2009:TAPRO 02

SOFTWARE ENGINEERING

DEPARTMENT OF ECONOMICS AND IT

MASTER THESIS

Automatic Subtitle Generation for Sound in Videos

Author:

Boris Guenebaut

May 2009

Summary

The last ten years have been the witnesses of the emergence of any kind of video content. Moreover, the appearance of dedicated websites for this phenomenon has increased the importance the public gives to it. In the same time, certain individuals are deaf and occasionally cannot understand the meanings of such videos because there is not any text transcription available. Therefore, it is necessary to find solutions for the purpose of making these media artefacts accessible for most people. Several software propose utilities to create subtitles for videos but all require an extensive participation of the user. Thence, a more automated concept is envisaged. This thesis report indicates a way to generate subtitles following standards by using speech recognition. Three parts are distinguished. The first one consists in separating audio from video and converting the audio in suitable format if necessary. The second phase proceeds to the recognition of speech contained in the audio. The ultimate stage generates a subtitle file from the recognition results of the previous step. Directions of implementation have been proposed for the three distinct modules. The experiment results have not done enough satisfaction and adjustments have to be realized for further work. Decoding parallelization, use of well trained models, and punctuation insertion are some of the improvements to be done.

Author:	Boris Guenebaut		
Examiner:	Steven Kirk		
Advisor:	Jonas Amoson		
Programme:	Software Engineering, Master of Science		
Subject:	Software Engineering	**Level**:	Master
Date:	May 22, 2009	**Report Number**:	2009:TAPRO 02
Keywords	Audio Extraction, Java Media Framework, Speech Recognition, Acoustic Model, Language Model, Subtitle Generation, Sphinx-4		
Publisher:	University West, Department of Economics and IT S-461 86 Trollhättan, SWEDEN		

"If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization."

One of Murphy's Laws of Technology

Contents

AUTOMATIC SUBTITLE GENERATION FOR SOUND IN VIDEOS

Boris Guenebaut

Software Engineering
Institution of Economics and IT
Högskolan Väst, 461 86 Trollhättan
Contact: boris.guenebaut@student.hv.se

ABSTRACT

The last ten years have been the witnesses of the emergence of any kind of video content. Moreover, the appearance of dedicated websites for this phenomenon has increased the importance the public gives to it. In the same time, certain individuals are deaf and occasionally cannot understand the meanings of such videos because there is not any text transcription available. Therefore, it is necessary to find solutions for the purpose of making these media artefacts accessible for most people. Several software propose utilities to create subtitles for videos but all require an extensive participation of the user. Thence, a more automated concept is envisaged. This thesis report indicates a way to generate subtitles following standards by using speech recognition. Three parts are distinguished. The first one consists in separating audio from video and converting the audio in suitable format if necessary. The second phase proceeds to the recognition of speech contained in the audio. The ultimate stage generates a subtitle file from the recognition results of the previous step. Directions of implementation have been proposed for the three distinct modules. The experiment results have not done enough satisfaction and adjustments have to be realized for further work. Decoding parallelization, use of well trained models, and punctuation insertion are some of the improvements to be done.

Key words: Audio Extraction, Java Media Framework, Speech Recognition, Acoustic Model, Language Model, Subtitle Generation, Sphinx-4.

1. INTRODUCTION

Video has become one of the most popular multimedia artefacts used on PCs and the Internet. In a majority of cases within a video, the sound holds an important place. From this statement, it appears essential to make the understanding of a sound video available for people with auditory problems as well as for people with gaps in the spoken language. The most natural way lies in the use of subtitles. However, manual subtitle creation is a long and boring activity and requires the presence of the user. Consequently, the study of automatic subtitle generation appears to be a valid subject of research. This master thesis attempts to answer the following problematic:

In which way could video sound be extracted and analysed in order to produce corresponding accurate text formatted according to subtitle standards?

Nowadays, it exists many software dealing with subtitle generation. Some proceed on copyright DVDs by extracting the original subtitle track and converting it in a format recognized by media players, for example *ImTOO DVD Subtitle Ripper* [1], and *Xilisoft DVD Subtitle Ripper* [2]. Others allow the user to watch the video and to insert subtitles using the timeline of the video, e.g. *Subtitle Editor* [3], and *Subtitle Workshop* [4]. It can also be found subtitle editors providing facilities to handle subtitle formats and ease changes, for instance *Jubler* [5], and *Gaupol* [6]. Nonetheless, software generating subtitles without intervention of an individual using speech recognition have not been developed. Therefore, it seems necessary to start investigations on this concept.

A section dedicated to literature review will present earlier published work related to the thesis topic. It will contain meaningful material in order to produce a suitable experimental program. In other words, the literature review is going to deepen our knowledge of media

video and audio, voice recognition and time synchronization. Besides, it will provide resources for the purpose of choosing the most convenient programming language to be used to develop the experimental program as well as samples to understand Application programming interface (API) features.

1.1. Project Description

The current thesis work principally tends to answer our problematic by presenting a potential system. Three distinct modules have been defined, namely audio extraction, speech recognition, and subtitle generation (with time synchronization). The system should take a video file as input and generate a subtitle file (sub/srt/sst/son/txt) as output. The figure 1 below gives an overview of the experimental system named AutoSubGen.

Audio Extraction The audio extraction routine is expected to return a suitable audio format that can be used by the speech recognition module as pertinent material. It must handle a defined list of video and audio formats. It has to verify the file given in input so that it can evaluate the extraction feasibility. The audio track has to be returned in the most reliable format.

Speech Recognition The speech recognition routine is the key part of the system. Indeed, it affects directly performance and results evaluation. First, it must get the type (film, music, information, home-made, etc...) of the input file as often as possible. Then, if the type is provided, an appropriate processing method is chosen. Otherwise, the routine uses a default configuration. It must be able to recognize silences so that text delimitations can be established.

Subtitle Generation The subtitle generation routine aims to create and write in a file in order to add multiple chunks of text corresponding to utterances limited by silences and their respective start and end times. Time synchronization considerations are of main importance.

1.2. Report Overview

Our investigation of the problem follows a systematic sequence. We first study related works and reusable material. Then, we describe the system shaping and its implementation. The final phase consists in realizing experiments, discussing the results and concluding the thesis. The sections include:

Section 2. Background Study:
Deals with the literature review and provides background material to understand the specificities of such a system.

Section 3. Experimental System:
Describes the analysis, design and implementation of the experimental program.

Section 4. Experiments and Results:
Explains the different tests and measures realized on the developed apparatus and shows the results.

Section 5. Discussion:
Discusses the efficiency and reliability of such a system.

Section 6. Conclusion:
Gives the conclusion and the further work.

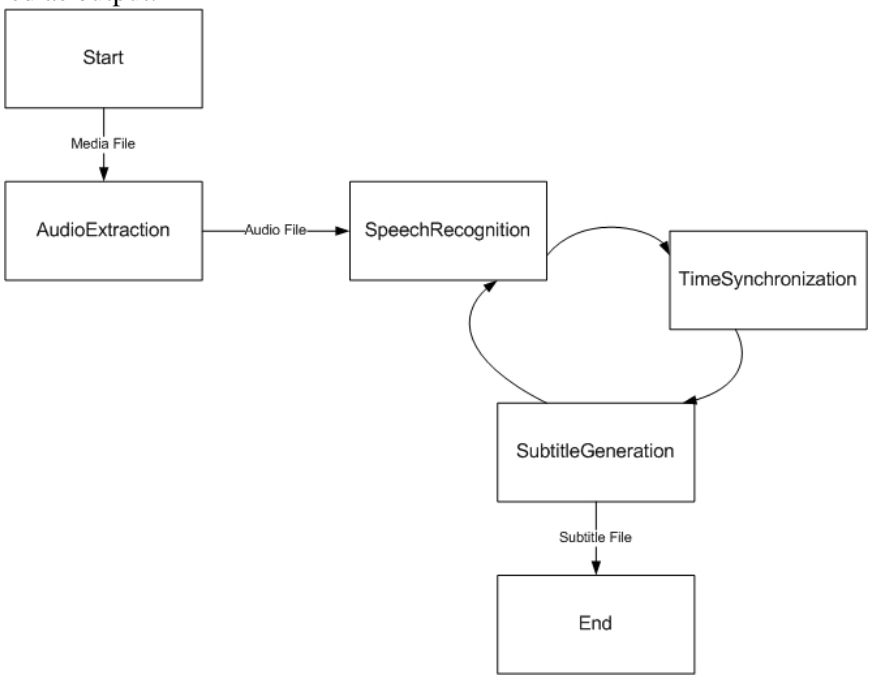
2. BACKGROUND STUDY

2.1. Suitable Programming languages Discussion

There were several programming languages that could have been used in the creation of AutoSubGen. A quick overview on the Internet indicated to focus on C/C++ and Java. We now analyze the positive points of the two aforementioned languages based on the study of J.P. Lewis and Ulrich Neumann [7]. On the one hand, C++ provides remarkable characteristics towards speed, cross-systems capabilities, and well-tested packages. On the second hand, Java offers an intuitive syntax, portability on multiple OS and reliable libraries. They both are used in the Sphinx speech recognition engine [8]. Java proposes the Java Media Framework (JMF) API allowing developers to deal with media tracks (audio and video) in an efficient way. In the article of Lewis, it appears Java performances are relatively similar to C/C++ ones independently of the hardware. Besides, Java performances should be even higher in theory. The absence of pointers, the use of efficient garbage collector and the run-time compilation play a key role in Java and their enhancement should soon permit to go beyond C++ performances.

Consequently, Java seemed to be the most suitable language and was elected to realize AutoSubGen. The JMF API features (described in details later on, cf. 2.2.2) facilitate audio extraction. An implementation of the Sphinx Speech Recognition Engine has been done in

Fig. 1. Breakdown structure of the AutoSubGen experimental system. A media file (either video or directly audio) is given as input. The audio track is extracted and then read chunk by chunk until the end of track is reached. Within this loop happen successively three tasks: speech recognition, time synchronization, and subtitle generation. Finally, a subtitle file is returned as output.



Java, Sphinx-4 [9] and is the core of the speech recognition part. The subtitle generation uses Java file capabilities.

2.2. Audio Extraction

2.2.1. Basics

A media movie is generally composed of a video track and an audio track. During the montage, the tracks are gathered together and the final artefact is a single file. In this section, we study the feasibility to isolate the audio track from the rest of the file in order to solely process sound. A google research on the Internet for 'extract audio from movie' provides multiple links to software but few tutorial or tip to implement it on his own, especially in Java for this thesis concern. We oriented the work on the JMF API [10]. This API provides many interesting features for dealing with media objects. The next subsection describes the different aspects of it.

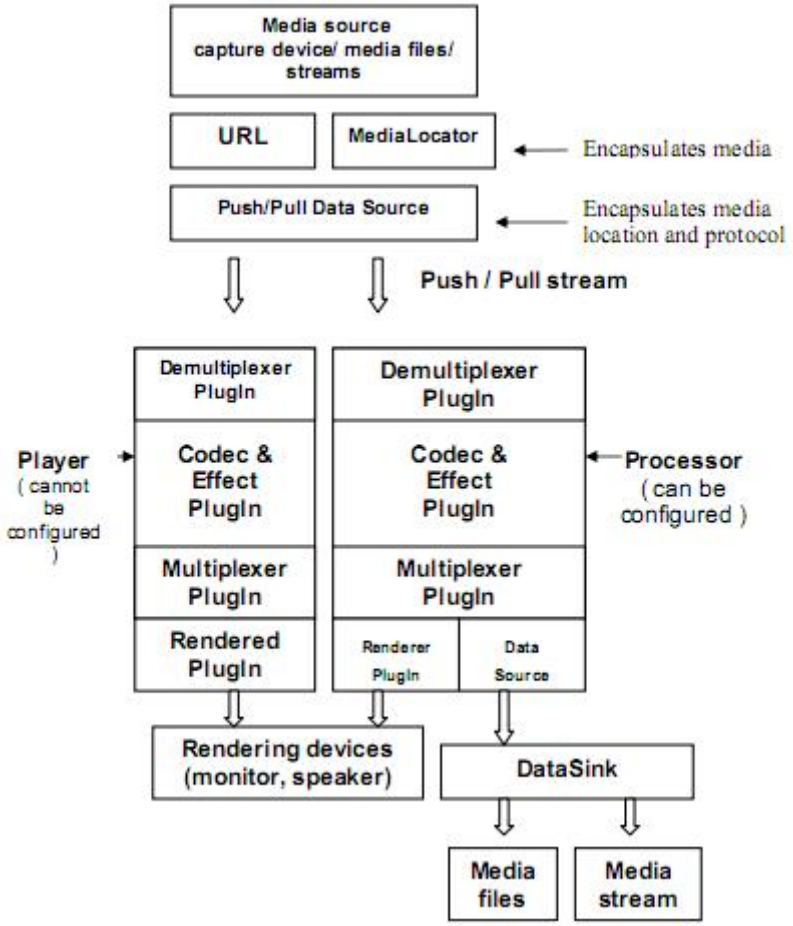
2.2.2. Java Media Framework

We present an overview of the JMF API [10] and focus a bit deeper on insightful functionalities related to audio extraction. It exists an interesting tutorial [11] to start with the JMF and it is thus used in the following parts. The official documentations are also pertinent resources [12] and [13]. Furthermore, a complete book describing JMF API [14] was published in 1998.

