Speech coding using personalized speech repository

Index

1
INTRODUCTION & MOTIVATION

The project deals with the idea of achieving compression by coding a person's speech using digital signal processing, clustering and vector quantization algorithms.

People download a lot of audio and video over the Internet. Generally it takes a huge lot of time to download the audio speeches. e.g. downloading news spoken by a news reader, commentary of a particular match which created some history in the concerned sport, budget presentation by the Finance Minister, important messages by the President for the general public etc. In such cases so as to optimize the time required to download these huge files, our project focuses on the speech compression by speech coding. Since the process has to be carried out individually for every person therefore the term "personalized" in the title.

This work is based on the intuition that in a speech sample of a particular person, similar *elementary sounds* are repeated. For example, when a person says "cricket" and "club", the initial "kk" sound in both words will have similar characteristics. Significant reduction in storage could result if the actual signal information for both these sounds is not stored. Instead the *elementary sound* is stored just once and wherever this sound appears the same stored sound is played.

E-mail is good only for text, and for graphics transmission. Standard sound formats that encode human speech, produce extremely large outputs that are improper for e-mail communication. However, if certain assumptions are made about features of human speech, the communication will be efficient.

The speech profile of a person can be created which will contain the collection of elementary sounds uttered. This profile will be a one-time download for the listeners. The actual audio messages can be encoded based on the profile. The users will only need to download the encoded data (which will be much smaller than the actual audio data). This can be decoded using the profile stored earlier by the user, and the audio can be regenerated. As only the binary codes are transferred rather than the speech signals themselves, huge bandwidth compression can be obtained.

2
PROBLEM STATEMENT

The project involves building a system for exchanging voice messages over mail, using very high speech compression. The sender will record his voice message and transform it into the coded, compressed file using the encoder module. The coded file is transferred as an email attachment. The receiver passes the attached file through the decoder module, which reproduces the original speech. Both the encoder and decoder will use a repository of speech segments. This repository will be pretty large in size and may need to be transported by CDs etc.

The entire system (encoder, decoder and repository generator) needs to be prepared and coded for Linux. The project should deliver a easy-to-use package (it may be set of command-line tools) which will enable the proposed exchange of voice messages. The encoder tool should just take a sound file (maybe in the WAV format) and convert it into a compressed binary file. The decoder tool does the opposite job. The repository-generator tool works on a large sample of speech to generate the corpus.

3
REQUIREMENT ANALYSIS

3.1. Introduction

The project involves building a system for exchanging voice messages over mail, using very high speech compression, as described above. The sender will record his voice message and transform it into the coded, compressed file using the encoder module. The coded file is transferred as an email attachment. The receiver passes the attached file through the decoder module, which reproduces the original speech. Both the encoder and decoder will use a repository of speech segments. The repository may be transported by CDs, or may be made available for download, etc. The entire system (encoder, decoder and repository generator) will be prepared and coded for Linux.

The project will deliver an easy-to-use package which will enable the proposed exchange of voice messages.

- The repository-generator tool works on a large sample of speech to generate the corpus using clustering and Mel-frequency cepstrum coefficients (MFCC) feature extraction processes.
- The encoder tool will take a sound file and convert it into a compressed binary file, using the repository.
- The decoder tool does the opposite job.

3.2. Steps of the process

1. Repository generation

A recorded lecture will be obtained. All experiments will be conducted using this sample (sampling rate: 11025 Hz, single channel and 8-bits/sample.).

A 15-minute sample will be extracted for repository generation. This file will be divided into 45000 files of 20 ms duration each. 12 MFCC features (Mel-frequency cepstral coefficients) will be computed for each of these *sound-slices*. MFCC features are perception-based features, which are widely used in the speech recognition arena.

It is assumed that 10000 different *elementary sounds* will be enough to characterize the range of sounds produced by a person. This number will be arrived at empirically. The 45000 sound samples will then be clustered into 10000 clusters based on their Mel-frequency cepstrum coefficients (MFCC) features.

A variant of the k-mean algorithm will be used for clustering. For each of these clusters, a sample that is closest to the centroid will be chosen as the representative. These 10000 representative sound samples will then be assigned unique codes (the cluster numbers have been used as the codes). This collection of representative sounds and their codes will be the repository, using which other sound samples can now be encoded. Both the encoder and decoder will use a repository of speech segments. The repository may be transported by CDs, or may be made available for download, etc.

Purpose

To create a repository that represents the phonetically balanced characteristics of the particular user.

Inputs

A speech file which has been recorded in the .wav format of at least 20 min duration. Speech should be Mono and of uncompressed format.

Input should be sampled at 11025 Hz with 8 bits per sample (Microsoft standard for telephone quality speech).

Outputs

A speech repository (frame files characterizing the speech features of the user) that automatically gets created in the user's system. This repository should then be made publicly available by the creator.

Repository size is around 2 MB for each user. Every repository consists of empirically decided (10000) representative frames and the codebook which associates the frames with their corresponding parameters.

Repository generator stores the repository in a directory named as per the user's email id. Error messages have been handled by standard c++ handling mechanisms such as try, throw, catch etc.

2. Encoding

A new 10-second sample will be taken and divided into 20 ms slices. MFCC features will be extracted from the 500 sound-slices created this way. Each of these feature vectors will be taken and a closest match will be found from the 10000 feature vectors of the representative samples of the profile. This will be done by determining the minimum Euclidean distance in the 12 dimensional feature space.

Thus, for each of the 500 sound-slices, a representative sound from the profile will be identified. The encoded file will consist of this sequence of codes of the representative sound samples. The sender will record his voice message and transform it into the coded, compressed file using the encoder module. The coded file will be transferred as an email attachment.

Purpose

The encoder tool will take a sound file and convert it into a compressed binary file, using the repository.

Inputs

A speech file which has been recorded in the .wav format. Speech should be Mono and of uncompressed format. Input should be sampled at 11025 Hz with 8 bits per sample (Microsoft standard for telephone quality speech).

Outputs

The code file that has to be transmitted over the internet to the receiver.

For an input file of 10 sec duration, an output file (code file) of around 2.2 KB will be generated. This codefile will also contain the user's email id for identification purposes.

Error messages have been handled by standard c++ handling mechanisms such as try, throw, catch etc.

3. Decoding

The decoding will be done using the encoded file and the repository (i.e. 10000 representative sound-slices). The resultant audio will be created by successively concatenating the representative sound samples indicated in the encoded file. Smoothing will improve the quality of the resulting decoded sample. The receiver will pass the attached file through the decoder module, which will reproduce the original speech.

Purpose

The decoder tool will take a code file and convert it into a decoded speech file formed by concatenating representative frames from the repository.

Inputs

An encoded speech file (which has been encoded using this software itself)

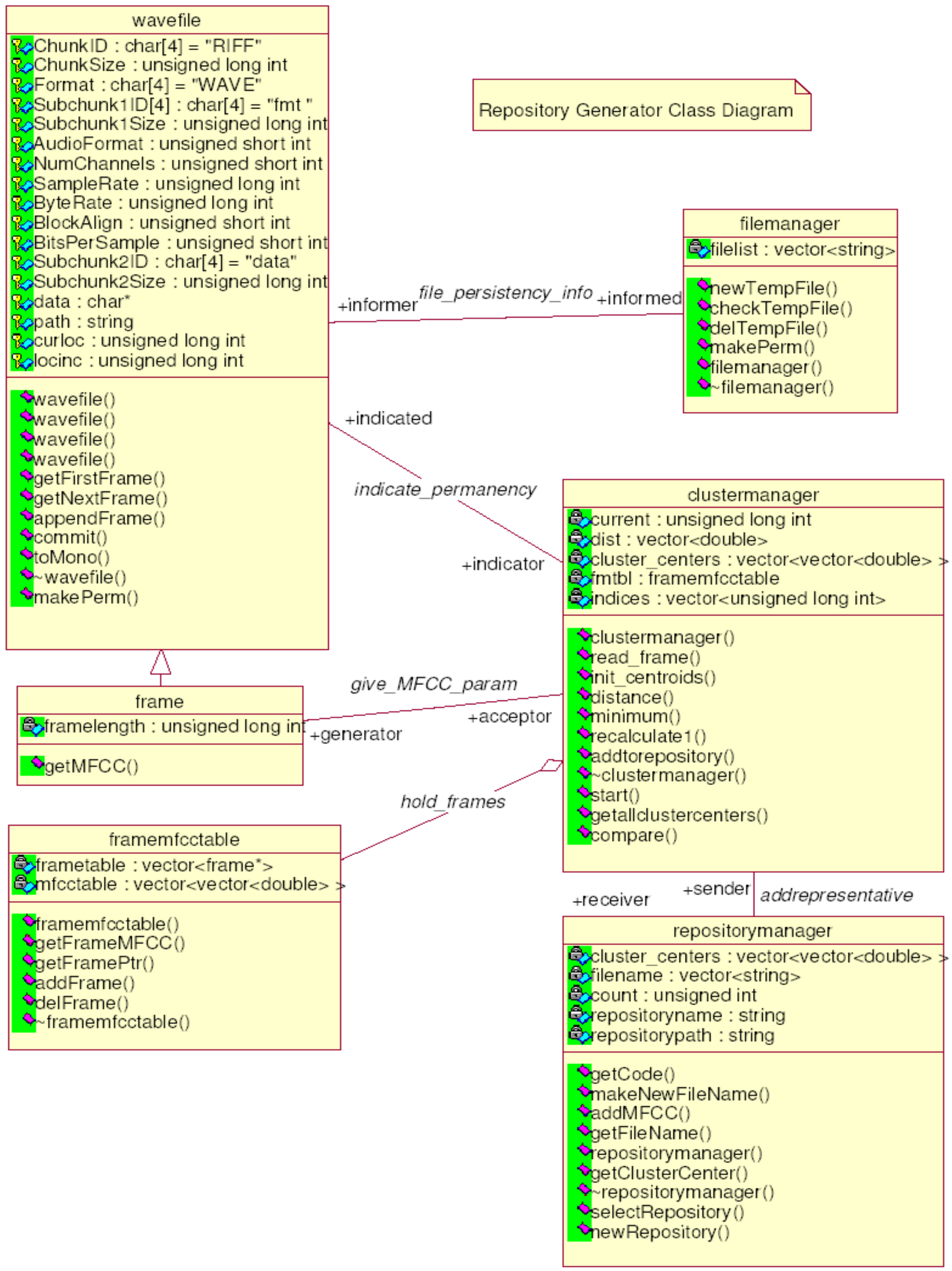
Code file should have been created by using the repository that is present at the decoder end. i.e. the user should possess the repository of the sender. If not available he can get it.

