# Speak2Me

# A Chrome Speech Interface

*Annica Ivert* *Christos Kakouros*
aivert@kth.se kakouros@kth.se

*Speech and Speaker Recognition*
*2013*

*ABSTRACT: With the advances in speech recognition technology, integration of speech recognition support is becoming more prominent in personal computer environments. With Internet applications occupying the largest timeshare of the average user's daily use of personal computers [1] web browsers are one of the main target applications for speech-based control. In this report we examine the use of speech in controlling Chrome web browser. We refer to existing solutions and examine two major architectures, i.e. a Chrome extension and a personal computer application. We then use Python Dragonfly and Windows Speech to implement Speak2Me, a speech interface for different Chrome operations.*

## Introduction

Automatic speech recognition (ASR) has followed a slower rate of growth compared to other areas of pattern recognition, largely connected to the growth of computing power over the last two decades [2]. Today, automatic speech recognition is used in a broad range of applications, from call centers to military applications [3] and personal computer environments. In the latter proprietary speech recognition engines have been introduced to perform tasks that users have been performing through other means of interaction with their computers, i.e. mostly tactical moves. Speech is used either as a complementary mean of interaction or in cases of physical impairment as the only mean of interaction between the user and their computer [4]. Internet applications occupy more than half of the timeshare of the average user's daily use of personal computers [1] and web browsers can be one of the main target applications to be controlled using speech.

In this report we present Speak2Me, a speech interface for Chrome. We begin with investigating the architecture of two ways of implementation, i.e. a distributed architecture implemented as a Chrome extension and a local architecture developed as a personal computer application. We continue with describing the methodology and features of our system that is based on the second architecture. We summarize our conclusions and experiences from the system development and usage.

# Basic Architectures

## *Chrome Extension*

One of the ways to deploy the speech interface is to implement it as a Chrome extension. Extensions are small programs that are integrated in Chrome window to add new features to the browser and personalize the user's browsing experience [5]. Google provides APIs i.e. the tools needed to interact and control different components of the browser, like opening a new tab or manipulating bookmarks. Along with the contribution of third party communities they offer extended documentation for the APIs and the development of new extensions. The main technologies used are JavaScript and HTML.

There are two approaches in developing a speech extension that are summarized in figures 1 and 2 accordingly. Both approaches include a call from the extension to a server on which a Speech Recognition engine runs. The extension first records the user's voice and then sends it to the server where it is transcribed to text. Afterwards the server sends the text back to the extension. With this implementation recognition is performed entirely remotely.

In figure 1, the extension runs flash player to record the user's voice and then uses RTMP (Real Time Messaging Protocol) to contact the media server (Red5 or Adobe Flash Media Server) and send the recording. The media server passes the recording to the Speech recognition application. The latter includes the Speech engine and the interface to it that feeds it the waveforms, specifies the grammar that the extension uses to perform its operations on the browser and receives the textual outputs from the engine. The textual output is returned to the extension using any web server.