2.2.2.1. Features

JMF gives applications the possibility to present and playback multimedia content, capture audio and video through respectively microphone and camera, do real-time streaming of media over the Internet, process media (like changing media format, adding effects, etc...) and store media into file. The most popular media formats are enabled: JPEG, MPEG-1, MPEG-2, QuickTime, AVI, WAV, MP3, GSM, G723, H263, and MIDI. It also offers a support for media access protocols such as file, HTTP, HTTPS, FTP, RTP, and RTSP. JMF provides a platform-neutral framework for dealing with multimedia since Sun's implementation of JMF has a version for each particular platform so that benefits can be retrieved from each one. JMF offers material to handle time and its progression. The Figure 2 summarizes the several aspects of multimedia treatment handled by JMF.

Fig. 2. JMF Architecture. Shows the several parts JMF surrounds. It enables management of media-related objects such as DataSource, Player, Processor, DataSink. Extracted from [11].



2.2.2.2. Utilization

JMF is available in several versions depending on the target platform. There are currently four versions: JMF 2.1.1 FCS Cross-Platform (implementation of JMF with only Java byte codes and without any native codes), JMF 2.1.1 FCS with the Windows Performance Pack (optimized for Windows platform), JMF 2.1.1 FCS with the Solaris Sparc Performance pack, and JMF2.1.1 FCS for Linux from Blackdown. If a platform-specific version has been installed, the JMF directory contains three subdirectories: doc (containing only a readme file), bin (containing the applications of the JMF - cf. [11] for deeper information), and lib (the one we have interest for - containing the necessary libraries to create multimedia handling applications).

2.2.2.3. Focus on Helpful Functionalities

The JMF documentation provides various examples demonstrating use of JMF features. Luckily, a pertinent example related to audio extraction is included. Indeed, the instance called "*Splitting tracks from an input*" [15] presents a way to separate different tracks (audio and video) contained in a media file given as input. The number of output files of individual elementary track is equal to the number of multiplexed tracks composing the input media file. Thereby, it is an insightful base to develop the audio extraction module. We now describe the key parts of the aforementioned sample code.

The program only runs in command line (see the source page for more details). The file *Split.java* contains the class **Split** which itself contains four inner classes: **StateWaiter**, **SplitDataSource**, **SplitStream**, and **FileWriter**, all used as auxiliary classes to reach the aim of track splitting. The **main** method first retrieves the different

arguments given to the program. If unexpected inputs are provided, the program displays an help and exits. Then, it tries to locate the input file and and if successful, launches the split process. This phase consists in verifying the video and audio formats and accordingly applying some modifications, defining the number of different elementary streams and instantiating a **SplitDataSource** object for each one, and finally writing each track in specific file with the extensions given in input.

2.2.3. Alternative

A preliminary review of Sphinx-4 showed up that a restricted number of audio files can be supported so far and thus the audio extractor presented above could be of limited usage. Fortunately, an open source software dealing with media content called *VLC media player* [16] supplies media format conversion functionalities and is therefore able to convert most formats into the only Sphinx-4 compatible audio format *WAV*. This option could be retained in the case the Java program described previously cannot complete the task well.

2.3. Speech Recognition

2.3.1. Basics

A high level overview of how speech recognition works is provided here [17]. The article "*A Speech Recognition and Synthesis Tool*" [18] offers a clear and concise overview of this field while essential methods used in this process are well exposed in "*Statistical Methods for Speech Recognition*" [19]. This technology permits a computer to handle sound input through either a microphone or an audio file in order to be transcribed or used to interact with the machine. A speech recognition system can be designed to handle either a unique speaker or an infinite number of speakers. The first case, called speaker-dependent model, presents an accuracy rate greater than 90% with peaks reaching 98% under optimal conditions (quiet room, high quality microphone, etc...). Modern speech recognitions engines are generally based on the Hidden Markov Models (HMMs). Ghahramani [20] provides a thorough introduction to HMMs. ElAarag and Schindler [18] explains HMM as "*a statistical model which attempts to determine the hidden components by using the known parameters*". We can also pick up their example to ease the understanding: "*if a person stated that he wore a raincoat yesterday, then one would predict that it must have been raining. Using this technique, a speech recognition system may determine the probability of a sequence of acoustic data given one word (or word sequence)*". The most likely word sequence can be elucidated according to Bayes' theorem:

$$P(word|acoustics) = \frac{P(acoustics|word)P(word)}{P(acoustics)} \tag{1}$$

with P = Probability

Following this theorem, it is observed P(acoustics) is a constant for any given sequence of acoustic data and thereby, can be ignored. P(word) represents the probability of the word related to a specific language model. Further readings include the work of Ghahramani [20] and the Wikipedia article about Bayes' theorem [21] proposing clear examples.

Even if word recognition has reached a very satisfying rate of 80-90%, grammar still needs multiple enhancements. Punctuation determination involves to distinguish stressed syllables or words in an utterance. In a natural conversation, it is straightforward to define if the speaker affirms ('.'), exclaims ('!') or interrogates ('?'). Nonetheless, it is much more difficult for speech recognition systems to make the difference and they usually only recognize which word was pronounced. Therefore, stresses and intonations are rarely detected and punctuation remains a manual process.

Acoustic Model We now introduce the concept of acoustic model. It is a file containing a statistical representation of the distinct sounds that make up each word in the language model. We differentiate two acoustic models:

- **Speaker Dependent Acoustic Model**: it is an acoustic model that has been designed to handle a specific individual's speech. This kind of model is generally trained using audio from the concerned person.

- **Speaker Independent Acoustic Model**: it is supposed to recognize speech from different people, especially the ones who did not participate to the training of the acoustic model. It is obvious a speaker independent acoustic model requires much more speech audio training to provide correct results.

Language Model or Grammar A language model groups a very broad list of words and their probability of occurrence in a given sequence. In a grammar, a list of phonemes is associated to every word. The phonemes correspond to the distinct sounds forming a word.

Decoder The following definition for '*Decoder*' comes from the VoxForge website [22]: "*Software program that takes the sounds spoken by a user and searches the Acoustic Model for the equivalent sounds. When a match is made, the Decoder determines the phoneme corresponding to the sound. It keeps track of the matching phonemes until it reaches a pause in the users speech. It then searches the Language Model or Grammar file for the equivalent series of phonemes. If a match is made it returns the text of the corresponding word or phrase to the calling program*".

2.3.2. Sphinx

We present the different tools provided by the CMU Sphinx Group with focus on speech recognition since it is the core part of the AutoSubGen system. The following explanations are based on the official website [8] and documentations and tutorials provided by the group, cf. [23], [9].

The Sphinx Group currently provides four decoders: PocketSphinx (fastest version but less accurate than Sphinx-3 or Sphinx-4 - useful for live applications), Sphinx-2 (predecessor of PocketSphinx - still largely used in interactive applications), Sphinx-3 ("*state-of-the-art large vocabulary speech recognition system*" from [8] - the most accurate version), and Sphinx-4 ("*state-of-the-art speech recognition system written entirely in the Java(tm) programming language*" from [8] - capabilities close to the Sphinx-3). A comparison using regression tests between Sphinx-3 and Sphinx-4 has been realized by the CMU group to evaluate the potential of each version:

> "*We have some regression tests comparing sphinx4 to s3 (flat decoder) and to s3.3 (fast decoder) in several different tasks, ranging from digits only to medium-large vocab. s3 (flat decoder) is often the most accurate, but Sphinx4 is faster and more accurate than sphinx3.3 in some of these tests.*"

These results confirmed our will to use Sphinx-4 in order to provide a suitable system entirely in the Java environment.

In addition to the decoders, the CMU Sphinx project makes auxiliary tools available. Indeed, an acoustic model training tool is provided, SphinxTrain, as well as two language model trainers, namely cmuclmtk and SimpleLM.

In order to use the Sphinx tools in an optimal way, several software are needed: Perl to run SphinxTrain scripts, C Compiler (Visual C++ can be used) to compile Sphinx sources, the Java platform and Apache Ant for using Sphinx-4. Besides, a word alignment program is advised for the purpose of evaluating the accuracy of the decoder. The latest versions have been used for this work and have not presented any trouble when running Sphinx software. It is recommended to keep those necessary programs up-to-date.

2.3.3. Sphinx-4

2.3.3.1. In a Nutshell

Sphinx-4 is an open source project led by Carnegie Mellon University, Sun Microsystems Inc. and Mitsubishi Electric Research Laboratories. A white paper by Walker et al. [24] presents an overview of the framework. As previously noted, it is completely written in Java. It offers a highly modularized and flexible architecture as well as versatile APIs, supports any acoustic model structure, handles most types of language models, provides new algorithms in order to get word level hypotheses, and accepts multimodal inputs.

2.3.3.2. Architecture

This paragraph explains the main aspects of the Sphinx-4 architecture. An overall representation is depicted in Figure 3. We distinguish three principal modules: the **FrontEnd**, the **Decoder** and the **Linguist** getting material from the **Knowledge Base**. The FrontEnd gets a single or several input signals and computes them so that a sequence of **Features** is created. The Linguist generates a **SearchGraph** by translating any kind of standard language model, with the aid of pronunciation information contained in a **Lexicon**, also called Dictionary and structural information stored in sets of **AcousticModel**. The **SearchManager** component located in the Decoder uses the aforementioned Features as well as the SearchGraph in order to realize the decoding. **Results** are produced. **Controls** might be emitted to any enabled component and become an effective partner in the process.

A **ConfigurationManager** eases the configuration of a large number of parameters acting on the system performance tuning. The particularity of Sphinx-4 lies on the fact that it can dynamically load and configure modules at run time, making the system flexible and pluggable. Multiple **Tools** are included in the framework and permit to monitor decoder statistics (i.e. word error rate, runtime speed, memory usage, etc...). Moreover, a set of **Utilities** is enclosed within Sphinx-4. Utilities support application-level processing of recognition results and for instance can retrieve result lattices, confidence scores and natural language understanding.

2.3.3.3. Auxiliary Elements

The tuning of Sphinx-4 is truly flexible and allows developers to manage most of their Automatic Speech Recognition (ASR) system from the configuration file. Thanks to that, the Java code required to make the system run is rather brief. Useful guidelines are provided on the official website [26] and give precious tips to setup Sphinx-4. The instrumentation also plays a crucial role in monitoring the decoding process. Indeed, it offers the possibility to display warning and error messages, logging and tracing messages, recognition results, accuracy results, speed statistics, memory footprint statistics, and configuration information. Detailed information is proposed on the Sphinx-4 website [27].

2.4. Use of Subtitles and Standards

We mentioned earlier different types of subtitles (cf. Part 1.1). After an overall research on the Internet, it appears the SubRip (SRT) file format is currently the most used. We therefore focus uniquely on that format for this thesis work. The following code is an example of SRT file:

```
1 1
2 00:03:25,785 --> 00:03:29,356
3 Here is a dialog text...
4
5 2
6 ...
7 ...
8
9 3
```

Line 1 corresponds to the number of the subtitle that follows (starting with 1). Line 2 represents the start timecode followed by the string "–>" and then the end timecode. Timecodes are in the format HH:MM:SS,MIL (hours, minutes, seconds, milliseconds). Line 3 and consecutive lines contain the text of the subtitle. Two successive subtitles are separated from each other by a blank line.