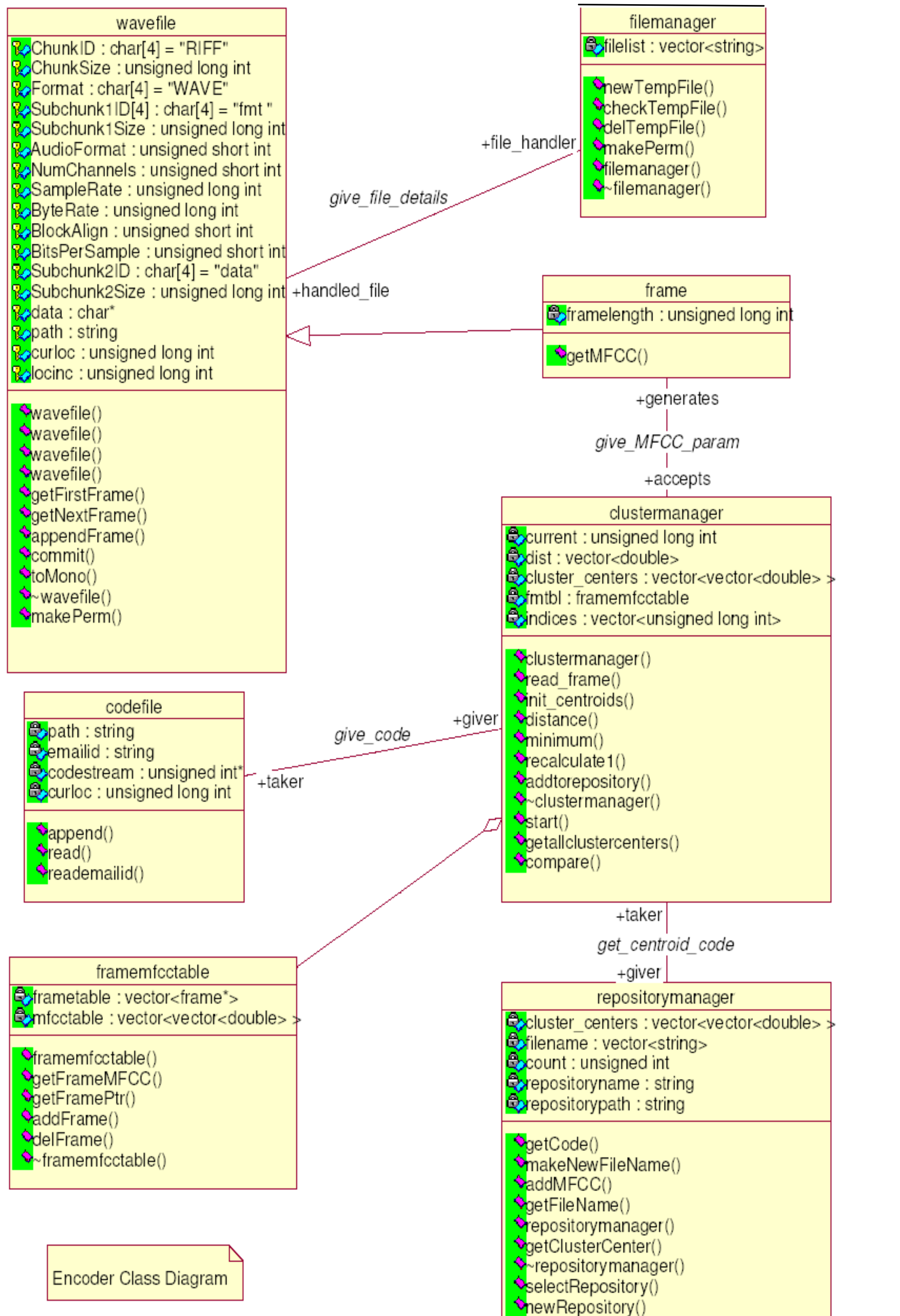
Outputs

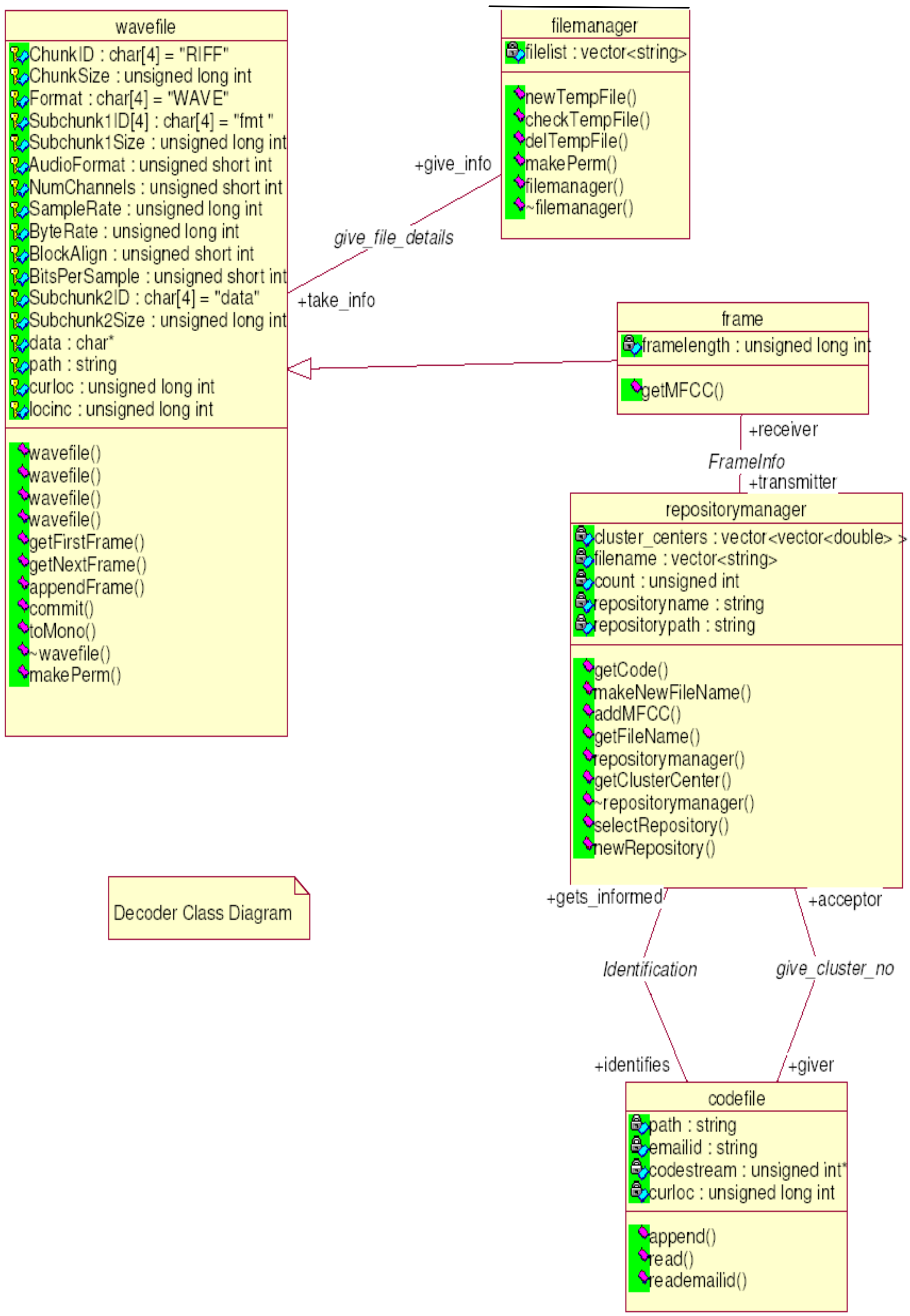
A .wav file that the user can listen to.

Error messages have been handled by standard c++ handling mechanisms such as try, throw, catch etc.

