HELPING THE ELDERLY USE THE INTERNET: IMPLEMENTATION DETAILS FOR VOICE CONTROL CONTROLLED WEBSITES

Cole D. Noble

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

# Introduction

# Implementation

## Implementation overview

An implementation overview of all the JavaScript used to achieve voice enabled WebPages, automatic link highlighting, voice feedback for either Numerical Voice Referencing or Spoken Link Name referencing is highlighted below in Figure 1.

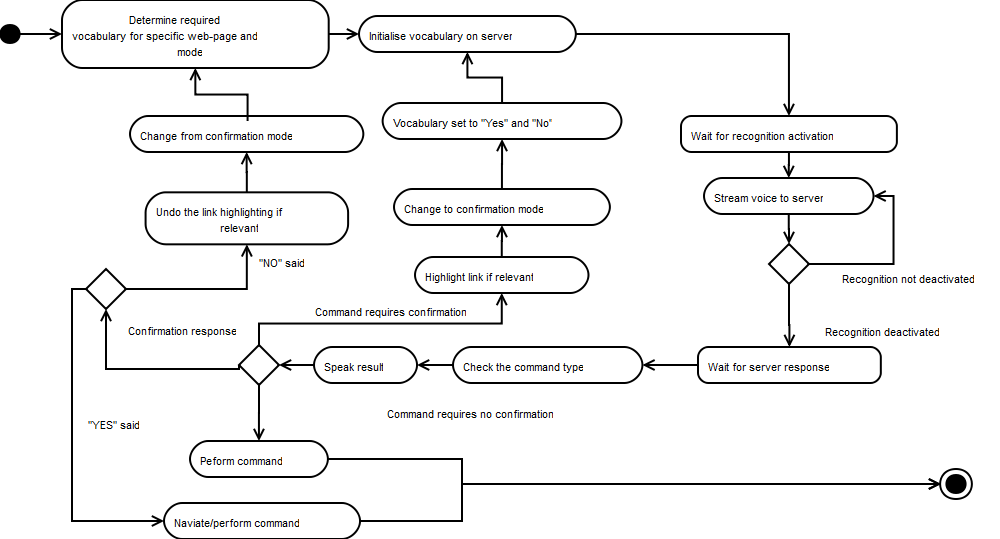


Figure 1: Total overview of voice navigation

## Using the speechAPI

All voice recognition is processed using an online API from SpeechAPI.com [1].

This API provides a flash object, *SpeechAPI.swf*, which can be embedded onto an HTML page using a *SWFObject.* This flash object is used to stream audio between the client and server. The streaming starts whenever the client gives the flash the command to start recognition. *SpeechAPI.js,* provided by SpeechAPI.com,acts as an interface with the flash object and allows the client to issue commands to the flash component. Once a stop recognition command has been issued by the client, the server sends the client back a result string indicating either that server was unable to recognise the command or the voice command was perceived to be. Once the result is returned various JavaScript methods are used to interpret what to do with the result. The following paragraphs provide a little more detail into the process described above.

Once the flash has been successfully loaded, the *onLoaded* (see Figure XXX)method is called. Importantly, this method is called to initialise the server with the particular vocabulary needed for a specific web page. Depending on what the flow of the program has been set to (Numerical Voice Referencing or Spoken Link Name referencing), the vocabulary is either set to include certain numbers, for Numerical Referencing, or names, for Spoken Link Name referencing.

When the flash component receives the command to start recognition it starts streaming audio to the server (see section xxxx). The server will try and match in given input to one of the specified vocabularies. A string result is then returned indicating a closest match result or the string, “recognition error” if no result was found.

## Keeping the API usage simple

The API also lets you set it up so that it recognises alternative words for the same thing (so one could potentially say “go up” or “scroll up” etc.). This would give the application a more natural interface. Beyond this, the API also lets you construct grammar segments in such a way that parts of a sentence are not necessarily necessary. If thus functionality was used, links could be followed in this way instead using highlighting to indicate what words need to be spoken. However, the scope was limited to highlighting at this stage so that a basic comparison between numerical referencing and spoken link name referencing could be conducted.

## Flow Control

### Controlling flow and notifications

Once a string result has been returned from the server (see section XXX above), processing needs to be carried out to determine what action to perform given a certain result has been returned. The main processing of results from the server (including setting up connections) is conducted in *speechprocessor.js*. Since two primary streams of tests were undertaken, two main courses of the program are determined by the state of the Boolean variable, *numericalReferenced*. If this variable is true, the program will divert to a stream which performs the required functions for numerical functions. Otherwise, the code diverts on a course that is implicitly, spoken name referencing.

Another Boolean variable which helps determine the course of the code is the variable, *confrimationMode*. This variable is set to true every time a result is returned from the server that indicates some sort of confirmation is required before either following a link or performing a command (for instance, following link 1 or performing the “backwards” command” require the user to confirm the Speech API’s interpretation of the spoken command). The process introduced with the confirmation mode can be seen in section XXX.

