7. Evaluation

Evaluation is done by selecting various kind of CEB call employees with a different kind environments. Different employees used to evaluate the pronunciation effect on the application and how the implementation a reach to overcome different pronunciations. Also different environments used to evaluate how the environmental aspects especially artificial and natural noise effect to the recognition process

8. Discussion

This report's intend to discuss the interim progress of the research. As a summary, over all this report discuss the purpose and intend to do the research and how the progress up to current stage. Design and implementation chapters include high level facts regarding the Software and Identified Functional and Non Functional requirements to support the hypothesis.

Literature review contains some details regarding related works done by the different people and some technologies used to do them. Main difference in this research is Sinhala language model to identify Sinhala words. Although we have lot of implemented language models for other languages, for Sinhala it is very rare to find out good language model. By doing that research is planning to use Sphinx 4, which is a framework consist of several java libraries to support voice recognition.

Basically this research output implementation has major functionalities to cater with. They are as follows

1. Create and Train Acoustic model
2. Read and record user input from microphone and pass it to recognizer
3. Analyze the audio input and using the trained acoustic model to identify the digit and pass it to the front end
4. Get the recognizer identified result and display it in the application. Handle the input events such as start of speech, end of speech etc...

From above, Step 1 and step 3 are the most critical and researching parts of the overall activity.

9. Conclusion

In this research we have try to achieve to implement efficient digit recognition system in Sinhala language to support Ceylon electricity board call center application. Once the system identify the account number of the complaint call center application can retrieve his/her details from the database for further processing. Implementation of the research used JAVA based speech libraries to work with trained acoustic model for archiving the research targets. But still those acoustic models needs to do some tuning and training to make the recognition more accurate

10. Future Work

As the future work this research need to conduct to identify Sinhala words, then call center application can log the customer complain at the first place too. That is the main target of this research. So lot of training and recognition needs to do to create good dictionary for Sinhala language. Once it build we have to update and train it day by day to improve the efficiency.

Since Ceylon electricity board has its call centers all around the county and more age variation within its employees, with the time progress we can create good trained dictionary to Sinhala language with different pronunciations used around the country

11. References

[1] M. Daryl Ning, "Developing an Isolated Word Recognition System in MATLAB," 2010. [Online]. Available: http://in.mathworks.com/company/newsletters/articles/developing-an-isolated-word-recognition-system-in-matlab.html.

[2] Google, "Speech Processing," 2015. [Online]. Available: http://research.google.com/pubs/SpeechProcessing.html.

[3] Kanishka Rao, Fuchun Peng, Franc,oise Beaufays, "Automatic Pronounciation Verification For Speech Recognition," [Online]. Available: http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43262.pdf.

[4] sourceforge, "CMU Sphinx," 2015. [Online]. Available: http://cmusphinx.sourceforge.net/.

[5] G. Brandl, "sphinx-doc.org," 2015. [Online]. Available: http://sphinx-doc.org/sphinx.pdf.

[6] D. Paul, "Speech Recognition Using Hidden Makrov Models," 2010. [Online]. Available: https://www.ll.mit.edu/publications/journal/pdf/vol03_no1/3.1.3.speechrecognition.pdf.

[7] Bronto, "javasphinx User's Guide," 2012. [Online]. Available: https://bronto.github.io/javasphinx/.

[8] A. Hunt, "JSpeech Grammar Format," 2015. [Online]. Available: http://www.w3.org/TR/jsgf/.