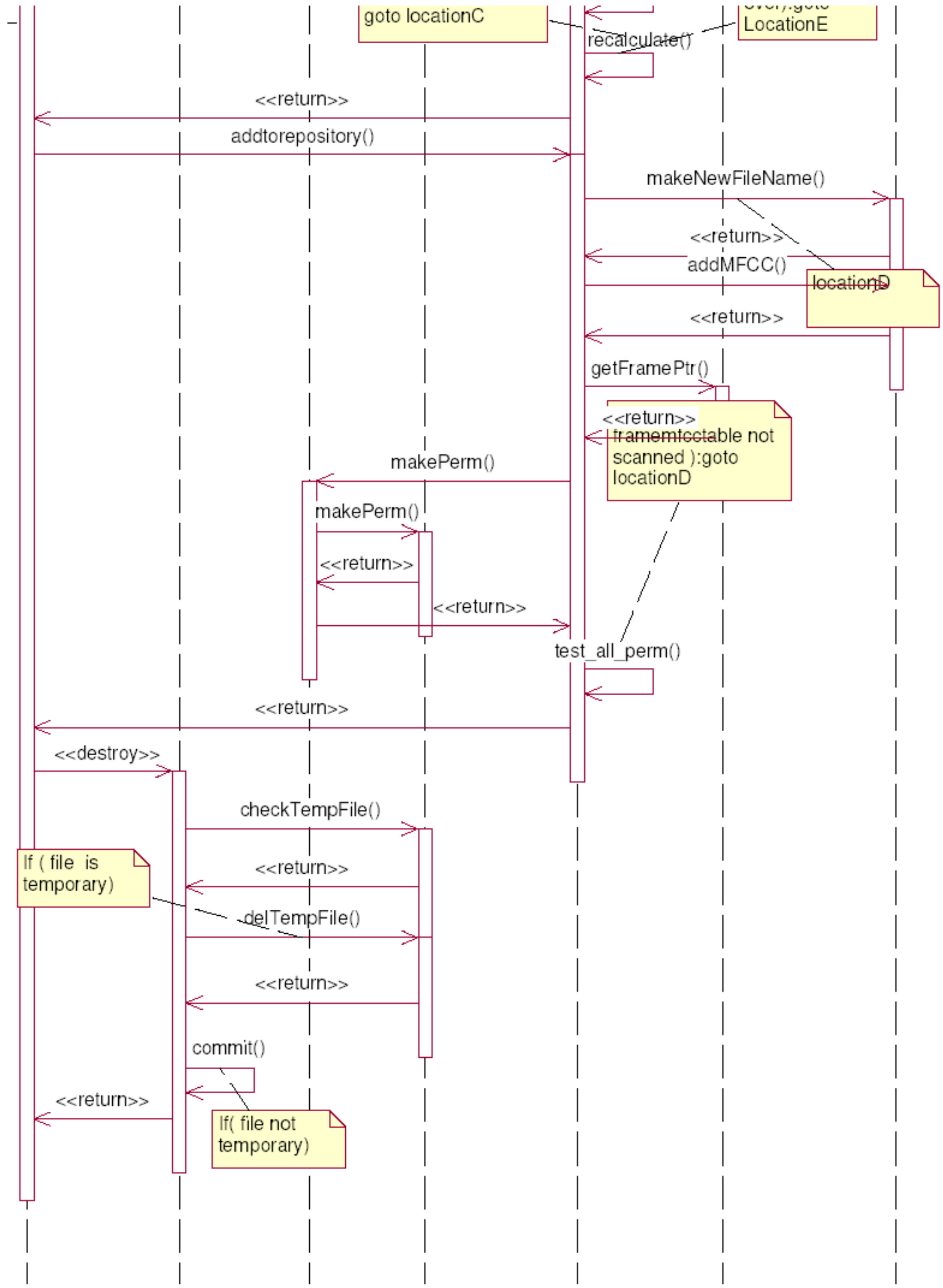
4.1. CLASS DIAGRAMS

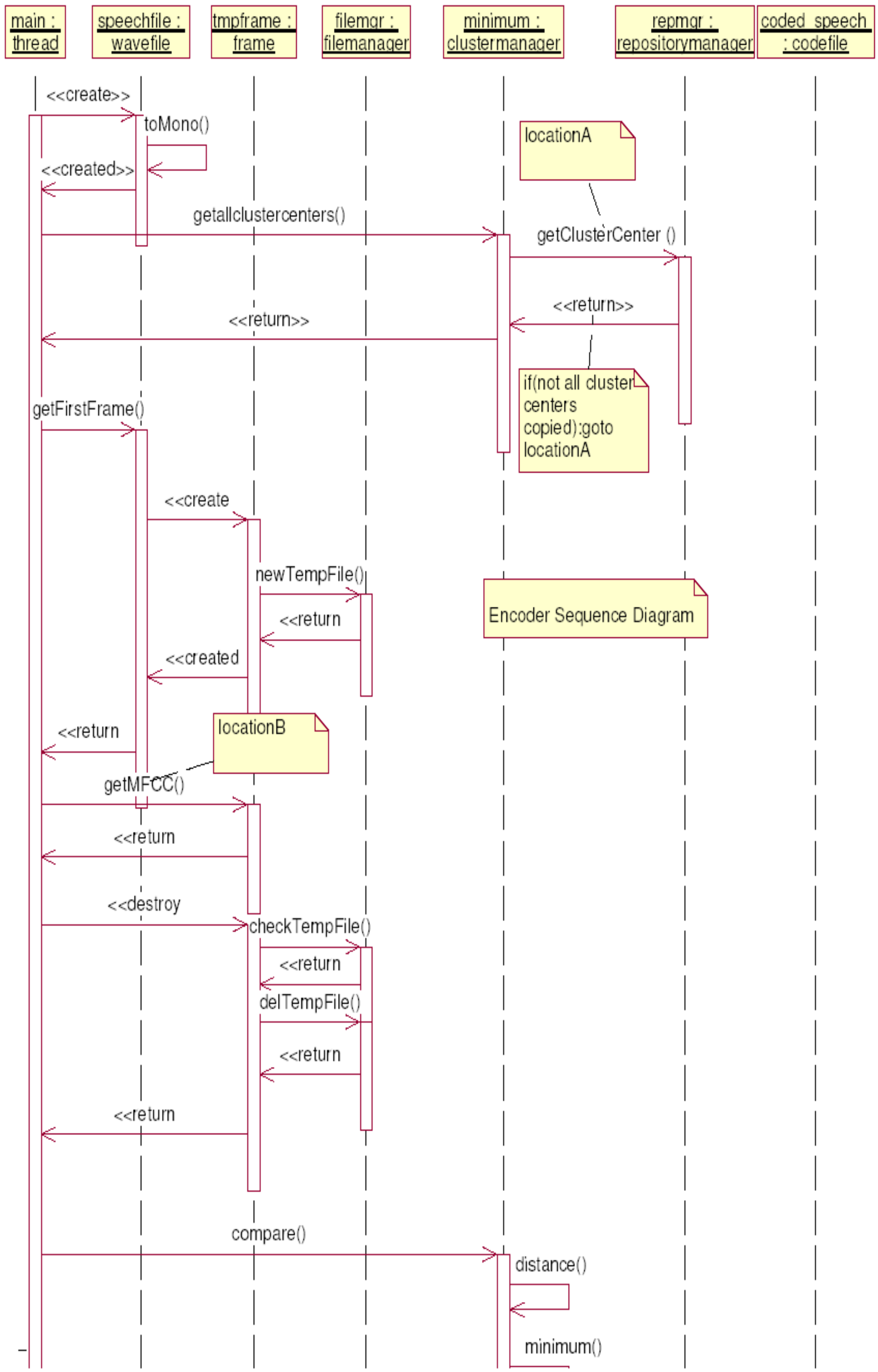


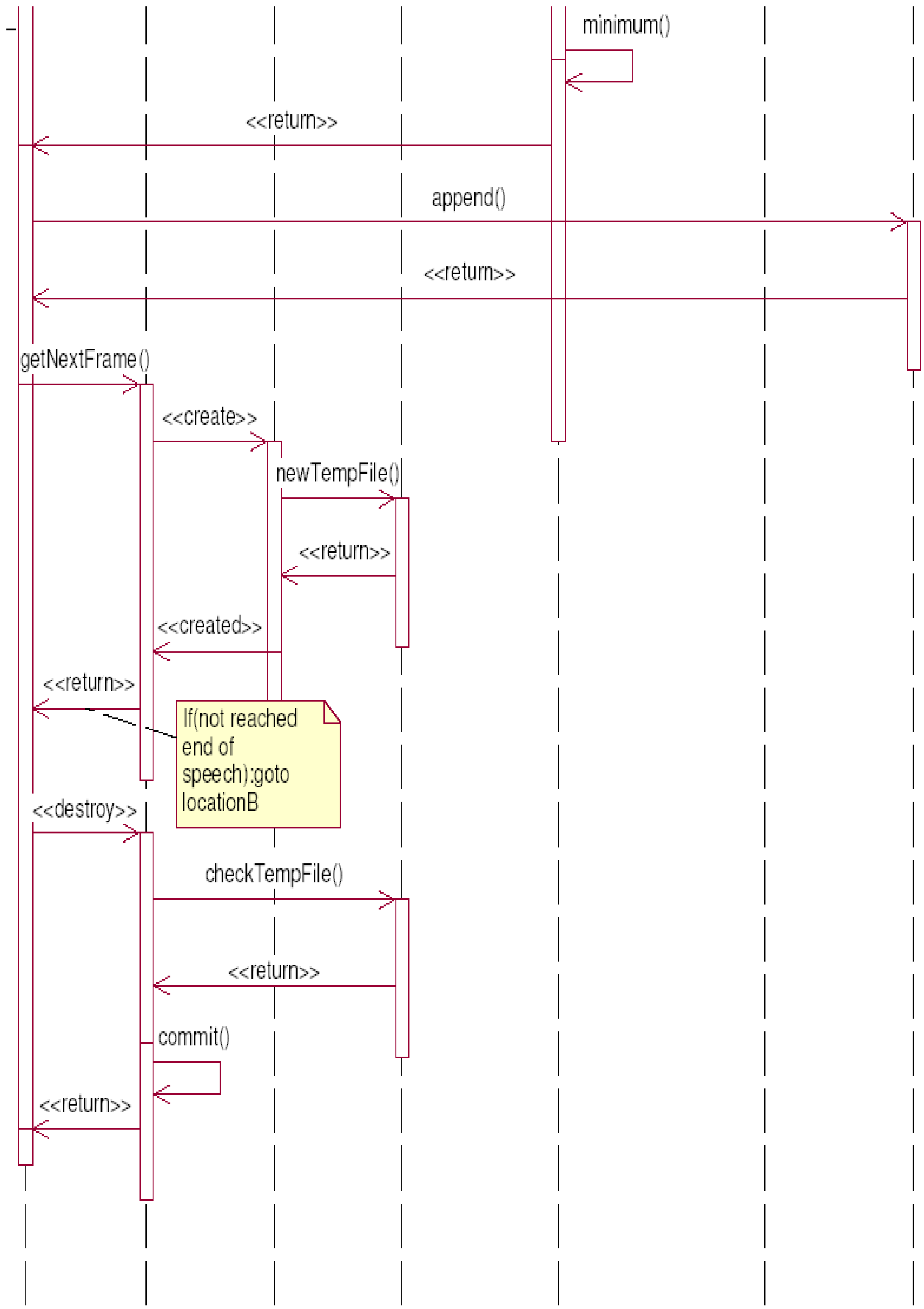

wavefile
ChunkID : char[4] = "RIFF"
ChunkSize : unsigned long int
Format : char[4] = "WAVE"
Subchunk1ID[4] : char[4] = "fmt "
Subchunk1Size : unsigned long int
AudioFormat : unsigned short int
NumChannels : unsigned short int
SampleRate : unsigned long int
ByteRate : unsigned long int
BlockAlign : unsigned short int
BitsPerSample : unsigned short int
Subchunk2ID : char[4] = "data"
Subchunk2Size : unsigned long int
data : char*
path : string
curloc : unsigned long int
locinc : unsigned long int
wavefile()
wavefile()
wavefile()
wavefile()
getFirstFrame()
getNextFrame()
appendFrame()
commit()
toMono()
~wavefile()
makePerm()
filemanager
filelist : vector<string>
newTempFile()
checkTempFile()
delTempFile()
makePerm()
filemanager()
~filemanager()
+file_handler
give_file_details
+handled_file
frame
framelength : unsigned long int
getMFCC()
+generates
give_MFCC_param
+accepts
clustermanager
current : unsigned long int
dist : vector<double>
cluster_centers : vector<vector<double> >
fmtbl : framemfcctable
indices : vector<unsigned long int>
clustermanager()
read_frame()
init_centroids()
distance()
minimum()
recalculate1()
addtorepository()
~clustermanager()
start()
getallclustercenters()
compare()
codefile
path : string
emailid : string
codestream : unsigned int*
curloc : unsigned long int
append()
read()
reademailid()
give_code
+giver
+taker
+taker
get_centroid_code
+giver
framemfcctable
frametable : vector<frame*>
mfcctable : vector<vector<double> >
framemfcctable()
getFrameMFCC()
getFramePtr()
addFrame()
delFrame()
~framemfcctable()
repositorymanager
cluster_centers : vector<vector<double> >
filename : vector<string>
count : unsigned int
repositoryname : string
repositorypath : string
getCode()
makeNewFileName()
addMFCC()
getFileName()
repositorymanager()
getClusterCenter()
~repositorymanager()
selectRepository()
newRepository()
Encoder Class Diagram