3. EXPERIMENTAL SYSTEM

This section presents the experimental system to be setup. First, the requirements as well as the analysis and design of such a system are presented. The second section discusses the implementation phase in which are described the use of existing software and the creation of customized modules.

3.1. Analysis and Design

3.1.1. System Overview

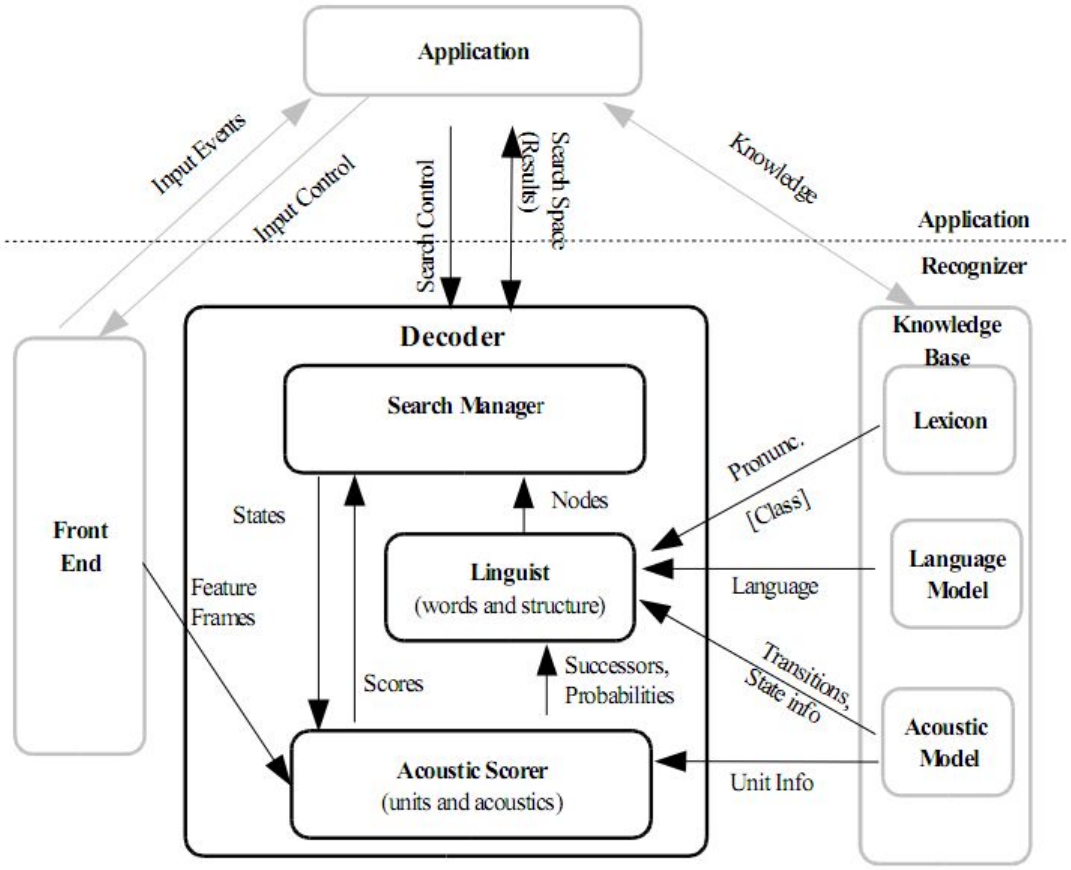
We presented the basic requirements of our system in section 1. We now propose a deeper analysis in order to clearly evaluate the real needs. The programming language comparison in the previous section gave us the desire to mainly use the Java environment to implement the system. We also discovered Java components being usable for audio extraction and speech recognition. A smart adaption of those artefacts is essential to offer a suitable system. The subtitle generation will use the different symbols and tags produced by the speech recognizer in order to generate SRT files.

3.1.2. Audio Extraction

This module aims to output an audio file from a media file. An activity diagram of the module is presented in Figure 4. It takes as input a file URL and optionally the wished audio format of the output. Next, it checks the file content and creates a list of separated tracks composing the initial media file. An exception is thrown if the file content is irrelevant. Once the list is done, the processor analyzes each track, selects the first audio track found and discards the rest (note that an exception is thrown if there is more than one audio track). Finally, the audio track is written in a file applying the default or wished (if mentioned) format.

Audio Formats Discussion It was previously talked about the audio formats accepted by Sphinx. At the moment, it principally supports WAV or RAW files. It is therefore a complicated task to obtain directly the right format. It often requires various treatments and conversions to reach the aforementioned purpose. The module described previously thus provides a way to extract audio from a media file but does not offer conversion tools. Otherwise, it is envisaged to use the facilities of VLC

Fig. 3. Sphinx-4 Architecture. There are three main blocks: the FrontEnd, the Decoder, and the Knowledge Base. From Sphinx-4 documentation [25].



(cf. previous section 2.2.3) for the purpose of getting suitable audio material. An explanation is proposed in the implementation section.

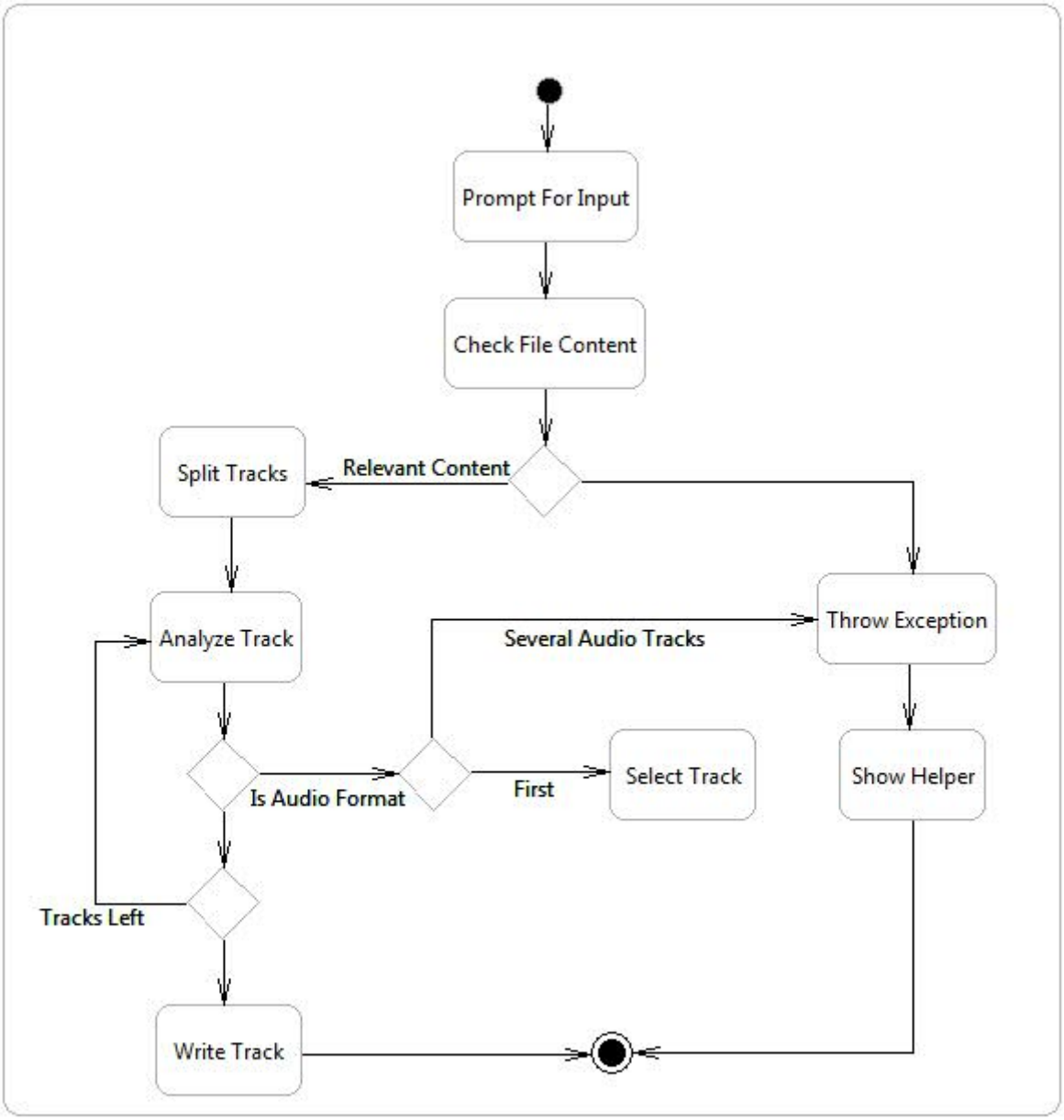
3.1.3. Speech Recognition

According to the limited duration of this thesis work and because of the complexity of the task, it would not have been feasible to design and realize a new ASR module. In section 2, we talked about the different Sphinx systems. They have been developed for nearly two decades and have proved their reliability within several ASR systems where they were integrated in. It is therefore natural we opted for the Sphinx-4 decoder. As it was introduced in the background study, Sphinx-4 has been entirely written in Java, making it totally portable, and provides a modular architecture, allowing to modify configurations with ease. The architecture overview was shown in the background section (cf. Figure 3). We aim to derive advantage of Sphinx-4 features in order to satisfy the needs of the AutoSubGen speech recognition module. The latter is expected to select the most suitable models with regards to the audio and the parameters (category, length, etc...) given in input and thereby generate an accurate transcript of the audio speech. The Figure 5 resumes the organization of the speech recognition module.

3.1.4. Subtitle Generation

The module is expected to get a list of words and their respective speech time from the speech recognition module and then to produce a SRT subtitle file. To do so, the module must look at the list of words and use silence (SIL) utterances as delimitation for two consecutive sentences. The Figure 6 offers a visual representation of the

Fig. 4. Activity Diagram for Audio Extraction. Describes the successive steps of the audio extraction module in order to obtain an audio file from a media file given in input.



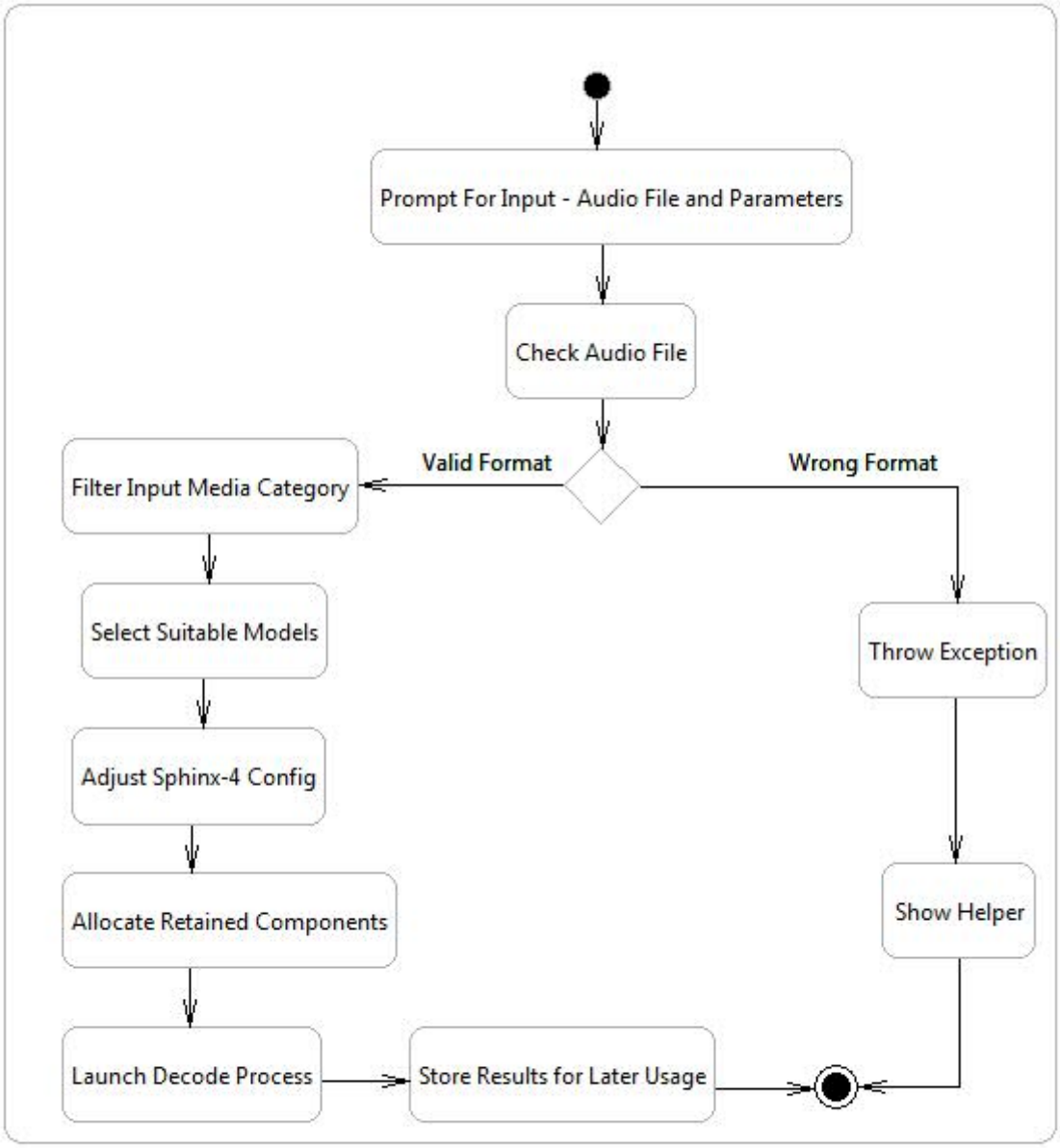
steps to follow. However, we face up to some limitations. Indeed, it will not be able to define punctuation in our system since it involves much more speech analysis and deeper design. Orientations about this problem will be discussed in the conclusion.