12. Appendixes

Following code sniffs outline the major interface and functions of the implementation

```
package edu.cmu.sphinx.decoder;

import edu.cmu.sphinx.util.props.Configurable;

/**
 * Some API-elements shared by components which are able to produce
<code>Result</code>s.
 *
 * @see edu.cmu.sphinx.result.Result
 */
public interface ResultProducer extends Configurable {

    /** Registers a new listener for <code>Result</code>.
     * @param resultListener*/
    void addResultListener(ResultListener resultListener);


    /** Removes a listener from this <code>ResultProducer</code>-
instance.
     * @param resultListener*/
    void removeResultListener(ResultListener resultListener);
}
```

```
package edu.cmu.sphinx.recognizer;

import edu.cmu.sphinx.decoder.Decoder;

import edu.cmu.sphinx.decoder.ResultProducer;

import edu.cmu.sphinx.decoder.ResultListener;

import edu.cmu.sphinx.instrumentation.Monitor;

import edu.cmu.sphinx.instrumentation.Resetable;

import edu.cmu.sphinx.result.Result;

import edu.cmu.sphinx.util.props.*;


import java.util.ArrayList;

import java.util.Collections;

import java.util.List;


/**
```

```
 * The Sphinx-4 recognizer. This is the main entry point for Sphinx-
4. Typical usage of a recognizer is like so:
 * <p/>
 * <pre><code>
 *  public void recognizeDigits() {
 *      URL digitsConfig = new URL("file:./digits.xml");
 *      ConfigurationManager cm = new
ConfigurationManager(digitsConfig);
 *      Recognizer sphinxDigitsRecognizer
 *          = (Recognizer) cm.lookup("digitsRecognizer&quot");
 *      boolean done = false;
 *      Result result;
 * <p/>
 *      sphinxDigitsRecognizer.allocate();
 * <p/>
 *     // echo spoken digits, quit when 'nine' is spoken
 * <p/>
 *      while (!done) {
 *           result = sphinxDigitsRecognizer.recognize();
 *           System.out.println(&quot;Result: &quot; + result);
 *           done = result.toString().equals(&quot;nine&quot;);
 *      }
 * <p/>
 *      sphinxDigitsRecognizer.deallocate();
 *   }
 * </code></pre>
 * <p/>
 * Note that some Recognizer methods may throw an
IllegalStateException if the recognizer is not in the proper state
 */
public class Recognizer implements Configurable, ResultProducer {
```

```
    /** The property for the decoder to be used by this recognizer.
*/
    @S4Component(type = Decoder.class)
    public final static String PROP_DECODER = "decoder";

    /** The property for the set of monitors for this recognizer */
    @S4ComponentList(type = Monitor.class)
    public final static String PROP_MONITORS = "monitors";

    /** Defines the possible states of the recognizer. */
    public static enum State { DEALLOCATED, ALLOCATING, ALLOCATED,
READY, RECOGNIZING, DEALLOCATING, ERROR }

    private String name;
    private Decoder decoder;
    private State currentState = State.DEALLOCATED;

    private final List<StateListener> stateListeners =
Collections.synchronizedList(new ArrayList<StateListener>());
    private List<Monitor> monitors;

    public Recognizer(Decoder decoder, List<Monitor> monitors) {
        this.decoder = decoder;
        this.monitors = monitors;
        name = null;
    }

    public Recognizer() {
    }

    /* (non-Javadoc)
```

```
     * @see
edu.cmu.sphinx.util.props.Configurable#newProperties(edu.cmu.sphinx.u
til.props.PropertySheet)

     */

    @Override

    public void newProperties(PropertySheet ps) throws
PropertyException {

        decoder = (Decoder) ps.getComponent(PROP_DECODER);

        monitors = ps.getComponentList(PROP_MONITORS, Monitor.class);


        name = ps.getInstanceName();

    }




    /**

     * Performs recognition for the given number of input frames, or
until a 'final' result is generated. This method

     * should only be called when the recognizer is in the
<code>allocated</code> state.

     *

     * @param referenceText what was actually spoken

     * @return a recognition result

     * @throws IllegalStateException if the recognizer is not in the
<code>ALLOCATED</code> state

     */

    public Result recognize(String referenceText) throws
IllegalStateException {

        Result result = null;

        checkState(State.READY);

        try {

            setState(State.RECOGNIZING);

            result = decoder.decode(referenceText);

        } finally {
```

```
            setState(State.READY);
        }
        return result;
    }



    /**
     * Performs recognition for the given number of input frames, or
until a 'final' result is generated. This method
     * should only be called when the recognizer is in the
<code>allocated</code> state.
     *
     * @return a recognition result
     * @throws IllegalStateException if the recognizer is not in the
<code>ALLOCATED</code> state
     */
    public Result recognize() throws IllegalStateException {
        return recognize(null);
    }



    /**
     * Checks to ensure that the recognizer is in the given state.
     *
     * @param desiredState the state that the recognizer should be in
     * @throws IllegalStateException if the recognizer is not in the
desired state.
     */
    private void checkState(State desiredState) {
        if (currentState != desiredState) {
            throw new IllegalStateException("Expected state " +
desiredState
```

```
                    + " actual state " + currentState);
        }
    }

    /**
     * sets the current state
     *
     * @param newState the new state
     */
    private void setState(State newState) {
        currentState = newState;
        synchronized (stateListeners) {
            for (StateListener sl : stateListeners) {
                sl.statusChanged(currentState);
            }
        }
    }

    /**
     * Allocate the resources needed for the recognizer. Note this
method make take some time to complete. This method
     * should only be called when the recognizer is in the <code>
deallocated </code> state.
     *
     * @throws IllegalStateException if the recognizer is not in the