### Confirmation mode

Confirmation was a feature only introduced at Iteration 3 of the program. It was introduced to compensate for misinterpreted user commands. From the results of the earlier iterations it was clearly evident that results would often be returned as something they were not. If a result is interpreted by the speech engine as being significant in nature, that is if it will result in some form of navigation, a confirmation is required of the user.Confirmation means that the user must either say, “yes” to confirm a returned result or, “no” do disapprove of a returned result. So, for instance, if a user commands “Home”, “Backwards”,””Forwards” or speaks any link selection, the user will have to first confirm the APIs interpretation before the command is carried out on the webpage.Although users preferred not to have to perform a confirmation step, See Appendix XXX for Results, it is at this stage necessary.

## Main processing details

### Initialising and loading

The script found in *speechprocessor.js* firsts calls the *initialise* function which sets up the flash component, establishes a connection to the server and embeds the flash component on the *HTML* page. From here, the program will wait until the *onLoaded* call-back is called by *SpeechAPI.js* (see Section XXX) code indicating that the flash component has successfully been embedded. The processes involved in this *onLoaded* call-back are illustrated in Figure 2. As can be seen from Figure 2, the *onLoaded* method’s chief concern is setting up the appropriate vocabulary for the SpeechAPI server.

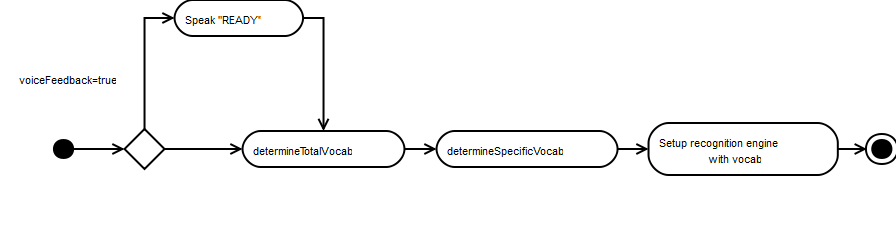


Figure 2: OnLoaded Overview

In the *onLoaded* method, the vocabulary is calculated using *determineTotalVocabulary* (Figure 3) and then *determineSpecificVocab* (Figure 4). *DetermineTotalVocabulary* in short determines a broad, rough, idea of what the vocabulary should be. *DetermineSpecificVocab* simply narrows down the vocabulary to what is specifically needed for a particular webpage.

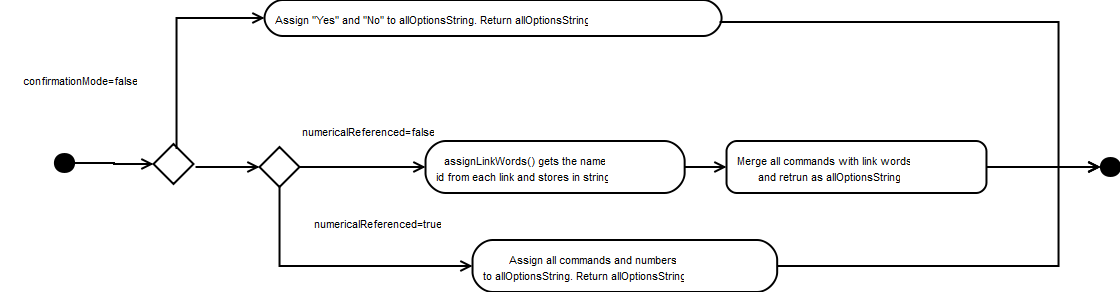


Figure 3: Determine total vocabulary overview

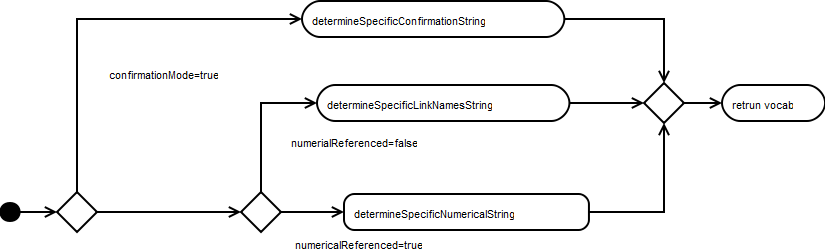


Figure 4:Determine specific vocab

*DetermineSpecificVocab* then calls specific functions: *determineSpecificConfirmationString*, *determineSpecificLinkNamesString* and *determineSpecificNumericalString* depending on whether or not confirmation mode is enabled (through the confirmation mode variable, see Section XXX) or if numerical referencing is being used (See Section XXX). These functions will divide a string which contains all the command strings and all the link referencing vocabularies into two strings. Once string containing commands and the other string containing link associated referable words.

