INFOKIOSK: AN INFORMATION KIOSK WITH TEXT-FREE USER INTERFACE

A THESIS IN
Computer Science

Presented to the Faculty of University
of Missouri-Kansas City in partial fulfillment of
the requirement for the degree

MASTER OF SCIENCE

by

PRASHANT SUNKARI

B.Tech, Jawaharlal Nehru Technological University, India, 2008

Kansas City, Missouri

2010

INFOKIOSK: AN INFORMATION KIOSK WITH TEXT-FREE USER INTERFACE

Prashant Sunkari, Candidate for the Master of Science Degree
University of Missouri-Kansas City, 2010

ABSTRACT

Even though computer usage may seem very intuitive to almost everyone, they have minimum usability requirements that the user's ability to read is in the language being used. In developing countries such as India, where the adult literacy rate is 66% [1], this basic requirement for computer usage is its major hindrance. Some other hindrances to accessing modern technology are socio-economic inequality and cultural diversity. InfoKiosk is an end-user application as a step towards providing a text-free user interface (UI) using an existing architectural framework [2]. InfoKiosk UI is designed using features such as action images to represent types of information, mouse-over audio for navigation help, and universal help videos throughout every screen of the application. User inputs and outputs to InfoKiosk are kept intuitive and easy to understand. Two kinds of possible user inputs are – audio and mouse click. In response, the user will receive streaming audio and/or video from YouTube or the InforKiosk server. The design and development of InfoKiosk involve working with technologies such as Java Sound API and, Lumenvox text-to-speech translator.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of School of Computing and Engineering, have examined a thesis titled “InfoKiosk: An Information Kiosk With Text-Free User Interface”, presented by Prashant Sunkari, candidate for the Master of Science degree, and certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Deep Medhi, Ph.D., Committee Chair
Computer Science and Electrical Engineering

Yugyung Lee, Ph.D.
Computer Science and Electrical Engineering

Praveen Rao, Ph.D.
Computer Science and Electrical Engineering

CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACKNOWLEDGMENTS

Completion of any task requires two things passion to work on the topic and moral support. I thank God for giving me the interest to work on such an interesting topic. I am also grateful to Dr. Deep Medhi for his support and feedback as my advisor. It has been a pleasure working under his guidance. I am thankful to Venkata Rama Krishna Jamithireddy, Chitti Babu Ruddaraju for their contribution and assistance in my thesis work.

I would like to thank my parents, Dr. Vishwanatham Sunkari and Aruna Sunkari, and my sister Sai Madhavi Sunkari for their encouragement and moral support throughout my life. I am also thankful to my close friends - Rahul Reddy Varakala and Wendy Seelbinder- for their support and encouragement .

CHAPTER 1

INTRODUCTION

Currently, majority of people use computers for business, education and entertainment through laptops, desktop computers and mobile phones. If we take a careful look at section of people who can interact with these computers, then we find computers have always been a productive tool for literate people with at least a minimum ability to read and understand. These devices have been a powerful source of information and knowledge. If one is using such devices then one must be, at minimum, able to read the text in the language in use. Such a restriction renders millions of people, who are non-literate or semi-literate, unable to access world of information that could be of their importance. InfoKiosk is a step towards making the information available to people irrespective of their literary skills. InfoKiosk is an end-to-end user application with a text-free user interface (UI) and interacts with user based on audio, video and image display. In my work I have considered India as a case in point because of reasons such as

1. India is house to world's largest number of adult illiterates [1],
2. India offers diversity challenges by being a multi-lingual and, multi-religious country [2].
3. India also has vast socio-economic and cultural diversity [2].

Even though I will talk about InfoKiosk application considering the challenges in India, its architecture and application design is quite adaptable to any region of the world facing similar challenges.

India has adult literacy rate of 66% [33, 26] which well below the world literacy rate of 83%. According the UNESCO Institute for Statistic's worldwide survey [1], in 2008, 796 million adults were reported non-literate. Due to illiteracy and poverty people are unable to make use of the information which is easy and, sometimes, freely available. They are left behind in the world where, if googled, one can get all kind of information on web. Some of information has universal understandability when provided in multimedia formats – audio, video and images. For example, a person in a village in India would cherish availability of public health information from a nearby information kiosk. And for a farmer equally valuable would be any information about kind of fertilizer to be used for a certain crop. Also, information about vaccination, immunizations, employment, government schemes are useful when communicated on time. For making such information accessible to the varied population with characteristic like non-literate or semi-literate, multi-lingual and multi-cultural, it is important to concentrate on how it is presented to user. The information on web and in general can be broadly divided into four kinds - text, audio only, audio-video and images. The last three together are also called multi-media. InfoKiosk application heavily uses video/audio output (YouTube), text-to-audio conversion, audio feedback and images. InfoKiosk application ensures that non-literate users do not feel restricted while using applications on computer.

Generally, Information & Communication Technologies enabled public health projects reach users in one of the following ways: [9]

> Exclusive Websites for diseases

> Training material available on internet such as medical journals and periodicals for nurses and paramedics.

> e-Learning facilities such as web-based portals.

> Virtual knowledge centers for information dissemination.

> e-consultation of sensitive information such as AIDS

All the current and similar applications have exceedingly high level of complexity and diversity.

My goal while working on, InfoKiosk, was to create a text free UI with an architectural capability to enable non-literate or semi-literate people to interact and obtain essential information from an intuitive computer application. My implementation is inspired from the innovative work in the paper [2] by Chowdhury and Medhi. The authors have proposed an architectural framework for electronic access and retrieval of public health system in developing countries like India. There architecture was built considering myriad of factors like socio-economic diversity, multilingualism. The final design has qualities like inclusive, dialectic, adaptive, evolving, robust, and people sensitive.

There have been few attempts to create applications, projects and products which are aimed to help illiterate population in developing countries. In early 2000, seven Indian scientist and engineers at Simputer Trust [30], designed a self-contained handheld computer, called Simputer [29], as an alternative to Personal Computer. “Simputer” stands for Simple Inexpensive Multilingual People’s computer. Simputer has Linux Operating System, text-to-speech software, smart card reader/writer, web interfacing, touch screen operated by stylus and simple handwriting recognition software. The device was though aimed to help citizens in villages of India it ended up being used for other purposes such as land records procurement by Indian state of Karnataka, automobile engine diagnostics, and so on. Some of the success inhibitors could have been high license cost, lower cost of laptops and PDAs and lack of support from Government and NGOs.

Another work in the area of application development for illiterate population and one of the biggest influences to my work has been text-free UI application by Medhi which was discussed in [27]. She suggested some innovative design principles, for developing text-free

UIs, through an ethnographic design process. The process involved working with 250 people and more than 300 hours for field work at urban slums in Bangalore, India. These designed principles [7, 26] have been used for some test applications such as one developed for informal labor job search. While developing InfoKiosk, I have followed these proposed design principles and also added few additional principles. She has also explored similar design principles for mobile phones and also performed a usability test with design suggestions such as general text based, using multimedia and with spoken dialog feedback. It was concluded that non literates and semi-literate had faster completion rates and required less assistance while using spoken-dialog system. It may be noted that design methods for mobile interface for non-literate users is addressed in [16].

A major influence in the design and development of InfoKiosk is the Digital Green project [3]. This project based on a participatory learning framework and helps small and marginal farmers in India to access targeted agricultural information. This information is stored in a video repository and the content is generated by farmers and domain gurus. The produced videos are facilitated to the farmers for dissemination and training. The aim is to deliver the targeted information and enable farmer to efficiently perform their farming operations. Some of the important findings from the digital green project are as follows:

1. Video clips which attracted more attention have an entertaining flavor, such as women group singing folk songs.

2. It was influential when presentations and interviews where with the farmers who have farming experience or have benefited from the practices that are being preached.

3. It was well understood when a brief overview of (3-5 minutes) of topic is presented before conveying the actual point.

4. Farmers felt comfortable when familiar farmers from local vicinity performed demonstrations.

5. It was observed that farmers felt the need to review the video for 5 times on average.

The above findings are also important in our design of InfoKiosk.

InfoKiosk is a desktop based client-server application where all the processing tasks such as decoding the speech, retrieving appropriate video playback, text-to-text and text-to-audio translation and more takes place on the server side. Client is a text-free user interface developed using Java Swing and Java Sound technology. The user interface as described above has been inspired by design principles discussed in [7, 27]. InfoKiosk is a desktop based application but with these design principles one could develop an equally competent web based application or a phone application.

CHAPTER 2

OVERVIEW

While creating a text free UI I have concentrated on design principles recommended in [7, 27]. One of the authors of [27], Medhi has done extensive field work while developing these design principles and her field work included travelling and meeting people from 400 villages in India and South Africa. The thesis documentation has been divided into chapters to discuss - InfoKiosk components choice, architecture behind the application and future for InfoKiosk. In chapter 3, I tried to justify the choice of each component used in the InfoKiosk application. A detailed discussion on each module of the architecture behind the application is provided in chapter 4. Also in the same chapter is the high level discussion about interaction between those modules. The five stages of application design, software components used, and application tests run are all elaborated in three sections of chapter 5.

Following are the four main stages of operation when user interacts with InfoKiosk:

1. Selecting “kind of information” sought (On Client Side)

For example: Public health Information, Agricultural Information

2. User recording the query (On Client Side)

For example: Malaria Samachar, Polio Vartalu or in a pre-defined format <<Disease>> <<Language>>

Audio bytes are transmitted over to server using HTTP Post protocol.

3. Audio to Text translation using Lumenvox (On Server Side)

Lumenvox is a speech recognition engine used to decode the disease and language.

4. Requesting Interpreter and Translator (On Server Side)

After decoding the disease and language information, following steps are performed:

i. Using the disease and language information find the Youtube video ID for the requested information.

ii. If the video is unavailable in desired language then there are two possibilities:

a. “Text is available in desired language”: Translator is called to speak the text

b. “Text is available in different language”: Translator is called to convert the text in desired language and speak it.

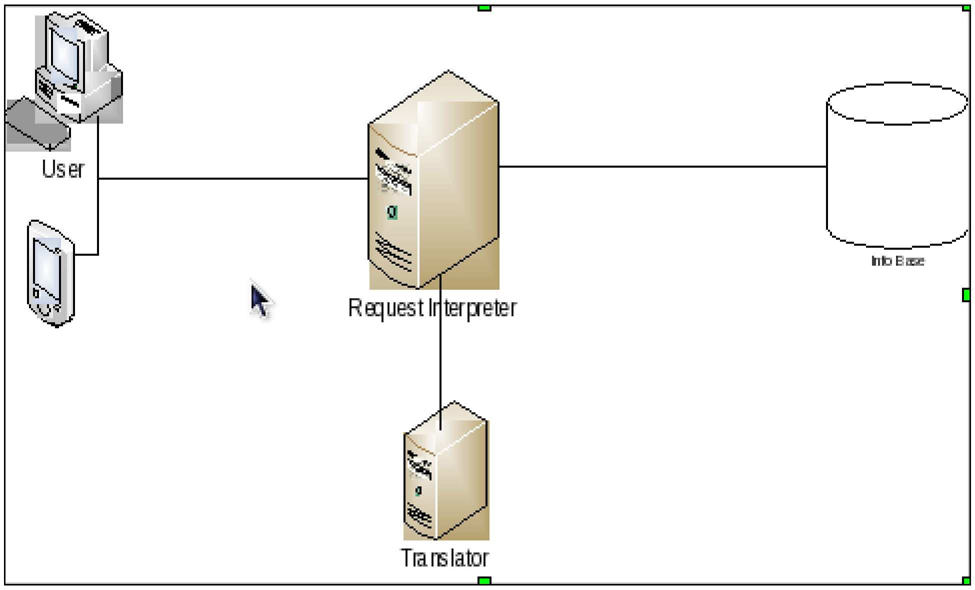


Fig 2.1: InfoKiosk Network Architecture

This architecture is originally outlined in [2]; the implementation of the core communication of this architecture is discussed in [11], while my work focuses from the perspective of a text-free user interface for this architecture.

A more detailed explanation about each module is given in chapter 4. A view of the modules involved in the network architecture behind InfoKiosk is show in Figure 2.1.

CHAPTER 3

APPLICATION COMPONENT CHOICES

Design, development and implementation of the components in InfoKiosk involved trying various possible choices for each and finding the optimal solution where all the components perfectly work together. For example for Audio-to-Text translation I tried an open source and a subscription based SREs (Speech Recognition Engine) available in the market. The open source option, Julius [15], is a two-pass large vocabulary continuous speech recognition (LVSR) decoder software. Julius requires acoustic model, which I tried creating using VoxForge but the model was efficient to handle various user input. Finally, Lumenvox was used for Speech-to-Text translation.

While working on the InfoKiosk I studied various topics such as Unicode for data storage and translation, SREs for audio to text translation and few Java APIs for interface development. Detailed information about each of the topics is given below sections and an overview can be obtained using Table 3.1

Table 3.1: InfoKiosk Technologies Used

	Technologies	**Used for**
1.	Java Swing API	User Interface Design
2.	Java Sound API JMF Youtube API Native Swing API	Audio, Video Interaction
3.	Lumenvox API Unicode Julius	Speech to Text Translation
4.	SGRS Grammar	Grammar for Lumenvox Speech Engine

Continued…

	Technologies	**Used for**
5.	Java Servlet	Simulate HTTP Server
6.	Jffmpeg	Codec
7.	Fedora 11, Eclipse and NetBeans	Development Platform

3.1. Unicode

I approached Unicode seeing it as a standard way to represent and translate between different languages using its features such as language tagging. Though Unicode has language tag functionality but it was not used and not recommended to use it for language translation.

Unicode [35] is an industry standard that allows computers to represent and manipulate text from majority of world's writing systems. Language tag is specified by a string of characters U+E0001 as tag characters. Tag values are spelled out in a format underlined by Internet Request for Comments (RFC) 4646 [28], i.e., using user defined tag or registered tag which begins with "x-". Unicode consortium advice for Unicode users is to avoid the language tags in plain text because of the additional overhead of implementation. [35]. whenever it is implemented few points must be considered

1. Consider protocols such as MIME, HTML as they may also provide language attributes.
2. Consider effect of tags on syntactic meaning of text.