<code>DEALLOCATED</code> state
     */
    public void allocate() throws IllegalStateException {
        checkState(State.DEALLOCATED);
        setState(State.ALLOCATING);
```

```
        decoder.allocate();

        setState(State.ALLOCATED);

        setState(State.READY);

    }



    /**

     * Deallocates the recognizer. This method should only be called
if the recognizer is in the <code> allocated

     * </code> state.

     *

     * @throws IllegalStateException if the recognizer is not in the
<code>ALLOCATED</code> state

     */

    public void deallocate() throws IllegalStateException {

        checkState(State.READY);

        setState(State.DEALLOCATING);

        decoder.deallocate();

        setState(State.DEALLOCATED);

    }



    /**

     * Retrieves the recognizer state. This method can be called in
any state.

     *

     * @return the recognizer state

     */

    public State getState() {

        return currentState;

    }
```

```
    /** Resets the monitors monitoring this recognizer */
    public void resetMonitors() {
        for (Monitor listener : monitors) {
            if (listener instanceof Resetable)
              ((Resetable)listener).reset();
        }
    }


    /**
     * Adds a result listener to this recognizer. A result listener
is called whenever a new result is generated by the
     * recognizer. This method can be called in any state.
     *
     * @param resultListener the listener to add
     */
    @Override
    public void addResultListener(ResultListener resultListener) {
        decoder.addResultListener(resultListener);
    }


    /**
     * Adds a status listener to this recognizer. The status listener
is called whenever the status of the recognizer
     * changes. This method can be called in any state.
     *
     * @param stateListener the listener to add
     */
    public void addStateListener(StateListener stateListener) {
```

```
        stateListeners.add(stateListener);
    }


    /**
     * Removes a previously added result listener. This method can be
called in any state.
     *
     * @param resultListener the listener to remove
     */
    @Override
    public void removeResultListener(ResultListener resultListener) {
        decoder.removeResultListener(resultListener);
    }



    /**
     * Removes a previously added state listener. This method can be
called in any state.
     *
     * @param stateListener the state listener to remove
     */
    public void removeStateListener(StateListener stateListener) {
        stateListeners.remove(stateListener);
    }
    /* (non-Javadoc)
    * @see java.lang.Object#toString()
    */
    @Override
    public String toString() {
        return "Recognizer: " + name + " State: " + currentState;
    }}
```

```
package speechrecognition.view;

import java.awt.EventQueue;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

import edu.cmu.sphinx.frontend.util.Microphone;
import edu.cmu.sphinx.recognizer.Recognizer;
import edu.cmu.sphinx.result.Result;
import edu.cmu.sphinx.util.props.ConfigurationManager;
import speech.SpeakString;
import speechrecognition.digits.english.EnglishRecognizer;
import speechrecognition.digits.english.MainRecognizer;

public class AccountDetailView {

	private JFrame frame;
	private static JTextField accNoTxt;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					AccountDetailView window = new
AccountDetailView();
```

```
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });

        try {

            SpeakString spk = new SpeakString();
            EnglishRecognizer engRec = new EnglishRecognizer();
            String str = "";
            accNoTxt.setText(str);

            ConfigurationManager cm;
            cm = new
ConfigurationManager(MainRecognizer.class.getResource("englishdigits.
config.xml"));

            Recognizer recognizer = (Recognizer)
cm.lookup("recognizer");
            recognizer.allocate();

            // start the microphone or exit if the program if
this is not possible
            Microphone microphone = (Microphone)
cm.lookup("microphone");
            if (!microphone.startRecording()) {
                //System.out.println("Cannot start
microphone.");
                spk.dospeak("Cannot start microphone.",
"kevin16");
                recognizer.deallocate();
```

```
                        System.exit(1);
                    }

                    spk.dospeak("Enter Account Number. Please speak out
number by number.", "kevin16");

                    Result result = recognizer.recognize();

                    String resultText =
result.getBestFinalResultNoFiller();

                    str =
str.concat(engRec.convertStringToDigit(resultText));


                    while (str.length() < 10 && str.length() > 0) {

                        Result resultInLoop = recognizer.recognize();

                        str =
str.concat(engRec.convertStringToDigit(resultInLoop.getBestFinalResul
tNoFiller()));

                        accNoTxt.setText(str);

                    }


                    spk.dospeak("Data is Loading Wait For While !!!!",
"kevin16");
            } catch (Exception e) {


            }
        }


        /**
         * Create the application.
         */
        public AccountDetailView() {
            initialize();
        }


        /**
```

```
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 450, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        JLabel lblAccountNumber = new JLabel("Account Number");
        lblAccountNumber.setBounds(10, 31, 121, 14);
        frame.getContentPane().add(lblAccountNumber);

        accNoTxt = new JTextField();
        accNoTxt.setBounds(141, 28, 158, 20);
        frame.getContentPane().add(accNoTxt);
        accNoTxt.setColumns(10);

        JButton btnLoadDetails = new JButton("Load Details");
        btnLoadDetails.setBounds(309, 27, 115, 23);
        frame.getContentPane().add(btnLoadDetails);
    }
}
```