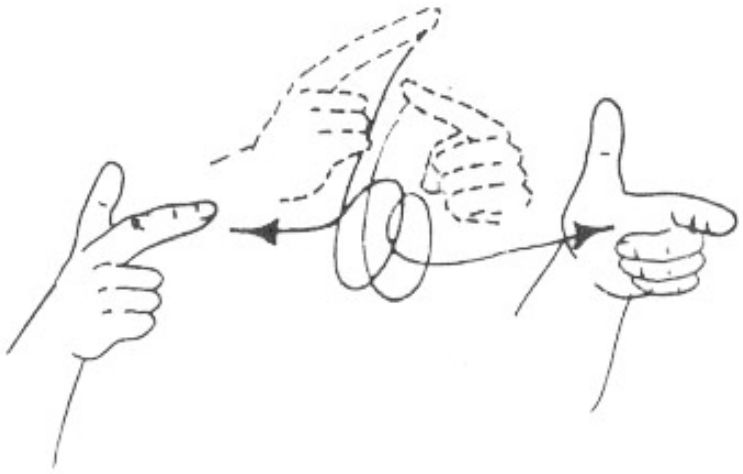
L.I.S.T.E.N.

Language Interpreter and Sign Translator for Educational Needs



Design Document

Authors:
Jennifer Crowell
Kulvir Sing
Abraham Evangelista
Susan Phillips
Sugnesh Patel

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide an overall description of the design and architecture of LISTEN. It will provide for technical and design issues and will also provide all the information required to recreate the database and the LISTEN application.

1.2 Scope

The scope of this document is to contain all information regarding the database and LISTEN application in as close to the final state as possible. It will use writing and UML diagrams to describe all the components of LISTEN. The UML diagram is for all the engines and all the classes that are required for each engine.

1.3 Definitions

GUI: Graphical User Interface.

LISTEN: Language Interpreter and Sign Translator for Educational Needs.

ASL: American Sign Language. This version of Sign Language does not have the same grammatical structure as the conversational English used by the non-deaf community.

Signed English: Signed English is a version of Sign Language that shares the same grammatical structure as the conversational English used by the non-deaf community.

Fingerspelling: The act of translating a word using the sign language alphabet.

Interpreter: A member of the non-deaf community whose occupation is to translate spoken word into either Signed English or ASL for a member of the deaf community.

1.4 Intended Audience

The intended audience for this document is developers only. Since this document is technical, it is recommended that the readers have background knowledge in order to understand some of the sections in this document. The reader should have some knowledge of UML diagramming, databases, OOP, and C# to understand the document at a high level.

1.5 Methodology, Tools, and Techniques

The waterfall methodology is being used for the development of the LISTEN application.

The following tools were used to develop the application: C# .NET, SQL Server 2005, and Dragon Naturally Speaking 9. A more in depth description of these will appear in the later sections of this document.

The techniques used include Object Oriented Programming. A more in depth description of this will be in a later section of this document.

1.6 Overview of the Remainder of the Document

The remainder of this document will cover system design and detailed design of all the classes for the LISTEN application. The content in the remainder will require some technical knowledge and the ability to read UML diagrams. The document will be as descriptive as possible, therefore the writers assume the reader has the technical knowledge necessary to understand the document.

2. Design Overview

2.1 Application Overview

The LISTEN application is divided into 4 major parts (3 engines and a mode). The Main engine is responsible for calling each mode and reading text from standard input. The Grammar engine is responsible for matching each string to a sign in the database. The Output engine is responsible for outputting the video indexed to the token. The four modes are Classroom, Presentation, Tutor, and Playback. The rest of the document will cover a more in-depth look at these 4 parts.

2.2 Constraints

Accuracy of translation is limited by the accuracy of the text stream provided by Nuance's Dragon Naturally Speaking 9.

2.3 Assumptions and Dependencies

2.3.1 Related Software and Hardware

C#/C++ .NET, and SQL Server 2005 will be used for the development of the LISTEN application, but the system also needs a third party software such as Nuance's Dragon NaturallySpeaking 9 to convert speech into text.