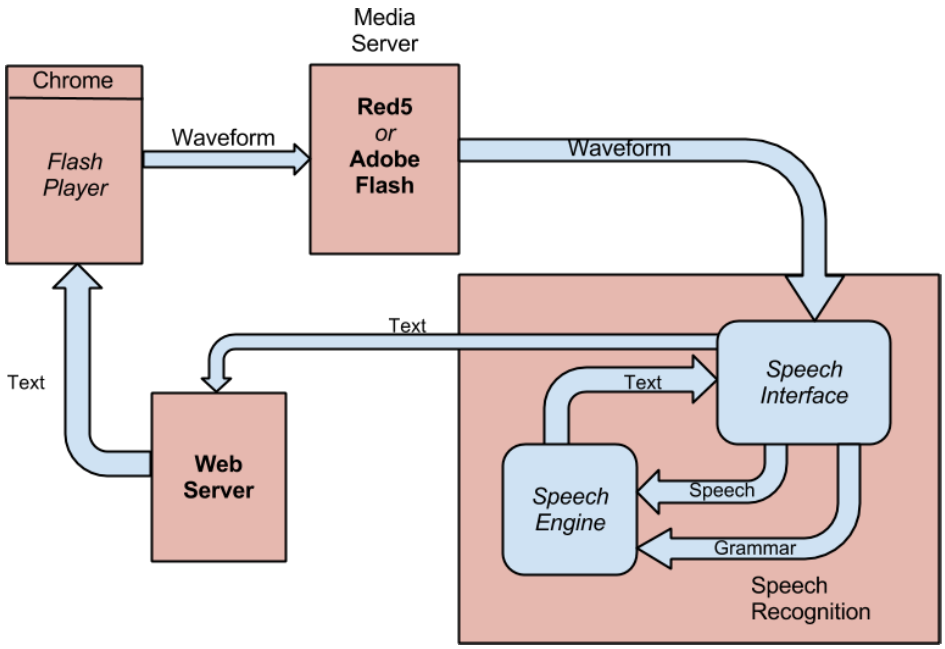


***Figure 1:*** *Extension using media server and remote speech recognition*

Figure 2 demonstrates another newly adopted approach. The extension uses Web Speech API that is a specification published by the Speech API Community Group and not yet a W3C Standard [6]. It is a JavaScript API that aims to enable web developers to produce speech scripts for web browsers. The API itself is agnostic of the underlying speech recognition and synthesis implementation and can support both server-based and client-based/embedded recognition (for one-shot and continuous speech) and synthesis. Grammars can be specified and used in the web page context and speech recognition results are provided to the web page as a list of hypotheses, along with other relevant information for each hypothesis. Currently, the API is in experimental status and only the recognition part is supported in Chrome 25. In figure 2 we have used Google speech engine.

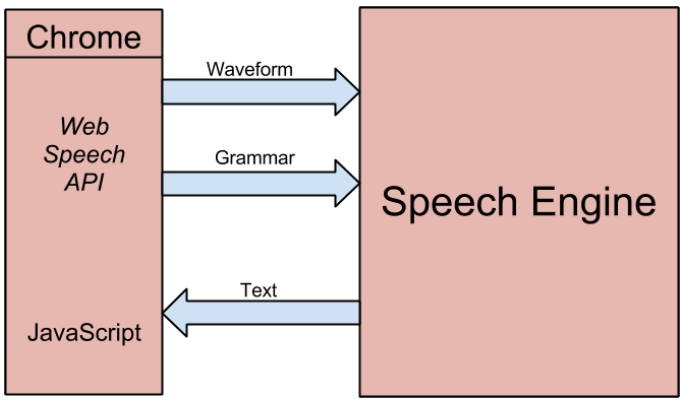


***Figure 2:*** *Extension using JavaScript Web Speech API and remote speech recognition*

## *Personal Computer Application*

Another approach to building a Chrome speech interface is to develop it as a personal computer application that can run in parallel with the browser and control its functions using speech input from the user. Everything is performed locally on the user's computer. The basic architecture is given in figure 3. The application comes with a predefined grammar that specifies the set of operations/rules that it supports and is used when communicating with the Speech Recognition engine. It receives the recognized rules as textual output from the engine and translates them into an action that it then performs on the browser. Speech input from the user's microphone and the grammar are fed to the speech engine through the application.