3.2. Implementation

3.2.1. General Settings of the System

The first implementation of the system has been realized on a personal machine of which the characteristics are described in Table 1. The current section provides

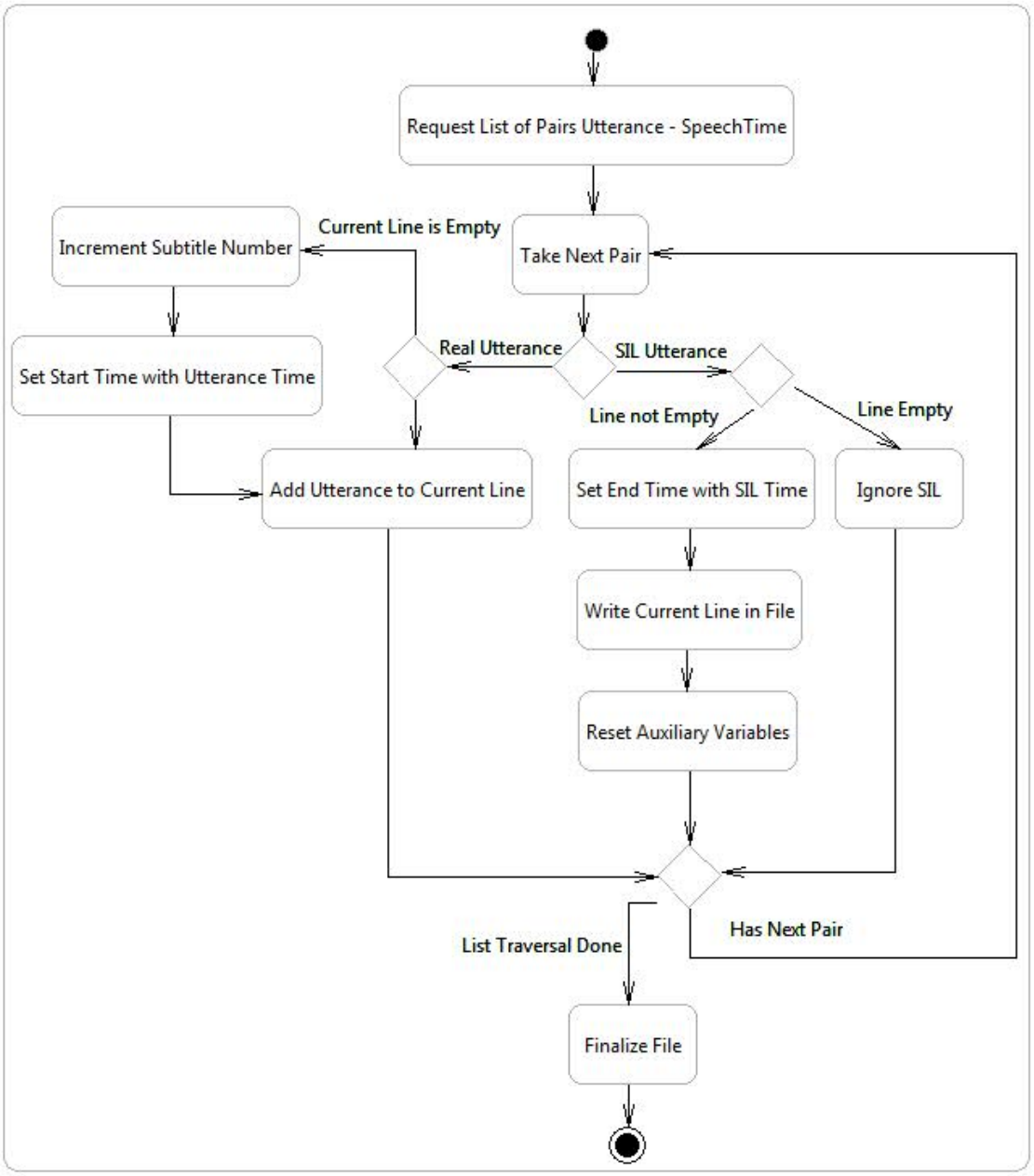
Fig. 5. Activity Diagram for Speech Recognition. Shows the successive statements to be executed at the time of the speech recognition process. An audio file and some parameters are given as arguments to the module. First, the audio file is checked: if its format is valid, the process continues; otherwise, an exception is thrown and the execution ends. According to the category (potentially amateur, movie, news, serie, music) given as argument, related acoustic and language models are selected. Some adjustements are realized in the Sphinx-4 configuration based on the set parameters. Then, all the components used in the ASR process are allocated required resources. Finally, the decoding phase takes place and results are periodically saved to be reused later.



some tips about the use of external software necessary to make the AutoSubGen system run.

Apache Ant A complete manual [28] and a wiki [29] are provided by the Apache Ant project. An instruc-

Fig. 6. Activity Diagram for Subtitle Generation. Exhibits the principal statements of the subtitle generation module. First, it receives a list of pairs Utterance-SpeechTime. Then, it traverses the list till the end. In each iteration, the current utterance is checked. If it is a real utterance, we verify if the current line is empty. If so, the subtitle number is incremented and the start time of the current line is set to the utterance speech time. Then, the utterance is added to the current line. In the case, it is a SIL utterance, we check if the current line is empty: if not, the end time of the current line is set to the SIL speech time. If the line is empty, we ignore the SIL utterance. Once the list has been traversed, the file is finalized and released to the user.



tive book [30] about the relation between Java and Ant is also available. First of all, it is required to download one of the binary distributions available here [31]. The next step consists in extracting the root directory and copy it at the target location. Then, the *ANT_HOME* environment variable should be set to the directory where Ant is installed ; the **bin** directory must be added to the user's path ; and the *JAVA_HOME* environment variable should

Table 1. Presents the main characteristics of the machine used to run the AutoSubGen system.

OS Name	Microsoft Windows Vista Business
System Manufacturer	Notebook
System Type	X86-based PC
Processor	Intel(R) Core(TM)2 Duo CPU T7250 @ 2.00GHz
RAM	3,00 GB

be if necessary set to the directory where the JDK is installed as shown below for Windows:

```
set ANT_HOME=C:\\apache-ant-1.7.1
path = %PATH%;%ANT_HOME%\bin
set JAVA_HOME=C:\\Program Files\Java\jdk1.6.0_13
```

Afterwards, it is interesting to execute a simple test to check if Ant has been successfully installed:

```
C:\>ant -version
Apache Ant version 1.7.1 compiled on June 27 2008
```

3.2.2. *Audio Extraction*

We expressed a certain satisfaction concerning the sample code provided in the JMF documentation (cf. part 2.2.2.3 - since it permits to retrieve audio tracks). The next task was to figure out how to convert the output audio file into a format recognized by Sphinx-4. Despite the fact we followed guidelines to do so in Java [32], we did not obtain the expected result. Consequently, we orientated this part of the system on the use of VLC media player introduced in the audio extraction alternative 2.2.3. The trick thus consists in successive straightforward steps:

1. Install VLC media player.
2. Launch VLC and go in Media -> Convert / Save...
3. Select media file - several files of a same directory can be selected at a time.
4. In Encapsulation check WAV.
5. In Audio codec check Audio, choose WAV in Codec, set the Bitrate at 256 kb/s and set Channels at 1.
6. Define output file path and click on Save button.
7. The resulting file is quite heavy but can be perfectly used in Sphinx-4.

3.2.3. *Speech Recognition*

3.2.3.1. *Sphinx-4*

One of the advantages of Sphinx-4 lies in the fact it is entirely implemented in Java and thus, it only needs a Java environment and Apache Ant (cf. 3.2.1). Based on the official manual of Sphinx-4 [9], we explain now the different phases we went through in order to obtain a fully operational speech recognition system on a Windows Operating System (OS) (cf. Table 1). We first download the **sphinx4** archive containing the source code and auxiliary tools and extract it. We obtain a root directory called **sphinx4**. We then configure the environment to support the Java Speech API (JSAPI):

1. Go to the lib directory.
2. Type .\jsapi.exe and view the BCL.
3. If the BCL is acceptable, accept it by typing "y". The jsapi.jar file will be unpacked and deposited into the lib directory.

Henceforth, we can build Sphinx-4 - we repeat the three first lines if it is a new command prompt:

```
set ANT_HOME=C:\\apache-ant-1.7.1
path = %PATH%;%ANT_HOME%\bin
set JAVA_HOME=C:\\Program Files\Java\jdk1.6.0_13
...\sphinx4>ant
```

It is useful to remember the command permitting to delete all the output from the build to give a fresh start:

```
...\sphinx4>ant clean
```

The ultimate step in this part is the Javadocs generation. We choose to generate documentation for classes of any access restriction:

```
...\sphinx4>ant -Daccess=private javadoc
```

3.2.3.2. *Language Model Creation*

This task requires the installation of CMU-Cambridge Statistical Language Modeling toolkit [33]. As the implementation is done on a Windows machine, we install the software on Cygwin [34]. We describe briefly the successive steps:

1. Install Cygwin and CMU-Cam_Toolkit_v2.tar.gz.
2. Extract CMU-Cam_Toolkit_v2.tar.gz in C: \cygwin\home\boris.
3. Launch Cygwin.bat and move to CMU-Cam_Toolkit_v2/src.
4. Execute the command *make install*. The executable files are generated in the **bin** directory.
5. Copy all the executable files contained in **bin** to the **bin** directory of Cygwin.

We now present the phases to generate a language model from a text file containing sentences given a text file called *sentences.text* located in the current directory:

1. Compute the word unigram counts:

```
...>cat sentences.txt | text2wfreq >
    sentences.wfreq
```

2. Convert the word unigram counts into a vocabulary:

```
cat sentences.wfreq | wfreq2vocab > sentences
    .vocab
```

3. Generate a binary id 3-gram of the training text, based on the above vocabulary:

```
cat sentences.text | text2idngram -vocab
    sentences.vocab > sentences.idngram
```

4. Convert the idngram into a binary format language model:

```
idngram2lm -idngram sentences.idngram -vocab
    sentences.vocab -binary sentences.binlm
```

5. Generate an ARPA format language model from a binary format language model:

```
binlm2arpa -binary movies.binlm -arpa movies.
    arpa
```

The 'make' file we used to generate our LMs can be analysed in Appendix A.