3.2. Text Free UI

The text free UI for InfoKiosk has been designed and developed following design principles [7, 27] and using Java Swing. For me the motivation and guiding parameter has been [7, 27] in the field of creating text-free interfaces for non-literate and semi-literate users with application such as Employment search, Health symptoms based search application. She has done extensive field work enabling people, with low literacy and minimum computer skills, to connect and utilize computers. The common base for all successful text-free UI applications has been ethnographic design process and eliminating need for text. Ethnographic design process helped to understand that user feedback is sought after every step. The complete text elimination is compensated with audio feedback for all functional units, mouse over-actions, and semi abstracted cartoon. All the ideas have been assimilated into following principles of text free UI:

i. Minimal use of text:

- The reason for saying "minimal use" when target is text-free UI is that research has proven that numbers (1, 2, 3 etcetera) are readable by majority of people even if they have are illiterate.

ii. Abstracted cartoons are preferred over simplified graphics:

-The hand drawn diagrams with action as visual representation of activity were easily understood.

iii. Voice feedback on all functional units:

- It has been observed that action on mouse over whenever possible is very helpful for user. All the functional elements on the application screen should have voice audio feedback in the language selected. Such a feedback assures users whether they have selected what they intended to.

For example- In my implementation after selecting for information on “public health” user is given voice feedback “You have selected to get information about “public health” (In appropriate language).

iv. Full content video to dramatize the intent and mechanism of an application.

According to [7, 27], using the above design principles led to increase in task completion percentage to 100% from 30%. For the Job search application the phases were:

Intro page ---> Location Page ---- > Job Listing ---> Job Info

For our application

Language selection ---> Information type selection ---> Recording query ---> Video output

It must be noted that user-centralized design in developing countries depends on factors such as:

i. Requirement of innovation in design process

ii. Tweaking established process to fit the context.

iii. Designer must spend much time as possible engaging with potential user keeping in mind the cultural differences.

In another work [7], test results were presented for experiments with application using static images, text static drawing, hand-draw animation and videos - with & without voice annotation. Some of the important results were:

i. Voice annotation helps in speed of completion.

ii. Richer info can be confusing and are not always better.

iii. Dynamic images were more understandable than static images.

Results from this work proved some obvious and non obvious things. Obvious things were

i. Use of graphical icons may not be always useful.

ii. Minimal use of text was productive

iii. Voice annotation was very helpful.

iv. Easy navigability helps in faster completion rate.

v. Degree of interaction with subject is important.

Not so obvious results were:

i. Dynamic images were easy to understand than the static images.

ii. Hand-drawn images were easier to recognize than regular photographs.

iii. Visual representation was an important matter.

The main challenges of designing a text free UIs has been that the visual representation must be comprehendible to all irrespective of culture. Major outcome of this work which was very useful for my thesis is that voice annotations and semi abstracted drawings were found to be best for non-literate user. I have introduced few additional principals which add on to the principles discussed in [7]:

i. Universal help feature on all the stages of interaction with InfoKiosk.

ii. Audio Input: User interacts with InfoKiosk has been restricted to two types – button clicks and audio input. Audio Input feature makes it easy to add data on the server side without changing the user interface.

iii. Audio or video output only: The InfoKiosk user would receive response as a Youtube video played on Java Web player or an audio/video played using JMStudio player.

3.3 Speech -To-Text Translation

Speech to text translation is a method of converting a given voice sample in to a text in the selected language. This is also called automatic speech recognition or speech recognition. Speech Engines are capable of recognizing individual words but they don’t really understand speech in same way as humans do. A Speech engine [35] communicates to speech application about what the user said and then the application decides on how to handle it. Some of the examples of dynamic speech-based application are Voice activated dialing, Bill payment, Doctor’s Appointments, Order Status, Flight Information and Phone Shopping. Speech recognition and voice recognition are different; voice recognition deals with identifying individual voices and not what the speaker said. Following are the general stages that speech recognition engines go through:

1. The engine loads a grammar and speaker audio.
2. The speaker audio, represented as waveform, is compared to waveform in acoustic model.
3. The Engine compares the results in step 2 with the words in grammar.
4. The closest matched word from the grammar is returned.

In the development of InfoKiosk application I have tried two speech-to-text applications – Julius, an open source option and Lumenvox, market leaders in speech recognition software industry [24]. These applications have been discussed in the sections, 3.3.1 and 3.3.2. Figure 3.2 gives a high level diagram for speech recognition engine with respect to Julius application.

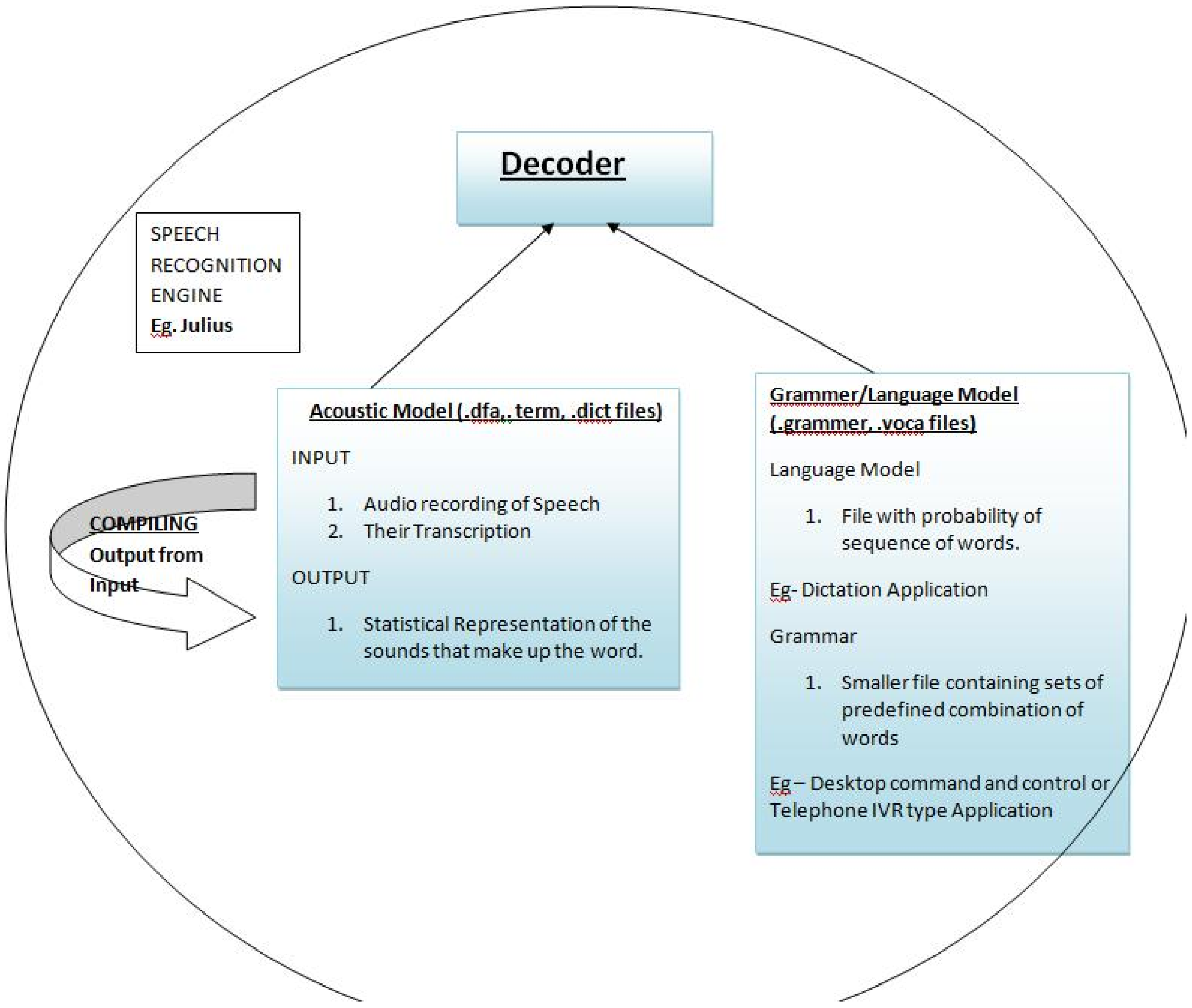

Decoder
SPEECH RECOGNITION ENGINE
Eg. Julius
COMPILING
Output from Input
Acoustic Model (.dfa,, term, .dict files)
INPUT
1. Audio recording of Speech
2. Their Transcription
OUTPUT
1. Statistical Representation of the sounds that make up the word.
Grammer/Language Model (.grammer, .voca files)
Language Model
1. File with probability of sequence of words.
Eg- Dictation Application
Grammar
1. Smaller file containing sets of predefined combination of words
Eg – Desktop command and control or Telephone IVR type Application

Fig 3.1: General SRE Decoder

3.3.1. Julius

Julius [15] is one of the best open source options available for speech recognition. Two of the important features of Julius which made me to try it are its two phase LVSR decoder software, and it being open source product. To run a Julius speech recognizer we need two things for your language - Language model and Acoustic Model. And this was biggest hurdle in adopting Julius as the audio-to-text translation module for InfoKiosk.

Julius is language independent decoding program which needs – a language model and then for the chosen language we need an acoustic model- to make a recognizer. The recognition accuracy depends on these models. One of the biggest problems for open-source SREs is the Acoustic models are not open source. Acoustic model is created using speech audio and that speech audio is not available freely. VoxForge tries to address this problem by creating speech audio and transcriptions, and by helping to create Acoustic Models for Julius and other open source SREs.

An acoustic model is used to describe the statistical representation of phonemes in the defined in the SRE specific language. These statistical representations, also called Hidden Markov Models (HMMs), are created using large samples of Speeches and special training algorithms. As shown in the Figure 3.1, once an acoustic model is ready, the decoder listens for distinct user voice inputs and matches with the HMMs in the acoustic model. All the matching HMMs and their corresponding phonemes are recorded till a pause is reached. After the pause, decoder will look in pronunciation dictionary for the recorded sequence of phonemes. Once the word is determined a predefined grammar is scanned to get the display format.

I tried to create Speaker dependent Acoustic model using Hidden Markov Toolkit (HTK) and Julius. Hidden Markov Models (HMMs) for representing sound in Speech Recognition are developed using HTK. There are three main steps involved in this process - Data Preparation and create Monophone HMMs, create Tied-state Triphones. Firstly, a phonetically balanced pronunciation dictionary has to be created with sorted list of words in grammar. Data preparation stage involved recording audio for the sample sentences which involve words in the grammar and more. After following the remaining steps the acoustic model prepared failed to decode recognize user input. One of the reasons for failure has been the insufficient resources available to create Monophones, the second step of the complete process. The Monophone creation is a vital step in the creation of Hidden Markov Model and the tutorial is customized to create an HMM with general set of sentences. These words are associated with a particular frequency and are used in Monophone creation. The creation of HMM model was not flexible with every grammar. Also the creation of a successful acoustic model requires two important things – an in-depth knowledge of Hidden Markov Model and HTK toolkit and large amount of training data.

3.3.2. Lumenvox

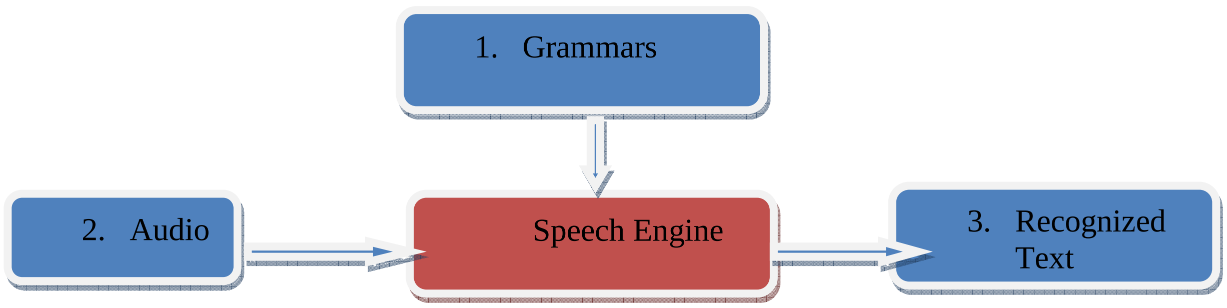


Fig 3.2 Speech Engine

Lumenvox provides speech recognition tools which are generally used in IVR and other applications. It is not a speech verification tool. Some of the products made by them are:

i. Speech Engine

ii. Speech Tuner

We are interested in Lumenvox Speech Engine. It takes audio as input (file/live input) and is guided by grammar to recognizes the audio. It is speaker independent and no dictation is needed.

Why Lumenvox?

There are several reasons to select Lumenvox for the project. They are as follows:

i. Efficient development & run-time platform by allowing dynamic language, grammar & audio formats.

ii. Grammar formation is easy. Grammar consists of a set of words or it can also be expressed using Speech Recognition Grammar Specification (SRGS).

iii. User independent. Lumenvox is independent of audio source. Speech recognition can be performed on any audio data.

iv. Has built in support for Voice Activity Detection (VAD), which is used for noise cancellation, NBest Results (Gives top few expected solution with probability), SRGS standard support.

Though Lumenvox is optimized for IVR solutions and turned out to be good dedicated application software.

There are several standards that Lumenvox supports such as Media Resource Control Protocol (MRCP), Semantic Interpretation for Speech Recognition (SISR), VXML, and SRGS. We are interested in SISR - a W3C proposal that allows programmer to define a specific interpretation for user input. For example, if the user inputs "May sixth two thousand and four" the application will understand "2004-05-06". Lumenvox defines SISR as “SISR allows grammar authors to embed snippets of JavaScript code into their SRGS grammars, to automatically transform what a speaker says into a format understandable to an application.”

Grammar Design:

For Lumenvox to work for our project, we need to design a SRGS(Speech Recognition Grammar Specification) grammar as an additional module for Lumenvox. Once the grammar is defined, Lumenvox uses the defined grammar to interpret the user audio input. I have designed a simple two word SRGS grammar which efficiently serves our purpose. The grammar is designed to accept two words - Information sought and native language translation for word “news”. For example if the users wants information about Malaria in Hindi then they would say “Malaria Samachar”, where “Malaria” is the information sought and “Samachar” is the Hindi translation for word “news”.

