UNIVERSITY OF GOTHENBURG

Increasing Accuracy of Speech Recognition Applications

On Windows platform using .net framework

Ali Bahaloo

Bachelor of Applied Information Technology Thesis

Report No. 2010:018
ISSN: 1651-4769

University of Gothenburg
Department of Applied Information Technology
Gothenburg, Sweden, May 2010

Increasing Accuracy of Speech Recognition Applications on Windows platform using .net framework

Ali Bahaloo

alibahaloo@gmail.com

IT University of Gothenburg, Software Engineering and Management, Gothenburg, Sweden

Abstract

The intention behind developing speech recognition applications is to automate tasks. That is by accurately recognizing the user's input and taking action accordingly. There are many algorithms and techniques that one can apply in developing an ASR system, however the development resources play main factor in choosing these techniques. This paper focuses on .net framework as the main development resource and compares the possible approaches on designing ASR systems using this development resource. From this comparison the most effective and accurate one will be selected. Finally, we propose a method in designing an ASR system using .net framework that results in highest accuracy.

1. Introduction

Speech Recognition (SR) is the process of converting spoken speech to text. The terms voice recognition and speech recognition differ in their definitions. Voice recognition refers to an engine in which, a part of it is speech recognition, and has the ability to be trained for a particular user (with its unique vocal patterns). Voice recognition is also known as Speaker recognition. Speaker recognition's main role is to identify the "Speaker”, in other words; identifies the user. However, speech recognition's main role is to recognize what the speaker (user) has spoken (inputted) [1].

The concept of speech recognition as a tool for automation of tasks came to live in the 1990's, the era of “call centers”. Calls centers are handling a lots of tasks, among these tasks are; rotating incoming calls to the desired department, when a customer is asking for a specific assistance. Large companies such as AT&T were looking for a way to automate these tasks in their call centers. Back then the number of agents in call centers could be very high, since all of these tasks were handled manually by agents [2]. Now consider a call center, where these tasks are handled by a machine that can recognize the customer's request by recognizing their speech on the phone. In that case the tasks are automated, reducing the large operating cost of companies. This capability has been provided by Automated Speech Recognition (ASR) technologies. AT&T in 1992 introduced their Voice Recognition Call Processing (VRCP) service that handles 1.2 billion voice transactions every year [2].

Speech recognition, nowadays have many different usages; in Health care; for deaf people, it's been used to convert the speech to text for better understanding of the deaf users. It's been noticed that, Electronic Medical Record (EMR) applications are more effective when they are combined with a speech recognition engine, since filing forms and writing reports and such are performed faster by voice rather than keyboard [3]. SR has been used in military, in high-performance fighter aircraft and helicopters. It is also widely used in the field of mobile and telephony.

2. Speech Recognition (Windows platform - .net framework)

Microsoft provides an API that allows developers to use speech recognition and speech synthesis engines in windows applications. Speech recognition engine is used for speech-to-text conversion, while speech synthesis provides access to text-to-speech conversion engine. The Speech API, in short SAPI, can be seen as an interface between the Application and SR/TTS engines [4].

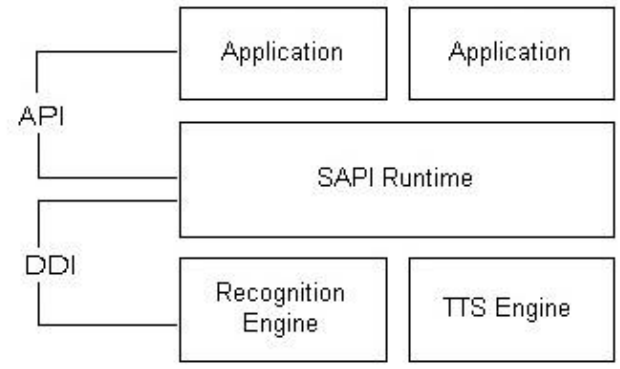

Figure 1: SAPI Overview [4]

SAPI is the API that is developed by Microsoft in order to implement speech applications. Implementation via SAPI can be done either via .net framework namespaces that wrap the functionalities of SAPI, or via SAPI COM component. COM, short for Component Object Model, represents a number of interfaces in order to use the API's functionalities. SAPI acts like a connecting bridge between engines and application, while .net framework handles the interaction between the application and SAPI.

The interaction between the application and SAPI is handled by .net framework namespace; i.e. System.Speech. This namespace comes along with .net framework version 3.0 and above and provides certain number of classes to interact with speech recognition engine.

In this paper, as it is illustrated in figure 2, the focus is on the interaction of the application and the SAPI runtime, which is handled by .net framework.

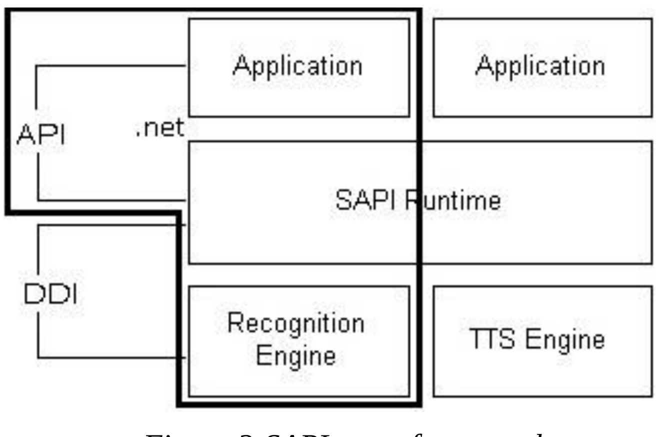

Figure 2:SAPI - .net framework

3. Problem Statement

As it was explained above, development of speech application on windows platform can be done in two ways; using .net framework or SAPI COM. This research considers .net framework as the main resource, which is because we want to develop a managed speech application that is by using .net framework namespaces, therefore calling SAPI COM will not be examined here. The aim of this research is to develop a managed application, i.e. using .net framework. However using SAPI COM in developing an ASR system makes the application a native one, which is not the concern of this paper. The problem that rises is to design/develop managed speech applications with reliable accuracy for windows platform using .net framework. In order to do so, first we need understand the concept of accuracy in speech recognition, and then we need to make assumptions. Assumptions are needed in order to enlighten the path of research. Accuracy in ASR is about recognizing the correct command, and acting according to the user's (input's) request. Accuracy plays the main role in ASR, which is because in order for a system to fulfill requests, it should be able to accurately recognize it first.