2.3.2 Operating System

The operating system that the LISTEN application will be developed on is Microsoft Windows XP Professional. However, the LISTEN application will be tested on all Microsoft Windows 2000 and later (includes Microsoft Windows Vista).

3. Interface

3.1 Overview

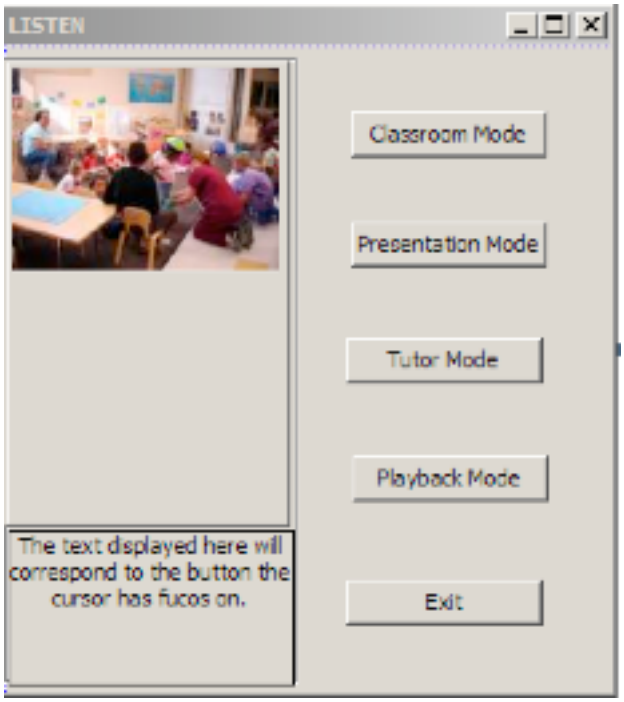
The User Interface is built around a Graphic Display window, which displays the translated Sign Language video. Depending on the selected mode of the system, this window can be accompanied by many other windows, which include a display of the token stream, a list of available words, and a control bar.

3.2 Interface Specifications

3.2.1 Main Menu

3.2.1.1 Three Selection Mode Radio Buttons: At the start of the Application, the following three selection buttons will be displayed.

- English Sign Translation Mode
- Tutor Mode
- Presentation Mode



3.2.1.2 *English Sign Translation Mode* - Upon selecting this mode, the text to English Sign translation Graphic Window displays.

3.2.1.3 *Tutor Mode* - This mode should take the User to the Tutor Window.

3.2.1.4 *Presentation Mo*de - This mode should set the Graphic Window Display and Tutor Window for the presentation display.

3.2.2 Graphic Display Window

This window will have the following parts:

3.2.2.1 Upper Window Pane: This pane displays the video or graphic clip. The window must be large enough to see the complete output clearly.

3.2.2.2 Lower Window Pane: This window pane can display any two of the following:

- **3.2.2.2.1** English text that is being translated to Signed English.
- **3.2.2.2.2** Text Description of the video or graphic clip when in Tutor mode.
- **3.2.2.2.3** Hide Button which hides the Lower Window Pane

3.2.2.3 Control Bar: The control bar can be displayed either on the bottom of Upper Window Pane or on the bottom of both panes. The bar will have the following components:

- **3.2.2.3.1** *Play and Pause Button:* Play button is used to play the Video or Graphic clip. While the video or graphic clip is playing, the Play text should change to Pause. The Pause button pauses the graphic display output.
- **3.2.2.3.2** *Rewind Button:* Rewind button will rewind the Graphic display output.
- **3.2.2.3.3** *Forward Button:* Forward button will forward the graphic output. This button is only active in playback mode.
- **3.2.2.3.4** *Repeat Button:* Repeat button will repeat the graphic output from start.
- **3.2.2.3.5** *Speed Slider:* Speed slider will control the speed of graphic output. Speed can go from slower to normal.