Table 3.2: User Audio Query Format

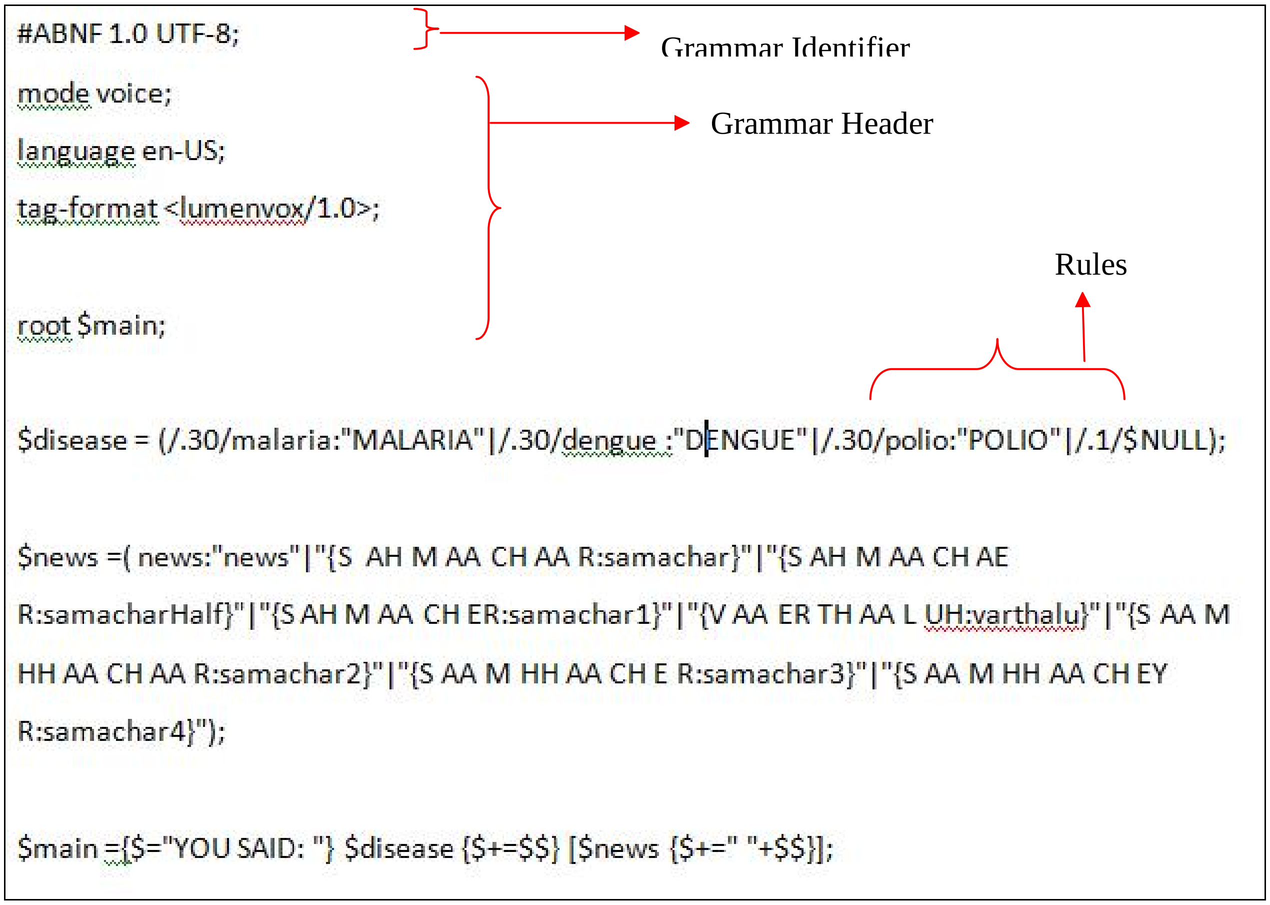
		Information Sought (RETRIEVED)	Native Language translation for word “news” (RETRIEVED)	Information sought in language (DERIVED)
1.	Malaria Samachar	Malaria	Samachar	Hindi
2.	Malaria Varthalu	Malaria	Varthalu	Telugu
3.	Malaria News	Malaria	News	English

Grammar format:

The Lumenvox Speech Engine supports grammars must be written according to SRGS grammar rules. The basic structure of a grammar file would consist of Grammar Identifier, Grammar Header and Rules. Grammar Identifier declares the type of grammar being used, for example ABNF grammar. The language of interaction, root rule definition (where the engine begins search) and expected interactive mode are the three section under

Grammar Header. Finally, the Grammar Rules specify combination of words the Engine can recognize. Each rule has a name starting with $ character and immediately after the = sign is the rule expansion. The rule expansion contains the words associated with the rule and optionally can include the word pronunciation with its probability of occurrence.

Grammar used for InfoKiosk application is as show in Figure 3.3



```
#ABNF 1.0 UTF-8;

mode voice;

language en-US;

tag-format <lumenvox/1.0>;


root $main;


$disease = (/.30/malaria:"MALARIA"|/.30/dengue :"DENGUE"|/.30/polio:"POLIO"|/.1/$NULL);


$news =( news:"news"|"{S  AH M AA  CH AA R:samachar}"|"{S AH M AA CH AE
R:samacharHalf}"|"{S AH M AA  CH ER:samachar1}"|"{V AA  ER TH AA L UH:varthalu}"|"{S  AA M
HH AA CH AA R:samachar2}"|"{S AA M HH AA CH E R:samachar3}"|"{S AA M HH AA CH EY
R:samachar4}");


$main ={$="YOU SAID: "} $disease {$+=$$} [$news {$+=" "+$$}];
```

Fig 3.3 Lumenvox Grammar

Running Lumenvox Service and Output:

InfoKiosk application uses Speech Engine Lite - License service of LumenVox Speech Engine [20]. This license version can be used to recognize up to 500 words/pronunciations per interaction. LumenVox Speech Engine supports multiple languages and performs speech recognition with audio feed. Speech Engine API provides the application with a speech port to communicate with it. Following are the steps that InfoKiosk goes through to decode the user audio data:

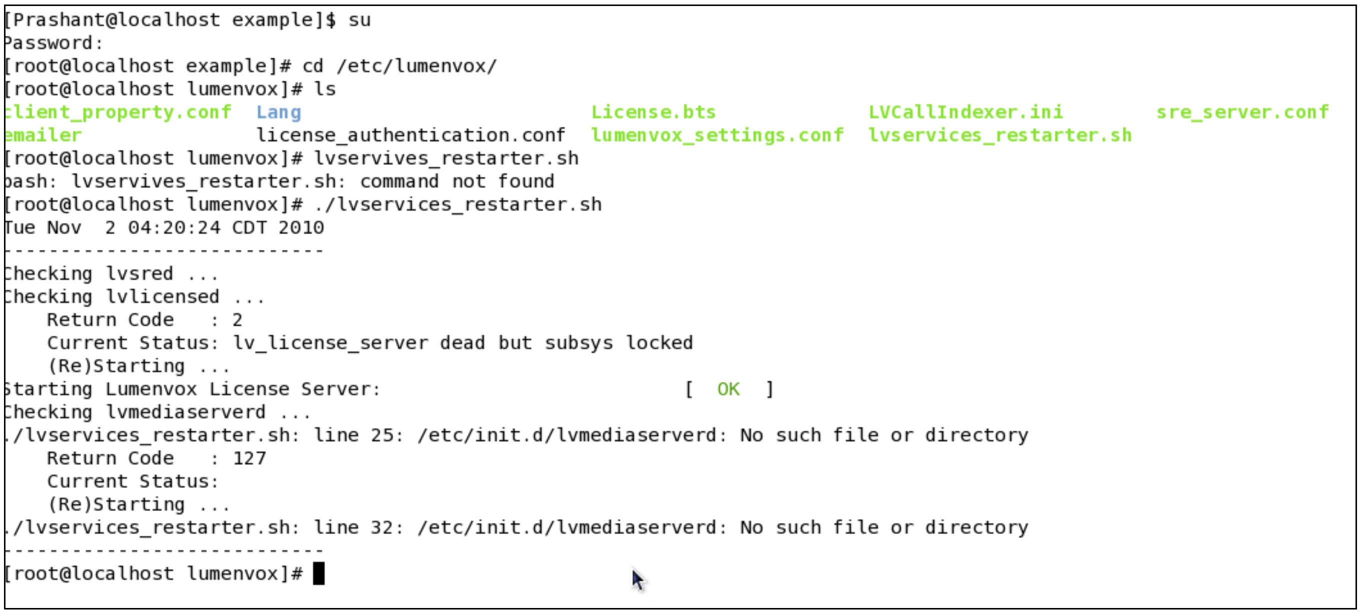
i. Opening a new port to connect InfoKiosk to Speech Engine

ii. Load a grammar

iii. Load audio data

iv. Instruct the engine to get result.

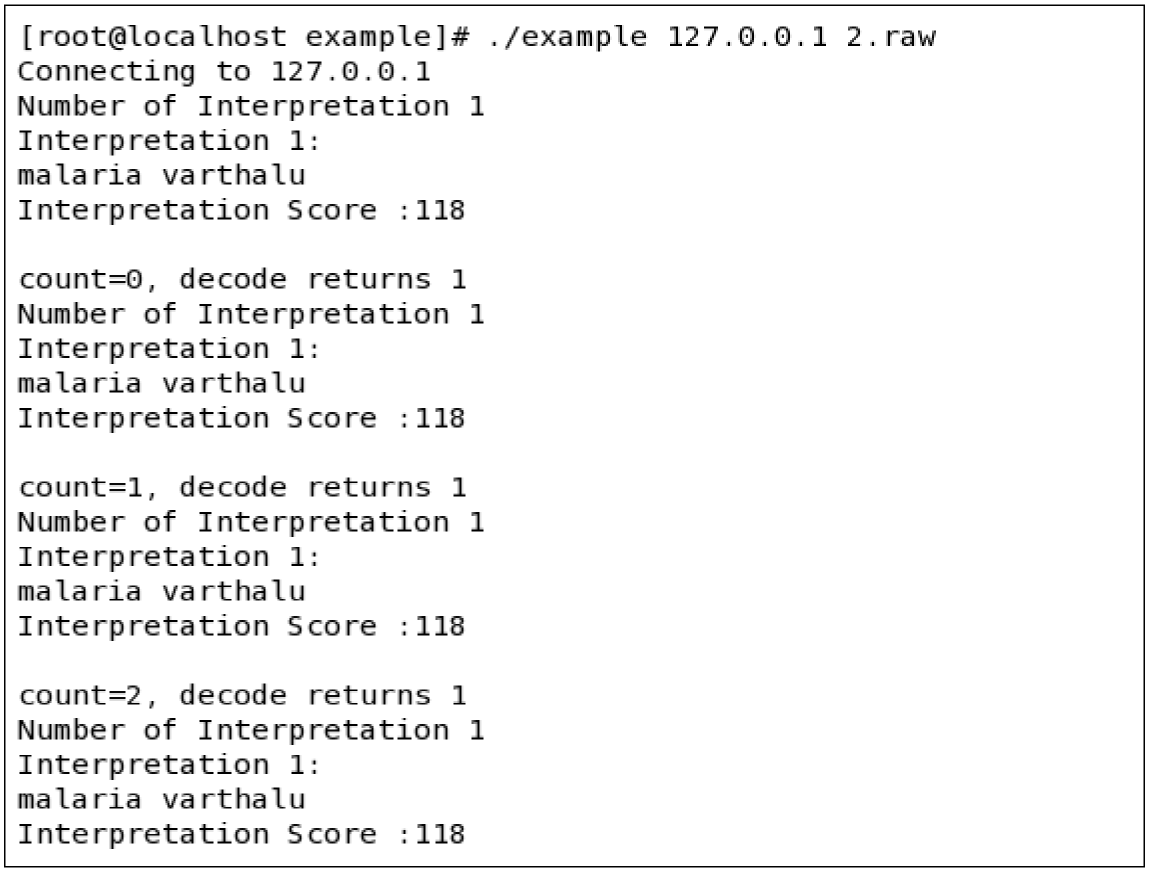
The Speech Engine Lite gives access to use speech recognition resource on per channel basis. Before running the program to connect to Speech Engine the LumenVox license server is turned on.



```
[Prashant@localhost example]$ su
Password:
[root@localhost example]# cd /etc/lumenvox/
[root@localhost lumenvox]# ls
client_property.conf  Lang                        License.bts            LVCallIndexer.ini        sre_server.conf
emailer               license_authentication.conf lumenvox_settings.conf lvservices_restarter.sh
[root@localhost lumenvox]# lvservives_restarter.sh
bash: lvservives_restarter.sh: command not found
[root@localhost lumenvox]# ./lvservices_restarter.sh
Tue Nov  2 04:20:24 CDT 2010
----------------------------
Checking lvsred ...
Checking lvlicensed ...
    Return Code    : 2
    Current Status: lv_license_server dead but subsys locked
    (Re)Starting ...
Starting Lumenvox License Server:                          [  OK  ]
Checking lvmediaserverd ...
./lvservices_restarter.sh: line 25: /etc/init.d/lvmediaserverd: No such file or directory
    Return Code    : 127
    Current Status:
    (Re)Starting ...
./lvservices_restarter.sh: line 32: /etc/init.d/lvmediaserverd: No such file or directory
----------------------------
[root@localhost lumenvox]# █
```

Fig 3.4: Start Lumenvox Services

Command for restarting LumenVox Services as seen in Figure 3.4 is "/etc/lumenvox/lvservices_restarter.sh".



```
[root@localhost example]# ./example 127.0.0.1 2.raw
Connecting to 127.0.0.1
Number of Interpretation 1
Interpretation 1:
malaria varthalu
Interpretation Score :118

count=0, decode returns 1
Number of Interpretation 1
Interpretation 1:
malaria varthalu
Interpretation Score :118

count=1, decode returns 1
Number of Interpretation 1
Interpretation 1:
malaria varthalu
Interpretation Score :118

count=2, decode returns 1
Number of Interpretation 1
Interpretation 1:
malaria varthalu
Interpretation Score :118
```

Fig 3.5: Lumenvox Audio Decode Output

Once the services are started the C++ decode program [Appendix: I] is run to connect to Speech Engine using new grammar and audio. Figure 3.5 shows the output of that program.

How Speech Engine works with Grammar:

The Speech Engine begins audio decoding from root rule, "main" in InfoKiosk grammar as show in Figure 3.3. It then steps through the legal expansions. The control logic moves into the rules "$disease" and "$news", as it can match against combination of both rules. The final result, indicated using $main rule, is a concatenation of the results from $disease and $news rules. Figure 3.4 show an example where “2.raw” audio file has pronunciation “malaria varthalu” where $disease rule returns “malaria” and $news rule returns “varthalu”. The output of the decode program is shown in Figure 3.5.