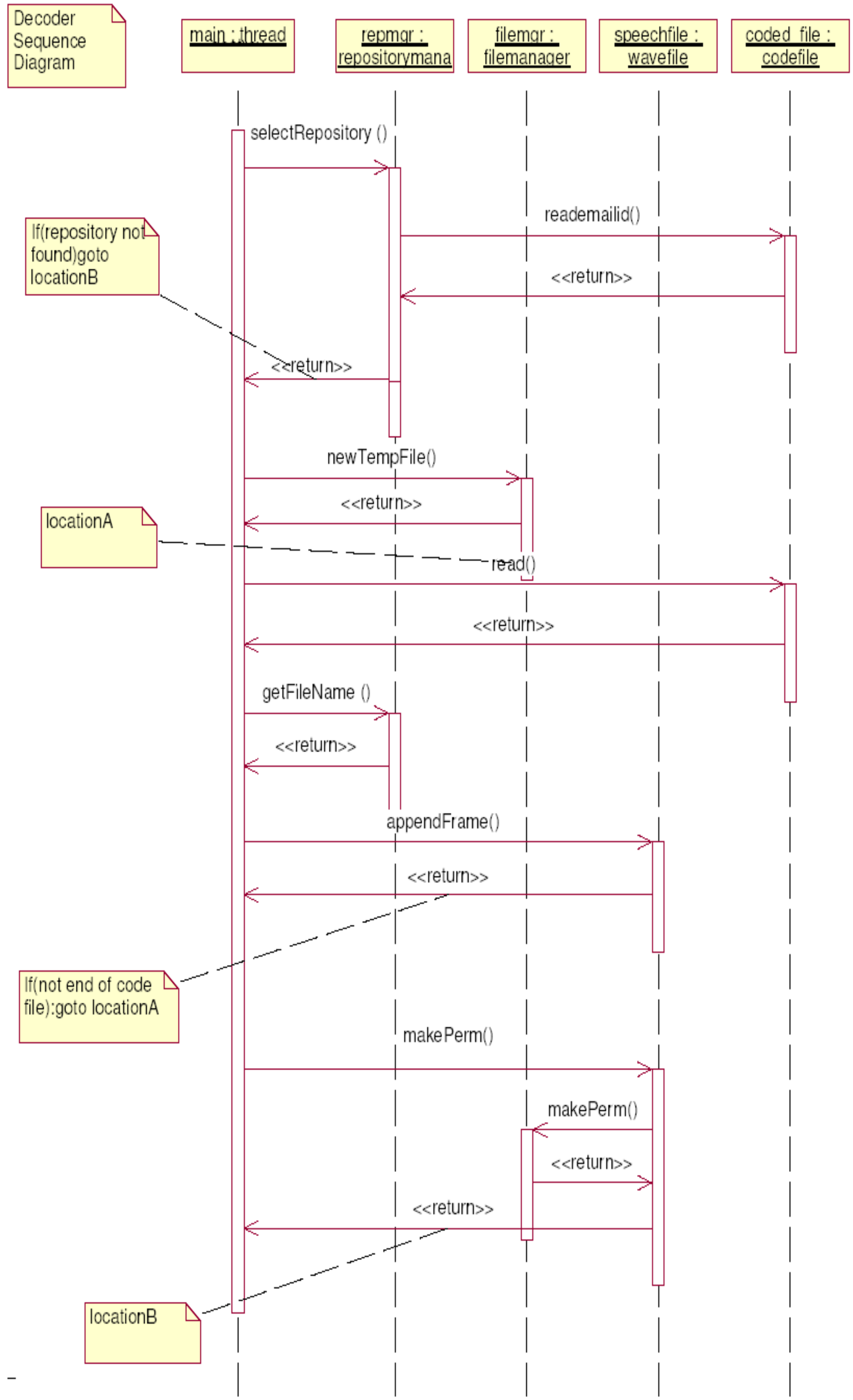

wavefile
ChunkID : char[4] = "RIFF"
ChunkSize : unsigned long int
Format : char[4] = "WAVE"
Subchunk1ID[4] : char[4] = "fmt "
Subchunk1Size : unsigned long int
AudioFormat : unsigned short int
NumChannels : unsigned short int
SampleRate : unsigned long int
ByteRate : unsigned long int
BlockAlign : unsigned short int
BitsPerSample : unsigned short int
Subchunk2ID : char[4] = "data"
Subchunk2Size : unsigned long int
data : char*
path : string
curloc : unsigned long int
locinc : unsigned long int
wavefile()
wavefile()
wavefile()
wavefile()
getFirstFrame()
getNextFrame()
appendFrame()
commit()
toMono()
~wavefile()
makePerm()
filemanager
filelist : vector<string>
newTempFile()
checkTempFile()
delTempFile()
makePerm()
filemanager()
~filemanager()
+give_info
give_file_details
+take_info
frame
framelength : unsigned long int
getMFCC()
+receiver
FrameInfo
+transmitter
repositorymanager
cluster_centers : vector<vector<double> >
filename : vector<string>
count : unsigned int
repositoryname : string
repositorypath : string
getCode()
makeNewFileName()
addMFCC()
getFileName()
repositorymanager()
getClusterCenter()
~repositorymanager()
selectRepository()
newRepository()
Decoder Class Diagram
+gets_informed
+acceptor
Identification
give_cluster_no
+identifies
+giver
codefile
path : string
emailid : string
codestream : unsigned int*
curloc : unsigned long int
append()
read()
reademailid()

4.2. SEQUENCE DIAGRAMS




main : thread
speechfile : wavefile
tmpframe : frame
filemgr : filemanager
minimum : clustermanager
repmgr : repositorymanager
coded_speech : codefile
<<create>>
toMono()
<<created>>
locationA
getallclustercenters()
getClusterCenter ()
<<return>>
<<return>>
if(not all cluster centers copied):goto locationA
getFirstFrame()
<<create
newTempFile()
<<return
Encoder Sequence Diagram
<<created
<<return
locationB
getMFCC()
<<return
<<destroy
checkTempFile()
<<return
delTempFile()
<<return
<<return
compare()
distance()
minimum()


minimum()
<<return>>
append()
<<return>>
getNextFrame()
<<create>>
newTempFile()
<<return>>
<<created>>
<<return>>
If(not reached end of speech):goto locationB
<<destroy>>
checkTempFile()
<<return>>
commit()
<<return>>


Decoder Sequence Diagram
main : thread
repmgr : repositorymana
filemgr : filemanager
speechfile : wavefile
coded_file : codefile
selectRepository ()
reademailid()
<<return>>
If(repository not found)goto locationB
<<return>>
newTempFile()
<<return>>
locationA
read()
<<return>>
getFileName ()
<<return>>
appendFrame()
<<return>>
If(not end of code file):goto locationA
makePerm()
makePerm()
<<return>>
<<return>>
locationB

5
IMPLEMENTATION DETAILS

Detailed description of components
The various components used in the modules, as shown above, are listed below, module-wise:

repositorygenerator.cpp

Identification	**Repository generator**
Type	A module
Purpose	To create a repository that represents the phonetically balanced characteristics of the particular user.
Function	Split into frames Find MFCC parameters Perform clustering Prepare repository
Subordinates	- Wavefile - File Manager - Frame - Cluster Manager - Frame-MFCC Table - Repository Manager
Dependencies	This phase should be done before encoding and decoding.
Interfaces	vox –r training_file emailid It interfaces with the Edinburgh speech tools siq2fv functionality to get the MFCC parameters. Input and output restrictions have been mentioned earlier.
Resources	Internet connection to publish repository Heavy Memory requirements , CPU requirements I/O channels cdwriters to publish repository, libraries, and system services
Processing	A recorded lecture will be obtained. All experiments will be conducted using this sample (sampling rate: 8000 Hz, single channel and 16-bits/sample.). A feature balanced sample of duration of a few minutes will be utilized for repository generation. This file will be divided into a number of files of FRAME_LENGTH duration each. MAX_DIM number of MFCC features (Mel-frequency cepstral coefficients) will be computed for each of these *sound-slices*. MFCC features are perception-based features, which are widely used in the speech recognition arena. It is assumed that 10000(NO_OF_CLUSTERS) different *elementary sounds* will be enough to characterize the range of sounds produced by a person. This number will be arrived at empirically. The sound samples will then be clustered into NO_OF_CLUSTERS clusters based on their MFCC features.