3.1. Research Assumptions

In this research, it is assumed that .net framework is the main development resource (rather than SAPI itself) and Microsoft Windows is considered to be the platform. In other words; the research focuses on accuracy with the regard to .net framework and Windows platform.

There are two factors affecting the accuracy while considering .net framework [12] [5]; first is the approach on the design of the recognition engine and second is the approach on designing the language model. Recognition engine is the main core of the application that handles the conversion of speech into text. The language model holds the definition of the commands that the engine can recognize, which is loaded into the engine.

Each of these two factors can be designed with two different approaches, which makes four possible combinations in designing ASR systems in general. Every one of these approaches result in different levels of accuracy and performance. So the problem is to find the combination which results in highest accuracy.

4. Aim and Objective

The aim and objective of this research is to find and investigate an approach to design accurate speech recognition application while .net framework is being considered and used as the main development resource. This is mainly done by studying different approaches on designing the language model and the engine of the ASR system. These approaches will be compared together to see which one results in highest accuracy.

The research question that can be asked is;

How to design and develop an accurate (managed) ASR system using .net framework for Windows platform?

Designing the language model and the recognition engine are the main factors affecting the accuracy of such system, therefore the main focus and aim of this paper is to investigate possible approaches on designing these two factors. In this way the problem of accuracy is studied.

In the end, the desired result would be a method that one can apply to develop an accurate speech recognition application for windows platform using .net framework. A prototype will be developed to illustrate the idea.

The research is mainly done by studying possible approaches in designing the language model and the engine. That is because; the design of the language model and the recognition engine defines the level of accuracy of a speech application [3].

In this paper, first we define the two possible approaches on designing the engine and the language model. We then describe the possible combinations. Finally we test and compare these possible combinations to investigate the effectiveness of each one in regard to accuracy problem.

5. Literature Review

In this section, information about the process of data collection and literature review. Basically, literature review has been used to access the published materials related to this research topic. Data was extracted from different research papers to have a better understanding of the speech recognition concept and the problem of accuracy in this field. Number of research papers has been studied; these papers are mostly published by Microsoft. That is since SAPI is developed by Microsoft and ever since there's been number of researches done by Microsoft Research – Speech technology group.

As it was explained above, the two factors affecting the accuracy of a speech recognition application are the Language Model and the Engine. In order to digest these two concepts, different papers has been reviewed;

Microsoft Windows Highly Intelligent Speech Recognizer [12]: whisper is a research paper done at Microsoft Research Group that introduces a new technology called Whisper, and explains the approach they took in order to have a more efficient, usable and accurate speech recognizer. This paper talks about factors such as dialect, accent and cultural background that can affect the recognition error rate. It also considers noise to be another important factor and tries to solve the accuracy problem in this regard. With the use of context-free grammars (CFG) and set of rules in it, applying a new efficient normalization procedure that differentiates noise from speech during normalization; that is by separating speech from noise and confidence measures used for noise rejection, they successfully increased the accuracy of recognition. They also suggest that speaker adaptation, by applying adaptation algorithms reduces the error rate by 50%.

There is another paper that focuses on speech recognition improvement for fast talkers. In this paper, authors observed that just like the presence of background noise, speaking rate also affects the recognition error rate. They purpose cepstrum length normalization (CLN) that normalizes the speech duration by stretching the length of the utterance in the cepstrum domain so it matches the acoustic model. Experiments show that normalizing the duration and smoothing it results in 13% error rate reduction [7].

Other articles have been reviewed that explicitly focuses on the language model. Use and acquisition of Semantic Language Model [11] is an article that focuses the usage and effect of semantics in designing language models. It mainly talks about the problem regarding the intention of the user during speech recognition. That is, since any spoken language understanding system must deal with this critical issue; accurately inferring user's intention from speech. Authors of this paper suggest that one can extend the statistical pattern recognition framework used in Automatic Speech Recognition (ASR) to the spoken language understanding (SLU). In this way, the pattern to be recognized by ASR turns into a tree of semantic objects that represents the domain entities and tasks that describe the user's intention. Authors applied the same algorithm on an existing application (MiPad) to see the results and observed that because of it, the model can be trained directly to optimize the understanding accuracy. The result was 17%

understanding accuracy improvement.

The concept of semantically structured language model (SSLM) is focused on [10]. The paper illustrates that SSLM results in decreasing the understanding error rate by 50%. The suggested model by the authors consists of both statistical word n-gram and CFG. Applying such model significantly reduces the understanding error rate compared to when the language model is only word n-gram. Authors concluded that SSLM not only reduces the human involvement in designing language models, therefore reduces the possibility of errors made by human, it also results in achieving a better understanding accuracy.

Another related and interesting paper is discussing name recognition [5]. Name recognition is the same concept of command recognition, since a “name” is nothing more than a simple phrase; just like a command. In this paper, authors explain the benefits of applying user modeling (UM) in speech recognition application in order to increase the accuracy and at the same time decrease perplexity of such application. The authors define UM as modeling of user’s behavior patterns; that is, the process of tailoring the application to the user’s personal needs. They suggest that recognition accuracy and usability of ASR systems can be improved if the system can automatically adapt to the usage patterns of individual users, which is training itself for that specific user. The result of experiments in this paper shows that there can be up to 62% reduction on out-of-vocabulary (OOV) rate as well as 99.5% reduction on the language model size. As a matter of accuracy, the improvement is very obvious that even with factors such as strong accent or slightly incorrect pronunciation of the names, there is over 99% accuracy. The perplexity also reduced by 99.9%, and with that there is a significant improvement on speed. From the experiments, the authors concluded that, with a system which continuously and automatically monitors user’s usage pattern and models user’s behavior, the accuracy increases significantly. That is since the ASR system dynamically adapts itself to the user, instead of user adapting him to the system.