3.2.3.3. Acoustic Model Package Creation

We follow the Robust group's Open Source Tutorial [23] and present here the steps on a Windows OS (Table 1) equiped of Cygwin. The initial step consists in getting data. We gather data from the Speech Audio File Repository of the VoxForge website. We create a root directory for the task called **asg-am** in the Cygwin user home directory.

In order to get relevant results from the decoder, it is imperative to use the SphinxTrain. We download the file **SphinxTrain.nightly.tar.gz** containing the source code in C and extract it in Cygwin user home directory. The next phase is the compilation of the trainer: from the Cygwin command line, move to the SphinxTrain directory and type the following commands:

```
.../SphinxTrain>./configure
.../SphinxTrain>make
```

Once the SphinxTrain has been installed, we need to configure the task. We move to the asg-am directory and input the command:

```
.../asg-am>../SphinxTrain/scripts_pl/
    setup_SphinxTrain.pl task asg-am
```

The next step consists in preparing the audio data. We put the gathered audio in **asg-am/wav** folder and the necessary property files in the **asg-am/etc** folder. Those files which have to be created from the audio material are listed here:

- asg-am.dic: includes each word and the phonemes that make up each word.
- asg-am.filler: includes generally filler sounds for <s>, <sil>, and </s>.
- asg-am.phone: includes the phonemes that are part of the training set (must not have unused phonemes).
- asg-am_train.fileids: contains the names of the audio files of the asg-am/wav folder without extension - one on each line in the same order as the audio files order.
- asg-am_train.transcription: contains the transcription of each audio file - one on each line in the same order as the audio files order and surrounded by the markers <s> and </s>.
- feat.params: generated by SphinxTrain.
- sphinx_train.cfg: generated by SphinxTrain.

After adjusting the configuration in sphinx_train.cfg, the model creation can take place by first generating the feature files from the WAV files and then by running all the perl scripts to complete the process. Here are the two successive commands used:

```
.../asg-am>./scripts_pl/make_feats.pl -ctl etc/asg
    -am_train.fileids
.../asg-am>./scripts_pl/RunAll.pl
```

For our try, we used 9229 audio files from different native english speakers. We successfully created the features for all the files in about ten minutes. Later on, we encountered a fatal error in the module 'Training Context Independent models' that stopped the process. We then tried with less of the same data and it yielded to the same error. This issue may come from various sources such as erroneous file or audio file, uncomplete configuration or others. It does not represent a critic problem within the scope of this thesis work. It just limits the experiments to use existing acoustic models and thus might distort results when testing media content since customization cannot be proposed.

3.2.4. Subtitle Generation

Based on the analysis and design realized previously, a module generating subtitles in SRT format has been implemented in Java. To do so, we created three classes **TimedToken**, **Subtitle** and **SubtitleGenerator**. The class diagram is provided in Figure 7. The Subtitle class permits to encapsulate a subtitle in SRT format as described in the dedicated part of the background study 2.4. The SubtitleGenerator class provides static methods to create Subtitle instances using **Result** objects from the Sphinx-4 apparatus. Then, the method *getTimedBestResult()* from Result is used to retrieve both tokens and times in string form. The String is then parsed and each token and its times are used as input to instantiate a TimedToken. From this point, a list of TimedToken objects is available. Finally, the list is traversed and different operations are made according to the type of token. The algorithm was described in Figure 6. Each Subtitle text is delimited by two silences.

The SubtitleGenerator class is used by the **AutoSubGenProcessor** class. This class handles the allocation of resources for the Sphinx-4 module and runs the recognition phase until the end of the audio file is reached. Then, a subtitle file is created from the list of subtitles generated during the recognition process. The **Main** class provides just a **main** method that retrieves necessary parameters: path of the output subtitle file, path of the configuration file, and path of the audio file to be decoded.

4. EXPERIMENTS AND RESULTS

This section presents the different aspects to take in consideration in the process of experiment. It explains the auxiliary tasks to setup, namely data gathering, experiment machine and Sphinx-4 configuration and details the different experiments to be processed.

4.1. Data Gathering

In order to obtain interesting results, it is necessary to use numerous data of various content. We have defined five global categories: amateur, movie, news, serie, song. It appears evident more specialized categories would bring more accuracy since acoustic and language models are category-specific. In other terms, the more limited the category is, the more accurate the results are. We already discussed about the worldwide presence of several websites based on video media (e.g. YouTube, Dailymotion, ...). We thus estimate insightful the use of their material to generate our tests since they are used by hundred of thousand people. It exists different tools offering to retrieve videos from those websites. We opted for *KEEPVID* [35] that permits to download the desired video in MP4 format from its URL.

4.2. Considered Experiments

In this section, we explain the parameters to setup for the purpose of realizing experiments. The initial step lies in the experiment machine choice. We then consider suitable media categories where results can be expected and thereby we select a set of related media data. Next, we describe the language and acoustic models being deemed within those experiments. The following phase deals with the Sphinx-4 decoder tuning. The final subsection covers a typical experiment run.

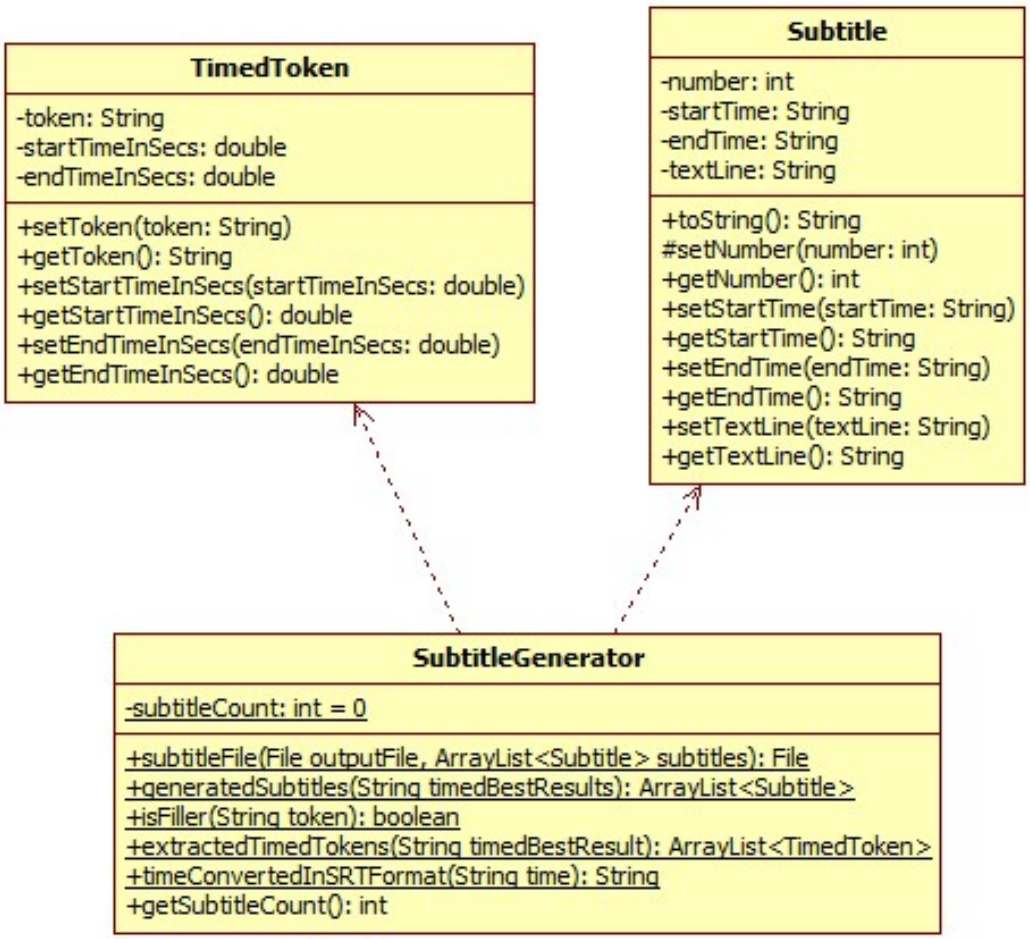
4.2.1. Machine Configuration

The first experiments run on the implementation machine we discussed earlier. Its characteristics are exposed in Table 1. It is a correct computer proposing similar properties as most machines used by the public. More powerful processors and larger RAM would permit execution time optimization but it is not of absolute necessity within the scope of this thesis work.

4.2.2. Experiment Data

Media Categories In the previous section, we introduced the categorization of media content within five distinct classes: amateur, movie, news, serie, song. Because of time constraints, we cannot afford the testing of

Fig. 7. Class Diagram for Subtitle Generation.



all classes. Therefore, we opt for news and excerpts of movies and series as input material.

Elected Data A set of news items as well as movie and serie samples from video and audio websites has been retained and is available in Table 2.

4.2.3. Choosing Suitable Models

There already exists a language and acoustic model trained for news content called HUB4 corpus. It gives suitable material to experiment news media. It is more complicated concerning movie and serie samples since new models must be created. You can refer to sections 3.2.3.2 and 3.2.3.3 to review the processes for such tasks. However, HUB4 corpus is also used for the samples because customized models have not been successfully generated during the training process. Besides, runs using the newswire text provided in the English Gigaword corpus (1200 Millions words from different news sources) are also realized. The acoustic model remains HUB4.

4.2.4. Sphinx-4 Tuning

The entire XML configuration file is proposed in Appendix B. The main parameters are now considered. They permit to control the beamwidths used in the search and their value affects both speed and accuracy of the recognition. The parameter **wordInsertionProbability** has an effect on the probability of word changes. As for the parameter **languageWeight**, it sets the weight of the language model compared to the phone scores. The main parameters are listed below:

- absoluteBeamWidth=10000
- relativeBeamWidth=1E-80

Table 2. Lists the media items retained to participate to the experiments.

Id	Category	Title	Duration
n1	News	Ron Paul on Fox Business News	00:09:55
n2	News	Peter Schiff 1/15/09 - Wall Street Journal	00:03:48
m1	Movie	Pulp Fiction: Foot Massage	00:01:02
s1	Serie	One Tree Hill - Girlfight	00:01:38
voa1	News	A Doctor Who Left His Mark on a World...	00:03:57
voa2	News	A Handful of Findings About Finger Length	00:03:55

- absoluteWordBeamWidth=20
- relativeWordBeamWidth=1E-60
- wordInsertionProbability=0.2
- languageWeight=10.5

4.2.5. Recognition Evaluation Measures

In order to efficiently evaluate the Sphinx-4 system, we align the output text called hypothesis with the actual transcription named reference. Three error types are distinguished in this process:

- Substitution: deals with words that are wrongly recognized (including singular and plural differentiation).
- Insertion: additional word in the hypothesis that is absent in the reference.
- Deletion: word present in the reference but not in the hypothesis.

Two measures permit to quantify the errors, namely Word Accuracy (WA) and Word Error Rate (WER). They are determined as:

$$WA = \frac{\textit{total words} - \textit{substitutions} - \textit{deletions}}{\textit{total words}} \quad (2)$$

$$WER = \frac{\textit{total word errors}}{\textit{total words}} \quad (3)$$

where

$$\textit{total word errors} = \textit{substitutions} + \textit{insertions} + \textit{deletions} \quad (4)$$

and *total words* is the number of words in the reference transcript.

4.2.6. Sequence of Operations

First of all, a research on media content websites has to be done in order to determine potential items being used in the experiments. The main criteria include sufficient speech sequences, reasonable length - limited to 10 minutes and satisfactory audio quality. Once an item has been elected, basic information on its origin and content are saved. It is then treated by VLC, cf. 3.2.2, and an audio file in WAV format is output. The audio is then given as input to the AutoSubGen **main** method and the speech recognition and subtitle generation processes can start. The instrumentation functionalities permit to monitor accuracy using alignment principle. Once the process is done, log and subtitle file can be retrieved and analysed.