	A variant of the k-means algorithm will be used for clustering. For each of these clusters, a sample that is closest to the centroid will be chosen as the representative. These NO_OF_CLUSTERS representative sound samples will then be assigned unique codes (the cluster numbers have been used as the codes). This collection of representative sounds and their codes will be the repository, using which other sound samples can now be encoded. Both the encoder and decoder will use a repository of speech segments. The repository may be transported by CDs,or may be made available for download, etc.
Data	Repository containing the frames and a codebook.

encoder.cpp

Identification	**Encoder**
Type	A module
Purpose	The encoder tool will take a sound file and convert it into a compressed binary file, using the repository
Function	Split into frames Find MFCC parameters Perform vector quantization Prepare encoded message.
Subordinates	- Wavefile - File Manager - Frame - Cluster Manager - Frame-MFCC Table - Repository Manager - Code File
Dependencies	This phase should be done before decoding and after the repository for the concerned person has been generated.
Interfaces	vox –e speech_file emailid output_file It interfaces with the Edinburgh speech tools sig2fv functionality to get the MFCC parameters. Input and output restrictions have been mentioned earlier.
Resources	Internet connection to send encoded file via email. Heavy Memory requirements ,

	CPU requirements I/O channels libraries, and system services
Processing	A small message file to be encoded will be taken and divided into slices each of size given by FRAME_LENGTH. MFCC features will be extracted from the sound-slices created this way. Each of these feature vectors will be taken and a closest match will be found from the NO_OF_CLUSTERS feature vectors of the representative samples of the profile. This will be done by determining the minimum Euclidean distance in the MAX_DIM dimensional feature space. Thus, for each of the sound-slices, a representative sound from the profile will be identified. The encoded file will consist of this sequence of codes of the representative sound samples. The sender will record his voice message and transform it into the coded, compressed file using the encoder module. The coded file will be transferred as an email attachment.
Data	Encoded file to be transmitted as the message.

decoder.cpp

Identification	**Decoder**
Type	A module
Purpose	The decoder tool will take a code file and convert it into a decoded speech file formed by concatenating representative frames from the repository.
Function	Receive the encoded file Get the individual codes Select the repository Get the frames Concatenate them Apply smoothening algos.
Subordinates	- Wavefile - File Manager - Frame - Repository Manager
Dependencies	This phase should be done after the repository generation and the encoder phases.
Interfaces	vox –d encoded_file decoded_file

	Input and output restrictions have been mentioned earlier.
Resources	Internet connection to receive encoded file via email. Memory requirements , CPU requirements I/O channels and system services
Processing	The decoding will be done using the encoded file and the repository (i.e. NO_OF_CLUSTERS representative sound-slices). The resultant audio will be created by successively concatenating the representative sound samples indicated in the encoded file. Smoothing will improve the quality of the resulting decoded sample. The receiver will pass the attached file through the **decoder module**, which will reproduce the original speech.
Data	The encoded message has to be stored. Finally the decoded message file is obtained.

clustermanager.cpp

This class is responsible for identifying representative frames corresponding to the cluster centers obtained by performing k-means clustering on the training data set or on the message file.

Data members:

Visibility	Datatype	Variable name	Description
private	long int	Current	current cluster number being processed
private	vector<double>	Dist	distance of each cluster center from the current data point
private	vector<double>	Centroid	MFCC parameters of a particular cluster center.
private	vector<vector<double> >	cluster_centers	centers of the clusters
private	vector<unsigned long int>	Indices	indices of frames (in mfcc table) to be added to the repository
private	vector<int>	Count	// count of members in each cluster currently
Public	framemfcctable	Fmtbl	Stores the mfcc values for all the frames

Member Functions

Visibility	Return type	Name	Parameters	Description
Public	-	clustermanager	Void	constuctor for the clustermanager class
Public	void	showcenters	Void	Display all the cluster centers
Public	int	initcentroids	int iter	Initializes cluster centroids by randomly selecting tuples from the mfcc table
Public	int	Start	Void	Initiates clustering algo
Public	int	Distance	Void	calculates the distance between the current data point taken from mfcc table and the cluster centroids.
Public	int	distance	vector<double > mfcc	calculates the distance between the current data point passed as parameter and the cluster centroids.
Public	int	minimum	Void	Finds the minimum distance of current frame from all other cluster centroids
Public	int	recalculate1	int min	Recalculates the new cluster centroid after the current frame has been added to the cluster
Public	vector<unsigned long int>	getIndices	Void	Gets indices of the representative cluster centroids' mfcc parameters from mfcc table
Public	vector<vector<double> >	getcentroids	Void	Gets mfcc values of the representative cluster centroids
Public	int	getallclustercenters	string email	Gets the cluster centers from the codebook which is being managed by repositorymanager
Public	unsigned int	compare	vector<double > mfcc	Combines the functionality of distance() and minimum() to find representative for the frame passed as the parameter

wavefile.cpp

This class is responsible for representing the wavefile and performing operations related to it like creation,getting MFCC parameters,breaking wavefile into frames,making wavefile from constituent frames.

Visibility	Datatype	Variable name	Description
Protected	Char[4]	ChunkID	Contains the letters "RIFF" in ASCII form(0x52494646 big-endian form)
Protected	unsigned long int	ChunkSize	36 + SubChunk2Size, or more precisely: 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size) This is the size of the rest of the chunk following this number. This is the size of the entire file in bytes minus 8 bytes for the two fields not included in this count:

			ChunkID and ChunkSize
Protected	Char[4]	Format	Contains the letters "WAVE" (0x57415645 big-endian form)
Protected	Char[4]	Subchunk1ID	Contains the letters "fmt " (0x666d7420 big-endian form)
Protected	unsigned long int	Subchunk1Size	16 for PCM. This is the size of the rest of the Subchunk which follows this number
Protected	unsigned short int	AudioFormat	PCM = 1 (i.e. Linear quantization) Values other than 1 indicate some form of compression
Protected	unsigned short int	NumChannels	Mono = 1, Stereo = 2
Protected	unsigned long int	SampleRate	8000, 44100, etc.
Protected	unsigned long int	ByteRate	== SampleRate * NumChannels * BitsPerSample/8
Protected	unsigned short int	BlockAlign	== NumChannels * BitsPerSample/8 The number of bytes for one sample including all channels.
Protected	unsigned short int	BitsPerSample	8 bits = 8, 16 bits = 16, etc.
Protected	Char[4]	Subchunk2ID	Contains the letters "data" (0x64617461 big-endian form)
Protected	unsigned long int	Subchunk2Size	== NumSamples * NumChannels * BitsPerSample/8 This is the number of bytes in the data. You can also think of this as the size of the read of the subchunk following this number
Protected	char *	Data	The actual sound data.
Protected	String	Path	Location of open wavefile
Protected	unsigned long int	locinc	Size of Each Subchunk2Size of each frame
Protected	unsigned long int	Curloc	Current frame start location
Protected	FILE*	Fptr	Associated with mfcc.fil

Member Functions

Visibility	Return type	Name	Parameters	Description
Public	-	Wavefile	void	Constructor: Implicit constructor used to create wavefile with header but no data
Public	unsigned long int	getlocinc	void	Gets the location increment
Public	-	Wavefile	char* frdata,unsigned long int frsize	Constructor: Used to create wavefile using the data passed as parameter
Public	-	wavefile	string wavepath	Constructor: opens the file specified by the path and initialises all private variables, allocates buffer for data, and copy data
Public	-	wavefile	wavefile& wv	Copy Constructor: copy all private variables except for the path, reallocates buffer for data, and copy data
Public	-	wavefile	const wavefile& wv	Copy Constructor: copy all private variables except for the path, reallocates buffer for data, and copy data
Public	int	makeMono	int type	Converts this wavefile to monochannel, if it is multichannel type=1 => Sum; type = 2 => Avg; ret = -1 => clip
Public	int	makePerm	string dest	makes a wave file permanent
Public	int	getFirstFrame	char *frmdata,unsigned long int* frsize	Copies first frame of locinc samples (if needed, padding is done) in wf and returns the length of frame
Public	int	getNextFrame	char *frmdata,unsigned long int* frsize	Copies next frame of locinc samples (if needed, padding is done) in wf and returns the length of frame
Public	int	getFrame	unsigned long int i, string framename	Copies i^{th} frame of locinc samples (if needed, padding is done) in wf and returns the length of frame
Public	vector<double>	getMFCC	int *status	Gets all the parameters one by one from fptr
Public	int	appendFrame	string fpath	appends a frame to this wavefile without smoothing
Public	int	getData	char* frmdata,unsigned long int* frsize	Gets data for the current frame or wavefile with the length indicated by frsize
Public	int	commit	void	copies all private variables & data back to the location specified by path
Public	int	showDetails	void	Displays the header information of wavefile
Public	-	~wavefile	void	Destructor: closes the file specified by the path and copies all private variables & data back, deallocates buffer for data, and try to delete the temporary file
Public	unsigned long int	nFrames	void	Returns Subchunk2Size/getlocinc()

filemanager.cpp

This class is responsible for handling the various frame file operations such as creating frame files with a unique name, deleting temporary ones and saving the permanent ones as the repository.

Visibility	Datatype	Variable name	Description
Protected	vector<string>	filelist	List of all the temporary files that have been created

Member Functions

Visibility	Return type	Name	Parameters	Description
Public	-	filemanager	void	Constructor: Implicit constructor
Public	-	~filemanager	void	Destructor: Deletes all the temporary files
Public	string	newTempFile	void	Provides a new unique name to the frame file
Public	int	checkTempFile	string path	Checks whether the file is temporary.Returns 1 if temporary else 0
Public	int	delTempFile	string path	Deletes the file if it is found to be temporary
Public	int	makePerm	string dest, string src	Makes the file permanent by renaming it if it is found to be non-temporary

frame.cpp

This class is responsible for representing the frames and performing operations as performed by the wavefile class.
This class publicly inherits from the wavefile class.

Data members

Inherited from the wavefile class.

Member functions

Except for the constructors, it inherits all the functionality of the wavefile class. Other member functions are as follows

Visibility	Return type	Name	Parameters	Description
Public	-	frame	void	Constructor: Implicit constructor.Calls wavefile()
Public	-	frame	char* frdata,unsigned long int frsize	Constructor: Calls wavefile(frdata,frsize)
Public	-	Frame	string wavepath	Constructor: Calls wavefile(wavepath)
Public	-	Frame	wavefile& wv	Copy Constructor: Calls wavefile(wv)
Public	-	Frame	const wavefile& wv	Copy Constructor: Calls wavefile(wv)
Public	-	Frame	frame& wv	Copy Constructor: Calls wavefile(wv)
public	-	~frame	Void	Destructor

framemfcctable.cpp

This class is responsible for populating and retrieving mfcc parameters from the mfcc table for current frame.

Data members

Visibility	Datatype	Variable name	Description
private	vector<vector<double > >	mfcctable	Stores the 12 mfcc parameters for each frame

Member functions

Visibility	Return type	Name	Parameters	Description
Public	-	framemfcctable	void	Constructor: Implicit constructor
Public	int	addFrame	vector<double > mfcc	Adds the mfcc parameters for the current frame into the mfcc table
Public	vector<double>	getFrameMFCC	int i,int *status	Gets the mfcc parameters for the current frame from the mfcc table
Public	unsigned long int	nFrames	wavefile& wv	Returns the size of the mfcc table

codefile.cpp

This class represents the codefile that is the output of encoder and used as an input to the decoder. It is responsible for holding the emailid and codes and for the operations on these data members.