There are also other techniques to improve speech recognition accuracy; in [6] a training technique has been introduced to discriminatively train both front end and back end of ASR systems. The front end is responsible for extracting the acoustic features while the back end which scores transcription of hypotheses for sequences of those acoustic features extracted by the front end. Experimental results show that joint discriminative training can be fast and accurate, achieving 5.75% word error rate (WER).

The other article that is related to language model concerns concept acquisition in example-based grammar authoring. The paper explains that automatic learning procedures reduces the involvement of humans in grammar design therefore reduces errors. It also improves the overall understanding accuracy of the system. Automatic learning procedures in ASR are referred to example-based grammars. Experiments show that the accuracy of recognition improves by 60% when a grammar is example-based [9].

Another paper titled “Grammar Learning for Spoken Language Understanding” is focused on automatic creation of semantic-based grammars [8], which leads to less consumed time and less possibility of being error-prone. Speech recognition application must use domain-specific grammar in order to meet the requirements of that domain. In order to do so, the developer needs to manually develop a grammar that is specified for that domain, which is both very time consuming and error-prone. It also requires expertise knowledge on the domain. The paper focuses on automating this process. That is, by taking advantage of many different sources of prior information to ease the grammar development process. Results of the experiments show that machine aided grammar development achieves better understanding accuracies with less effort that manual development.

Another research paper done at Microsoft Research Group is about language modeling for voice search [13]. This article is quite related the research area of this thesis since one of the functionalities of the speech applications to search for a spoken phrase. The paper mainly focuses on the usage of translation in language modeling. Authors of this paper first describe the process of recognition and then purpose a statistical translation model that automatically converts the original form of listing to its query forms, which works as training data for the voice search language model. Results suggest that applying such algorithm where the spoken phrase is used for training the same language model decreases perplexity and OOV rate, therefore increases the accuracy of ASR.

The concepts discussed in the above papers are well researched by Microsoft, and researcher tried to improved different aspects of ASR; i.e. understanding Accuracy, perplexity and human involvement, with the aim of improving the overall system. In general these papers focused on designing the language model so that an ASR system can have a reliable accuracy.

Although these papers have not addressed .net framework as their main development resource, but algorithms and techniques suggested by them are quite useful and relative to the focus of this paper and therefore can be applied.

6. Research Method

The aim of this research is to extend the capabilities of speech recognition using .net framework in windows platform, that is, by increasing the accuracy effectively so that speech could be a part of input for applications. Therefore, it can be said that this research seeks to extend the boundaries of capabilities by creating new and innovative artifacts. Such research is considered to be in design science (DS) paradigm. In DS, knowledge and understanding of a problem domain and its solutions are achieved in the building and application of the designed artifact [14]. In this case, the artifact would be the speech prototype that is developed using different approaches, that is to investigate the most effective method that results in high accuracy.

Fundamentally speaking, DS is a problem solving paradigm [14]. This research is a problem solving research, since it focuses on solving the problem of accuracy of speech recognition, and it seeks to define an idea (method) in producing accurate speech recognition applications.

In [14] it is mentioned that an artifact is often the object of study. This is basically true for this research, since the main goal of this research which is the investigation of methods to design ASR systems, is studied via examining the artifact (the speech prototype).

Here, we try to create and evaluate the artifact that is intended to solve the accuracy problem. In order to do so, first the prototype needs to be developed using our approach, then compared with existing approaches and finally analyzing the result this comparison. From this analysis, it would be clear for the researcher that which one of these approaches leads to the highest accuracy. This analysis leads to quantitative evaluation of the artifact, that is, numerical analysis. Hevner explains in his paper that a mathematical basis for design allows many types of quantitative evaluation of the artifact, including analytical simulation and quantitative comparisons with alternative designs [14], which is the exact case of this research. The evaluation of the artifact provides feedback and a better understanding of the problem, which is useful for improving the quality of the product and the design. This makes us to be involved in both design process and design artifact.

The process of constructing the prototype and examining it enables us to understand the problem addressed by this research and the feasibility of our approach to the solution will be examined [ref]. In DS, design is both a process and a product, that is, set of activities (methods) and artifact (prototype) [14].

In this research, mathematical methods are used to evaluate the quality and effectiveness of the prototype. As in DS, computational and mathematical methods are primarily used for evaluation of such factors. This leads to a quantitative analysis, that is, by comparison of different approaches [14].

The guidelines applied on this research would be; creating a purposeful speech prototype specifically for the problem domain of accuracy. Then the evaluation of this prototype takes place which is comparison. This results in solving a known problem in more effective manner. This artifact is of course coherent and internally consistent. Finally the results will be described to illustrate the solution. These guidelines are the ones applied on every design science research [14].

Based on the above discussion, Design Science paradigm has been chosen to be applied on this research. Moreover, it is also a quantitative research since numerical analysis and comparison does exist in this context.

As mentioned above, there are activities in a design science research; build, evaluate, theorize and justify. Basically the output of such research would representational constructs, models, methods and instantiations [15]. This research's aim is at improving Information Technology performance, i.e. improving performance (accuracy) of ASR systems. Therefore it is a knowledge-using activity rather than knowledge producing. In other words; it's a prescriptive research rather than a descriptive research [15]. Here we attempt to create idea that serves for human's purposes [15], that is, increasing accuracy of speech recognition so that users (human) can use it as an alternative way of input (alternative to keyboard and mouse). We try to produce and apply knowledge of speech recognition in order to create an effective artifact, which is the key factor in design science [15].

According to [15] There are four types of design science products; Contracts, models, methods and implementations. In this research, the desired product is a method. That is because; we develop a way to perform a goal-directed activity, i.e. a method in designing accurate ASR systems using .net framework [15].

7. Implementation

There are two factors that indicate the overall accuracy of an ASR system [12] [5]; the design of the engine and the design of the language model. These two factors are essential for an ASR system and needs to be designed carefully so that the whole system would be reliable in accuracy. Typically in an ASR system, each one of the mentioned factors should be designed separately and then combined together. Here, first we will describe the possibilities in designing each factor, and then we'll discuss the possible combinations of them.