4.3. Results

The media contents listed in Table 2 have been tested several times during the experiment phase using slightly modified configurations for each run. Outcomes for those media items have not satisfied expectations and present irrelevant text transcription, especially for movie and serie samples. WA for those cases is lower than 1%. Table 3 recapitulates the results for the item *voa1*. It can be observed the WA increases with the WER. Figure 8 presents a sample output from Sphinx-4 decoding. The reference sentence **REF** and the hypothesis **HYP** are listed. The lines in Figure 8 contain **ALIGN_REF** and **ALIGN_HYP** that are aligned so that differences are easily observed. **ALIGN_REF** lists the reference sentence where substitutions are written in capital letters. **ALIGN_HYP** lists the hypothesis sentence where substitutions are also written in capital letters and deletions are indicated by stars. Although recognition results are irrelevant for item *n2*, the corresponding SRT subtitle output is shown in Appendix C. The file format suits the requirements but the transcription is a complete nonsense. It can also be remarked just few words are

Table 3. Results of decoding for item *voa1*: A Doctor Who Left His Mark on a World That Lives in Fear of Ebola 3 minutes and 57 seconds long from VOANews.

Run	Time	Words	Matches	Errors	Subs	Ins	Del	WER(%)	WA(%)
1	00:50:23	403	7	399	188	3	208	99,007	1,737
2	00:41:50	403	15	530	317	142	71	131,514	3,722
3	00:59:55	403	17	554	313	168	73	137,469	4,218

recognized while it is a video of continuous speech of 3 minutes 48 seconds. Those issues are approached in the next section 5.

5. DISCUSSION

First of all, the audio extraction module has known a change of orientation because of the restricted audio formats handled by Sphinx-4. Although ways of converting audio files in appropriate formats were found, a proper implementation of the module has not been successfully reached. Therefore, we used the alternative described in the background study, i.e. VLC media player. It has provided satisfying results concerning audio conversion. The positive point rests on the vast number of input formats supported. It can directly take an audio video format and generate the WAV equivalent. We can however regret it does not offer a standard command-line interface for Microsoft OS, in which case we could have used the capabilities of Java [36] to encapsulate VLC necessary commands. Consequently, the alternative way has been used for our experiments and gave expected results. The initial direction can be envisaged in further work.

In a second time, speech recognition was approached. The use of Sphinx-4 appeared obvious for two reasons. Thanks to the modular design and the simple configuration, the setup was straightforward. Besides, the existence of pre-trained acoustic and language models resulting from the HUB4 corpus plus the Gigaword corpus was an interesting base to plan experiments and expect correct results for news content. Nevertheless, news results have not been very successful as it was observed in Table 3. Concerning our second category of video media being tested, namely movie and serie samples, the outcome is close to null. This hitch can be explained by the failure to make specific acoustic and language models operational. Thereby, a language model based on serie dialog was run but yielded to a runtime error. As for the acoustic model, fatal errors were encountered early in the SphinxTrain creation process. The issues were investigated in vain. Fatal errors remain. Therefore, the experiments of movie and serie samples have also been generated using HUB4 corpus, that can explain their irrelevant accuracy. Another explication relies on the loss of quality through conversions. A video stored on a media website such as YouTube has undergone a conversion from its original format to a compressed format, e.g. Flash Video (FLV). It is then retrieved for the experiment purpose and converted in suitable WAV format. However, vital information has been lost during the compression and quality of the final WAV audio is nearly equivalent to the compressed one. Besides, the input file should be of same format and quality as that with which the acoustic model was trained. Possibilities of getting interesting results using HUB4 to decode compressed news videos is thus low. A solution would be to train an acoustic model using FLV encoded samples. The training of acoustic models using transcoded speech is studied by Besacier et al. [37].

The last part of the thesis work consisted in returning subtitles formatted according to standards. The SRT standard was chosen because of its current popularity within the Internet community. The module described in section 3 respected its requirements and generates subtitle files that can be added to videos in media player software. The timecode of each subtitle seems to be precise and the user can benefit of the text. At the moment, each utterance delimited by two silences serves as input for a subtitle occurrence. This could be refined by merging together words of which times of apparition are very closed.

The system has shown some interesting characteristics and seems to have a great potential. It has demonstrated a possibility of running on Microsoft OS. Besides, it is facile to bring it on GNU/Linux systems since the main system is implemented in Java and VLC is a C software also designed for the open-source OS. It has been observed the decoding process requires a large amount of RAM, about 250 Megabytes, when using large

Fig. 8. AutoSubGen Output Sample using item of id *voa1*. It contains the reference and hypothesis sentences as well as the aligned reference and hypothesis.

```
REF:       health officials in the democratic republic of
congo have declared the end of an outbreak of ebola virus

HYP:       she i mean you are womb or ten you should

ALIGN_REF: HEALTH OFFICIALS IN   THE DEMOCRATIC REPUBLIC
OF CONGO HAVE DECLARED THE END OF AN OUTBREAK OF EBOLA VIRUS
ALIGN_HYP: SHE    I         MEAN YOU ARE        WOMB
OR TEN   YOU SHOULD   *** *** ** ** ******** ** ***** *****

REF:       cases were first identified in december in the
southern province of kasai occidental

HYP:       he has pulled

ALIGN_REF: CASES WERE FIRST  IDENTIFIED IN DECEMBER IN THE
SOUTHERN PROVINCE OF KASAI OCCIDENTAL
ALIGN_HYP: HE    HAS  PULLED ********** ** ******** ** ***
******** ******** ** ***** **********

REF:       the world health organization says officials
reported a total of thirty-two cases

HYP:       the room i i she will

ALIGN_REF: the WORLD HEALTH ORGANIZATION SAYS OFFICIALS
REPORTED A TOTAL OF THIRTY-TWO CASES
ALIGN_HYP: the ROOM  I      I            SHE  WILL
******** * ***** ** ********** *****

Accuracy: 2,326%    Errors: 42  (Sub: 18  Ins: 0  Del: 24)
Words: 43   Matches: 1    WER: 97,674%
Sentences: 3   Matches: 0   SentenceAcc: 0,000%
```

vocabulary corpus (e.g. HUB4). It is therefore important to choose a correct machine. As already mentioned in the previous sections, most computers used either for private usage or in laboratories can bear this charge. Then the capacity of the processor and the configuration determine the time in which the audio is decoded.

We can examine the impact such software could have on the public. Even if the results obtained during experiments are low, cf. Table 3, it has been shown in different related works recognition of news material reaches a worthwhile level with average accuracy of 70% - e.g. [38], [39], [40]. Such rates can only be attained with very trained and specific models like HUB4 specially dedicated for news environment. Our attempt to use customised models specially trained for series failed despite a total respect of the guidelines. The irrelevant rate for movie and serie material decoded using HUB4 demonstrates the importance of creating and applying specific models for each media category. Besides, models for most of the media contents apart news appear to be truly complex to be defined since they surround a large domain of themes. A pertinent idea would be to develop an acoustic model and a language model for each theme

within a category. The idea is recapitulated in the following non-exhaustive list:

- **Amateur**: Birthday, Wedding, Unforeseen Reportage.
- **Movie**: Action, Romance, Children-Oriented, Comedy, Thriller, Science-Fiction.
- **Serie**: One acoustic and one language model for each serie.
- **Song**: Rock, Pop, R'nB, Disco.

Eventually, most categories listed above involve background noise and silence periods. The Sphinx-4 FrontEnd has at its disposal utilities to prevent and lessen noise impact - configuration file available in Appendix B. Unfortunately, it is not sufficient and a deeper work would be necessary. That task would act on the audio by differentiating the different kinds of audio. An interesting work on audio classification has been realised in the master thesis cited previously [38]. It is explained a way to differentiate speech and music. Thereby, it diminishes the task of the decoder since only speech segments are used as input.

6. CONCLUSION

We proposed a way to generate subtitles for sound videos. A complete system including the three required modules introduced in section 1 could not be realized since the audio conversion needed more resources. VLC gave an appropriate solution but a custom component coded in Java is expected in further work so that portability and installation of the system is rendered uncomplicated. Nonetheless, the expected output for each phase has been reached. The audio extraction module provides a suitable audio format to be used by the speech recognition module. This one generates a list of recognized words and their corresponding time in the audio although the accuracy is not guaranteed. The former list is used by the subtitle generation module to create standard subtitle file readable by the most common media players available.

In a cyberworld where the accessibility remains insufficient, it is essential to give each individual the right to understand any media content. During the last years, the Internet has known a multiplication of websites based on videos of which most are from amateurs and of which transcripts are rarely available. This thesis work was mostly orientated on video media and suggested a way to produce transcript of audio from video for the ultimate purpose of making content comprehensible by deaf persons. Although the current system does not present enough stability to be widely used, it proposes one interesting way that can certainly be improved. The next section deals with further work that could be realized to enhance the concept.

6.1. Further Work

In the first place, it is important to think about implementing a specific module for audio conversion into suitable WAV format. Directions of development in the Java programming language have been described earlier. This achievement is primordial for the completeness and the portability of the AutoSubGen system.

In such a system like AutoSubGen, parallelization could be seriously envisaged. A possible direction would consist in smartly dividing the audio track in numbered subtracks to keep the order. Next, each subtrack would be given as parameter to a process standing for an instance of an AutoSubGen subroutine allowing decoding and subtitle generation from the subtrack. Finally, a subroutine would merge the various generated subtitle files within one respecting the numbered order.

When talking about the subtitle generation module, we emphasized the insertion of punctuation was a complicated task to be performed by an ASR apparatus. It could be interesting to lead a study towards this subject because the outcome of an ASR system is generally a raw text, in a lower-text format and without any punctuation mark while the former plays a significant role in the understanding of talk exchanges. Several methodologies should be deemed such as the use of transducers or language model enhancements.

Abbreviations

API : **A**pplication **P**rogramming **I**nterface
ASR : **A**utomatic **S**peech **R**ecognition
CFG : **C**ontext **F**ree **G**rammar
CMU : **C**arnegie **M**ellon **U**niversity
FSG : **F**inite-**S**tate **G**rammar
FST : **F**inite-**S**tate **T**ransducer
GUI : **G**raphical **U**ser **I**nterface
HMM : **H**idden **M**arkov **M**odel
IDE : **I**ntegrated **D**evelopment **E**nvironment

JMF : **J**ava **M**edia **F**ramework
JRE : **J**ava **R**untime **E**nvironment
LM : **L**anguage **M**odel
OS : **O**perating **S**ystem
SDK : **S**oftware **D**evelopment **K**it
URL : **U**niform **R**esource **L**ocator
WA : **W**ord **A**ccuracy
WER : **W**ord **E**rror **R**ate
XML : **E**xtensible **M**arkup **L**anguage

Acknowledgements

I investigated this topic as part of my Master of Science in Software Engineering. Dedication and intense research work have been determining factors in the writing of this thesis.

I would like to thank the SourceForge.net community for giving insightful tips about technical concerns when needed. I also would like to express my gratitude towards my supervisor Jonas Amoson and my examiner Steven Kirk for offering precious advices concerning main issues of such a thesis work.

References

[1] Imtoo dvd subtitle ripper. URL http://www.imtoo.com/dvd-subtitle-ripper.html.

[2] Xilisoft dvd subtitle ripper. URL http://www.xilisoft.com/dvd-subtitle-ripper.html.

[3] Subtitle editor, 2005-2009. URL http://home.gna.org/subtitleeditor/.

[4] Subtitle workshop, 2006. URL http://subtitle-workshop.en.softonic.com/.

[5] Jubler subtitle editor in java, 2005-2008. URL http://www.jubler.org/.

[6] Gaupol. URL http://home.gna.org/gaupol/.

[7] J.P. Lewis and Ulrich Neumann. Performance of java versus c++. 2003. URL http://www.idiom.com/~zilla/Computer/javaCbenchmark.html.

[8] The cmu sphinx group open source speech recognition engines, 1999-2001. URL http://cmusphinx.sourceforge.net/.

[9] Sphinx-4 a speech recognizer written entirely in the javatm programming language, 1999-2008. URL http://cmusphinx.sourceforge.net/sphinx4/.

[10] Java media framework api (jmf), 1994-2009. URL http://java.sun.com/javase/technologies/desktop/media/jmf/.

[11] Tutorial : Getting started with the javatm media framework. URL http://www.ee.iitm.ac.in/~tgvenky/JMFBook/Tutorial.pdf.

[12] Java tm media framework api guide, 1999. URL http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/guide/index.html.

[13] Java media framework 1.0 programmers guide, 1998. URL http://java.sun.com/javase/technologies/desktop/media/jmf/1.0/guide/index.html.