Data members:

Visibility	Datatype	Variable name	Description
private	String	Emailed	uniquely identifies repository and also name of the repository directory
private	string	Path	system path of the directory under which

			all the repositories are stored
private	vector<unsigned int>	Codestream	buffer to be emptied into the codefile
private	unsigned long int	curloc	current location inside the codestream

Member Functions

Visibility	Return type	Name	Parameters	Description
Public	-	codefile	string cf_path ,string email_id,unsigned long int size	constructor to be called by the encoder module
Public	-	codefile	string cf_path	constructor to be called by the decoder module
Public	int	append	unsigned int code	Appends the code specified as parameter to this codefile
Public	unsigned int	read	void	Gets all the codes in this codefile into the codestream.
Public	unsigned int	getcode	Void	Returns the next code from the codesream. Returns END_OF_CODESTREAM when reached end of codestream
Public	int	distance	vector<double > mfcc	calculates the distance between the current data point passed as parameter and the cluster centroids
Public	string	reademailid	Void	Returns the emailid's value embedded in this codefile
Public	-	~codefile	void	Destructor

repositorymanager.cpp

This class is used to manage a single repository that is the output of the repository generator, and used by both the encoder and decoder. The repository is identified by the emailid ad is stored as a directory containing a codebook and representative frames.

Data members:

Visibility	Datatype	Variable name	Description
private	String	Emailed	uniquely identifies repository and also name of the repository directory
private	string	Path	system path of the directory under which all the repositories are stored

Member Functions

Visibility	Return type	Name	Parameters	Description
Public	-	repositorymanager	void	Implicit constructor
Public	-	repositorymanager	string email_id, int create=NOCREATE	Creates a repository with the name as specified by the parameter,email_id when used in repository generation. Used to access the repository in the encoder and the decoder phases.
Public	string	makeNewFileName	int i	Generates a new file name as specified by the email_id and integer i
Public	vector<double>	getClusterCenter	unsigned int i	Gets all the cluster centroids for this repository
Public	int	addMFCC	vector<double> mfcc	Insert the mfcc parameters of the cluster center in the codebook
Public	string	getFrameName	unsigned int code	Gets the filename for the specified code
Public	-	~repositorymanager	void	Destructor

vox.cpp

NAME
    vox : Voice eXchange

SYNOPSIS

vox options filename module_specific_options

options:

-r	repository generation
-e	encoding
-d	decoding

filename: path of the input file

module_specific_options:
If option=="-r" then emailid
If option=="-e" then output_filename
If option=="-d" then output_filename

DESCRIPTION:
A system for exchanging voice messages over mail, using very high speech compression. The sender can record his voice message and transform it into the coded, compressed file using the encoder module. The coded file can be transferred as an email attachment. The receiver may then pass the attached file through the decoder module, which reproduces the original speech. Both the encoder and decoder use a repository of speech segments generated using the repository generator module.

Parameter name	Typical value	Description
SUCCESS	1	Denotes successful completion of the routine
FAILURE	0	Denotes failure in the routine due to some error
END_OF_CODESTREAM	0xFFFFFFFF	Denotes the end of the code file
REP_PATH	"repositories/"	Path of the directory where the repositories are stored
CODEBOOK	"/rep_file.bin"	Name of the codefile
MAXPATH	256	Maximum size of the path
voxtemppath	"tmp"	Denotes the directory name where the temporary files are stored
FRAMELENGTH	0.02	Denotes the length of the frame in seconds
SAMPLERATE	8000	Denotes the sampling rate in samples per second
BPS	16	Denotes the number of the bits per sample
MAX_DIM	12	total number of dimensions involved
k	10000	number of clusters
VERY_HIGH_VALUE	99999.99999	Denotes a very high value
NO_OF_ITER	6	Number of iterations
CREATE	1	Denotes that a repository needs to be created
NOCREATE	0	Denotes that a repository need not be created as it already exists

6
TECHNOLOGIES USED

6.1. Linux

Here are some of the benefits and features that Linux provides over single-user operating systems (such as MS-DOS) and other versions of UNIX for the PC.

- Full multitasking and 32-bit support.
- GNU software support.
- The X Window System.
- TCP/IP networking support.
- Virtual memory and shared libraries.
- Audio & Multimedia.

6.2. STLs

Originally, the development of the STL (Standard Template Library) was started by Alexander Stepanow at HP in 1979. Later, he was joined by David Musser and Meng Lee. In 1994, STL was included into ANSI and ISO C++.

The STL provides general purpose utility classes which programmers can use in their applications and they even don't have to worry about allocating and freeing memory. These classes are array, link, stack, string, vector, iterator, map classes. And the STL provides general algorithms for sort, search, or reverse arrays or links. Besides these two things, the STL also provides some iterators and other options you can apply on these classes.

Features:

The STL's generic algorithms work on native C++ data structures such as strings and vectors. STL containers are very close to the efficiency of hand-coded, type-specific containers.

Advantages of the STL

- You don’t have to write your classes and algorithms. It saves your time.
- You don’t have to worry about allocating and freeing memory. That's a big problem when you create you own linked-list, queue or other classes.
- Reduces your code size because STL uses templates to develop these classes.
- You have to override your functions or classes to operate on different types of data while STL let you apply these classes on different kind of data.
- Easy to use and easy to learn.

6.3. Emacs

For programming on the CSE Unix system. Emacs features are as follows:

- source code coloring

- Automatic indentation
- Line numbers
- Split screen compilation
- Automatic line wrapping
- Automatic backups
- Free Windows version

6.4. C++ under LINUX

C++ is an "object oriented" programming language created by Bjarne Stroustrup and released in 1985. It implements "data abstraction" using a concept called "classes", along with other features to allow object-oriented programming. Parts of the C++ program are easily reusable and extensible; existing code is easily modifiable without actually having to change the code. C++ adds a concept called "operator overloading" not seen in the earlier OOP languages and it makes the creation of libraries much cleaner.
Overloading allows to declare a method with different parameters.

C++ maintains aspects of the C programming language, yet has features which simplify memory management. Additionally, some of the features of C++ allow low-level access to memory but also contain high level features.

C++ could be considered a superset of C. C programs will run in C++ compilers. C uses structured programming concepts and techniques while C++ uses object oriented programming and classes which focus on data.

C++ describes classes into header files, and body of methods into source files. By declaring instances of classes you can reuse set of variables and methods without having to define them again.

Memory management is unchanged. Classes inherit one from other and share their methods.

6.5. Makefiles

We need a file called a *makefile* to tell make what to do. Most often, the makefile tells make how to compile and link a program.

6.6. Edinburgh Speech Tools

The Edinburgh Speech Tools Library is library of general speech software, written at the Centre for Speech Technology Research at the University of Edinburgh.

The Edinburgh Speech Tools Library is written is C++ and provide a range of for common tasks found in speech processing. The library provides a set of stand alone executable programs and a set of library calls which can be linked into user programs.

sig2fv *Generate signal processing coefficients from waveforms*

sig2fv is used to create signal processing feature vector analysis on speech waveforms. The following types of analysis are provided:

- Linear prediction (LPC)
- Cepstrum coding from lpc coefficients
- Mel scale cepstrum coding via fbank
- Mel scale log filter bank analysis
- Line spectral frequencies
- Linear prediction reflection coefficients
- Root mean square energy
- Power

fundamental frequency (pitch)

6.7.Tk/tcl

Tool Command Language

The Tcl language and Tk graphical toolkit are simple and powerful building blocks for custom applications. The Tcl/Tk combination is increasingly popular because it lets you produce sophisticated graphical interfaces with a few easy commands, develop and change scripts quickly, and conveniently tie together existing utilities or programming libraries.

One of the attractive features of Tcl/Tk is the wide variety of commands, many offering a wealth of options. Most of the things you'd like to do have been anticipated by the language's creator, John Ousterhout, or one of the developers of Tcl/Tk's many powerful extensions. Thus, you'll find that a command or option probably exists to provide just what you need.

The tool command language Tcl (pronounced tickle) is an interpreted, action-oriented, string-based, command language. It was created by John Ousterhaut in the late 1980's along with the Tk graphical toolkit. Tcl and the Tk toolkit comprise one of the earliest scripted programming environments for the X Window System. Though it is venerable by today's standards, Tcl/Tk remains a handy tool for developers and administrators who want to rapidly build graphical frontends for command line utilities.

Tcl and Tk come bundled with most major Linux distributions and source-based releases are available from tcl.sourceforge.net. If Tcl and Tk are not installed on your system, the source releases are available from the SourceForge Tcl project: http://tcl.sourceforge.net/. Binary builds for most Linux distributions are available from rpmfind.net. A binary release is also available for Linux and other platforms from Active State at http://aspn.activestate.com/ASPN/Tcl

Tcl is built up from commands which act on data, and which accept a number of options which specify how each command is executed. Each command consists of the name of the command followed by one or more words separated by whitespace. Because Tcl is interpreted, it can be run interactively through its shell command, tclsh, or non-interactively as a script. When Tcl is run interactively, the system responds to each command that is entered as illustrated in the following example. You can experiment with tclsh by simply opening a terminal and entering the command tclsh.

Tcl's windowing shell, Wish, is an interpreter that reads commands from standard input or from file, and interprets them using the Tcl language, and builds graphical components from the Tk toolkit. Like the tclsh, it can be run interactively.

6.8. Pesq

PESQ stands for 'Perceptual Evaluation of Speech Quality' and is an enhanced perceptual quality measurement for voice quality in telecommunications. PESQ was specifically developed to be applicable to end-to-end voice quality testing under real network conditions, like VoIP, POTS, ISDN, GSM etc.

