimed) speaker model and make a decision about the identity of the speaker.

In such systems verification / identification errors can appear because (some) of the following reasons:

- **Natural constraints:** relatively small amount of information on speaker's identity in the speech signal (as compare to the message), features change in time (sensitivity to the train-test time difference), the human voice is highly sensitive to speaker's physiologic and psychological states, and to environment conditions (heat, coldness), bad pronunciation, extreme emotional states (e.g. anger), sickness / allergies / tiredness / thirst, aging, etc.
- **Impostors:** mimicry by humans, tape recorders and digital equipment for recording, editing and splicing sound, etc.
- **Environmental constraints:** channel transmission (telephone, internet), microphone transmission (different handsets, distances, angles), digitizer quality, channel mismatch (different channels for enrollment & verification request), environmental noise, poor room acoustics, etc.
- **System's and Application Constraints:** size of training database, duration of test utterance, time between training and testing, memory, etc.

The list above is not exhaustive.

The above error factors are generally outside the scope of the algorithms. They are usually corrected by better microphones, etc., and not by algorithms. However, these factors are important, because, no matter how good the algorithm is, human errors (misspeaking, misreading, etc.) limit the performance.

2. General Description of our Tool (NEEDS TO BE EXTENDED)

What we had in mind at the beginning of the project, as scenario, was a small robot that recognise (or not) a person immediately after speaking. We can imagine a small dialog:

Person: "Hello! How are you?"

Robot: "Hello, Monica! I am quite good today. Do you need some help?"

Or

Person: "Hello! How are you?"

Robot: "Hello. Can you please tell me your name? I do not know you."

What we did does not exactly fit the above scenario. Our application is a system for speaker identification, text dependent.

It has two components: the enrolling component and identification component. The general overview of our system is presented in Figure 2.

When adding a new user – enrolling -, the user should give in the name and repeat two times the password ("Hallo maschine!" – the text is chosen by the system). All the information is added in the user database XML file, and the .WAV and feature files are saved in the system.

When identifying, the user speaks into the microphone the chosen text ("Hallo maschine"). After extracting the necessary features, the signal is analysed by the verification module and the system makes the decision:

- the speaker is identified and he/she can ‘enter the system’ or ‘the robot says his/her name’, or
- the speaker is not identified

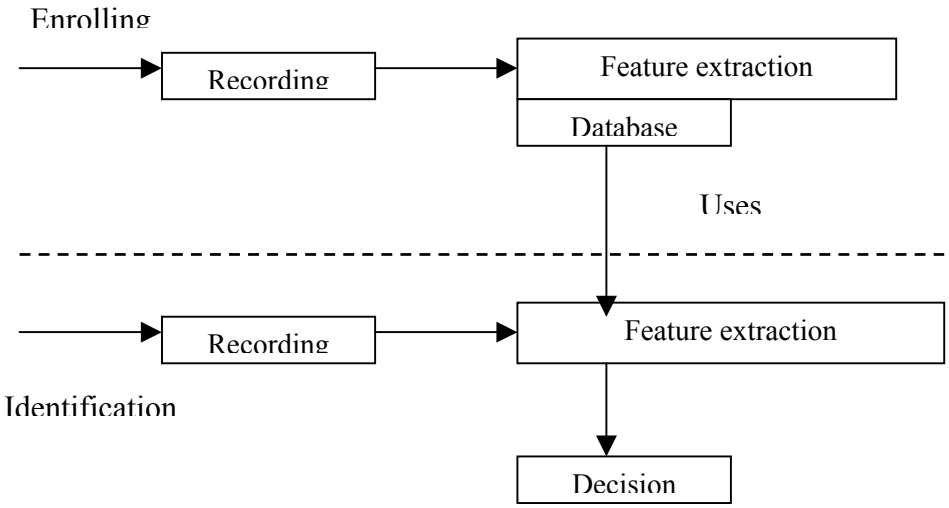

Figure 2. Overview of the system

The system is composed of three modules:

- the recording module,
- the feature extraction modules, and
- the analysing / verification module.

These three modules are presented in the following sections.

3. Recording Module

The recording module (as the whole application) has two components:

- Enrolling (Benutzer hinzufuegen) - it adds users to the XML database
- Identification (Benutzer identifizieren) - it records a .WAV file ("testuser.wav"). This is the file that should be compared with the ones in the database in order to decide if the user is identified or not.

The general structure of the recording module can be seen in Figure 3.