CHAPTER 4

ARCHITECTURAL FRAMEWORK FOR NETWORK BEHIND INFOKIOSK

The architectural framework for the network is adopted from the framework suggested in paper [2] by Chowdhury and Medhi. This architectural framework has 4 main components:

i. User-interface

ii. Information Base

iii. Request Interpreter

iv. Translator

The module interactions have been elaborately discussed as a thesis work by Jamithreddy [11] in his thesis work.

4.1. Modules

4.1.1 User Interface (UI)

User interaction with InfoKiosk begins at this point. As the targeted users for InfoKiosk are non-literate and semi-literate, I have created a text free user interface using the design principles [7, 27]. Text-free UI has been discussed in detail in section 3.2.

4.1.2 Request Interpreter (RI)

This module is one part of server which listens to the User-Interface module for http post request with audio of user request and other information such as language-response-requested. After receiving the audio bytes, the Request Interpreter, in simplest scenario, will pass the bytes to the speech-to-text sub module of Translator module and gets back text format of words pronounced in the audio. After receiving result from Speech-To-Text sub module, RI will check in Information base for a video containing requested information in requested language. Based on this text request interpreter has to make one of the following choices:

Case 1: If such a Youtube video is available then return the corresponding Youtube videoID to UI module.

Case 2: If the video is not available for the requested information or in requested language then we need to look for text information. In case of having text information in requested language, a call to Text-To-Speech sub module will give us the required audio. This Audio is returned back to UI module.

Case 3: If Case 1 and Case 2 are not satisfied then control logic comes to Case 3. Here a check is made to see if the requested information is present as text in a language other than requested language. In such a situation two sub modules of Translator module are used. Firstly, the Text-To-Text module is used to convert the available text in requested language. Secondly, the Text-To-Speech module is used to convert into audio bytes which are transmitted to UI module via Request Interpreter.

Case 4: If none of the above cases are applicable then HTTP “501 Not Implemented” response is sent back. On receiving this message at UI module, an audio informing user about unavailability of information is played back in the language user is interacting.

Table 4.1: Request Processing Steps

	Possible Outcomes -----------> | | | Example Situation | | v	Video or Audio Available in RL Case 1	Text Available in RL Case 2	Text Available in Another Language Case 3	No Text or Audio-Video is available Case 4.
1	**Requested Information**: Malaria **Requested Language(RL)**: Telugu	STEP1: Youtube VidoeID is returned to UI module	STEP 1: Text-To-Speech Sub module of Translator module is called. STEP 2: Audio bytes are returned to UI module	STEP 1: Text-To-Text(TTT) sub module is called STEP 2: TTT sub module calls Text-To-Speech Sub module (Sub modules are part of Translator Module) STEP 3: Audio bytes are returned to UI module	STEP 1: Audio saying that the requested information is played back at UI module

4.1.3 Translator

This module is second part of server, the first being the Request Interpreter. This module has three sub modules - Text-To-Speech, Speech-To-Text and Text-to-Text. For the Speech-To-Text translation I have tried Julius SRE and Lumenvox Speech Engine. Finally, based on the advantages I decided to use Lumenvox Speech Engine. Lumenvox SE translator needs two things - a grammar and C++ program to decode the recorded voice. Based on the decoding requested information, a Youtube videoID is selected by server in corresponding language and returned back to the User-Interface Module. The requested information - about a Disease or Crop - may not be available, in such a case it will be checked if the same information is available in text format in same or different language.

4.1.4 Information Base

For the information base, we decided to use a Youtube channel to store the videos required. Each of the Youtube videos has a video ID, which is returned to User-Interface module to be played for user. Information Base module also has videos, audios and text documents. If the requested information is not available as a Youtube video then other options are considered in the order of priority as discussed in table 5.1.

CHAPTER 5

APPLICATION DESIGN, IMPLEMENTATION AND TESTING

Text free UI Application design and implementation is very challenging mainly for two reasons - there has not been much application development done in this field, and the applications developed in this domain are very user centric and requires a constant developer-user interaction during the software development. In this chapter, I discuss the different stages of user interaction with InfoKiosk in section 5.1, the software tools used to build the UI are discussed in following section 5.2 and finally in the section 5.3 an application evaluation from user point of view has been made.

5.1 Stages of Design

InfoKiosk application has features which are universal to all its screens such as buttons with voice over, introductory video link on all screens and use of hand draw images to maximum extent. These features has been added keeping in mind the design principles of [7, 27]. InfoKiosk application design can be divided into five levels based on the interaction with the user. These stages are: Stage 1) Language selection, stage 2) Information domain selection, stage 3) User request (audio) submission, stage 4) Video or audio reply and stage 5) Help video.

Stage 1: Language selection

The language selection stage gives user a choice of language in which user wants to interact with the system. As the target users are non-literate and semi-literate, the possibility that they have low income level and less social interaction is high. These are some of the factors I have considered while identifying things users might relate to a particular language.

Following were the possible options for the images to be used to represent a language selection:

1. Local Political leader’s pictures.
2. Regional Sport person or Movie Star
3. Regional famous monuments
4. Words written in the regional language and use of numbers.

It is quite common that people tend to recognize their regional language if seen in written format. They may not be able to read it but they identify it as their language. We have built our Language Selection interface based on this thought. Figure 5.1 is a screenshot of the InfoKiosk language selection interface. We have several buttons each with an icon depicting words from that particular language. As we have discussed in the design principles [7, 27], people tend to comfortable with number so we have used numbers with different colors to identify the different languages along with the icons.

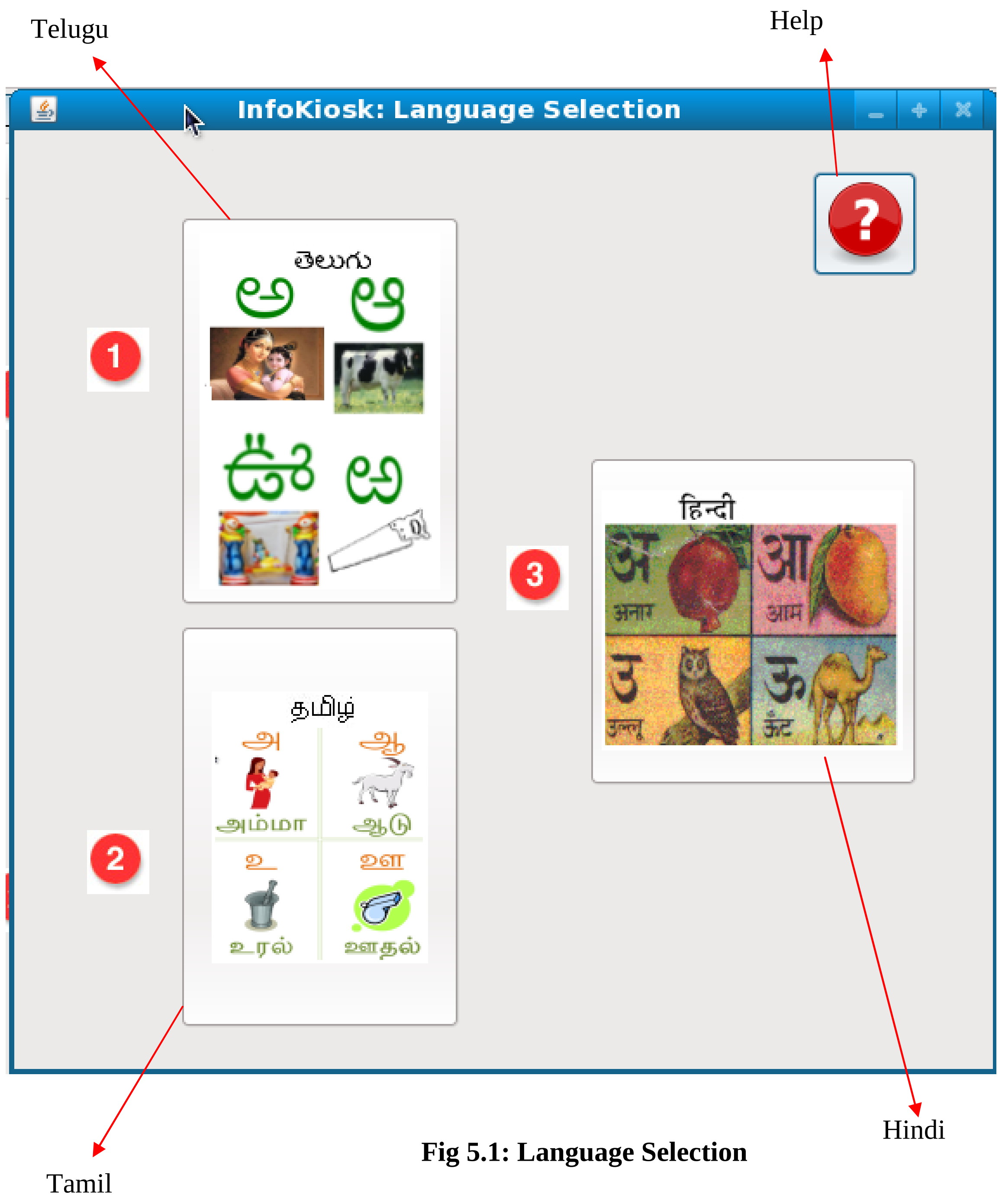


Fig 5.1: Language Selection

Stage 2: Information Domain Selection

As show in Figure 5.2, at this stage users have to make a choice of the kind of information they will be seeking. The range of available domains is currently limited for the demo purpose but the application is scalable to as many domains as desired. The domains can be accessed by clicking on buttons representing the domain. Each domain is represented by a hand drawn image which is displayed as icon on the button. As described in this thesis in chapter 2, the diagrams are made depicting an action taking place rather than a still image. It has been observed that images depicting an action are more easily perceived. Once the users make the choice of the information domain they are transferred to next stage. This stage also provides the user choice to get back to language selection screen.

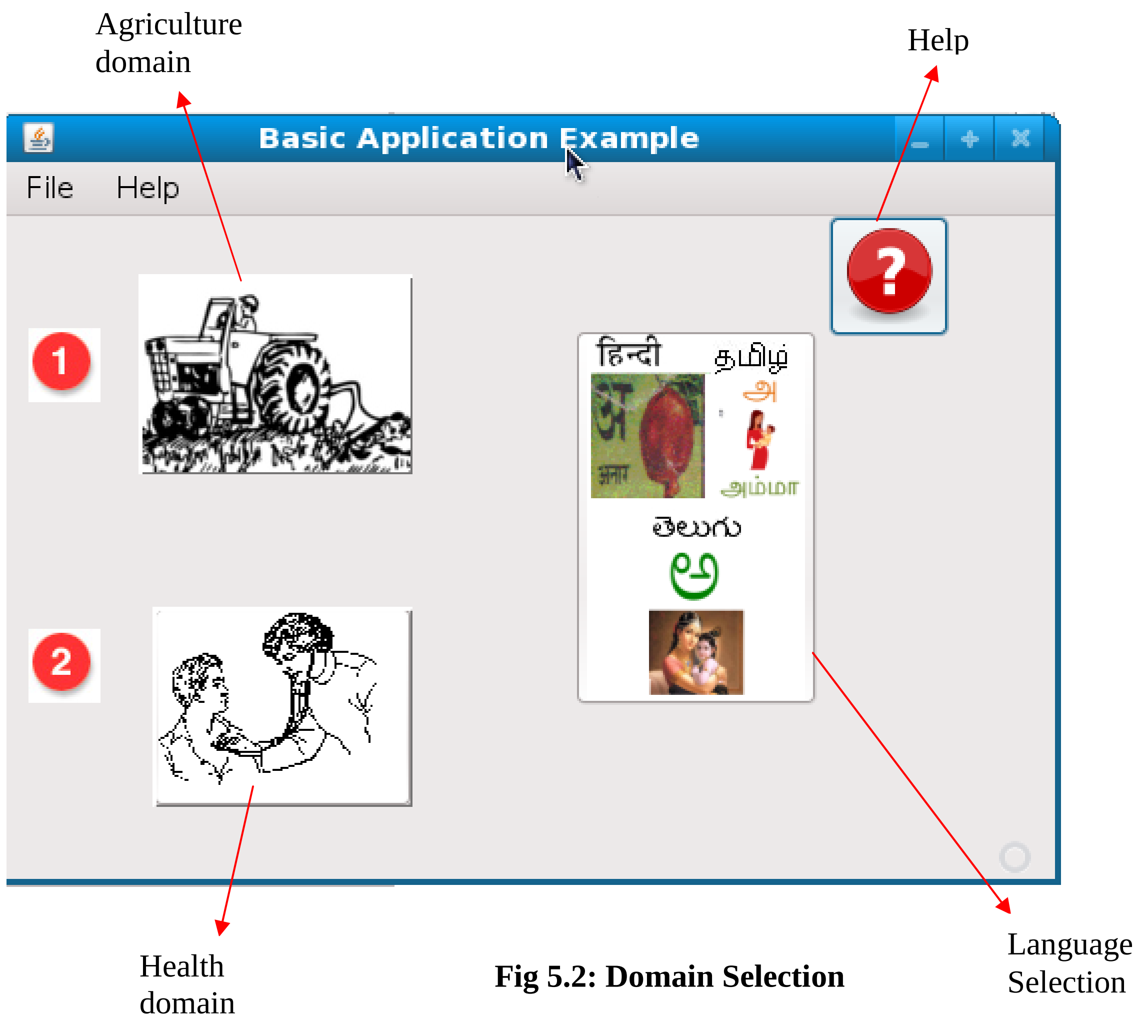


Fig 5.2: Domain Selection

Following are the information domains which are largely used by user and thus have been used for demonstration purpose:

1. Medical Domain Information

Medical information is direly needed by Indian people in villages and towns especially in the circumstances where there is no public health facility or doctor available. The kind of general information that is sought is about disease such as Malaria, Polio. Also general information about vaccinations and immunization can be provided through this domain.

2. Agricultural/farming Information

India has been an agricultural country for a long time with 60% of the population [5] living on it for livelihood. The livelihood of many and especially poor farmers has been made critical due to many factors like unfriendly climate and government ignorance. Also, majority of the Indian farmers suffer with lack of information of good modern farming practices. One of the reason farmers tend to follow their intuition or go by the hearsay of fellow villagers. These factors make agricultural related information a popular point of interest among villagers.

Other information domains where InfoKiosk can be used are rural employment information, general election information and government welfare schemes. Also the application of InfoKiosk is not limited to the domains discussed.