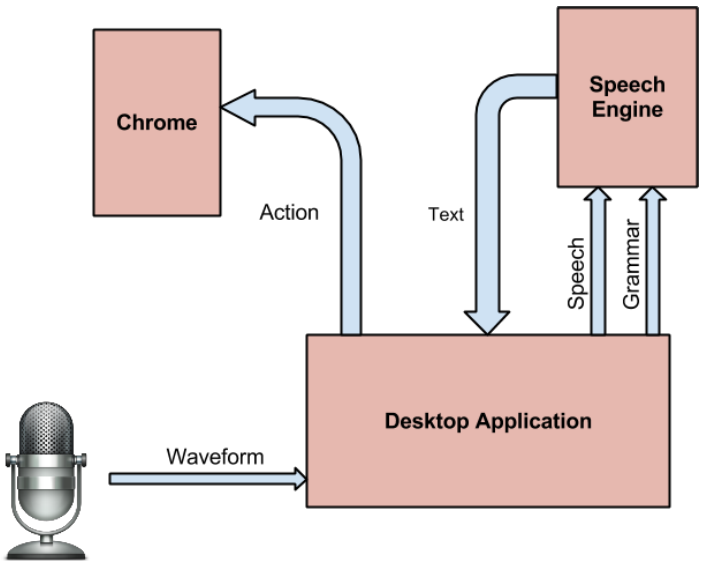


***Figure 3:*** *Personal computer application architecture*

## Speech Engines & Frameworks

There are various Large-Vocabulary-Continuous-Speech-Recognition (LVCSR) engines available in the market [7] and different frameworks to develop the interfacing application. Below we summarize some of them in tables 1 and 2 respectively.

| | **CMU Sphinx** | **Julius** | **Dragon Naturally Speaking** | **Microsoft Speech** |
|---|---|---|---|---|
| **Developer** | Carnegie Mellon University | Nagoya Institute of Technology | Nuance | Microsoft |
| **Licence** | Open-source | Open-source | Proprietary | Comes shipped with Microsoft Vista or earlier |
| **Supported language models** | English<br>Chinese<br>French<br>Spanish<br>German<br>Russian | Japanese<br>English | English<br>French<br>German<br>Italian<br>Spanish<br>Dutch | U.S. English<br>U.K. English<br>Chinese<br>Japanese<br>Spanish<br>French<br>German |

***Table 1:*** *Overview of speech engines*

Julius is mainly directed to Japanese, though it has a built in language model for English while CMU Sphinx greatly encourages developers to develop their own language models [7]. Both Dragon Naturally Speaking (depending on the version) and Microsoft Speech support speaker-independent recognition [8][9]. They have training modules and support dictation mode, to insert text to fields or office suites, and command mode, to control the computer environment and running applications. Microsoft Speech offers more command features for Microsoft applications, requires clear speaking and continues training as long as the user interacts with the engine.

| | **Vocola** | **Microsoft Speech API** | **Dragonfly** |
|---|---|---|---|
| **Language** | Own | C# | Python |
| **Supported Speech Engine** | Dragon Naturally Speaking & Microsoft Speech | Dragon Naturally Speaking | Dragon Naturally Speaking & Microsoft Speech |

***Table 2:*** *Overview of frameworks*

Concerning the frameworks we should note that, Vocola and Dragonfly offer a higher level of abstraction and are friendlier to the developer while Microsoft Speech API is an API written in C# and demands greater programming skills. Finally, one of the major drawbacks of Vocola is that it does not support conditional statements or calling scripts from within Vocola code [10].

## Chrome Speech Interface

Our choice was to implement our speech interface using DragonFly and Windows Speech Recognition engine. The choice of architecture was mostly motivated firstly by the fact that Web Speech is still in a highly experimental level and secondly by the locality offered through a personal computer application, i.e. the application and the engine run locally and are always available to the user. The choice of Dragonfly was made based on the friendliness and ease on writing macros, the completeness of the language and the fact that it supports Windows Speech. The choice on the latter was mostly based on the fact that it is shipped along with Windows 7 OS. Dragon Natural Speech would have us missing some hundred dollars and the use of any of the open-source speech engines would need greater efforts with not always guaranteed results. Below we present Dragonfly and then go on with the description of our implementation.

## *DragonFly*

Dragonfly provides a framework in which it is possible to create Python macros used to interface the speech engine of choice [2]. To build a macro, a Grammar must first defined. A Dragonfly Grammar consists of a set of Rules which define what can be said and how this will be mapped to executable actions.