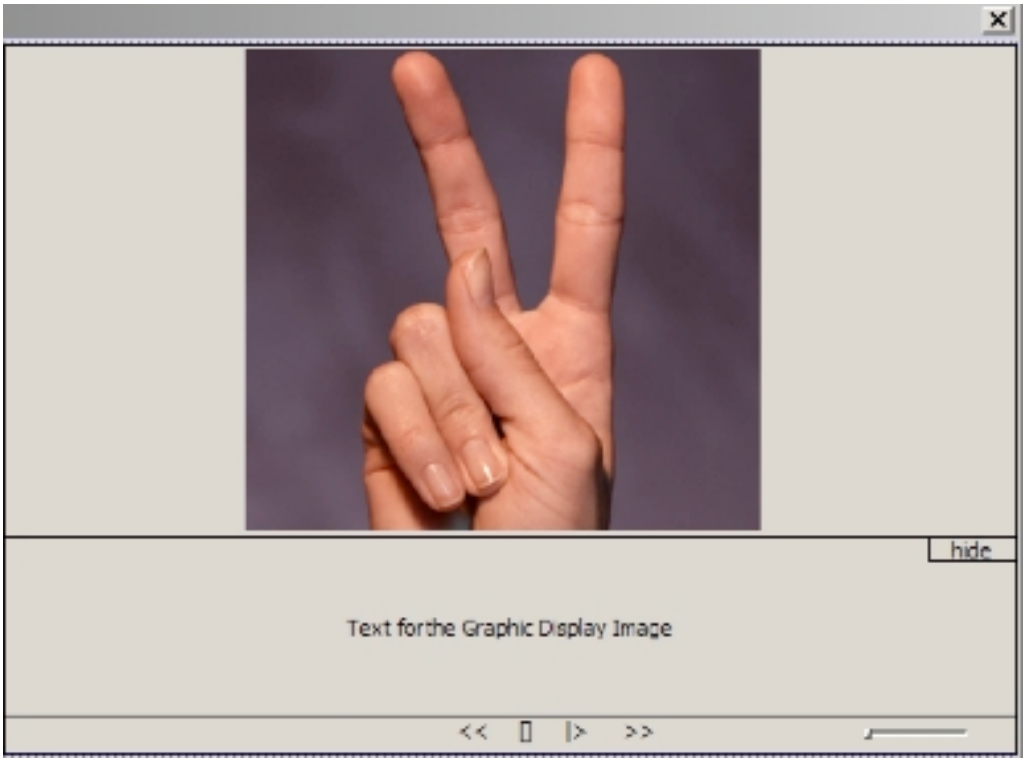


Figure 3.2.2 Graphic Display Window

3.2.3 Vertical Window Pane

This will be on left or right side of the Graphic Display Window. This will have the following parts:

3.2.3.1 Options Combo Box*:* This combo box will have Category, Sign English Dictionary, Alphabets and Numbers.

3.2.3.2 Expandable Words List*:* This contains the list of words. The list will display words depending on the selected option from the combo box. Following are the details about the Options combo box selection:

3.2.3.2.1 *Category:* If Category is selected from the Option combo box, all the Signed English categories will be listed.

3.2.3.2.2 *Signed English Dictionary:* This option will list all the English words existed in Signed English.

3.2.3.2.3 *Alphabetic and Numeric Characters:* This option will list the English alphabet and numbers.