PESQ (Perceptual Evaluation of Speech Quality) is a method of determining the voice quality in the telecommunications networks. It combines the time-alignment technique from PAMS (Perceptual Analysis Measurement System) with the accurate perceptual modeling of PSQM (Perceptual Speech Quality Measurement), the best features of each technique. It is applicable not only to speech codecs but also to end-to-end measurement. Defined by ITU-T recommendation P.862 in February 2001, PESQ has become the most widely accepted standard for measuring voice quality over VoIP networks. However, the use of PESQ is not limited to VoIP. It can be used effectively to test, for example, voice over frame relay (VoFR), voice over ATM (VoATM), wireless systems, and cable modem and DSL systems that carry speech.
PESQ takes into account filtering in analog components, variable delay, and coding distortion. It measures one-way quality and is designed for use with intrusive tests.
Meaning of PESQ Values The PESQ score is mapped to a MOS-like scale, a single number in the range of -0.5 to 4.5, where values close to 4.5 indicate very good speech quality, and values close to -0.5 indicate very bad speech quality. For most cases, the output ranges between 1.0 and 4.5. PESQ score 2 and below corresponds to degradation level that is difficult to understand. Further mapping to MO values is the fairly straightforward process.

A system that assesses the quality of speech must allow for the transmission of different voices. The source can be real or artificial speech. Input from real speech should be based on ITU-T P.830 and it is recommended the use of minimum of two male and female speakers. Artificial speech is recommended only if it can represent the temporal and phonetic structure of real speech signals. Test signals should include speech bursts that are separated by silent periods, that represent of natural pauses in speech. The typical duration of a speech burst is 1-3 seconds. PESQ can also be used to assess the quality of systems carrying speech in the presence of background or environment noise.

7
TEST CASES

Test case 1

Training File Parameters

Training file size	Sampling rate	Sample size	Number of Channels	Compression type used
~ 15 Minutes	8000 Hz	16 bits	1	PCM

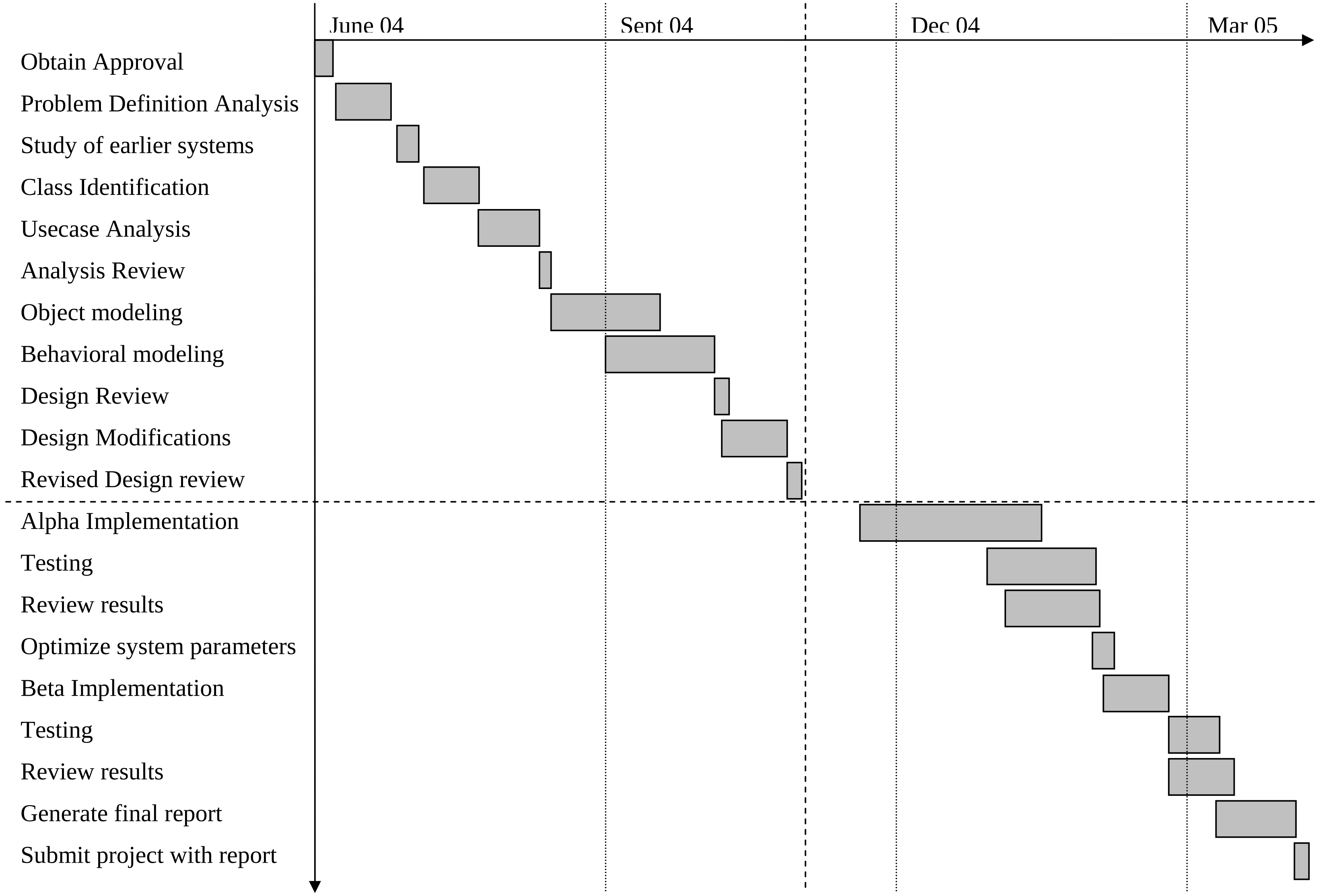
Repository Parameters

Repository Number	Number of Clusters	Frame length	MFCC features used	Number of Iterations	Size of repository obtained	Time required to generate repository
1	10000	20 milliseconds	0,1,2,3,4, 5,6,7,8,9, 10,11	6	~14 MB	~ 486 minutes
2	13000	20 milliseconds	0,1,2,3,4, 5,6,7,8,9, 10,11	6	~14 MB	~ 636 minutes

Message Parameters

Using Repository	Where is the message from	Length of message file	Length of coded file	PESQ
1	out repository	~ 1.9 MB	~ 24 KB	0.331
1	in repository	~ 250 KB	~ 4 KB	0.887
2	in repository	~ 250 KB	~ 4 KB	0.636

8
PROJECT TIMELINE



9
TASK DISTRIBUTION

Mumbai University recommends a group of 2-5 for the project work for the IV year BE projects. We formed a group of 3.

After understanding the project, we realized that it basically contains 3 modules from the statement of the problem. They were as follows:

1. Repository generator
2. Encoder
3. Decoder

On further analysis (this time aimed specifically at each module) we soon realized that all the modules depended on some basic classes of objects.

e.g. Wavefile class, a class to handle clustering, class to handle repository and code files, etc. So we sat together and decided on the different classes to be developed/reused and their interactions in various modules.

Then Apoorv started off with study and development of the wavefile class and its child class frame to handle various operations on .wav files. To handle multiple temporary frames, he also developed filemanager class. He was also instrumental in identifying the tools that can be used for MFCC generation.

Manish was handed the responsibility of handling the clustering algorithm (with the time and memory efficiency considerations) and vector quantization to be used and implemented as clustermanager class. He worked on the implementation of framemfcctable class, that is a part of clustermanager.

Sumeet was given the responsibility of handling the repositorymanager and codefile class which included considerations of how to represent the codefiles and the repository. He also put extra efforts for testing the program at his home and was instrumental in identification of someof the key parameters in system performance.

Finally, we decided to integrate our individual works to form 3 new classes to provide an abstraction interface between the user and these classes. Thus the combined effort led to development of repositorygenerator, encoder and decoder classes.

So as to create a complete command line-based tool we created the main file vox.cpp which presented the user with the desired module of the available three.

Finally to implement a GUI for our tool, we used Tk.

After having a working tool in our hand, we tested the system with different parameters which we had very cautiously isolated in parameters.cpp. We studied various test cases that were provided by our guide and those generated by us to improve the quality of the tool by deciding upon the appropriate parameter values

10
REFERENCES

- Ki-Seung Lee and Richard V. Cox, A very low bit rate speech coder based on a recognition/synthesis paradigm, *IEEE Transactions on Speech and Audio Processing, 2001*
- Suresh Balakrishna, Speech Recognition using Mel Cepstrum features, Mississippi State University, 1998
- http://www.it.iitb.ac.in/~chetanv
- http://www.speex.org/
- http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/kmeans.html
- http://www.festvox.org/
- http://www.sourceforge.org/
- http://www.opensource.org/
- http://www.psytechnics.com/downloads/2001-P02.pdf
- http://www.pesq.org/
- www.tcl.tk/

11
APPENDIX

User manual:

VoX is an acronym for Voice eXchange. VoX is a nifty command-line and GUI based tool that is used to encode speech files using a repository.

Sample passages can be used to generate a good training file. This will eventually affect the creation of repository. A good training file should be long and phonetically balanced. You may use open literature to generate the training file. Such literature is available at Project Gutenberg

Some of the sample commands for the command line are:

To create directory named vox in root directory.
$mkdir vox

To copy the compressed files vox.tar.gz to vox directory.
$cp vox.tar.gz vox

To change the directory.
$cd vox

To uncompress the compressed files.
$tar -zxvf vox.tar.gz

To run the make file of the vox tool.
$make

To view the man page of the vox tool.
$./vox

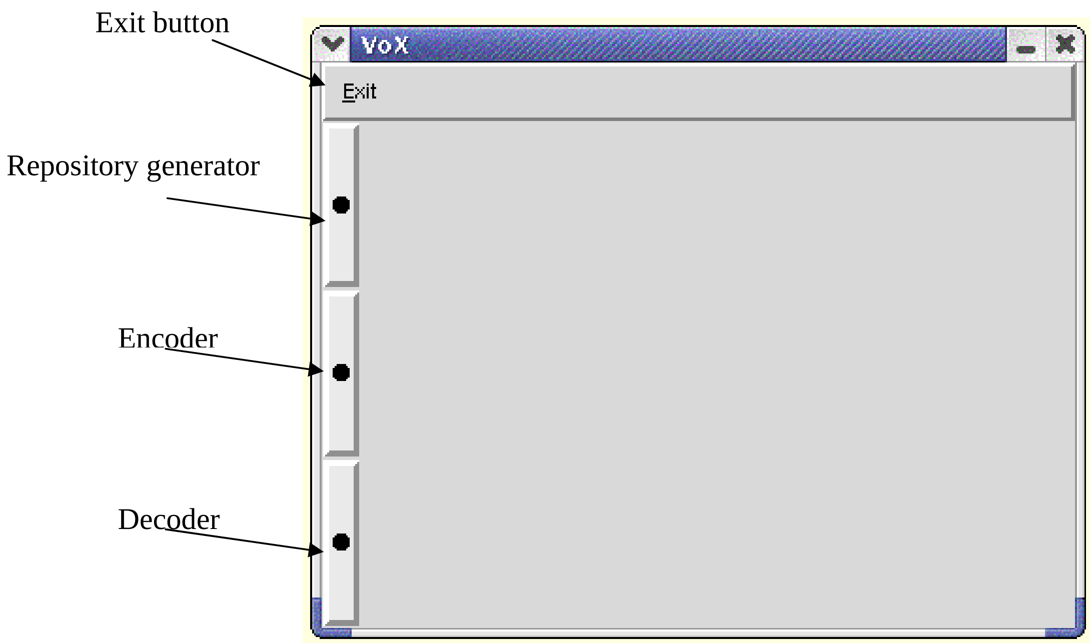
For repositorygenerator module :
$./vox -r yourbigspeechfile.wav youremailid@somehost.somedomain