Stage 3: User request (Audio) submission

Once the users are in this stage, they are ready to record their query in a predefined format and click on next button. The user query should be in pre-defined format of two words. For example if the user wants to query about information about malaria/polio in Hindi/Telugu i.e. firstly, selecting “Hindi/Telugu in Language Selection” Stage and secondly, selecting Public-health information in “Information Domain Selection“stage. The user audio input for such a case would look like:

i. Malaria Samachar: Where Samachar means News.

ii. Polio Varthalu: Where Varthalu means News.

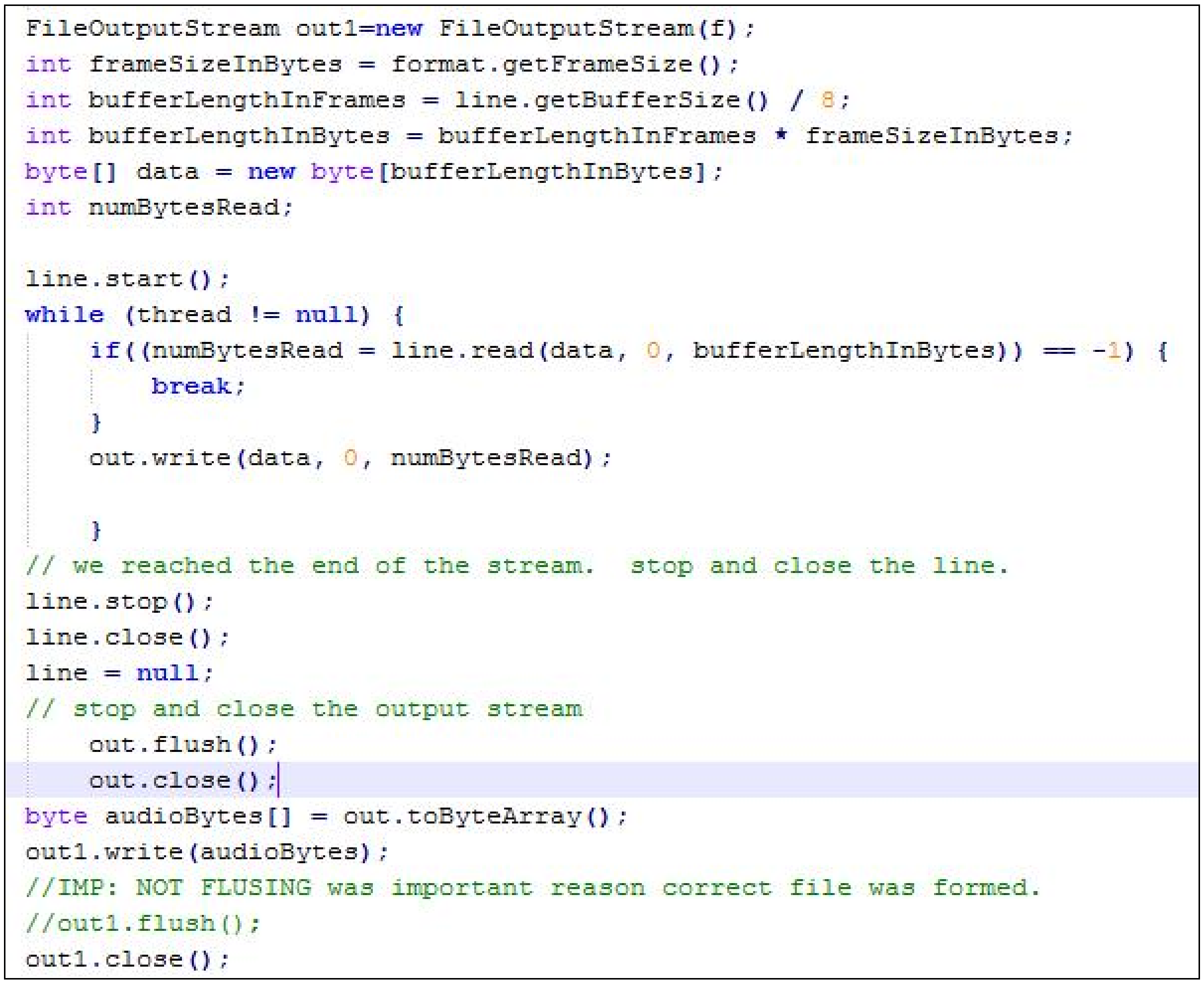
This interpretation of the users audio input is done at server side. Client application would send the recorded audio bytes over HTTP to the server or Requester Interpreter and server would decode the audio using the explained grammar to understand what information is requested. The server side audio to text translation is done using the C program. After understanding the information requested in the language, server side will look up if such information is present in the defined priority order. The priority order is:

a. Requested information in requested language in video format

b. Requested information in audio format.

Table 5.1: Possible Outputs From Request Interpreter

	Information return priority order	When is it possible?	What will client receive from server?	Notes
1.	Requested information in requested language in video format	If such a video is present on our Youtube channel	Youtube VideoID	
2.	Requested information in requested language in audio format	1. Priority order 1 cannot be satisfied. 2. Requested information is available in any language as Text. Or requested information is present in desired language as audio.	Audio bytes which will be combined to form an audio file.	If requested information is present in given language as text then Server makes call to text-to-speech module. If requested information is present in different language as text then Server has to make two calls -to text-to-text module and text-to-speech module.



```
FileOutputStream out1=new FileOutputStream(f);
int frameSizeInBytes = format.getFrameSize();
int bufferLengthInFrames = line.getBufferSize() / 8;
int bufferLengthInBytes = bufferLengthInFrames * frameSizeInBytes;
byte[] data = new byte[bufferLengthInBytes];
int numBytesRead;

line.start();
while (thread != null) {
    if((numBytesRead = line.read(data, 0, bufferLengthInBytes)) == -1) {
        break;
    }
    out.write(data, 0, numBytesRead);

    }
// we reached the end of the stream.  stop and close the line.
line.stop();
line.close();
line = null;
// stop and close the output stream
    out.flush();
    out.close();
byte audioBytes[] = out.toByteArray();
out1.write(audioBytes);
//IMP: NOT FLUSING was important reason correct file was formed.
//out1.flush();
out1.close();
```

Fig 5.3 Code To Capture Audio

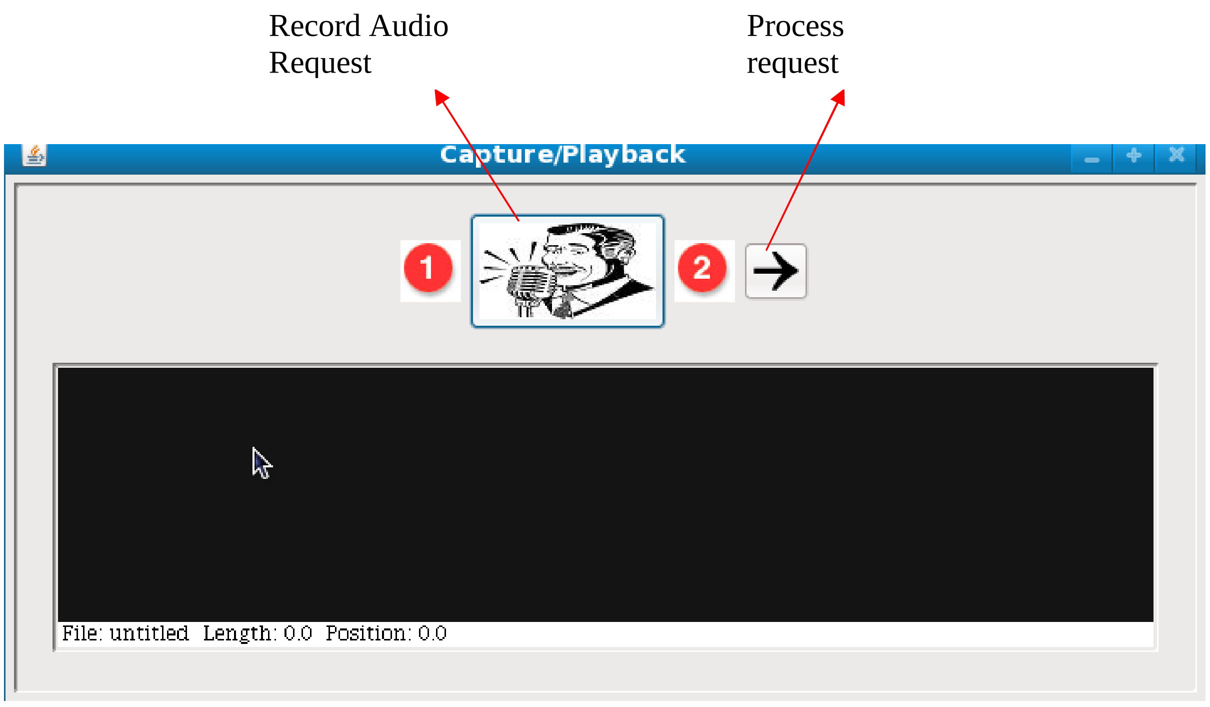


Fig 5.4: Audio Request Recording

Stage 4: Video and/or Audio Reply

InfoKiosk is capable of playing a typical Youtube video or standard format audio file using JMStudio player. Client application makes an informed decision of which player to choose based on the HTTP response from server.

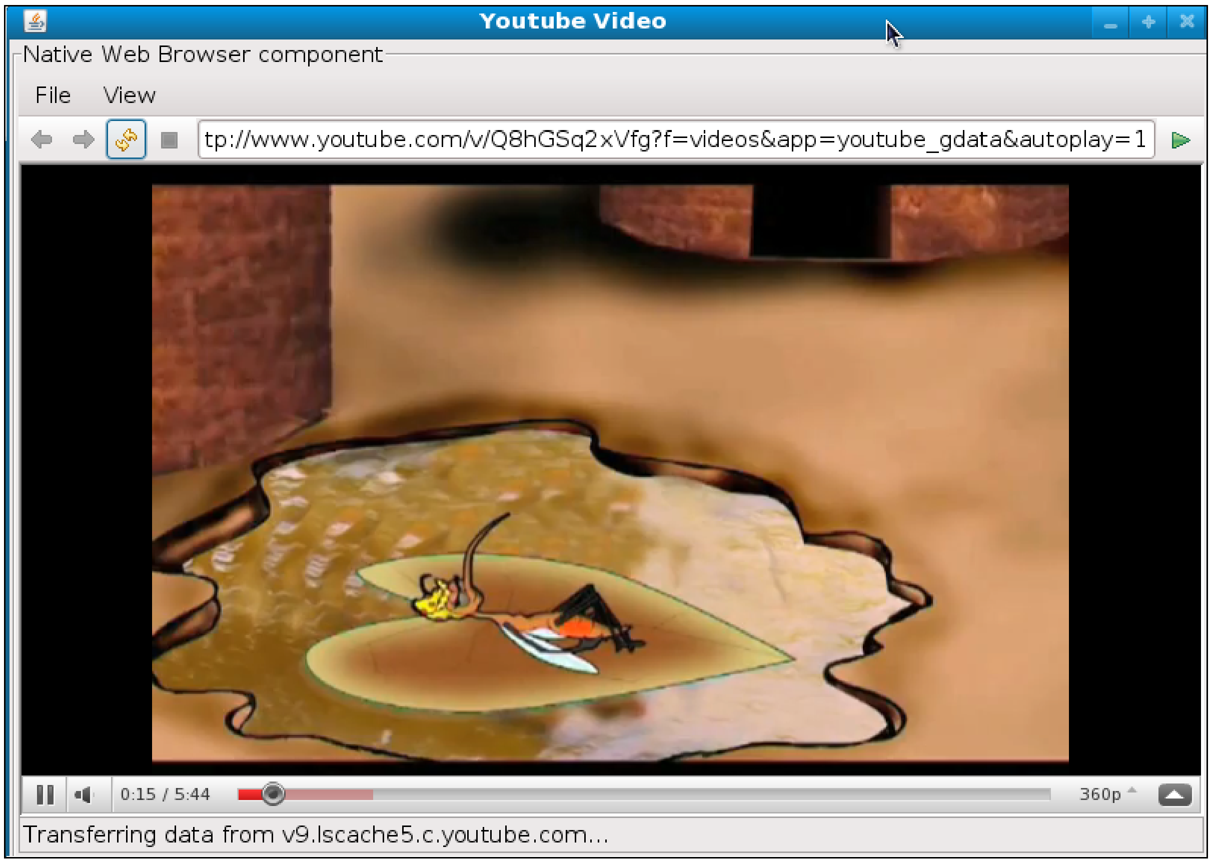

Youtube Video
Native Web Browser component
File
View
tp://www.youtube.com/v/Q8hGSq2xVfg?f=videos&app=youtube_gdata&autoplay=1
0:15 / 5:44
360p
Transferring data from v9.lscache5.c.youtube.com...