7.1. The language model

The language model (LM) is what holds the definitions of any ASR system. Without an LM, the ASR system is not able to recognize any phrase. In other words, LM tells the recognizer (engine) what to listen to and recognize. LM is a set of words and phrases that can be recognized by the recognizer. .net framework provides two approaches on designing LM:

7.1.1. Traditional Approach

The first one is a simple and traditional approach, where each phrase (or word) is defined in the LM as a separate item. For example, consider the phrases "search Wikipedia" and "open calculator". Each one of these (in a traditional approach) is defined as an item in LM. That is, for each phrase that the recognizer can recognize, there is an item describing that phrase in the language model.

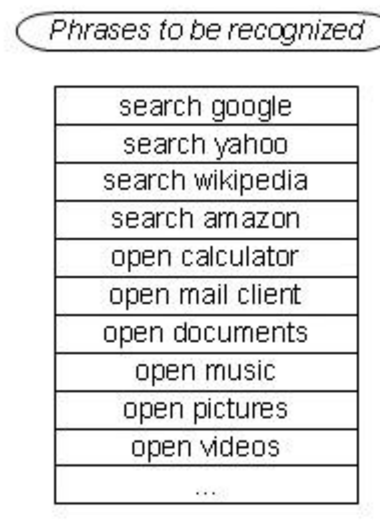

Figure 3: Traditional Language Model

7.1.2. Choice Approach

The second approach is to apply a set of choices for phrases. In speech recognition, phrases are usually combinations of words; for instance; "search Wikipedia". This item is consisted of two words; "search" and "Wikipedia". In Choice approach, this phrase for instance is split into two items, where the first item indicates the action and the second item indicates the domain of the action requested. The choice approach can be applied on every phrase regardless of the number of words in that phrase. Consider the next example; "search Wikipedia speech recognition". For this phrase, the set of choices would be; "search" which defines the action, "Wikipedia" which defines the domain of the action and "speech recognition" which defines the topic of the action.

Figure 2 illustrates the idea of applying choice approach in a language model. Note that; in this example, there are three sets of choices; initialize, action and domain. The first set of choices in Domain (Google, yahoo …) is connected to "search" action and the second set (calculator, mail client …) are connected to "open" action. In this way, by the time that recognizer recognizes the action, will move to the correspondent set of choices. Therefore the items to be listened for are limited, which increases the accuracy. That is, as soon as the recognizer recognizes an item in action (e.g. search) it will eliminate the rest of the choices (e.g. open), therefore the correspondent items in domain will also be removed (calculator, mail client, …), that is since the recognized item was search and the recognizer only listens for that set of choices (Google, Yahoo , …). This way the recognizer is constantly increasing the accuracy by reducing the number of items to be listened for.

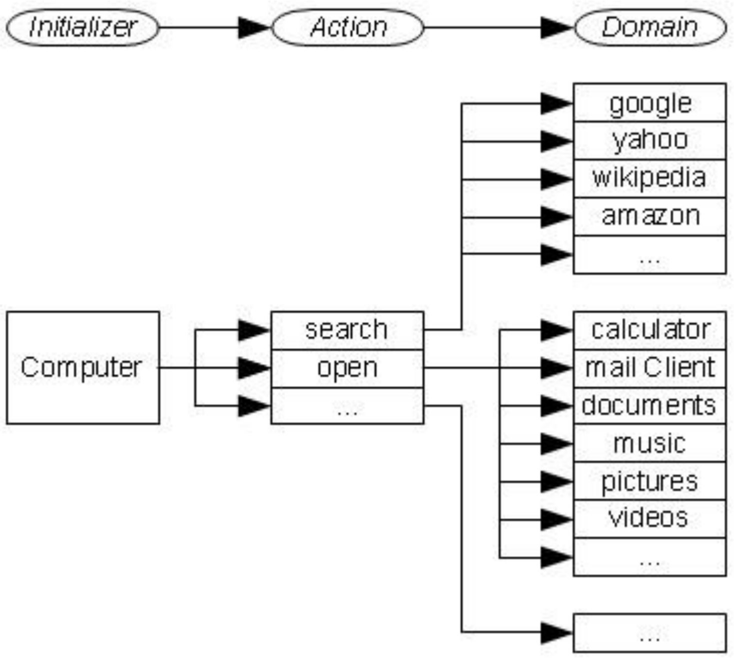

Figure 4: Choice-Designed LM

7.2. The engine (recognizer)

The recognition engine (or recognizer) is the component of any ASR system that makes the system able to recognize phrases from speech. Every time that the system receives an input (spoken phrase) the recognizer checks that input against a list of items; the language model, and retrieves the most appropriate one based on the confidence level. Retrieving the confidence level is done by SAPI which is a low level data [3] [4]. Based on this level the recognizer selects an item and passes it for further manipulation.

The recognizer can be implemented in two main ways. It can be either a shared recognizer, or it can be an in-process recognizer.

7.2.1. Shared Recognizer

With a shared recognizer, the ASR system transfers all of the data handling to the windows built-in recognition engine. That is, with the Microsoft operating system (windows XP and later) comes along a speech recognition application that is called Windows Speech Recognition which has both functionalities of dictation and command and control. This application is connected to a built-in recognition engine which is a part of SAPI. By a shared recognizer, this recognition engine is meant.

Using .net framework, developers can use the same recognition engine as Windows Speech Recognition to develop ASR applications. Implementing with this engine means using a recognition engine which is shared not only with Windows Speech Recognition but also with other ASR application implemented with this engine. In other words, applications implemented with this approach are all connected to a single shared engine, i.e. windows built-in recognition engine.

The key factor of this approach is that, since all of the applications are connected to single recognizer, it has to be loaded with those applications' entire language model. It is already loaded with a huge language model for windows built-in ASR application (Windows Speech Recognition). So the more applications connected to this engine, the more language models need to be loaded into it. In the other hand, we know the fact that the bigger a language model is, the possibility of error is higher [11] [13], and that is because the recognizer needs to check every input with a large list of phrases, and these phrases (usually) have similar pronunciations, and due to this similarity the task of recognizing the correct phrase gets harder for the recognizer.