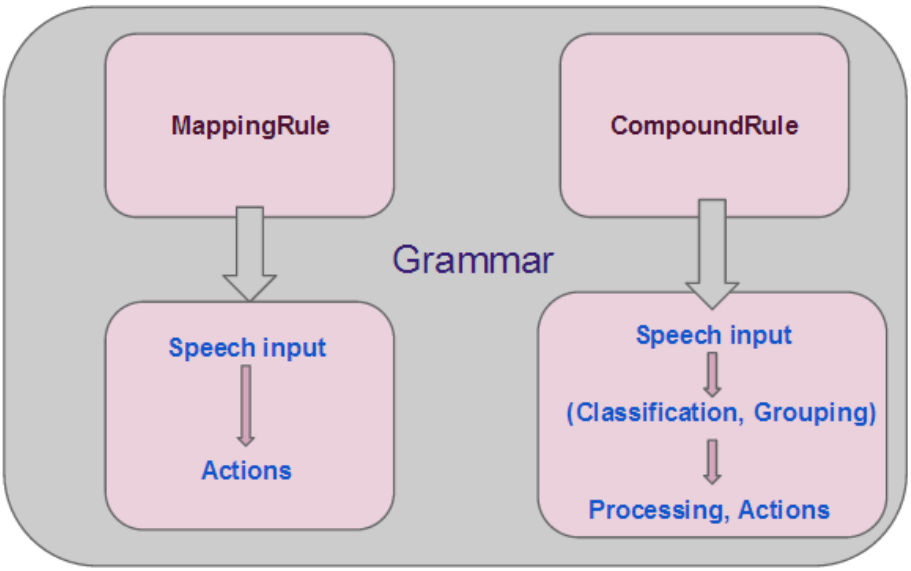


***Figure 4:*** *Example components of a DragonFly Grammar.*

Examples of DragonFly Rules are the MappingRule and the CompoundRule. A MappingRule defines the simplest rule, which is basically a direct mapping from a speech input to a set of actions to be executed without any intermediate processing. With a CompoundRule it is possible to group the speech input in different classes and do further processing of the data before deciding upon which actions to execute.

The translation from speech input to text is done via a speech engine. Currently DragonFly supports Dragon Naturally Speaking and Windows Speech Recognition. When the speech has been converted into its textual interpretation, it is matched against a grammar defined in the DragonFLy Rules. This grammar is not to be confused with the overall Grammar class, that defined all the rules for a given application.

The grammar that defines a valid speech input consists of a number of Elements. Some examples of Elements are:

1. *Sequence - A sequence of Elements.*
2. *Dictation - A free form Element that matches all inputs and allows the user to pass arguments to the functions. An example would be a "Save as <name>" command that lets the user specify the parameter name.*
3. *Alternative - Specifies that exactly one of a set of specified Elements will give a match.*
4. *IntegerRef - Matches integers within a specified interval.*

When an input is matching a specified grammar rule this is mapped to an Action. Common Dragonfly Actions include:

1. *Key - Is equivalent to pressing a sequence of keys on the keyboard.*
2. *Text - Inserts the specified text.*
3. *Mouse - Moves the mouse to the specified location.*
4. *Function - Calls a Python function that may do some preprocessing of the data before executing an Action/a set of Actions.*

# Implementation

For a speech interface to be user-friendly it needs to be complementary to traditional means of interaction between the user and the computer while offering all the basic functionalities with simple intuitive commands. Simple commands such as "Open tab" are quite straightforward. Other operations such as opening web pages and filling out forms need to be given a bit more though when it comes to satisfying the users needs.

Since Windows Speech Recognition does not by any means have a word error rate of zero and cannot, if not properly trained, recognize difficult web addresses it would be convenient to have a mapping from some simpler set of speech input to the actual web addresses. Also, it is easier to say "Read mail" than to have to spell out the actual adress, say "webmail.kth.se".