Fig 5.5: Youtube Web Player

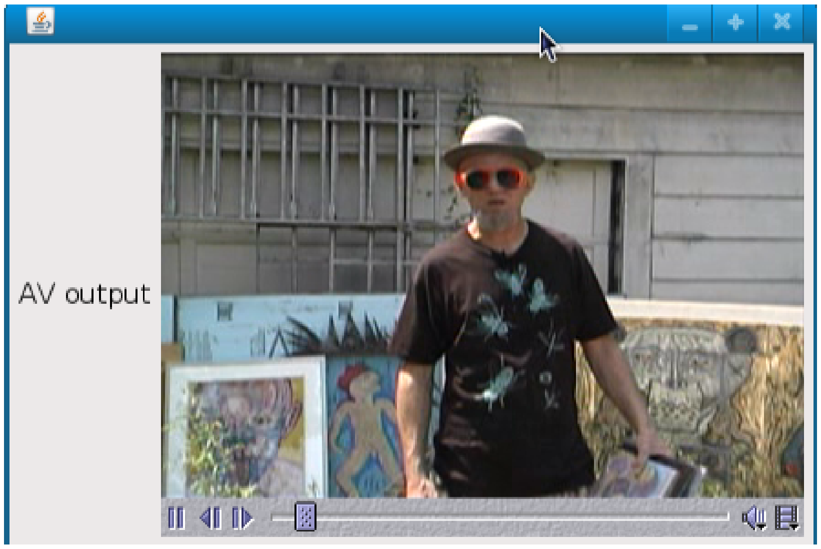
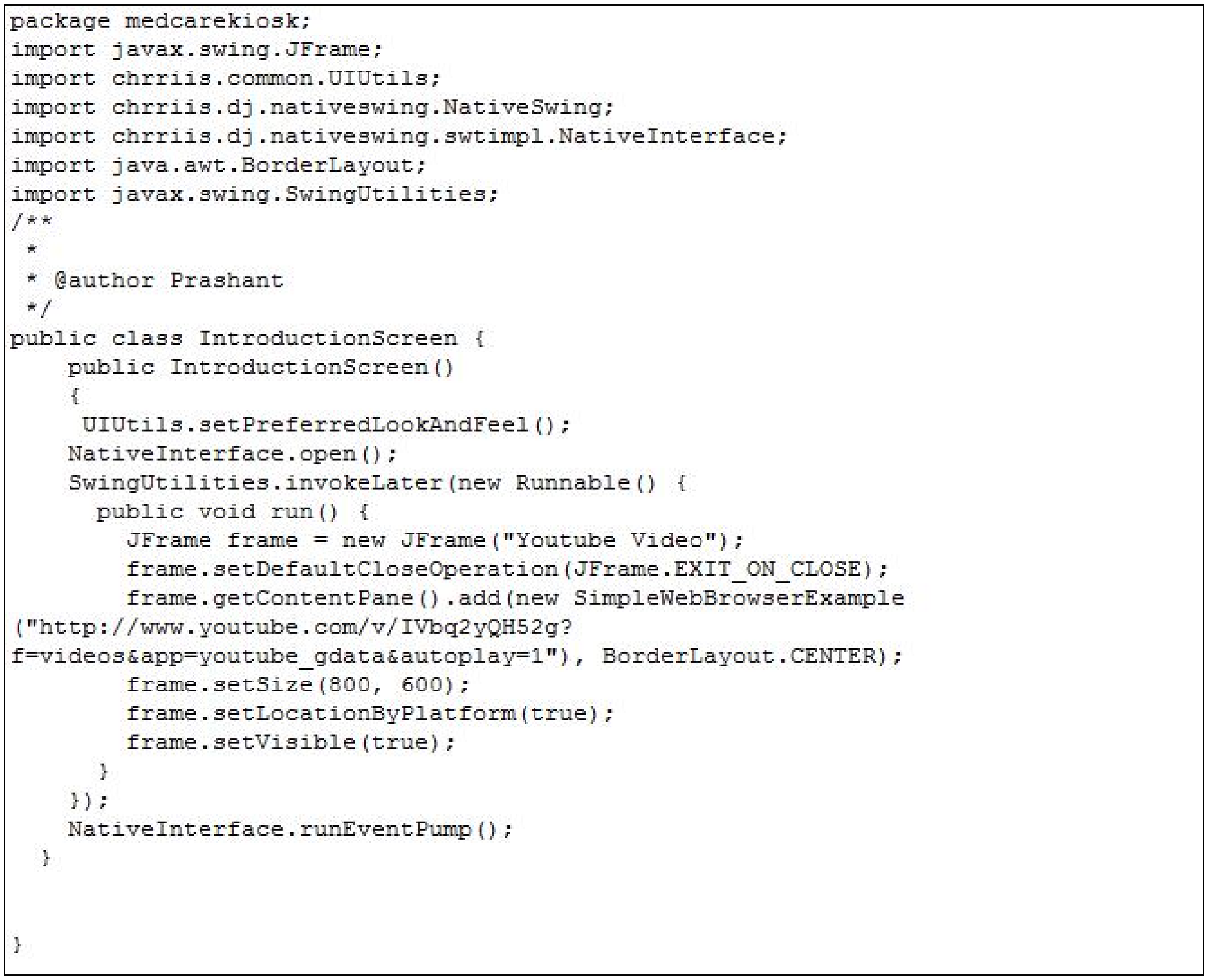


Fig 5.6: JMStudio Player

Stage 5: Help video

A help button is displayed to the user on all the screens. This gives user the choice of watching a Youtube video which has the usability demo of the entire application and its features.



```
package medcarekiosk;
import javax.swing.JFrame;
import chrriis.common.UIUtils;
import chrriis.dj.nativeswing.NativeSwing;
import chrriis.dj.nativeswing.swtimpl.NativeInterface;
import java.awt.BorderLayout;
import javax.swing.SwingUtilities;
/**
 *
 * @author Prashant
 */
public class IntroductionScreen {
    public IntroductionScreen()
    {
     UIUtils.setPreferredLookAndFeel();
    NativeInterface.open();
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        JFrame frame = new JFrame("Youtube Video");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new SimpleWebBrowserExample
("http://www.youtube.com/v/IVbq2yQH52g?
f=videos&app=youtube_gdata&autoplay=1"), BorderLayout.CENTER);
        frame.setSize(800, 600);
        frame.setLocationByPlatform(true);
        frame.setVisible(true);
      }
    });
    NativeInterface.runEventPump();
  }


}
```

Fig 5.7 Code for Introduction Help Screen

5.2 Tools and Software Used

5.2.1 Video Playback:

There are two players I have used to play videos and audio from the server. I have used Adobe Flash player and Java Sound JMStudio[13]. Playing Youtube video was challenging with respect to the format Youtube videos are returned. The Youtube videos are returned in SWF format which cannot be streamed on normal video players like JMStudio and advanced players like VLC. Note, the Youtube used to allow there video to be streamed from players like VLC but they do not any more. In order to play Youtube video I needed a Java Web player and I have used third party player from DJ Project [6].

The Youtube videos are automatically played full screen using this URL format. We use the following Youtube link format to stream the video.

http://www.youtube.com/v/VIDEO_ID?f=videos&app=youtube_gdata&autoplay=1

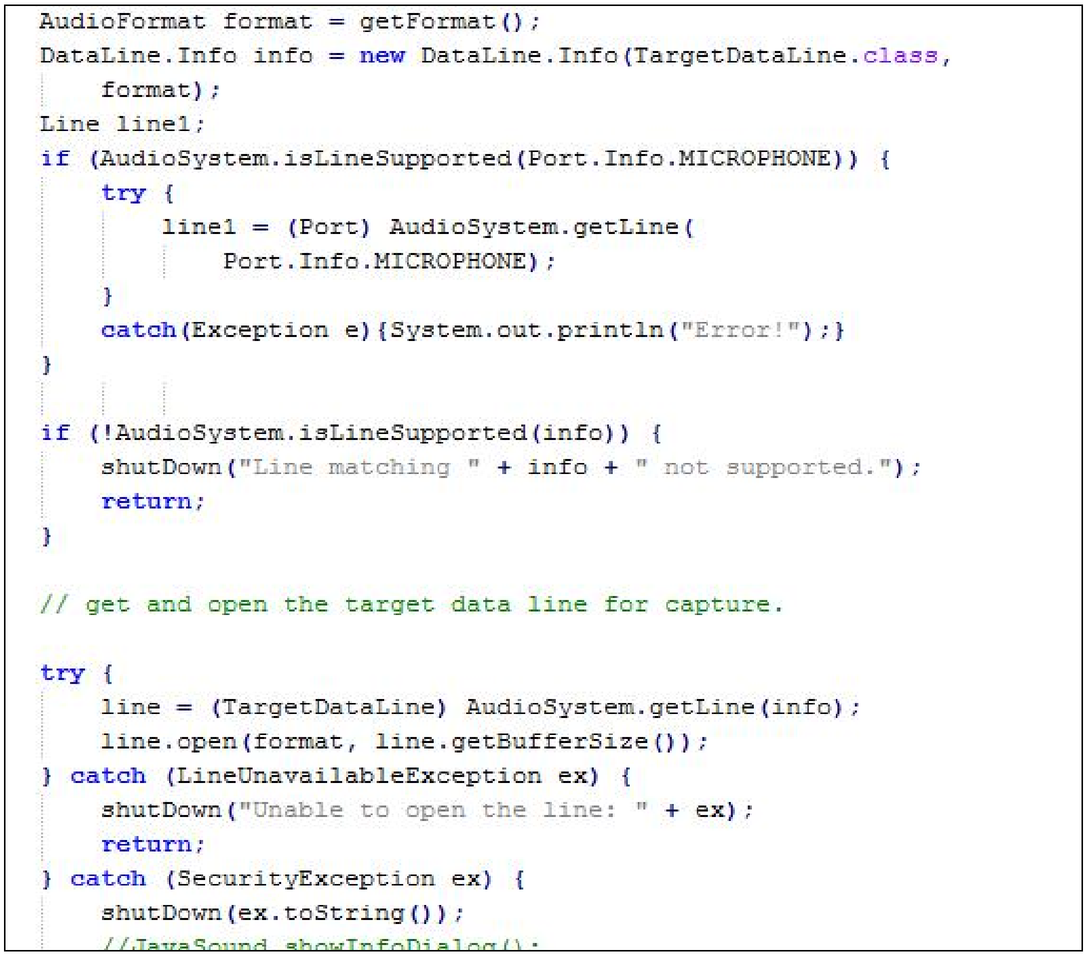
In such a link, we would only need Youtube videoID, obtained from InfoKiosk Information Base, to stream the video.

To play an audio file in common format like WAV we use JMStudio player which is provided by Oracle along with (Java Media Framework) JMF framework package. JMF is a module which handles audio and video files in Java; and included in it is media player called Java Media Studio (JMStudio). JMStudio finds the customized player, to play the audio, based on properties such as Audio codec and video codec. Often times it fails to find a codec and would be unable to handle the format. Jffmpeg plug-in [14] is used to playback a number of audio and video formats. For example, audio formats supported are MP3, AC3 and Vorbis; video formats supported are H263, MPEG, WMV and more.

In the case when the audio is sent from server it is first downloaded and played back using JMStudio player.

5.2.2 Audio Request Recording:

Voice recording has been done using Java Sound API. Java Sound accepts audio in predefined formats only and it also based on the audio-video support of the system. It was observed that the most generally used audio formats like mpeg, avi were not successfully handled using basic Java Sound API. In order to accommodate the deficiency, I have used JFFmpeg, free software licensed under the LGPL/GPL. It is a cross-platform solution to record, convert and stream media - audio and video.



```
AudioFormat format = getFormat();
DataLine.Info info = new DataLine.Info(TargetDataLine.class,
    format);
Line line1;
if (AudioSystem.isLineSupported(Port.Info.MICROPHONE)) {
    try {
        line1 = (Port) AudioSystem.getLine(
            Port.Info.MICROPHONE);
    }
    catch(Exception e){System.out.println("Error!");}
}

if (!AudioSystem.isLineSupported(info)) {
    shutDown("Line matching " + info + " not supported.");
    return;
}

// get and open the target data line for capture.

try {
    line = (TargetDataLine) AudioSystem.getLine(info);
    line.open(format, line.getBufferSize());
} catch (LineUnavailableException ex) {
    shutDown("Unable to open the line: " + ex);
    return;
} catch (SecurityException ex) {
    shutDown(ex.toString());
    //JavaSound.showInfoDialog();
```

Fig 5.8: Code to Set the Audio Format

The audio format supported by JMstudio player is defined in the class javax.sound.sampled.AudioFormat. Sound is defined by fields such big-endian or little-endian storage format, number of channels, type of encoding, and frame size, frame rate, sample size in bits and sample rate. The speech-to-text converter used in the thesis, Lumenvox, requires audio to be fed in a specific format. Following are the values of the parameters used:

i. Storage Format: little-endian format.

ii. Number of Channel: Mono channel.

iii. Sample Size: 16 bits

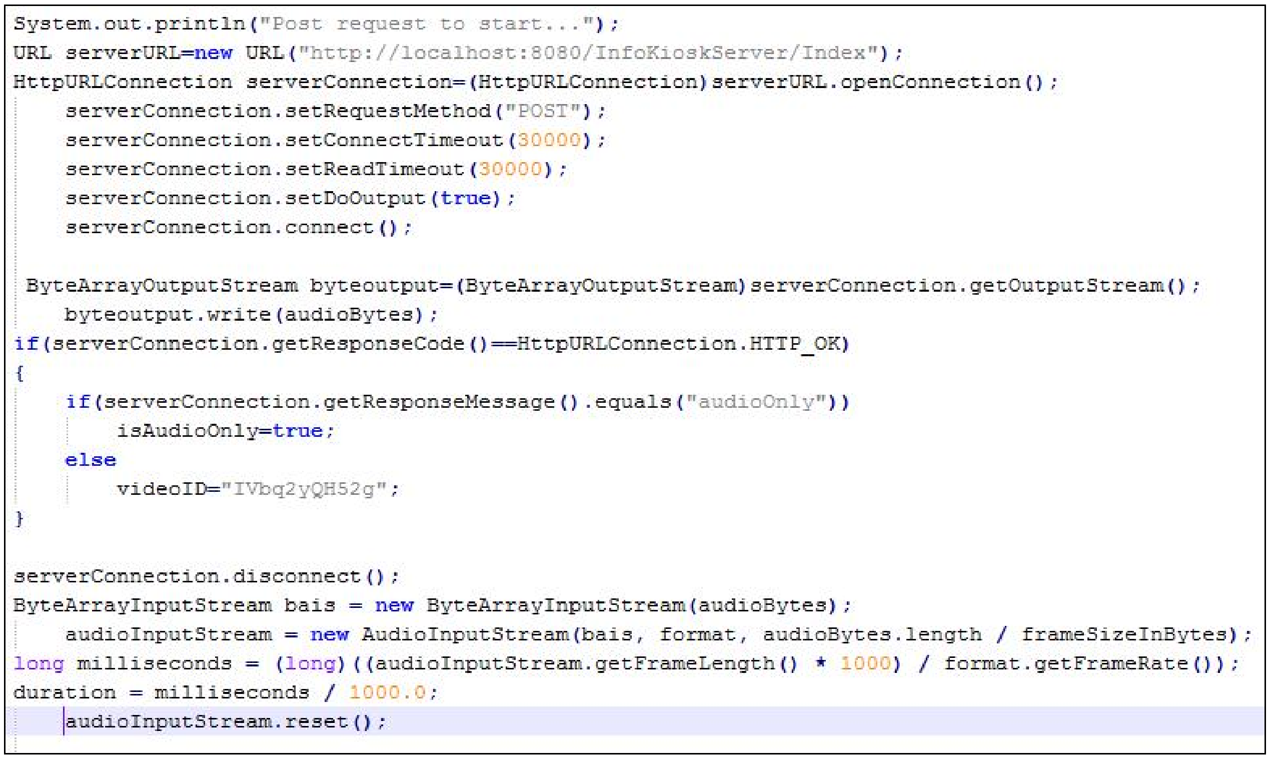
iv. Sample Rate and Frame Rate: 8KHz

v. Encoding: PCM Signed

vi. Frame Size: (Sample Size/8)*channels

5.3 Evaluation

InfoKiosk has been tested to check following features of the application – content understandability and ease of navigation. InfoKiosk has been designed keeping the non-literate or semi literate population in India as a case in point. In general, a user makes a request for certain information in a particular language. If the user is given the requested information in the desired language and format then the test case is said to have successfully passed.