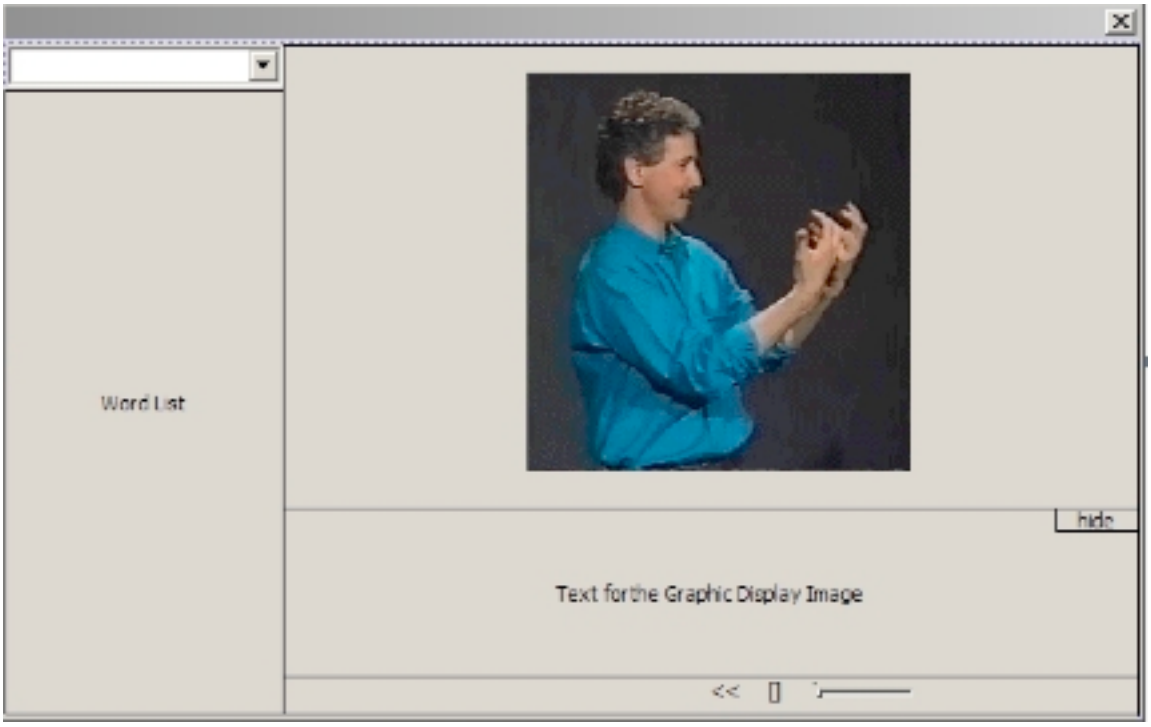


Figure 3.2.3 Graphic Display Window with Vertical Window Pane

4. Systems Architecture

4.1 Overview

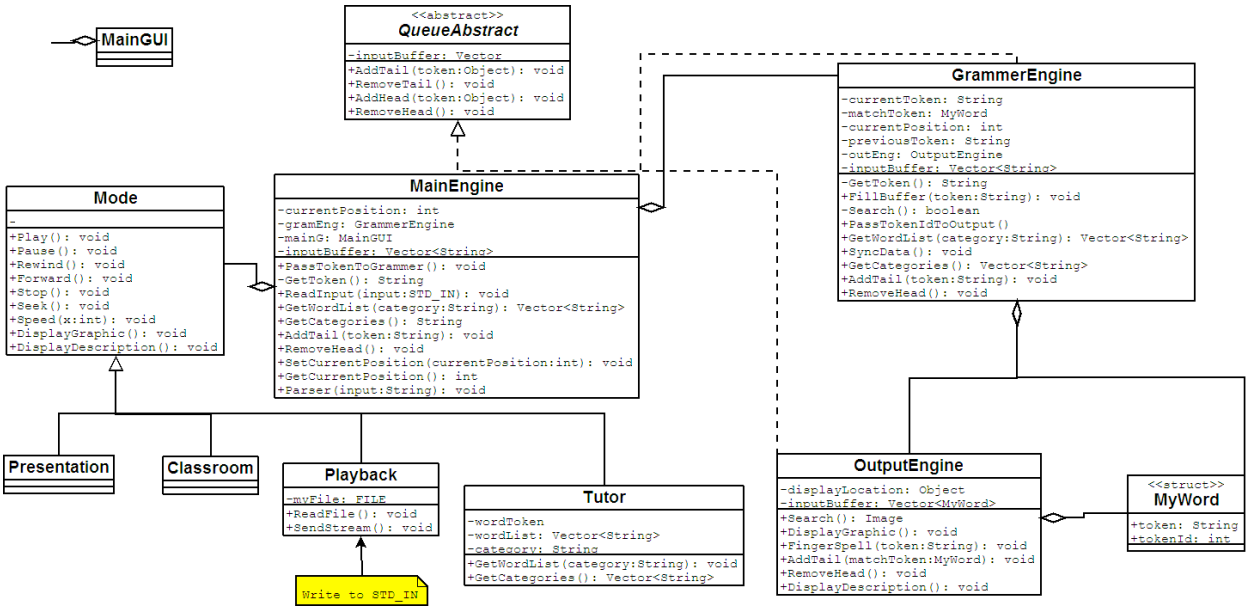


Figure 4.1 Class Diagram for LISTEN System

4.2 Main Engine

4.2.1 Main Engine Responsibilities

The Main Engine is responsible for the following functions:

1. Maintain an input buffer queue for the strings coming out from standard input
2. Keeps track of the current token send to Grammar Engine
3. Keeps track of the mode selected in the Main GUI

4.2.2 Main Engine Overview

The speech to text converter, Dragon, converts the audio to corresponding strings which are then passed from Standard input to the parser in Main Engine. The parser separates the words in the string and stores them in the input buffer queue. As the input buffer queue reaches its maximum storage, the contents of input buffer queue are transferred to a secondary storage which is also used in Playback Mode.

Depending on the mode selected the input buffer queue behaves as follows:

1. **Classroom Mode**

   The strings from Dragon are stored in the input buffer queue. The current position pointer of the buffer queue is set accordingly when the buttons in the control bar are clicked.
   Play: The pointer is at the current string passed to Grammar Engine.
   Pause: The pointer is at the current string passed to Grammar Engine, indicating the position of the string/token to be passed next to Grammar Engine when resumed.
   Rewind: The pointer is shifted one string backwards, which is then passed to the Grammar Engine.
   Forward: The pointer is shifted one string forward, which is then passed to the Grammar Engine.
   Seek: Depending on the position of the seek bar, the current position pointer is shifted back accordingly.

2. **Presentation Mode**

   Works the same as Classroom Mode.

3. **Tutor Mode**

   In tutor mode, the word selected from the list is put in the input buffer queue; then the word is passed to the Grammar Engine for a match.

4. **Playback Mode**