These sorts of concatenated strings were used for both *Confirmation Mode* and *Spoken Link Name Referencing* because this practice was used initially when the code was written to perform numerical referencing. Having all the vocab placed in one string meant that if one wanted to add additional commands or numbers one could all do so through a single string (the concacentated string). The other streams, namely spoken link names and confirmation mode, also revolved around this sort of concatenated string so that all the processing for all modes was similar in logic. This trade-off makes following the program flow easier, although many functions exist which are not essentially needed.

At this stage, the vocabulary has been winnowed down to only what a particular webpage needs. The flash component is then used to load the vocabulary onto the server.

### Assign referable names to links

For numerical referencing, the *neededNumbers* array (see section XXX) stores a list of all the needed numbers. These numbers can the be aligned to the link URLs as mentioned in section XXX. To extract the actual words for Spoken Link Names, the function *assignLinkWords* is used to extract all the link id’s (which have previously been assigned the appropriate names) which are then assigned to the array. This array’s (*neededNamesArray)* information is later used in the same way as the *neededNumbersArray*  to search for the correct index of a returned result and consequently navigate to a link (see section XXX).

### Processing the result

Once the server has processed a user’s voice for recognition duration, it will call the *onResult* call-back.

When the *onResult* call-back is called, the *processResult* function is called (See Figure 5).

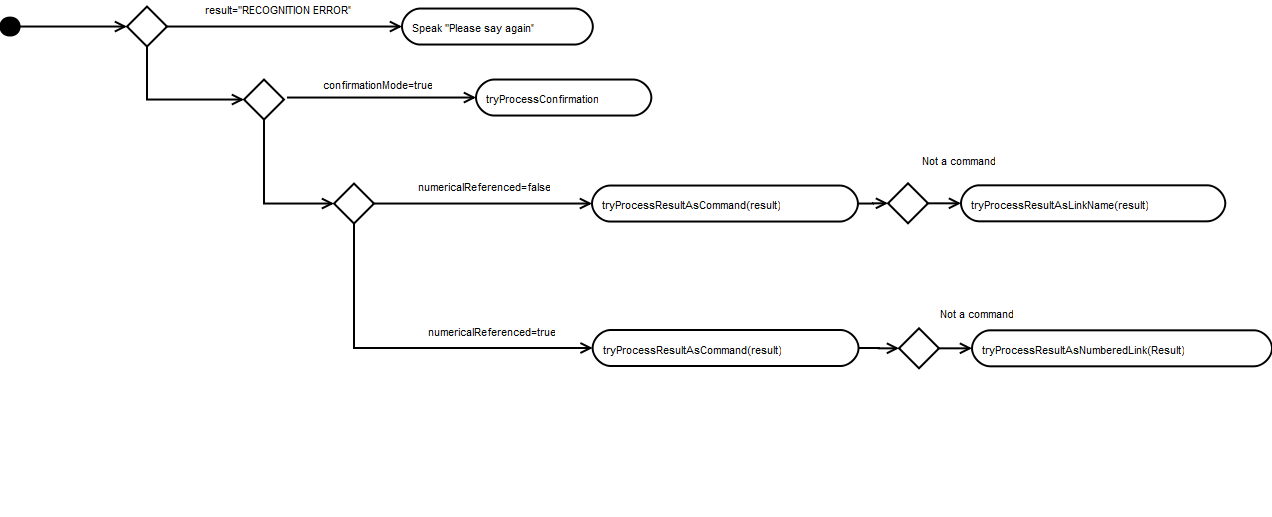


Figure 5: Result processing

This function performs a series of checks that determine how to treat the result. The function hieratically performs the indicated checks. The functions *tryProcessConfirmation* and *tryProcessResultAsCommand* simply conduct tests to check what type of confirmation or commands have been issued and perform the appropriate actions if a command is found. The methods *tryProcessResultAsLinkName* and *tryProcessResultAsNumberedLink* traverse the *neededNameArray* and  *neededNumbersArray* respectively to try and find a match to the returned result. If a result is found, the array index is recorded so that it can be used to find the required URL for navigation. Every link URL is then assigned to an array so that each one aligns with its corresponding link name or number index in the *neededNumersArray* or *neededNameArray*. The index from the traversal is then used to access and extract the appropriate URL from the links array. Once the right URL is had, the link is followed.

# Critical analysis

As was mentioned in Section XXX, the code was built in an evolutionary manner. That is, Numerical Referencing was first established followed by Spoken Link Name referencing and other more advanced features. This meant that the code is not well structured and functions are often created which perform similar calculations. Also, as mentioned, unnecessary functions were created to make the program easier to understand. If the functionality detailed above is ever implemented as a generic web-page modifier, it should be redesigned from scratch with the more advanced features in mind from the start. The current implementation worked more than adequately to perform what it was required to: A platform for testing the elderly. However, if further features need to be implemented on the current code base, developers might run into maintainability issues.

# Conclusion

# Bibliography

1. **Lord, Spencer.** Online Speech Recognition API. *SpeechAPI.* [Online] 2010. [Cited: 26 October 2011.] http://www.speechapi.com/.