7.2.2. In-process Recognizer

In this approach, instead of using the built-in recognizer, a dedicated recognizer is implemented. This means that, using this approach, every application is connected to its own (and dedicated) recognizer. This makes the recognizer to be responsible for only one application. In this way, unlike the shared approach, it does not need to be loaded with any other language model, only the one that is designed for that specific application.

The aim is to reduce the number of possibilities of recognition to those that are essentials. So giving access to one application, which is by designing an in-process engine that defines a specific language model for the recognizer, increases the accuracy. By default, an in-process recognizer is unloaded, that is empty from any language model, and therefore it cannot recognize anything. Then the language model is loaded into the recognizer. From there the recognition process is carried on.

The key factor of this approach is that, since there is only one language model connected to each recognizer, the possibility of recognizing the correct phrase is high. That is because of the fact that was explained above; the recognizer needs to check the input against a small list of items, therefore the possibility of correct recognition is higher.

7.3. The combinations

As it was explained earlier, there are two approaches on designing LM; Traditional and Choices. There are also two approaches on designing the recognizer; Shared and In-Process. An ASR system must contain both an LM and a recognizer; therefore there will be four possible approaches on designing an ASR system. Here, we combine all four possible approaches together and test them in order to find the most effective one.

7.3.1. Combination 1 (C-1):

Implementing a traditional LM along side with a shared engine:

This combination consists of a traditional-approached language model which is interacting with the windows built-in recognizer. That is, a sophisticated language model for testing purpose is loaded to the shared engine. Testing this combination shows the effectiveness of applying traditional approach on designing LM while choosing the shared approach for designing the engine.

7.3.2. Combination 2 (C-2):

Implementing a choice LM along side with a shared engine:

This combination consists of a choice-approached language model which is interacting with the windows built-in recognizer. A language model is designed using the choice approach for testing purpose and is loaded to the shared recognizer. Testing this combination enables us to rate the effectiveness of applying choice approach on designing LM while using a shared engine.

7.3.3. Combination 3 (C-3):

Implementing a traditional LM along side with an in-process engine:

This combination consists of a traditional language model loaded into an in-process engine. Here, we design a dedicated engine and load a traditional designed language model into it. Testing this combination enables to see the effectiveness of an in-process engine while it’s loaded with a traditional designed LM.

7.3.4. Combination 4 (C-4):

Implementing a choice LM along side with an in-process engine:

This combination consists of a choice designed language model loaded into an in-process recognizer. Testing this combination shows us the effectiveness of applying choice approach on designing LM and designing a dedicated engine for an application.

7.4. The test environment

As it was described in the combinations section, generally there are four possible approaches on designing ASR systems using .net framework. In order for our test results to be accurate, we use the same items (phrases) for all of the language models. In other words, we use the same inputs on every combination. The only difference between the LMs is the design approach applied on them.

As a matter of hardware, the very same system is used for all of the combinations. A noise-canceling microphone is used for giving inputs in order to reduce the noise and clarify the input, which makes the testing result more reliable since noise is one of the biggest concerns in speech recognition. The room and environment for testing is tried to be quite so no other noise would be recorded during testing.

7.5. The experiments

We conducted the experiment by first creating prototypes based on the four mentioned combinations. For the prototype we designed language models with the same items but different approaches and loaded them into the recognizer. We then gave series of inputs (20 items) via the microphone and compared the recognized phrase with the given input. From this comparison we drew table-1 which gives us the number of correct recognitions, in-correct recognitions and the number of times the system didn't recognize anything.

Combination No.	**No. of correct Recognition**	**No. of incorrect Recognition**	**No. of no Recognition**
C-1	16	4	0
C-2	15	5	0
C-3	18	2	0
C-4	19	1	0

Table 1: Comparision of combinations

From table-1 we concluded table-2 which illustrates the error rate. Error rate is concluded from the following formula:

$$\left(\frac{no.\,errors}{no.\,items}\right) * 100$$

In above equation, no. errors (number of errors) are the sum of not recognized and incorrect recognition. We assume both as error since it was not the correct recognition. The equation gives a percentage of error rates on every approach. With table-2 we compared the error rates of all four combinations and found the one with lower error rate. In this way we simply found the most accurate combination which results in lowest error rate.

Combination No.	**Error Rate (%)**
C-1	20
C-2	25
C-3	10
C-4	5

Table 2: Error Rate of combinations

Then from the error rate we concluded the accuracy percentage, simply by subtracting error rate from 100:

Combination No.	**Accuracy (%)**
C-1	80
C-2	75
C-3	90
C-4	95

Table 3: Accuracy percentage

7.6. Results

From the comparison of the four possible combinations, it was clear that which one results in the highest accuracy. Choosing an in-process recognizer in designing the engine significantly increases the overall accuracy of ASR system. Note that C-3 and C-4 only have 10% and 5% error rate which is quite satisfying. In the other hand, designing the language model with the choice approach also results in higher accuracy. However as it is shown in table 3, having a choice LM is not enough by itself (C-1 with 20% and C-2 with 25% error rate), it is most effective when the LM is designed with an in-process engine (C-4 with 5% error rate).

The result of this comparison suggests that the best approach for designing ASR system, when .net framework is considered to be the main development platform is to; combine an in-process recognizer with a choice-designed language model.

Certain hardware can also help in increasing the overall accuracy of the system. In our experience first we used a normal (non-noise-canceling) microphone and later on we shifted to a noise-canceling microphone. There is a noticeable difference in recognition performance when a noise-canceling microphone is used.

8. Conclusion and future work

Based on our experiments and comparisons, we suggest that the most effective approach to design an ASR system is to combine in-process recognition engines with choice-designed language model. Combining these two approaches, according to our experience results in 95% overall accuracy. Refer to appendix II for detailed design of the proposed system.

Other techniques can be applied in designing the language model to increase its performance and accuracy; use of annotations [8], example-based LM [9] and use of semantics in LM [10] [11] are applicable.