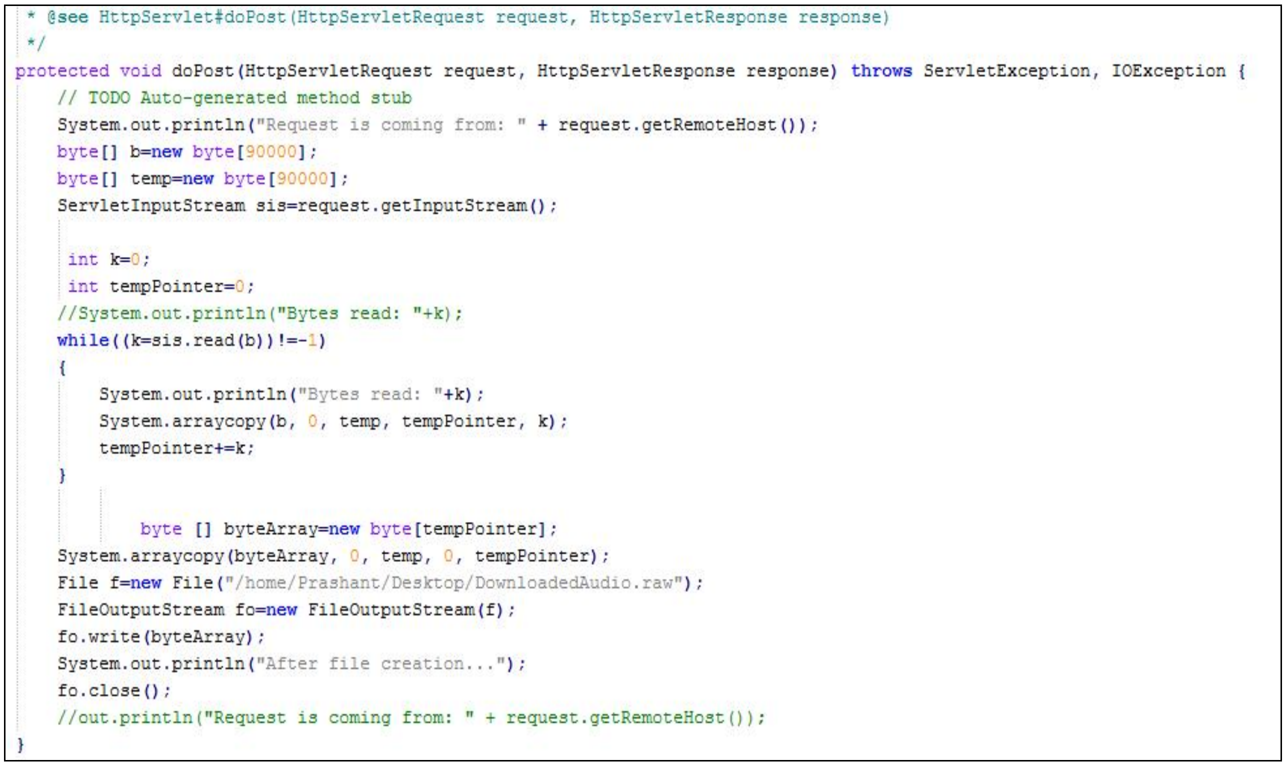
```
System.out.println("Post request to start...");
URL serverURL=new URL("http://localhost:8080/InfoKioskServer/Index");
HttpURLConnection serverConnection=(HttpURLConnection)serverURL.openConnection();
    serverConnection.setRequestMethod("POST");
    serverConnection.setConnectTimeout(30000);
    serverConnection.setReadTimeout(30000);
    serverConnection.setDoOutput(true);
    serverConnection.connect();

 ByteArrayOutputStream byteoutput=(ByteArrayOutputStream)serverConnection.getOutputStream();
    byteoutput.write(audioBytes);
if(serverConnection.getResponseCode()==HttpURLConnection.HTTP_OK)
{
    if(serverConnection.getResponseMessage().equals("audioOnly"))
        isAudioOnly=true;
    else
        videoID="IVbq2yQH52g";
}

serverConnection.disconnect();
ByteArrayInputStream bais = new ByteArrayInputStream(audioBytes);
    audioInputStream = new AudioInputStream(bais, format, audioBytes.length / frameSizeInBytes);
long milliseconds = (long)((audioInputStream.getFrameLength() * 1000) / format.getFrameRate());
duration = milliseconds / 1000.0;
    audioInputStream.reset();
```

Fig 5.9: Client Sending Audio bytes over HTTP

The test cases have been run against the InfoKioskServer, a server which accepts the client’s audio data and creates a “.raw” audio file from it. The InfoKiosk application and InfoKiosk server communicate using HTTP protocol. This audio file is fed to Lumenvox code for decoding and recognizing the information requested. Depending on the availability of the requested information and its format, a response is generated for the client. Figure 5.9 gives the code for connecting to InfoKioskServer using HTTP and sending audio data using POST method.



```
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
    System.out.println("Request is coming from: " + request.getRemoteHost());
    byte[] b=new byte[90000];
    byte[] temp=new byte[90000];
    ServletInputStream sis=request.getInputStream();

     int k=0;
     int tempPointer=0;
    //System.out.println("Bytes read: "+k);
    while((k=sis.read(b))!=-1)
    {
        System.out.println("Bytes read: "+k);
        System.arraycopy(b, 0, temp, tempPointer, k);
        tempPointer+=k;
    }

            byte [] byteArray=new byte[tempPointer];
    System.arraycopy(byteArray, 0, temp, 0, tempPointer);
    File f=new File("/home/Prashant/Desktop/DownloadedAudio.raw");
    FileOutputStream fo=new FileOutputStream(f);
    fo.write(byteArray);
    System.out.println("After file creation...");
    fo.close();
    //out.println("Request is coming from: " + request.getRemoteHost());
}
```

Fig 5.10 : InfoKioskServer handling the client data

The InfoKioskServer would accept such post request and form a RAW format audio file. The code handling the client request is shown in Figure 5.10.

Table 5.2 InfoKiosk Test Cases

	Test Case	Steps Involved	Results
1.	Requested Information is available as Youtube video. Sub Cases: - All requested Language - All requested domains	Client: a. Selecting a language. b. Selecting a domain of interest c. Recording and submitting the audio request. d. Stream the Youtube video based on VideoID. Server: a. Request is decoded using Lumenvox. b. Select corresponding Youtube VideoID and return to client.	Successfully Completed.
2.	Requested Information is available as video on server. Sub Cases: - All requested Language - All requested domains	Client: a. Selecting a language. b. Selecting a domain of interest c. Recording and submitting the audio request. d. Play the video after downloading it through link obtained from server. Server: a. Request is decoded using Lumenvox. b. Select corresponding video link and return to client.	Successfully Completed.

Table 5.2 continued …

	Test Case	Steps Involved	Results
3.	Requested Information is available in same or different language as text	Client: a. Selecting a language. b. Selecting a domain of interest c. Recording and submitting the audio request. d. Download the Audio and play Server: a. Request is decoded using Lumenvox. b. Select the available text and input it to Text-to-Text converter and then to Text-to-Speech converter. c. Send the audio link back to Client.	To Implement
4.	Requested Information available in different languages as audio/video	Client: a. Selecting a language. b. Selecting a domain of interest c. Recording and submitting the audio request. d. Download the Audio and play Server: a. Request is decoded using Lumenvox. b. Select the available audio/video and call speech-to-speech converter. c. Send the audio link back to client.	To Implement
5.	Requested information not available	Server a. Send Audio bytes which say “No Information” in the user selected language	To Implement

While I have tried to include all the test cases but there are cases which are not covered due to the lack of availability of such features. The above test cases do include cases when requested information is available in different language or when there is no information available. This case with information available in different language requires a text-to-text converter, which is seen as one of the future improvement to the application

Another aspect that has been studied under during the evaluation of the InfoKiosk application is its ability to portable to different environment – it other terms its flexibility. The two stages of InfoKiosk application – Language Selection, Information Domain Selection – provide the flexibility by defining the button-based selection procedure. Language selection in a button-based selection mode involves defining a language specific button with an appropriate image. So adding or removing a new language involves adding or removing a new button with a corresponding image. As shown in figure 5.1, with a small image size we can show 9 different language categories. Similarly the Domain selection, as shown in figure 5.2, also reflects similar flexibility. Figure 5.11 briefly shows kind of information exchanged in a typical InfoKiosk application.

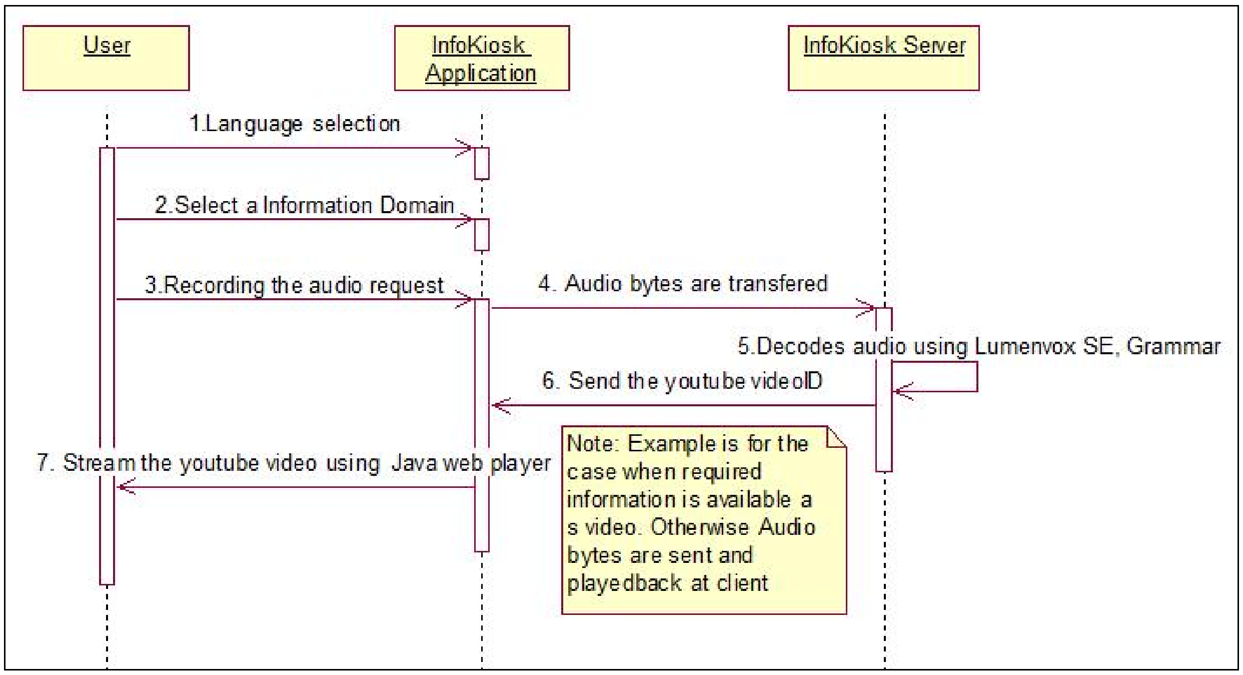

User
InfoKiosk Application
InfoKiosk Server
1.Language selection
2.Select a Information Domain
3.Recording the audio request
4. Audio bytes are transfered
5.Decodes audio using Lumenvox SE, Grammar
6. Send the youtube videoID
7. Stream the youtube video using Java web player
Note: Example is for the case when required information is available a s video. Otherwise Audio bytes are sent and playedback at client

Fig 5.11 Sequence diagram for InfoKiosk Application

CHAPTER 6

CONCLUSION AND FUTURE WORK

As the target user for InfoKiosk application is semi-literate and non-literate population, one way to imagine the possible amount of usage of such application can be the population of target audience. India currently has largest percentage of non-literate population of any nation on earth. The population of adult illiterates in India [17] is 291 million out of world count of 796 million. So the application has the chance to be influence a very large volume of consumers.

Application is scalable with respect to the languages and applicable domain. Each domain or language has can be selected by click of a button so adding a new domain or language would mean adding a new button. I have created application with an example of public health and agriculture information domain but it can be used for any domain of choice of the user.

I understand the importance of testing an application and regret that we cannot test the application using target users. In future if possible we can get feedback after testing application against target user. Such a feedback would let us improve the usability aspect of the application. For example - currently we have only one stage on the application which has majority audio only interaction i.e. the User Request (Audio) phase. If the user is seen comfortable to such an interaction we can extend it to other phases like languages selection.

India has around 617 million mobile subscribers [37], second largest in the world. The increasing number of phones with access to web will make an InfoKiosk mobile application a very lucrative option. India entered 3G arena in 2008 with Government led Bharat Sanchar Nigam Limited (BSNL) providing mobile and data services. The launch of 3G services by private mobile service provider starts November 2010 [38]. It would decrease the hardware and labor cost as it would require no installation and it also makes the product mobile, maintainable and more reachable to common people. InfoKiosk as a mobile application would follow same stages of design illustrated in Section 5.1, thus not deprecating users of its benefits. One of the hindrances of using InfoKiosk as mobile application would be the cost of a smartphone, a phone with advanced computing capability and connectivity. The price of a smartphone in India is Rs 20,000 and above [39]. With the audience for our application in mind, such a smartphone can be shared between the people in a under a community in a village. Developing text free UIs principles for mobile phones has also been suggested in [25, 16]. Lalji and Good [16] have used user-centered, incremental design approach and discuss mobile phone design approach for non-literate persons.

A sample android phone application for InfoKiosk was created and figure 6.1 and 6.2 are the screenshots.