   As the tokens from the selected text file are stored in the input buffer queue in Main Engine.
   The buttons in the control bar behaves similar to Classroom Mode and the current position pointer is set accordingly as in Classroom Mode.

4.2.3 Main Engine Methods

The Main Engine implements the following methods:

ReadInput (input:STD_IN):void
Reads the words sent to standard input

Parser(input:string):void
Called by ReadInput ( ). Separates the words from input string and passed to AddTail( )

AddTail(token:String):void
Adds the token to the end of queue

SetCurrentPosition(increment:Integer):void
Sets the position of the token to be sent to Grammar Engine

GetCurrentPosition( ): int
Returns the current position of pointer in input buffer queue

PassTokenToGrammer( ):void
Pass the token at current position to Grammar Engine

GetToken( ):String
Called by PassTokenToGrammer( )

GetWordList(category:String):vector<String>

GetCategories( ):String

RemoveHead( ):void
Called when the input buffer reaches the maximum storage and clears the input buffer

4.3 Output Engine

4.3.1 Output Engine Responsibilities

The Output Engine is responsible for the following functions:

1. Maintaining an input buffer queue for the MyWord tokens coming from the Grammar Engine
2. Maintaining a table of relationships between valid entries in the dictionary, and their corresponding graphic ouput.
3. Displaying the correct graphic output for a given token
4. Converting words not in the dictionary to finger-spelled equivalents
5. Logging words that are not in the dictionary, and words that are not fingersspellable to a file for later analysis

4.3.2 Output Engine Overview

The Output Engine contains the following objects:

- MyWord tokens, consisting of a String (which is an ASL phrase parsed by the Grammar Engine), and an Int (which is the database ID for the phrase.)
- A Vector:Words containing the MyWord token received from the grammar engine.
- (2) Ints, xLocation and yLocation containing the X and Y coordinates corresponding to the location of the Graphic Display Window.