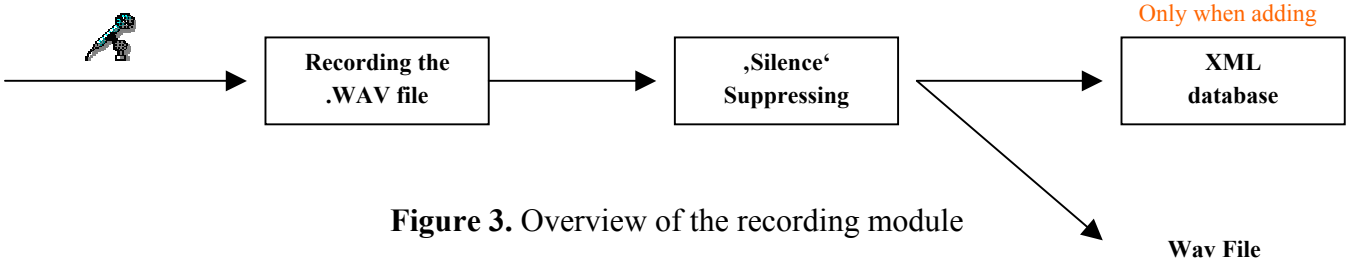

Figure 3. Overview of the recording module

The recording module does the following things:

- It records a .WAV file in a temporary file

- It removes the 'silence'[1] and saves the final .WAV file (The name of the final .WAV file depends on the component: enrolling or identification. If it is the identification component the name is "testuser.wav", else it has a special form that contains the user name. For details see section 3.2.)
- If it is the enrolling component, it adds the user in the XML user database
- It calls the other modules

For identifying there is taken one speech probe per user, but for enrolling there are taken two speech probes for each user and password. These two probes are needed by the analyzing / verification module.

All the .WAV files are kept in a special folder called "WavFiles". The user database is called "users.xml" and it is in the "Files" folder. The files from which the silence is not removed (what it is recorded at the beginning) are not saved by the application.

In order to record the .WAV files it is used the package javax.sound.sampled.* (Java Sound), for parsing the XML file it is used the DOM parser (org.w3c.dom.Document, org.w3c.dom.*), and for generating the graphical user interface (GUI) javax.swing.*.

"The Java Sound API is a low-level API for effecting and controlling input and output in audio media. It provides explicit control over the capabilities commonly required for audio input and output in a framework that promotes extensibility and flexibility." [*Java Sun*]

There is two different types of audio data that are supported the API: sampled audio data and Musical Instrument Digital Interface (MIDI) data. What the project uses, is the sampled audio data. This data can be thought of as a series of digital values that represent the amplitude or intensity of sound pressure waves. There are two Java packages (javax.sound.sampled, javax.sound.sampled.spi). They specify interfaces for capture, mixing and playback of digital audio.

The typically sampled audio data is captured in two steps:

- Usage of a microphone to convert the sound pressure waves to electrical voltages that mimic the waveform of the sound pressure wave.
- Usage of an analog-to-digital converter to measure the voltage at specific points in time and to convert that measurement to a digital value.

The audio format of the .WAV file that is recorded has the following characteristics:

- sample rate = 16000.0F (Other possible values: 8000,11025,22050,44100)
- sample size in Bits = 16 (Other possible value: 8)
- channels = 1 (Other possible value: 2)
- signed = true (Other possible value: false)
- big Endian = false (Other possible value: true)

Information about the XML parser and the GUI generation will be given in the following subsections.

[1] When removing 'silence', due to the algorithm, it is also removed part of the noise, and sometimes part of the recording. More details in section 3.1.

3.1. 'Silence' Suppressing

The 'silence' suppressing step is needed in order to correctly compare the files (needed by the DTW algorithm). In order to do this, we overtake the "SilenceSupressingAudioRecorder.java" file from the Tritonus project[2].

We modified the Tritonus silence suppression, by transforming the static silence detection into a dynamic one. The threshold, below which everything in the file is deleted, is calculated as the average of the frame averages. This way, the threshold is different for every file.

Before establishing this was of calculating the threshold, we did several experiments. Some of them are described in the following subsection.

3.1.1. Experiments

At the beginning the silence was removed by hand using Praat. This way the .WAV files were more or less clean and contain no errors / modifications. But we needed an automatic process that does this operation.

In order to establish the threshold for silence detection we did several experiments. Some of them are presented below.

At the beginning we tried to remove silence statically, by giving values to the threshold (as Tritonus does).

We tried to read the file as a byte array and get maximum value in absolute value, but it was always 128, no matter the file (tested on 10 files). We tried to get maximum, minimum and average of the frame averages, average of the file, sum of the elements, etc. In the table below can be seen some of the obtained results. – dynamical threshold.