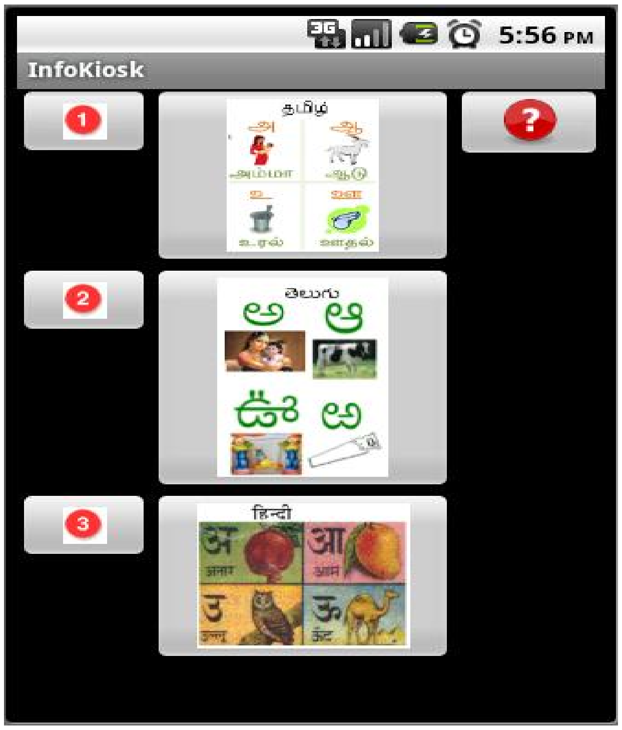


Fig 6.1: Prototype Android Application – Language Selection

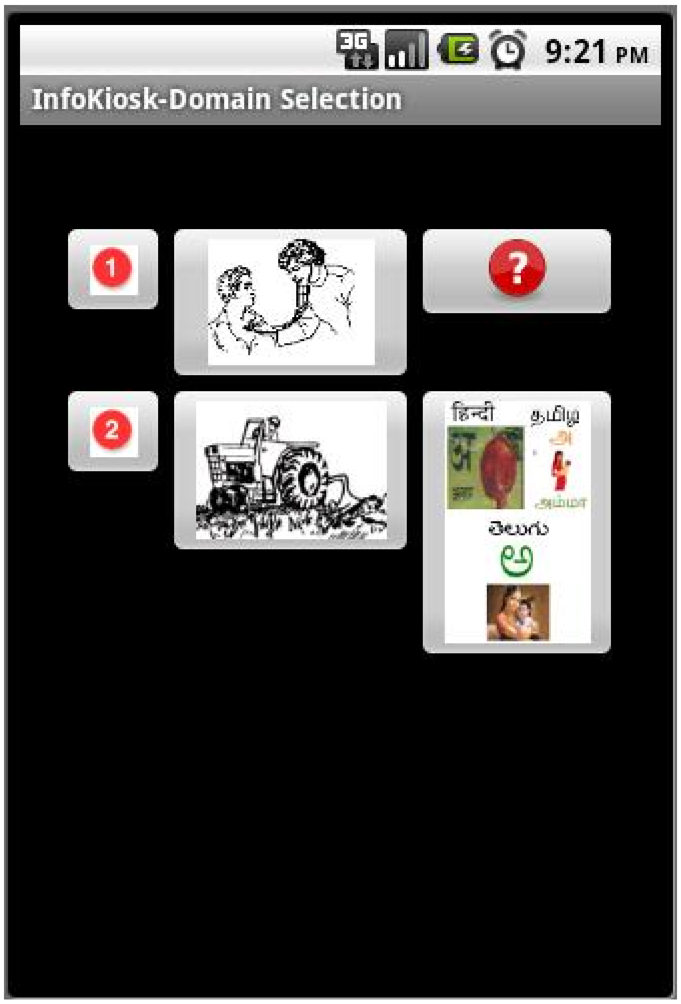


Fig 6.2: Prototype Android Application – Domain Selection

Each domain of InfoKiosk application is defined to handle specific set of requests and is not all compassing. For example, on selection of medical domain would handle request in a specific format such as <disease-name><request-in-a-specific-language>. Disease domain would give general information about a particular disease as available in the media such as internet article or Youtube video. InfoKiosk cannot provide a knowledgeable feedback if audio request is a question. The application would not be able to process request like “I have

102 F temperature, what medicine should I take?”. Its inability comes due to the fact that the grammar defined for Speech Recognition Engine is not defined to handle it.

APPENDIX

I. Lumenvox Program

```
#include <LVSpeechPort.h>

#include <iostream>
#include <fstream>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

void run_decode(std::string ip,std::string grammar_name)
{

         std::string FileName = grammar_name;
        SOUND_FORMAT audio_format = PCM_8KHZ;
       std::string grammar_fn="publicHealth.gram";

        LVSpeechPort port;
        //1. Open Audio file
        int FileHandle=open(FileName.c_str(), O_RDONLY, S_IREAD);

        if (FileHandle == -1)
        {
              printf("Cannot open the file %s \n",FileName.c_str());
              return;
        }

        struct stat temp_stat;
        //2. Getting statistics of the audio file e.g. length of file.
        if (stat(FileName.c_str(),&temp_stat) == -1)
          {
              close(FileHandle);
              printf("Cannot get size of file %s\n",FileName.c_str());
              return;
        }

        unsigned  long Length=temp_stat.st_size;
        //3. Creating map file for the the audio file.
        void *MAP = mmap(NULL,Length,PROT_READ,MAP_PRIVATE |
MAP_DENYWRITE,FileHandle,0);
        if (!MAP || MAP == (void *)0xffffffff)
        {
         close(FileHandle);
         printf( "Cannot create a mapfile from %s\n",FileName.c_str());
         return ;
        }

        char *C = (char *)MAP;

         //4. Point the client library to a local server and a remote server
        port.SetClientPropertyEx(PROP_EX_SRE_SERVERS,
PROP_EX_VALUE_TYPE_STRING,(void *)ip.c_str());
        unsigned int count = 0;
        int retval;

        printf("Connecting to %s\n",ip.c_str());

        while(count<5)
        {
        //5. Opening the Speech port
         if((retval=port.OpenPort())!=0)
          {
            printf("OpenPort() failed, errorcode returned %d\n",retval);
          }
        //6. Set properties of port
        port.SetPropertyEx(PROP_EX_DECODE_TIMEOUT, PROP_EX_VALUE_TYPE_INT,
(void*)50000, PROP_EX_TARGET_PORT);

        int vc = 1;
          //7. Adding Audio
        if((retval=port.LoadVoiceChannel(vc, C, Length, audio_format))!=0)
        {
          printf("LoadVoiceChannel failed, errorcode returned %d\n",retval);
        }

        //8. Working with Grammars
        if((retval=port.LoadGrammar("blah", grammar_fn.c_str())!=0))
        {
          printf("LoadGrammar() failed, errorcode returned %d\n",retval);
        }

        if((retval=port.ActivateGrammar("blah"))!=0)
        {
          printf("ActivateGrammar() failed, errorcode returned %d\n",retval);
        }

        //9.Decoding
        // this logs the responses that client receives to <installdir>/Lang/Responses directory
        port.SetProperty(PROP_SAVE_SOUND_FILES,1);
        int retv = port.Decode(vc, LV_ACTIVE_GRAMMAR_SET, LV_DECODE_BLOCK |
LV_DECODE_SEMANTIC_INTERPRETATION );

        //# of interpretations
        int numInterp = port.GetNumberOfInterpretations(vc);

        printf("Number of Interpretation %d\n",numInterp);
        for (int t = 0; t < numInterp; ++t)
        {
                printf("Interpretation %i:\n%s\n",t+1,port.GetInterpretationString(vc,t));
                printf("Interpretation Score :%d\n\n",(port.GetInterpretation(vc,t)).Score());
        }
        if(count<=3)
        printf("count=%d, decode returns %d\n", count, retv);

                ++count;
    //close port
        port.ClosePort();
        }

        msync(MAP,Length,MS_SYNC);
        munmap(MAP,Length);
        close(FileHandle);
}


int main(int argc,char *argv[])
{
        if(argc <2)
        {
                printf("Lumenvox Lite Command: %s SERVER_IP GRAMMAR_NAME \n",argv[0]);
                return -1;
        }
      run_decode(argv[1],argv[2]);

return 0;
```

REFERENCES

1. Adult And Youth Literacy: Global Trends in Gender Parity UNESCO Institute for Statistics, http://www.unesco.org/education/ild2010/FactSheet2010_Lit_EN.pdf ; accessed on 11/12/2010

2. Chowdhury, R., and Medhi, D. e-System for Public Health in India: Towards an Architectural Framework Incorporating Illiteracy and Linguistic Diversity, in Systems Thinking and e-Participations: ICT in the Governance of Society, edited by J. Cordoba-Pachon and A. Ochoa-Arias, IGI Global, pp. 69—91, 2010

3. Digital Green About page, http://www.digitalgreen.org/aboutus/; accessed on 11/12/2010

4. Digital Green Overview of farmer, http://www.digitalgreen.org/overviewfarmer/ ; accessed on 11/12/2010

5. Digital Green Standard of operation, http://www.digitalgreen.org/sop/ ; accessed on 11/7/2010

6. DJ Native Swing, http://djproject.sourceforge.net/ns/; accessed on 11/12/2010

7. Donner, J., Gandhi, R., Javid, P., Medhi, I., Ratan, A., Toyoma, K., and Veeraraghava, R. Stages of Design in Technology for Global Development. *IEEE Computer* 41(6) 34-41 (2008)

8. FAQ about UIS Literacy Data, http://www.uis.unesco.org/TEMPLATE/pdf/Literacy/FAQlit.pdf ; accessed on 11/12/2010

9. Garai, A., and Shadrach, B. *Processes and Appropriation of ICT in Human Development. In Rural India: Bridging the Research and Practice Gaps*. Retrieved on August 15th 2008, from http://www.dgroups.org/groups/oneworld/OneWorldSA/docs/TICTEIV_pdf.pdf

10. Indian Literacy projects, http://www.ilpnet.org/AboutILP ; accessed on 11/1/2010

11. Jamithreddy, V.R.K. , Socio-Cultural Communication System – A Communication Mechanism for Multi-Media Information Access System for Non-literate and Linguistically Diverse Users, MS Thesis, University of Missouri-Kansas City, 2010

12. Java sound and JMF: http://java.sun.com/products/java-media sound/techReference/javasoundfaq.htmlt#formats ; accessed on 11/25/2010

13. Java Sound Links, http://www.oracle.com/technetwork/java/index-jsp-140234.html; accessed on 11/12/2010

14. JFFMPEG : http://jffmpeg.sourceforge.net/download.html ; accessed on 11/25/2010

15. Julius Speech Recognition Engine, http://julius.sourceforge.jp/en_index.php; accessed on 11/12/2010

16. Lalji, Z. and Good, J., Designing new technologies for illiterate populations: A study in mobile phone interface design, *Interacting with Computers*, 20 (2008) 574-586.

17. Literacy in India: Wikipedia Article, http://en.wikipedia.org/wiki/Literacy_in_India; accessed on 11/12/2010

18. Literacy situation in India, http://www.roomtoread.org/Page.aspx?pid=304 ; accessed on 11/12/2010

19. Lumenvox Resources, http://www.lumenvox.com/resources/ ; accessed on 11/12/2010

20. Lumenvox Speech Engine: http://www.lumenvox.com/products/speech_engine/; accessed on 12/5/2010

21. Lumenvox Speech Recognition Engine, http://www.lumenvox.com/resources/tips/HowLVsoftwareUsed.aspx; accessed on 11/12/2010

22. Lumenvox Speech Recognition Introduction http://www.lumenvox.com/resources/tips/SpeechRecognitionSolutions.aspx ; accessed on 12/4/2010

23. Lumenvox Speech Recognition solution: http://lumenvox.com/resources/; accessed on 12/1/2010

24. Lumenvox Wikipedia page: http://en.wikipedia.org/wiki/LumenVox; accessed on 12/4/2010

25. Medhi, I., Profile, http://research.microsoft.com/en-us/people/indranim/; accessed on 11/12/2010

26. Medhi, I., Pitti B., and Toyama K. Text-Free UI for Employment Search. *Asian Applied Computing Conference*. Nepal, (2005).

27. Medhi, I., Sagar, A., and Toyama K. Text-Free User Interfaces for Illiterate and Semi-Literate Users. In Proceedings of IEEE*/ACM International Conference on Information and Communication Technologies and Development*, Berkeley, USA, 2006.

28. Phillips, A. and Davis, M. (Eds.), Tags for Identifying Languages, IETF RFC 4646, September 2006.

29. Simputer Media Coverage, http://news.bbc.co.uk/2/hi/science/nature/1442000.stm ; accessed on 11/12/2010

30. Simputer Organization, http://simputer.org/simputer/ ;accessed on 11/12/2010

31. Simputer Overview, http://en.wikipedia.org/wiki/Simputer ; accessed on 11/12/2010

32. UNESCO Institute of Statistics, http://www.uis.unesco.org/en/stats/statistics/literacy2000.htm; accessed on 11/12/2010

33. UNESCO Illiteracy Projection, http://www.uis.unesco.org/en/stats/statistics/UIS_Literacy_Country2002.xls ; accessed on 11/12/2010

34. UNICEF India Statistics, http://www.unicef.org/infobycountry/india_statistics.html ; accessed on 11/12/2010

35. Unicode Language Tagging: Ch 5 Pg150. http://unicode.org/versions/Unicode5.2.0/ch05.pdf; accessed on 12/5/2010

36. VoxForge, http://voxforge.org/ ; accessed on 11/12/2010

37. Volume of Indian mobile subscribers, http://www.ciol.com/Technology/Mobility/News-Reports/Mobile-subscriber-base-crosses-617-million-mark/138238/0/ ; accessed on 11/12/2010

38. Wireless services: 3G, http://en.wikipedia.org/wiki/3G; accessed on 11/18/2010.

39. Wireless phone prices in India : http://www.fonearena.com/mobile_phone_pricelist.html; accessed on 11/18/2010

VITA

Prashant Sunkari was born in Hyderabad, Andhra Pradesh, India and was awarded under-graduation degree in Computer Science & Engineering at Jawaharlal Nehru Technological University. He did his under-graduation final year project at Wipro Technologies, India on “Enterprise level VoIP application: Automated provisioning of Ekiga Instant Messaging tool”.

After completing his under-graduation, he decided to pursue his interest in learning more about Computer Networking and Software development at University of Missouri Kansas City. While doing his Master’s degree he gained valuable industrial experiences by working at Billing Tester Intern at National Insurance Producer Registry (NIPR) and Software Engineer Intern/Co-op at Garmin International.