Other general techniques in designing ASR systems such as; joint training [6], user modeling [5] and cepstrum length normalization (CLN) [7] are also applicable in order to increase the accuracy.

In-process recognizes have the ability to have audio files as input [3]. That can be a considerable factor; with audio files as input, it is possible to implement an algorithm in which after receiving an input from the microphone, that input could be saved as an audio file and sent to a noise canceling filter so that recorded audio file (the input) is noiseless. Then the filtered input could be sent again to the recognizer to be recognized. In this way the audio input would be clearer for recognition. Of course the feasibility of applying such algorithm depends on the technology of noise canceling and finding an appropriate noise canceling algorithm to be applied on speech recognition.

In the future, we would like to take SAPI instead of .net framework as the main development resource and examine the API itself. Based on our personal communication with one of the head developers of SAPI at Microsoft; Mr. Robert Brown, we know that there is no 1:1 mapping between SAPI and .net framework. That means that not all of the functionalities of SAPI are wrapped by .net framework. We believe that considering SAPI as the main resource could result in a design method that is even more accurate and effective that the one we propose in this paper. That is because with limited resources we could get 95%, we presume that considering and examining SAPI can result in a higher accuracy since there are more functionalities to be manipulated.

Acknowledgment

In the end, I would like to thank Björn Olsson and Mattias Grytting for their support and guidance during on .net framework. I also thank Robert Brown for his time and support on giving information regarding Speech development and SAPI.

References

*note: References follow Harvard referencing style: http://www.library.uq.edu.au/training/citation/harvard_6.pdf

1. Wikipedia, Speaker Recognition, March 2010, <http://en.wikipedia.org/wiki/Voice_Recognition>

2. BH. Juang, L.R. Rabiner, "Journal" , Automated Speech Recognition – A Brief History of the Technology Development , Published by CiteSeer

3. R. Brown, January 2006 , Exploring Speech Recognition and Synthesis APIs In Windows Vista, Microsoft Developer Network (MSDN) , <http://msdn.microsoft.com/en-us/magazine/cc163663.aspx>

4. Microsoft Developer Network (MSDN), Microsoft Speech API 5.3, Speech API Overview, <http://msdn.microsoft.com/enus/library/ms720151%28v=VS.85%29.aspx>

5. D. Yu, K. Wang, M. Mahajan, P. Mau, A. Acero, September 2003, "*In-proceedings*" , Improved name recognition with user modeling , Published by Microsoft.

6. J. Droppo, A. Acero, May 2006, Joint discriminative front end and back end training for improved speech recognition accuracy , "*In-proceedings*" , Published by Institute of Electrical and Electronics Engineers, Inc.

7. M. Richardson, M. Hwang, A. Acero, X. Huang, September 1999, Improvements on speech recognition for fast talkers, "*In-proceedings*", Published by Microsoft.

8. Y. Wang, A. Acero, 2001, Grammar Learning for Spoken Language Understanding, *"In-proceedings"*, Published by Institute of Electrical and Electronics Engineers, Inc.

9. Y. Wang, A. Acero, 2003, Concept Acquisition in Example-Based Grammar Authoring, *"In-proceedings"*, Published by Institute of Electrical and Electronics Engineers, Inc.

10. A. Acero, Y. Wang, K. Wang, 2004, A Semantically Structured Language Model, *"In-proceedings"*, Published by Microsoft.

11. K. Wang, Y. Wang, A. Acero, 2004, Use and Acquisition of Semantic Language Model, *"In-proceedings"*, Published by Association for Computational Linguistics.

12. X. Huang, A. Acero, F. Alleva, M. Hwang, L. Jiang, M. Mahajan, May 1995, Microsoft Windows Highly Intelligent Speech Recognizer: Whisper, *"In-proceedings"*, Published by Institute of Electrical and Electronics Engineers, Inc.

13. X. Li, Y. Ju, G. Zweig, A. Acero, March 2008, Language modeling for voice search: a machine translation approach, *"In-proceedings",* Published by ICASSP

14. A.R. Hevner, S.T. March, J. Park, S. Ram, 2004, Design science in information systems research. MIS Quarterly, vol. 28, 75-105.

15. S.T. March, G.F. Smith, 1995, Design and natural science research on information technology. Decision Support Systems, vol. 15, 251-266.

Appendix I – Process of Recognition

The process of recognition needs to be understood well, so the design of the system could be discussed. In the purposed design, an instance of an in-process recognizer is created and a proper audio device (microphone) is assigned to it. Then it is loaded with the proper language and model. That language model is created as a grammar instance. So in other words, the recognizer is loaded with the grammar.

The moment of recognizing a phrase is an event, which is happened when the recognizer detects an input. An event handler is assigned to this event so there is access to the property of the event, such as the text of the result of the recognition. It is important to notice that, there is not access to all of the properties of a recognition event. By that, it is meant; this event always returns a phrase, regardless of it being what was actually said by the user. For example, consider a user saying “life” and the system detects “like”. That is because these two words have almost the same pronunciation. In such case, the system carries on the actions as the value being “like”, which not something that the user inputted. The whole process of detection of words, that is, the process of translating spoken input (voice patterns) to text-words is handled by Speech API, where voice patterns are matched and from those phrases are created. There is no access to that sort of functionality. The .net framework provides a wrapper, which is System.Speech namespace that provides access only to some of functionalities on SAPI.

By initializing the recognizer and loading the proper language model, the system enters a state where is listening for any input, and any detected input triggers the event handler.