We therefore created a Python script that parses a text-file that can be filled in by any user without having to deal with actually rendering the Python code. An example subset of such a file may look like:

*mail: webmail.kth.se*
*news: www.dn.se*

In the same way information about the user that might be commonly used to fill out forms etc. can be stored in the settings file. A speech input address such as "Brinellvägen 68" would have no way of being recognized properly unless included in the dictionary somehow. '

In addition to this basic functionality, we implemented a speech calculator that let the user give speech input on the form *<multiply|divide> <number_1> by <number_2>* and get the result as a textual output.

We also added a simple speech reminder, where the user can add tasks to the memory bank by saying "Remember <task>" and later retrieve these by saying "Recall".

# Discussion

Our speech interface, Speak2Me, allows the user to interact with Chrome via speech commands. We solved the problem with extensive web addresses possibly containing non-English words by providing a settings file for the user to fill in "speech bookmarks". A more convenient way to do this would be to also add these bookmarks by voice commands on the form *"Bookmark this as <name>"*. This is a further improvement to be made. However this will require the python program to retrieve information from Google Chrome about the current web-page. Currently our implementation only speaks to Windows Speech Recognition and the computer input devices such as the keyboard and the mouse.

## Conclusions

The performance of our speech interface relies entirely on the performance of Windows Speech Recognition. The recognizer can be trained further to improve performance, but our interactions with Google Chrome were mostly satisfactory when the conditions, such as a decent microphone and a noise-free environment were met. We did also see the difference between different users and how this difference became smaller as the recognizer was trained with both voices. We also noted some user-independent difference in performance when using the system on a laptop connected to power supply and on the same laptop computer running on battery, with higher performance on the former. Possible explanation for this may be the usage of resources for power management in laptops. However, this is only our observations after having used the engine for a short period of time. More systematic approach will be needed to clarify these results.

DragonFly provides an easy way to extend the Windows Speech Recognition functionality to applications like Google Chrome. Even though speech recognition still has not reached the point where this works flawlessly, most commands were properly interpreted when the engine had been given some training and the testing was conducted in a noise-free environment.

The added features, like the calculator and the reminder function, hints to one of the advantages of a speech interface, besides providing a more natural interaction with the computer: With speech commands menus and buttons are unnecessary, providing a cleaner, less cluttered screen.

## References

[1] Beauvisage, T., 2009. Computer Usage in Daily Life. In *Conference on Human Factors in Computing Systems, Proceedings of the SIGCHI*, pp.575-584.

[2] O'Shaughnessy, D., 2008. Invited paper: Automatic speech recognition: History, methods and challenges. In *Pattern Recognition, Volume 41, Issue 10,* October 2008, Pages 2965-2979.

[3] Wikipedia, Speech Recognition. Online. 20 May. 2013
http://en.wikipedia.org/wiki/Speech_recognition

[4] Kakouros, C., Ramos, M., Edelstam, F., 2013. Speech Technology and Smart Homes. In *Speech Technology DT2112, Term Paper,* KTH.

[5] Google, Chrome Extensions. Online. 18 May. 2013
http://developer.chrome.com/extensions/index.html

[6] W3C, Web Speech API Specification. Online. 19 May. 2013
https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html

[7] Wikipedia, List of speech recognition software. Online. 25 May. 2013
http://en.wikipedia.org/wiki/List_of_speech_recognition_software

[8] Wikipedia, Microsoft Speech API. Online. 24 May. 2013
http://en.wikipedia.org/wiki/Microsoft_Speech_API

[9] Wikipedia, Dragon Naturally Speaking. Online. 24 May. 2013

http://en.wikipedia.org/wiki/Dragon_NaturallySpeaking

[10] Dragonfly, Project information. Online. 25 May. 2013
http://code.google.com/p/dragonfly/

[11] Dragonfly, Documentation. Online. 25 May. 2013
http://dragonfly.googlecode.com/svn-history/r105/trunk/dragonfly/documentation/index.html