[14] Robert Gordon, Stephen Talley, and Rob Gordon. *Essential Jmf: Java Media Framework*. Prentice Hall PTR, 1998.

[15] Splitting tracks from an input, 1994-2009. URL http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/solutions/Split.html.

[16] Vlc media player. URL http://www.videolan.org/vlc/.

[17] Engineered Station. How speech recognition works. 2001. URL http://project.uet.itgo.com/speech.htm.

[18] Hala ElAarag and Laura Schindler. A speech recognition and synthesis tool. *ACM SE'06*, pages 45–49, 2006.

[19] Frederick Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1999.

[20] Zoubin Ghahramani. An introduction to hidden markov models and bayesian networks. In *World Scientific Publishing Co., Inc. River Edge, NJ, USA*, 2001.

[21] Bayes' theorem. URL http://en.wikipedia.org/wiki/Bayes'_theorem.

[22] Voxforge. URL http://www.voxforge.org/home.

[23] Robust group's open source tutorial learning to use the cmu sphinx automatic speech recognition system, 1999-2008. URL http://www.speech.cs.cmu.edu/sphinx/tutorial.html.

[24] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition. In *SMLI TR2004-0811 2004 SUN MICROSYSTEMS INC.*, 2004. URL http://cmusphinx.sourceforge.net/sphinx4/doc/Sphinx4Whitepaper.pdf.

[25] Sphinx 4 for the javatm platform architecture notes. URL www.speech.cs.cmu.edu/sphinx/twiki/pub/Sphinx4/WebHome/Architecture.pdf.

[26] Configuration management for sphinx-4, 1999-2004. URL http://cmusphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/util/props/doc-files/ConfigurationManagement.html.

[27] Instrumentation for sphinx-4, 1999-2004. URL http://cmusphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/instrumentation/doc-files/Instrumentation.html.

[28] Apache ant 1.7.1 manual, 2004. URL http://ant.apache.org/manual/index.html.

[29] Welcome to the ant wiki, 2004-present. URL http://wiki.apache.org/ant/FrontPage.

[30] Erik Hatcher and Steve Loughran. *Java Development with Ant*. Manning Publications, 2002.

[31] Apache ant binary distributions. URL http://ant.apache.org/bindownload.cgi.

[32] Mp3 conversion in sphinx4. URL http://www.bakuzen.com/?p=14.

[33] Statistical language modeling toolkit, 1999. URL http://www.speech.cs.cmu.edu/SLM/toolkit.html.

[34] Cygwin, 2001-2008. URL http://www.cygwin.com/.

[35] Keepvid. URL http://keepvid.com/.

[36] Using runtime.exec to invoke child processes and programming with file attributes. URL http://java.sun.com/developer/JDCTechTips/2003/tt0304.html.

[37] L. Besacier, C. Bergamini, D. Vaufreydaz, and E. Castelli. The effect of speech and audio compression on speech recognition performance. In *IEEE Multimedia Signal Processing Workshop, Cannes : France*, 2001. URL http://hal.archives-ouvertes.fr/docs/00/32/61/65/PDF/Besacier01b.pdf.

[38] Kasper Winther Joergensen and Lasse Lohilahti Moelgaard. Tools for automatic audio indexing. Master's thesis, Kongens Lyngby, 2006.

[39] Jean-Luc Gauvain, Lori Lamel, Gilles Adda, and Martine Adda-Decker. Transcription of broadcast news. In *EuroSpeech*, 1997.

[40] Paul Deleglise, Yannick Esteve, Sylvain Meignier, and Teva Merlin. The lium speech transcription system: a cmu sphinx iii-based system for french broadcast news. In *Interspeech, Lisbon, Portugal*, 2005. URL http://www-lium.univ-lemans.fr/speechLIUMtools/Downloads/LIUM_Interspeech05.pdf.

A. MAKE LM SCRIPT UNDER LINUX OR CYGWIN

```
#
# Generates the language model for a movies environment
#
# uses:
#       autosubgen.txt - transcript of autosubgen
#
# generates:
#       autosubgen.vocab - vocabulary list of the 30,000 most repeated words
#       autosubgen.lm - arpa format of the language model
#       autosubgen.DMP - CMU binary format of the language model
#       autosubgen.transcript - transcript
#
#
# requires:
#       CMU language model toolkit:
#               http://www.speech.cs.cmu.edu/SLM_info.html
#       lm3g2dmp - utility to generate DMP format models:
#           http://cmusphinx.sourceforge.net/webpage/html/download.php#utilities#
# unix commands:
#       gawk uniq mv rmdir rm
#
# All commands should be in your path
#

bn=autosubgen.txt

# We want a closed vocabulary language model so we use
# extractVocab to extract just the sentences that entirely
# match our vocabulary (Not used in our case)

#gawk -f extractVocab.awk autosubgen.vocab  autosubgen.txt  > $bn.tmp.closed

# Copy content of original transcript when closed vocab is not used

cat $bn > $bn.tmp.closed

#
# We generate the 'test' file that can be used by the live decoder
# as the prompt for the user. We eliminate adjacent duplicate entries

#gawk -f genTranscript.awk < $bn.tmp.closed > autosubgen.transcript

#
# Generate the word frequencies
#

text2wfreq < $bn.tmp.closed > $bn.tmp.wfreq

#
# Generate the vocabulary (this should be a subset autosubgen.vocab if filter vocabulary file provided)
#

wfreq2vocab < $bn.tmp.wfreq > $bn.tmp.vocab

#
# Generate the idngram
#

text2idngram -vocab $bn.tmp.vocab < $bn.tmp.closed  > $bn.tmp.idngram


#
# Generates the language model
#

idngram2lm -vocab_type 0 -idngram $bn.tmp.idngram -vocab $bn.tmp.vocab -arpa $bn.arpa


#
```

```
70 # Generate the DMP version of the language model
71 #
72
73 mkdir dmp
74 lm3g2dmp $bn.arpa dmp
75 mv dmp/$bn.arpa.DMP autosubgen.DMP
76 mv $bn.arpa autosubgen.lm
77
78 #
79 # Cleanup
80 #
81
82 rmdir dmp
83 rm *.tmp.*
```

B. SPHINX-4 XML-CONFIGURATION

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ******************************************************* -->
<!-- Sphinx-4 Configuration file -->
<!-- ******************************************************* -->

<config>
  <!-- ******************************************************* -->
  <!-- frequently tuned properties -->
  <!-- ******************************************************* -->
  <property name="absoluteBeamWidth" value="10000"/>
  <property name="relativeBeamWidth" value="1E-80"/>
  <property name="absoluteWordBeamWidth" value="20"/>
  <property name="relativeWordBeamWidth" value="1E-60"/>
  <property name="wordInsertionProbability" value="0.2"/>
  <property name="languageWeight" value="10.5"/>
  <property name="silenceInsertionProbability" value=".05"/>
  <property name="frontend" value="wavFrontEnd"/>
  <property name="recognizer" value="recognizer"/>
  <property name="showCreations" value="false"/>
  <config>
    <property name="logLevel" value="SEVERE"/>
  </config>

  <!-- ************************************************ -->
  <!-- Batch mode -->
  <!-- ************************************************ -->
  <component name="batch" type="edu.cmu.sphinx.tools.batch.BatchModeRecognizer">
    <propertylist name="inputDataProcessors">
      <item>streamDataSource</item>
    </propertylist>
    <property name="skip" value="0"/>
    <property name="recognizer" value="${recognizer}"/>
  </component>

  <!-- ******************************************************* -->
  <!-- word recognizer configuration -->
  <!-- ******************************************************* -->
  <component name="recognizer" type="edu.cmu.sphinx.recognizer.Recognizer">
    <property name="decoder" value="decoder"/>
    <propertylist name="monitors">
      <item>accuracyTracker </item>
      <item>speedTracker </item>
      <item>memoryTracker </item>
      <item>recognizerMonitor </item>
    </propertylist>
  </component>

  <!-- ******************************************************* -->
  <!-- The Decoder configuration -->
  <!-- ******************************************************* -->
  <component name="decoder" type="edu.cmu.sphinx.decoder.Decoder">
    <property name="searchManager" value="wordPruningSearchManager"/>
  </component>

  <!-- ******************************************************* -->
  <!-- The Search Manager -->
  <!-- ******************************************************* -->
  <component name="wordPruningSearchManager" type="edu.cmu.sphinx.decoder.search.
      WordPruningBreadthFirstSearchManager">
    <property name="logMath" value="logMath"/>
    <property name="linguist" value="lexTreeLinguist"/>
    <property name="pruner" value="trivialPruner"/>
    <property name="scorer" value="threadedScorer"/>
    <property name="activeListManager" value="activeListManager"/>
    <property name="growSkipInterval" value="0"/>
    <property name="checkStateOrder" value="false"/>
    <property name="buildWordLattice" value="false"/>
    <property name="maxLatticeEdges" value="3"/>
    <property name="acousticLookaheadFrames" value="1.7"/>
```

```
    <property name="relativeBeamWidth" value="${relativeBeamWidth}"/>
  </component>

  <!-- ******************************************************* -->
  <!-- The Active Lists -->
  <!-- ******************************************************* -->
  <component name="activeListManager" type="edu.cmu.sphinx.decoder.search.SimpleActiveListManager">
    <propertylist name="activeListFactories">
      <item>standardActiveListFactory</item>
      <item>wordActiveListFactory</item>
      <item>wordActiveListFactory</item>
      <item>standardActiveListFactory</item>
      <item>standardActiveListFactory</item>
      <item>standardActiveListFactory</item>
    </propertylist>
  </component>

  <component name="standardActiveListFactory" type="edu.cmu.sphinx.decoder.search.
      PartitionActiveListFactory">
    <property name="logMath" value="logMath"/>
    <property name="absoluteBeamWidth" value="${absoluteBeamWidth}"/>
    <property name="relativeBeamWidth" value="${relativeBeamWidth}"/>
  </component>

  <component name="wordActiveListFactory" type="edu.cmu.sphinx.decoder.search.PartitionActiveListFactory">
    <property name="logMath" value="logMath"/>
    <property name="absoluteBeamWidth" value="${absoluteWordBeamWidth}"/>
    <property name="relativeBeamWidth" value="${relativeWordBeamWidth}"/>
  </component>

  <!-- ******************************************************* -->
  <!-- The Pruner -->
  <!-- ******************************************************* -->
  <component name="trivialPruner" type="edu.cmu.sphinx.decoder.pruner.SimplePruner"/>

  <!-- ******************************************************* -->
  <!-- The Scorer -->
  <!-- ******************************************************* -->
  <component name="threadedScorer" type="edu.cmu.sphinx.decoder.scorer.ThreadedAcousticScorer">
    <property name="frontend" value="${frontend}"/>
    <property name="isCpuRelative" value="false"/>
    <property name="numThreads" value="10"/>
    <property name="minScoreablesPerThread" value="10"/>
    <property name="scoreablesKeepFeature" value="false"/>
  </component>
  <!-- ******************************************************* -->
  <!-- The linguist configuration -->
  <!-- ******************************************************* -->
  <component name="lexTreeLinguist" type="edu.cmu.sphinx.linguist.lextree.LexTreeLinguist">
    <property name="logMath" value="logMath"/>
    <property name="acousticModel" value="hub4"/>
    <property name="languageModel" value="hub4Model"/>
    <property name="dictionary" value="dictionaryHUB4"/>
    <property name="addFillerWords" value="false"/>
    <property name="fillerInsertionProbability" value="1E-10"/>
    <property name="generateUnitStates" value="true"/>
    <property name="wantUnigramSmear" value="true"/>
    <property name="unigramSmearWeight" value="1"/>
    <property name="wordInsertionProbability" value="${wordInsertionProbability}"/>
    <property name="silenceInsertionProbability" value="${silenceInsertionProbability}"/>
    <property name="languageWeight" value="${languageWeight}"/>
    <property name="unitManager" value="unitManager"/>
  </component>

    <!-- ******************************************************* -->
    <!-- The Dictionary configuration HUB4                       -->
    <!-- ******************************************************* -->
    <component name="dictionaryHUB4"
        type="edu.cmu.sphinx.linguist.dictionary.FullDictionary">
        <property name="dictionaryPath"
```