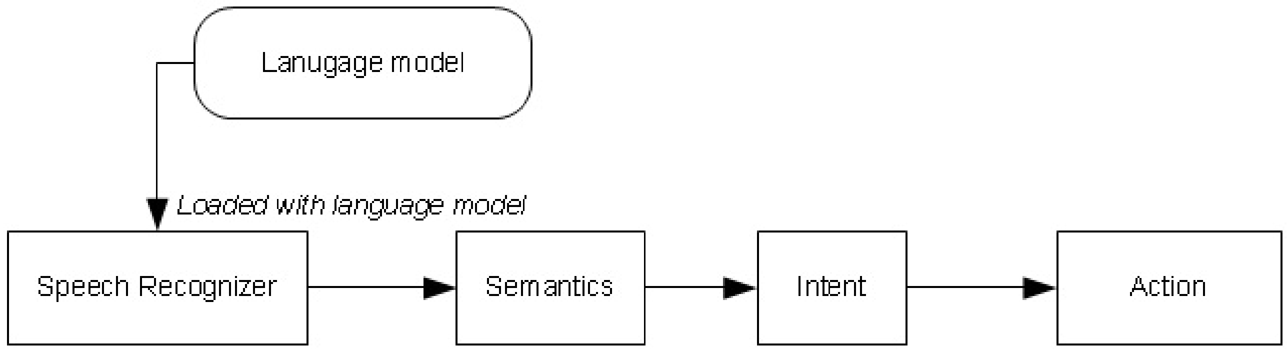

Figure 5: Recognition Process

The figure above illustrates the recognition process. The process of recognition first starts with a recognizer (either in-process or shared) which is loaded with a language model. This language model defines a set of semantics for the recognizer. Without these semantics it is impossible to recognize a phrase. For illustration of semantics meaning, consider words; yes, yeah, yep, ya. Although they are different in spelling and pronunciation, but they all have common semantic value which is yes, and that's why in a normal day communication, when we hear any of this words, our brain recognizes them as word yes. Semantics should be defined for the recognizer so that it can recognize the speech.

The next step of this process is intent. By intent, it is meant that once the recognizer recognizes a word, there is need to figure out what the user wanted. Consider the “life” example that was explained earlier, without intent, if the user speaks “life” and computer recognizes “like”, the recognizer actually takes the semantic value of the word “like” and takes action upon that. And the action will not be what the user requested. The intent part, in some sort of saying, defines the intention of user.

The action is where the system takes an action based on the decision that it made on recognition. Action is the end point of the process and usually after the action, the systems goes back to the entry point where it is listening for inputs.

In appendix II, handling every step in a recognition process is discussed.

Appendix II – System Detailed Design:

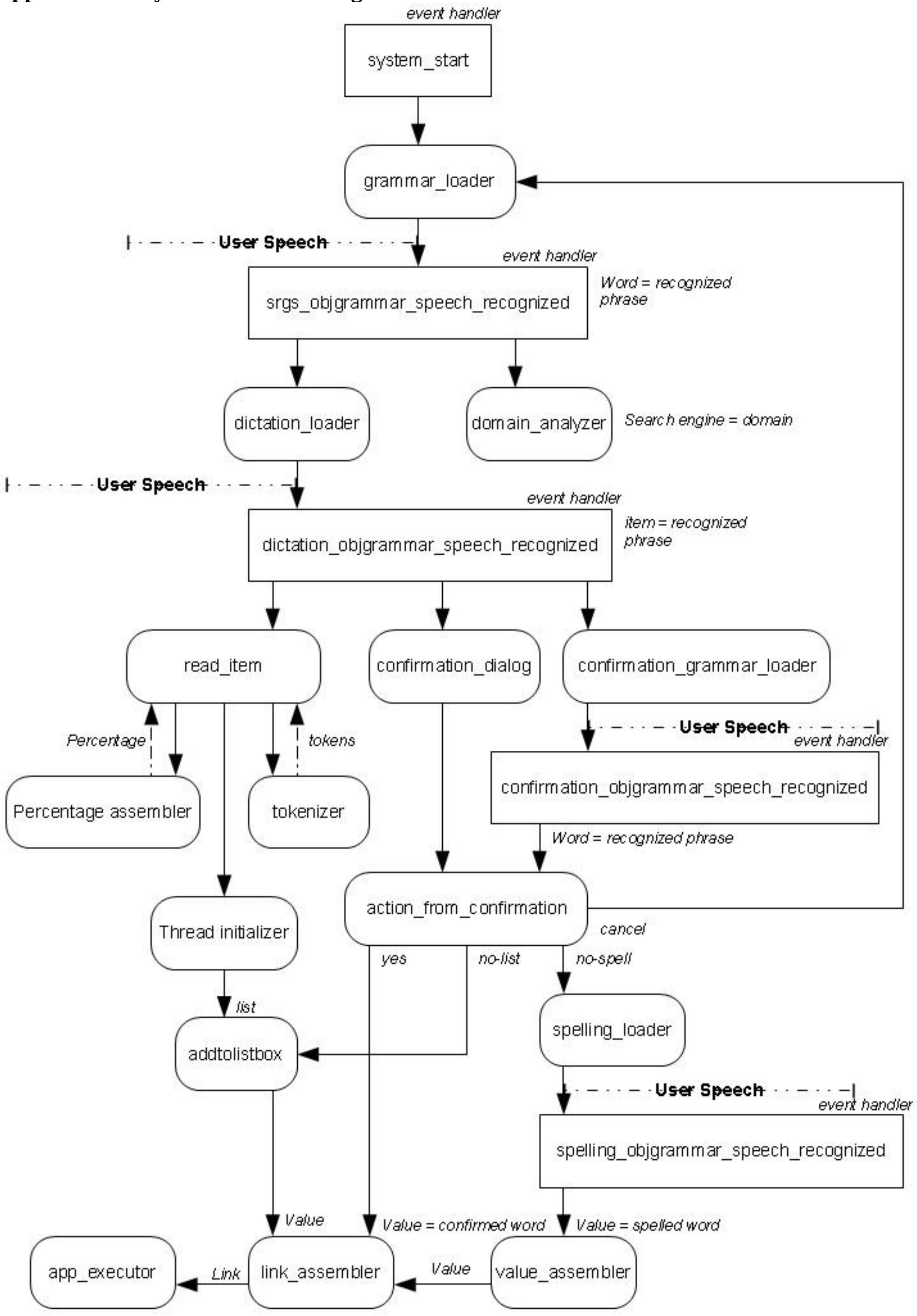

Figure 6:System Design

This section explains method of designing ASR systems using .net framework for windows platform, from the combination that was suggested in this paper:

The figure 6 illustrates the overall design of the system with its components and the interaction between them:

1. Initialization

Here, a possible algorithm will be introduced that results in high accuracy. Consider that, the algorithm will be introduced as a system.