When a MyWord token is received by the Output Engine, it is added to the tail of the inputBuffer.

If Words is not empty, the Output Engine performs the following actions on the MyWord token at the head of the vector in order:

- Search the Output Database by ID for an entry corresponding to the current token at the head of Words.
- If a graphic exists, display that graphic, then remove the token from the head of Words.
- If ID == -1 pass the token on to the FingerSpell() method, and wait for its completion.
- If a graphic does not exist, pass the token on to the FingerSpell() method, remove the token from the head of Words, and wait for its completion.

When FingerSpell() receives a token, it performs the following actions in order:

- Create a Vector:Letters to contain MyWord tokens.

- Convert the string into a character array.
- Iterate over the length of the array performing the following actions:
    i. Check the current character for an entry in the Output Database.
    ii. If a matching character exists, create a token corresponding to that character and add it to the tail of Letters
    iii. If a matching character does not exist, write the current word's string to the analysis file, delete Letters, and return control to the main function.
- Iterate over Letters starting from the tail, and perform the following actions:
    i. Add the token to the HEAD of Words.
    ii. Remove the token from Letters.
- Delete Letters, and return control to the Main() method.

4.3.3 Output Engine Methods

Output Engine implements the following methods:

Search(ID:int):Graphic
Searches the database for a graphic corresponding to ID:Int. Returns a pointer to the graphic, or Null if nothing is found.

DisplayDescription(token:MyWord):Void
Displays the String associated with MyWord.

DisplayGraphic(token:MyWord):Void
Displays the output graphic associated with MyWord.

AddHead(token:MyWord):void
Adds the token to the head of Words.

AddTail(token:MyWord):void
Adds the token to the tail of Words.

FingerSpell(token:MyWord):void
Breaks a token into more tokens corresponding to its constituent characters.

RemoveHead( ):void
Removes a token from the head of Words.

4.4 Grammar Engine

4.4.1 Grammar Engine Responsibilities

The Grammar Engine is responsible for the following functions:

1. Removing punctuation and token separators from the input stream
2. Determining if a token is part of the limited vocabulary of the system
3. Tokenizing the input stream (note that tokens can consist of more than one word)
4. Filling a queue with tokens to be displayed by the Output Engine

4.4.2 Grammar Engine Overview

The Output Engine contains the following objects:

- A Vector:String containing strings parsed by the Main Engine
- An Int, currentPosition, containing the value of the current position in the input stream
- (2) Strings, previousToken and currentToken
- A MyWord, matchToken

When a string is received by the Grammar Engine, it is added to the tail of the inputBuffer.

If the inputBuffer is not empty, the Grammar Engine performs the following actions on the string at the head of the vector in order:

- Removes all punctuation and formatting of the string.
- Compares the string against a finite list of multi-word tokens, if a match is found on the first word, checks the next string to determine if a multi-word token must be formed.
- If a multi-word token is necessary, creates that token, removing subsequent strings from the head of inputBuffer as necessary, and stores that token as currentToken.
- If a multi-word token is not necessary, creates a token from the initial string and store that token as the currentToken.
- Passes currentToken to Output engine.

4.4.3 Grammar Engine Methods

Grammar Engine implements the following methods:

FillBuffer(token::String)Void
Places the currentToken into the buffer so it can be accessed by the Output Engine.

Search():Boolean
Returns true if string matches a token in dictionary, false if there is no match.

SyncData():Void
Sets integer currentPosition to be equal to the position of the currentToken.

PassTokenIdToOutput()
Passes Token ID to Output Engine.

AddTail(token:MyWord):void
Adds the token to the tail of Words.

RemoveHead( ):void
Removes a token from the head of Words.