```
                value="resource:/edu.cmu.sphinx.model.acoustic.HUB4_8gau_13dCep_16k_40mel_133Hz_6855Hz.
                    Model!/edu/cmu/sphinx/model/acoustic/HUB4_8gau_13dCep_16k_40mel_133Hz_6855Hz/cmudict
                    .06d"/>
        <property name="fillerPath"
            value="resource:/edu.cmu.sphinx.model.acoustic.HUB4_8gau_13dCep_16k_40mel_133Hz_6855Hz.Model!/
                edu/cmu/sphinx/model/acoustic/HUB4_8gau_13dCep_16k_40mel_133Hz_6855Hz/fillerdict"/>
        <property name="addSilEndingPronunciation" value="false"/>
        <property name="wordReplacement" value="&lt;sil&gt;"/>
        <property name="allowMissingWords" value="false"/>
        <property name="unitManager" value="unitManager"/>
    </component>

  <!-- ******************************************************* -->
  <!-- The Language Model configuration HUB4 -->
  <!-- ******************************************************* -->
  <component name="hub4Model"
        type="edu.cmu.sphinx.linguist.language.ngram.large.LargeTrigramModel">
        <property name="logMath" value="logMath"/>
        <property name="maxDepth" value="3"/>
        <property name="unigramWeight" value=".5"/>
        <property name="dictionary" value="dictionaryHUB4"/>
        <property name="location"
            value="D:/lectures/Master_Soft_Eng/Thesis_Work(exd950)/ExperimentalSystem/AutoSubGen/models/
                language/HUB4_trigram_lm/language_model.arpaformat.DMP"/>
    </component>

  <!-- ******************************************************* -->
    <!-- The acoustic model configuration HUB4                    -->
    <!-- ******************************************************* -->
    <component name="hub4"
              type="edu.cmu.sphinx.model.acoustic.HUB4_8gau_13dCep_16k_40mel_133Hz_6855Hz.Model">
        <property name="loader" value="hub4Loader"/>
        <property name="unitManager" value="unitManager"/>
    </component>

    <component name="hub4Loader" type="edu.cmu.sphinx.model.acoustic.
        HUB4_8gau_13dCep_16k_40mel_133Hz_6855Hz.ModelLoader">
        <property name="logMath" value="logMath"/>
        <property name="unitManager" value="unitManager"/>
    </component>

  <!-- ******************************************************* -->
  <!-- The unit manager configuration -->
  <!-- ******************************************************* -->
  <component name="unitManager" type="edu.cmu.sphinx.linguist.acoustic.UnitManager"/>

  <!-- ******************************************************* -->
  <!-- The frontend configuration -->
  <!-- ******************************************************* -->
  <component name="wavFrontEnd" type="edu.cmu.sphinx.frontend.FrontEnd">
    <propertylist name="pipeline">
      <item>streamDataSource</item>
      <item>speechClassifier</item>
      <item>speechMarker</item>
      <item>nonSpeechDataFilter</item>
      <item>premphasizer</item>
      <item>windower</item>
      <item>fft</item>
      <item>melFilterBank</item>
      <item>dct</item>
      <item>liveCMN</item> <!-- batchCMN for batch mode -->
      <item>featureExtraction</item>
    </propertylist>
  </component>

  <component name="streamDataSource" type="edu.cmu.sphinx.frontend.util.StreamDataSource">
    <property name="sampleRate" value="16000"/>
    <property name="bitsPerSample" value="16"/>
    <property name="bigEndianData" value="false"/>
    <property name="signedData" value="true"/>
    <property name="bytesPerRead" value="320"/>
```

```
</component>

<component name="speechClassifier" type="edu.cmu.sphinx.frontend.endpoint.SpeechClassifier">
  <property name="threshold" value="11"/>
</component>

<component name="nonSpeechDataFilter" type="edu.cmu.sphinx.frontend.endpoint.NonSpeechDataFilter">
</component>

<component name="speechMarker" type="edu.cmu.sphinx.frontend.endpoint.SpeechMarker">
  <property name="speechTrailer" value="50"/>
</component>

<component name="premphasizer" type="edu.cmu.sphinx.frontend.filter.Preemphasizer">
  <property name="factor" value="0.9"/>
</component>

<component name="windower" type="edu.cmu.sphinx.frontend.window.RaisedCosineWindower">
  <property name="windowSizeInMs" value="25"/>
</component>

<component name="fft" type="edu.cmu.sphinx.frontend.transform.DiscreteFourierTransform"/>

<component name="melFilterBank" type="edu.cmu.sphinx.frontend.frequencywarp.MelFrequencyFilterBank">
  <property name="numberFilters" value="40"/>
  <property name="minimumFrequency" value="130.0"/>
  <property name="maximumFrequency" value="6800.0"/>
</component>

<component name="dct" type="edu.cmu.sphinx.frontend.transform.DiscreteCosineTransform"/>

<component name="liveCMN" type="edu.cmu.sphinx.frontend.feature.LiveCMN"/>

<component name="batchCMN" type="edu.cmu.sphinx.frontend.feature.BatchCMN"/>

<component name="featureExtraction" type="edu.cmu.sphinx.frontend.feature.DeltasFeatureExtractor"/>

<!-- ******************************************************* -->
<!-- monitors -->
<!-- ******************************************************* -->
<component name="accuracyTracker" type="edu.cmu.sphinx.instrumentation.BestConfidenceAccuracyTracker">
  <property name="confidenceScorer" value="confidenceScorer"/>
  <property name="recognizer" value="${recognizer}"/>
  <property name="showRawResults" value="true"/>
  <property name="showAlignedResults" value="true"/>
</component>

<component name="confidenceScorer" type="edu.cmu.sphinx.result.SausageMaker"/>

<component name="memoryTracker" type="edu.cmu.sphinx.instrumentation.MemoryTracker">
  <property name="recognizer" value="${recognizer}"/>
  <property name="showDetails" value="false"/>
  <property name="showSummary" value="false"/>
</component>

<component name="speedTracker" type="edu.cmu.sphinx.instrumentation.SpeedTracker">
  <property name="recognizer" value="${recognizer}"/>
  <property name="frontend" value="${frontend}"/>
  <property name="showDetails" value="false"/>
</component>

<component name="recognizerMonitor" type="edu.cmu.sphinx.instrumentation.RecognizerMonitor">
  <property name="recognizer" value="${recognizer}"/>
  <propertylist name="allocatedMonitors">
    <item>configMonitor</item>
  </propertylist>
</component>

<component name="configMonitor" type="edu.cmu.sphinx.instrumentation.ConfigMonitor">
  <property name="showConfig" value="false"/>
</component>
```

```
275
276   <!-- ********************************************************* -->
277     <!--  Miscellaneous components                                 -->
278     <!-- ********************************************************* -->
279     <component name="logMath" type="edu.cmu.sphinx.util.LogMath">
280         <property name="logBase" value="1.0001"/>
281         <property name="useAddTable" value="true"/>
282     </component>
283 </config>
```

C. SAMPLE SUBTITLE OUTPUT

```
1
00:00:00,349 --> 00:00:04,039
i roos rule for

2
00:00:04,489 --> 00:00:06,530
sean who huh who

3
00:00:07,190 --> 00:00:08,699
hold true shook

4
00:00:09,159 --> 00:00:10,970
off a apples or

5
00:00:11,710 --> 00:00:14,300
through old shrines

6
00:00:15,079 --> 00:00:17,500
and a routine or had

7
00:00:17,819 --> 00:00:21,239
a groove issue of

8
00:00:22,489 --> 00:00:22,489
for

9
00:00:24,319 --> 00:00:26,329
shrewd and bought

10
00:00:26,739 --> 00:00:30,170
war halftime shrewdly scholl
     loewe

11
00:00:31,090 --> 00:00:32,950
ruled moorhouse

12
00:00:33,470 --> 00:00:35,009
shunt lahore honk

13
00:00:35,689 --> 00:00:37,779
shuttled more movement now

14
00:00:38,810 --> 00:00:41,389
a harsher and more

15
00:00:41,990 --> 00:00:42,279
hum

16
00:00:42,799 --> 00:00:45,680
the u. k. caption

17
00:00:46,279 --> 00:00:48,610
the option of new

18
00:00:49,139 --> 00:00:51,049
bourque hum will

19
00:00:51,930 --> 00:00:53,750
humble ahead of a

20
00:00:54,090 --> 00:00:57,110
route de hunched lula

21
00:00:57,810 --> 00:00:59,700
a room by a little

22
00:01:00,630 --> 00:01:00,659
heavy

23
00:01:01,439 --> 00:01:04,069
i have trouble le

24
00:01:04,949 --> 00:01:06,230
shrewd helped blue

25
00:01:06,839 --> 00:01:09,230
more hatched ruled how

26
00:01:09,510 --> 00:01:12,430
i lured hulme and bought

27
00:01:13,470 --> 00:01:15,940
a hey commercial

28
00:01:16,529 --> 00:01:18,980
hell no draft kula

29
00:01:19,559 --> 00:01:21,569
him on a little

30
00:01:22,110 --> 00:01:24,500
fat old on gallucci

31
00:01:25,080 --> 00:01:28,169
and a who's sure of

32
00:01:28,629 --> 00:01:31,059
my asian hobnob

33
00:01:31,839 --> 00:01:32,419
shown walsh

34
00:01:33,059 --> 00:01:35,910
while lucia homolka

35
00:01:37,019 --> 00:01:41,870
shaw lemieux won't last
     hiroshima loewe

36
00:01:42,660 --> 00:01:42,660
lure

37
00:01:43,970 --> 00:01:46,540
orbach sure or true

38
00:01:47,309 --> 00:01:48,900
rouge ollie route

39
00:01:49,260 --> 00:01:51,879
to europe law and

40
00:01:52,599 --> 00:01:55,370
our losh home while

41
00:01:55,769 --> 00:02:00,269
the poor who are low act

42
00:02:02,440 --> 00:02:03,120
i come

43
00:02:04,099 --> 00:02:04,099
a

44
00:02:04,900 --> 00:02:08,110
the deal will i

45
00:02:08,570 --> 00:02:11,080
lose a loved all

46
00:02:11,839 --> 00:02:14,509
the while high ahead

47
00:02:14,899 --> 00:02:14,899
of

48
00:02:15,880 --> 00:02:18,479
the new moore and

49
00:02:18,979 --> 00:02:22,119
the law pollution

50
00:02:22,779 --> 00:02:24,460
humble yank moved more

51
00:02:24,779 --> 00:02:26,070
drawn on rwanda

52
00:02:26,610 --> 00:02:28,179
wanted to know how
```

```

53
00:02:28,529 --> 00:02:30,660
by a movie to a

54
00:02:31,100 --> 00:02:35,270
group hug of on luciano

55
00:02:35,809 --> 00:02:37,570
de option will

56
00:02:39,009 --> 00:02:40,179
on loesch awe

57
00:02:41,720 --> 00:02:44,539
yet it posh promotion on

58
00:02:44,809 --> 00:02:47,350
her role was packing law

59
00:02:48,100 --> 00:02:50,740
a lot a route not

60
00:02:51,679 --> 00:02:55,139
from a herschel aka

61
00:02:56,240 --> 00:02:59,449
owned by iowa who

62
00:02:59,880 --> 00:03:01,169
for all the

63
00:03:02,649 --> 00:03:04,020
o'leary then said

64
00:03:06,419 --> 00:03:09,380
you have good houle

65
00:03:10,289 --> 00:03:12,809
latin optional a.

66
00:03:13,279 --> 00:03:15,559
how hong orange

67
00:03:17,029 --> 00:03:19,050
laotian while drew

68
00:03:19,720 --> 00:03:24,630
a lukewarm notion on wahoo
      shulman issuer will

69
00:03:25,160 --> 00:03:26,600
have little power

70
00:03:27,320 --> 00:03:29,630
whom laura moon

71
00:03:30,720 --> 00:03:31,880
lockhart shovel

72
00:03:32,679 --> 00:03:34,089
humble schwab sure

73
00:03:34,720 --> 00:03:36,179
sharon on ruble lim

74
00:03:36,630 --> 00:03:39,339
a shrewd move holum

75
00:03:40,000 --> 00:03:40,000
hughes

76
00:03:40,800 --> 00:03:40,800
or
```