We also tried to normalize the file and than remove the silence statically. The tried values for the parameters (normalizing parameter, silence removal parameter) were: (0.6, 300) and (1,500). In both cases not all .WAV files were good, but from listening and analyzing the files with Praat, the first combination is better.

	Frame			**Elements**	
File	**Max(avg)**	**Min(avg)**	**Avg(Avg)**	**Sum**	**Avg/File**
User 1a	3424	0	683	683051	10
User 1b	3420	0	583	1966894	2
User 1c	3633	54	529	911767	11
User 1d	2863	76	771	11561	0
User 2a	3294	0	439	-216997	-2
User 2b	3597	0	640	-253232	-2

Table 1. Some experimental results

[2] Tritonus project represents an independent implementation of Java Sound API. More information in [*Tritonus*].

In all of the experiments there were cases when silence / noise was not completely removed or when the input text was cut off. The smallest number of files with problems was when calculating the threshold as average of the frame averages (column 4 in Table 1). This is why we considered that the best way of removing 'silence' is to calculate the threshold with this formula.

Observation: In order to establish the application needed thresholds there were recorded 60 files from 30 people (two speech probes per person). All the files contain as text "Hallo maschine!".

3.2. XML User Database

The structure of the XML user database file was thought in such a way, so that it can be easily extended, if needed:

users (*user (name, +password (passfile, passfile)))

- the **user tag** is the tag that contains all information about a user. It has an attribute **idUser**, which represents the current number of the user in the database. This attribute is used for making distinction between users with the same name (in case this extension will be needed)
- the **name tag** represents the name of the user. It has no attributes.
- the **password tag** contains **passfile tags**. It has two attributes:
    - **textPass** contains the text of the .WAV file. It is needed for generating the text that the user has to say when identifying (in case the system needs this extension).
    - **threshold** contains a user-password-dependent value (float), that is used in the identification decision. The last application module calculates this threshold and it is updated every time a new user is added. More information about the threshold calculation in section 5.
- the **passfile tag** contains the name of the .WAV file. It has an attribute: **version**, which has values in the set {1,2}. This number is used for making distinction between the two samples needed for a password.

The name of the .WAV file is idUser+name+version+".wav" and the name of the feature file is idUser+name+version+".feat"

Example of an entry in the user database:

```
<users>
<user idUsr=„xx“>
<name>testName</name>
<password textPass=„Hallo maschine“ threshold=„x“>
<passfile version=„1“>xxtestName1.wav</passfile>
<passfile version=„2“>xxtestName2.wav </passfile>
</password>
</user>
```

</users>

The Java DOM parser parses the XML file and the whole structure is kept in a tree structure that can be easily managed. There is a very easy method to transform at the end the DOM tree again in an XML file. This means that the XML database file is parsed, and the structure is kept in a tree. All the needed operations (searching information, adding a new user, etc.) are done using this tree representation. At the end, the tree is transformed in XML file.

The XML database is totally platform independent (being a normal text file) and very flexible. In case we want to extend our system (e.g. we want more passwords for one user), it is very easy to do this, without changing too much the inner structure of the program. Some problems may appear (too much time needed for processing) in case the database contains many entries.

3.3. Graphical User Interface (GUI)

Figure 4. Snapshots of the beginning GUI

The beginning GUI is quite simple and very easy to use. If the application will be extended, then also the GUI should be modified. This GUI was done in order to be able to easily add users in the database and make the necessary tests.

When adding a new user, the program verifies the user name. The application does not accept two users with the same name. In case the new name is already in the database, an error message is shown to the user. Also, it requires two immediate speech probes from a new user.

As can be seen, the recording is not done automatically – no speech detector incorporated. In order to start recording, a button should be pressed. This is valid also for stopping a recording.

The GUI is done in Java Swing and snapshots can be seen in Figure 4.

The GUI that shows the decision of the identification will be described in the following sections.

3.4. Connection with Other Modules

The connection with the other modules is done using files (the .WAV files) and a vector of structures. A structure contains the name of the .WAV files and the existing threshold for a certain user and password. These structures are needed in order to update the thresholds. The .WAV files are needed for extracting the features and creating the feature files (.FEAT).

4. Feature Extraction Module (NEEDS TO BE DEVELOPED)

5. Analysing / Verification Module (NEEDS TO BE DEVELOPED)

6. Installation (NEEDS TO BE DEVELOPED)

7. Evaluation (NEEDS TO BE DEVELOPED)

What we achieved at the end of the project is a system for speaker identification, text dependent. It can be extended and used in identification of a user when security matters are involved (e.g. access to information), or (only after extension) as a 'mini'-starting point for customizable electronic devices (as the small robot from the scenario at the beginning).