For encoder module :
$./vox -e yourmessage.wav codedfile.bin youremailid@somehost.somedomain

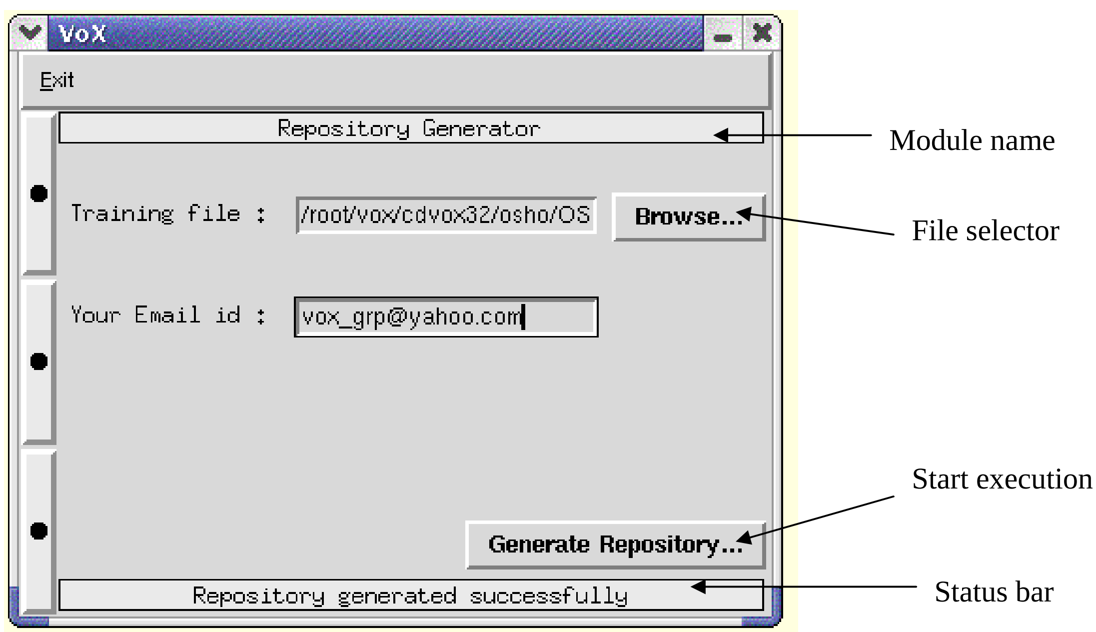
For decoder module:
$./vox -d codedfile.bin outputmessage.wav

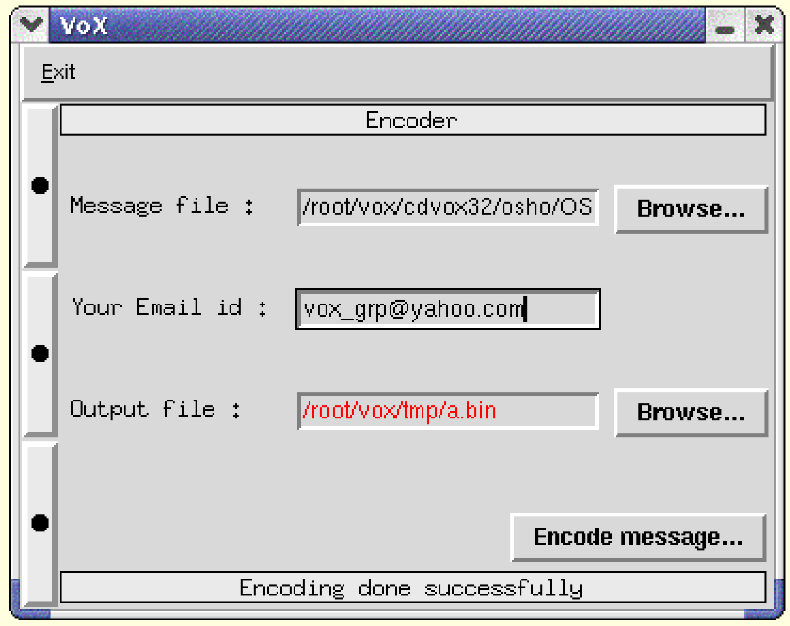
Graphical interface's screenshots are shown below:

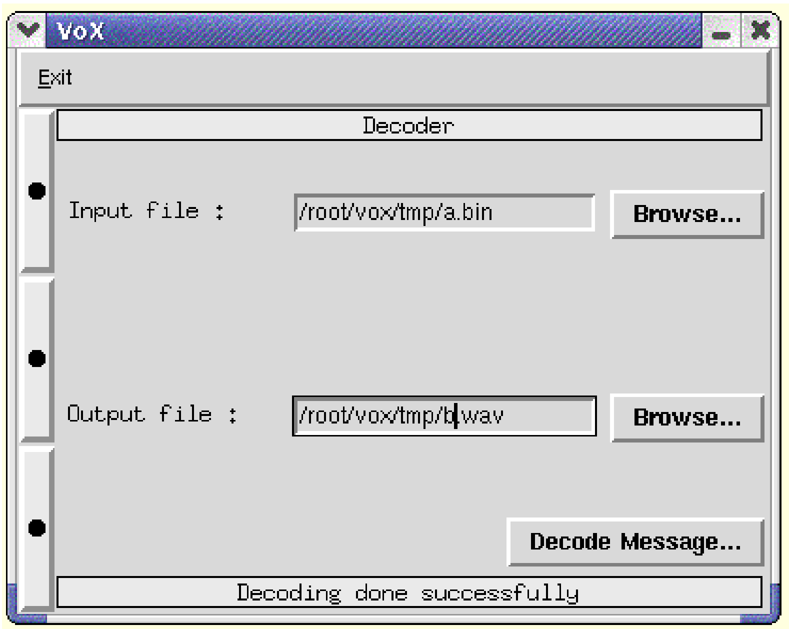
As you start you will see the following screen. Click one of the 3 buttons on the left hand side so as to start the desired module.



When you click the topmost button the following window opens up in which you need to enter the appropriate input as shown.







Some of the most frequently asked questions :

Q The program does not compile:
A Are all the source files together in a directory? If not, put them together and then try. Do you have the privilege to create or modify directories? If not, the program will not compile or will not run properly. Consult your root about this problem.

Q I am unable to run the program:

A The program may take a long time to finish. This is particularly true when you are creating a repository. It may even happen during encoding or decoding phase.

Q I get errors about MFCC stuff:
A Do you have sig2fv in the working directory of vox? If not, put it there. Is sig2fv executable? If not chmod it to 700. If you are getting errors about libtermcap or something like that, just get it from somewhere. sig2fv depends on it.

Q The repository generator is not working:
A The program may take a long time to finish. This is particularly true when you are creating a repository. It may even happen during encoding or decoding phase.

Q Help! VoX is stuck!!
A The program may take a long time to finish. This is particularly true when you are creating a repository. It may even happen during encoding or decoding phase.

Q The encoder is not working:
A The repository generator is not working: The program may take a long time to finish. This is particularly true when you are creating a repository. It may even happen during encoding or decoding phase.

Q The decoder is not working:
A The repository generator is not working: The program may take a long time to finish. This is particularly true when you are creating a repository. It may even happen during encoding or decoding phase.

For more information visit http://vox.sf.net

Technical manual

VoX should work on any Linux/Unix box.

VoX has been developed using g++ on Redhat Linux. It has been tested on Redhat Linux and Knoppix.

VoX makes use of sig2fv tool of Edinburgh Speechtools Library.

You will have to compile it seperately and place sig2fv in the working directory of VoX.

VoX is independent of

- speech recording software and hardware
- e-mail software and communication network
- sound reproduction software and hardware

Advantages of this system

The system will be user-friendly. Once the repository generation and exchange process is over, communication can begin almost instantly. The following are the most prominent advantages of this system:

- Efficient Bandwidth Usage: Since only codes are transmitted, and not actual speech, the system uses very little bandwidth, and is extremely speedy and cost effective.
- Clarity Of Communication: Expression and understanding of emotions are better in voice communication.
- Usable as a shared library
- Easy to use package

Applications

- News broadcast and archival: Consider the audio news downloads which appear on news websites. These news items are typically read out by one person (or a small group of persons). The actual news audio samples can be encoded based on the profile. The users will only need to download the encoded data. This can be decoded using the profile stored earlier by the user, and the audio can be regenerated.
- Streaming and audio conferencing: Instead of communication via e-mail, this system can act as a phone, so that two people can communicate in real-time. Extending this idea further, multicasting will help in creating a virtual conference, wherein the voice of speaker will be made audible to the entire audience.

For more information visit http://vox.sf.net

Hardware Requirements

Linux Compatible Machine (Pentium etc…Recommended Pentium III or equivalent).

Soundcard, Keyboard, Monitor, Speakers, Microphone (Not essential but Recommended)

Internet connection (Not essential but Recommended),RAM atleast 256 MB (Recommended).

Secondary Storage (Hard disc) : >5GB,CD-RW Drive (if Internet not available).

CD-RWs.

Software Requirements

Operating System: Linux

Playback Software: that supports uncompressed Wavefile at 8000Hz,Mono channel,8-bits/sample

Recording Software: (Not essential but Recommended) that supports uncompressed Wavefile at 8000Hz, Mono channel, 8-bits/sample.

CD-RW software: if CD-RW drive is present.

Web browser and E-mail client.

The project will be independent of all these:

- speech recording software and hardware
- e-mail software and communication network
- sound reproduction software and hardware