The errors that can be made the speaker identification / authentication systems are of two types: the false acceptance of an invalid user and the false rejection of a valid user. False acceptance errors are the ultimate concern of a high-security application. However, they can be traded off for false rejection errors.

During the tests we did we observed the following:

- If there is 'some' background noise (we made a rhythmic sound as touching a hard surface near the recording area) the user is recognized. If the noises become stronger, or the microphone is touched, or there are background voices (near the recording area), the user is not recognized anymore
- If the user is chewing gum he/she is not recognized
- If the user has a different accent, different intonation, or his health state is not the same (having colds), he/she is not recognized
- If the position of the microphone is (quite) different, the user is not identified.

8. Limitations and Possible Extensions (NEEDS TO BE DEVELOPED)

Among the possible extensions for out tool are the following:

- Transforming the system into a authentication application: this thing can be easily done by permitting the user to make an identity claim into the interface,
- Adding new 'passwords' and letting the system to choose one randomly, when identifying / authenticate a user,
- Transforming the recording module, so that it detects automatically when a person is speaking (without pushing buttons)

References

[*Java Sun*] Java Sun API http://java.sun.com

[*Tritonus*] Tritonus homepage www.tritonus.org

[*Arslan*] **Levent Arslan, Serhat Görgün, Umut Naci**, "*Handset Normalization for Voice Authentication (Voicify)*", www.srdc.metu.edu.tr/webpage/projects/hermesProject/documents/VOICIFY.doc

[*Cohen*] **Arnon Cohen, Yaniv Zigel**, "*Feature Selection in Speaker Verification Systems*", www.haifa.il.ibm.com/Workshops/ Speech2003/papers/IBM_03.pdf

[*Campbell*] **Joseph P. Campbell**, "*Speaker Recognition: A Tutorial*", Proceedings of the IEEE, Vol. 85, No. 9, September 1997

[*Xafopoulos*] **Alexandros Xafopoulos**, "*Speaker Verification (an overview)*", sigwww.cs.tut.fi/TICSP/PRESENTATIONS/ 2001%20Fulltexts/SpeakerVerif.pdf

Annex A – Class Structure for the 'Aufnahme Tool'

The main class of the application is **Main.java** (public class **Main** extends javax.swing.JFrame). It builds the beginning GUI, and calls the classes (methods) for addin or identifying a user

java.lang.Object
└java.awt.Component
└java.awt.Container
└java.awt.Window
└java.awt.Frame
└javax.swing.JFrame
└**Main**

All Implemented Interfaces:
javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, javax.swing.RootPaneContainer, java.io.Serializable, javax.swing.WindowConstants

Package Important: Contains the main classes for adding a new user and identifying a user

Class Summary	
BenutzerHinzufuegen1	Main class for adding a new user to the system.
BenutzerIdentifizieren	Class for the identification process
Structure	Class needed for sending information from the XML user database file to the other modules.

Package SilenceDetectionUtil: Contains classes used in silence removal; the classes are from the Tritonus project

Class Summary	
ArraySet	
AudioSystemShadow	Experimental area for AudioSystem.
Encodings	This class is a proposal for generic handling of encoding.
StringHashedSet	A set where the elements are uniquely referenced by their string representation as given by the objects toString() method.

TAudioOutputStream	Base class for classes implementing AudioOutputStream.
TDebug	
TNonSeekableDataOutputStream	A TDataOutputStream that does not allow seeking.
TSeekableDataOutputStream	A TDataOutputStream that allows seeking.
WaveAudioOutputStream	AudioOutputStream for Wave files.
WaveTool	Common constants and methods for handling wave files.

The package contains also two interfaces:

1. **AudioOutputStream**: Represents a one-time writing of audio data to a destination (file or output stream)
2. **TDataOutputStream**: Interface for the file writing classes

Package utilities: Contains utilities needed by the DOM parser, for adding a new user and the silence removal class

Class Summary	
DOMUtilities	DOM utilities functions
HinzufuegenUtilities	It contains methods needed when adding a user; most of them are in the connection with DOM tree information search.
SilenceSupressingAudioRecorder	It is a modification of the Tritonus project SilenceSupressingAudioRecorder.java (www.tritonus.org) It uses the SilenceDetectionUtil package that contains Tritonus files.

Annex B – Project Information

The project was done by students of the Hamburg University, Computer Science Department, during the summer semester 2004.

Coordinators:
Prof. Wolfgang Menzel
Dirk Knoblauch

Participants:
David Dannberg,
Alex Grupe,
Monica Gavrila,
Dennis Götsch,
(Lorenz Knies),
Daniel Liem,
Martin Siepak,
Marina Svagusa