The algorithm is consisted of different modules, and the interaction between these modules makes the recognition process possible. The desired outcome was to develop an application that can recognize certain phrases and act on them. The required actions are simple such as searching for something using a search engine. An example for input could be the one given previously. An input would be “Computer Search Google” followed by the subject to search for, i.e. “book”. Appropriate action on this input would to open the web browser and search for “book” on Google. The input consists of an initialize-word and two levels of choices, phrase “computer” is the initialize, “search” is the first level of choice which identifies the action, and “Google” is the third level, which identifies the search engine. In order to reach the final step, the recognizer needs to successfully recognize the initialize and first level.

2. Grammar Loader

The system starts by an event handler, this event could be a button click or even system load. That means the recognizer can be initialized at the moment that the system is loaded or at the moment that a button is clicked, both are considered to be events. In this event, sub-system grammar loader is initializing the recognizer. At this stage, the recognizer is defined to be an in-process one. The positive point of in-process engine is that, it is not loaded with any language model, and needs to be loaded with a sophisticated language model. In this way, the language model will hold only the definitions that are needed for the application. In other words; the recognizer will only listen to limited number of phrases, and that increases the accuracy noticeably.

The grammar loader initializes the recognizer, by first setting the input to the default system's audio input (usually the microphone), then loading the appropriate language model to it.

3. Speech Recognized Event

After initialization is completed, the system enters the listening mode. Here, it seems to be at halt. Actually the system is listening for any input given from the microphone, and as soon as it detects something, an event is triggered; speech recognized event. At this point, the inputted sound waves are detected and converted to string by Speech API, after conversion, the result is checked against the grammar that was loaded at initialization, and then it is passed to the event as parameter. The result data contains some properties that not all of them are useful for the purpose of accuracy increase. Among these properties is text which holds a string. The string is then manipulated in order to extract required information. The purpose is to find what was the intention of user, whether opening a calculator or searching for something on a web search engine. The domain analyzer sub-system sets the search engine by this manipulation. By a successful recognition, the process is carried on by entering the dictation state.

4. Dictation Loader and Dictation Event

When the manipulation is done by domain analyzer, the system enters diction state, and that is by unloading the current grammar and loading the dictation grammar. The explanation of dictation grammar will be on grammar model section. Dictation grammar also returns a string that is sent to dictation speech recognized event to be handled.

At this point, considering figure 5, the system reaches the intent part. Here since the probability of dictation is very high, due to large number of items that can be dictated, it is wise to check for intent, i.e. by confirmation.

5. Intent and Confirmation

In this algorithm, in order to handle and apply intent, three synchronized approaches are given:

Candidate list: In this approach a candidate list is created for the user to select from. This candidate list contains the matching items that are similar. Concept of similarity can be viewed from two perspectives, similarity in pronunciation and similarity in spelling. Here an algorithm has been applied to process this comparison of similarity. Percentage assembler sub-system checks the dictated phrase against the dictation grammar model, letter by letter and assembles a percentage value based on this comparison. For instance, consider words “like” and “love”, the result of comparing these two would be 50% similar, since there are two similar words at the same positions. That is, letter “l” in the beginning and “e” in the end, two similarities from four, makes it 50% similar. The value is then sent to another sub-system, i.e. read item, which reads the items from grammar, apply the algorithm, and returns a list of words that are above the given percentage, e.g. 60%. From this list, the user can select an item, if the recognized phrase was not the dictated phrase.

Confirmation Dialog box: another approach that is considered to be running at the same time of candidate list is a confirmation dialog. This dialog is a simple yes, no and cancel dialog, in exception that saying “no” can lead to two options. If the recognized phrase is not what the user has dictated, the user has the option to go back to graphical user interface and select an item from candidates list, or spell the word to computer. Note that, this dialog accepts mouse clicks and its life time depends on the next approach and considered to be an alternative way of confirmation. Each one these buttons represent a type of selection, which should be taken care of accordingly.

Confirmation Grammar: this approach enables user to confirm using his speech. At the start point of intent, the in-process recognizer is unloaded from current grammar and loaded with a confirmation grammar. Details on confirmation grammar will be on grammar model section. This grammar returns four different semantic values, each one representing a type of selection. Selections could be either yes, no-spell, no-candidate list and cancel. At the moment of recognition the same process as speech recognized event occurs, which returns the selection value.

By having these alternatives, the intent part is handled well with high accuracy, the percentage assembler reliably compares the given words against the grammar and assembles candidates list based on that, and confirmation grammar and dialog box, enable for both speech and mouse inputs.

6. Spelling loader and Spelling Event

Spelling loader sub-system is responsible for loading proper spelling grammar. Spelling grammar returns a word after each recognition process. That is, by retrieving the recognized phrase's semantic value. The spelling grammar contains definitions and specifications that will be explained in grammar model sections. At each recognition, an event is triggered which holds the recognized phrase as parameter. The process of speech recognized event is as same as other speech recognized events explained previously.

When the event is triggered, the phrase is manipulated and the semantic value of that phrase is extracted. This value is then sent to the value assembler sub-system. The responsibility of the noted sub-system is to construct a word from the spelled letters. Consider that the spelling process should be continued until requested to be done. A semantic value has been defined in the spelling grammar, when retrieved will cause the spelling to terminate, i.e. “finish”. Also considerable to note that each letter is recognized in a separate event, and value assembler needs to get input each time a phrase is recognized. This is what caused a problem, the problem was that each time the recognizer recognized something, the value gets set to the new phrase, and therefore the value assembler always would get one value. In order to solve the problem, the recognition process was set in a loop repeating while the recognized phrase is not the terminator phrase. Each time that recognition takes place, the phrase (letter) is added end of the value and a string is constructed.

As a matter of fact, an approach to increase understandability is to train the system. In this algorithm, after each spelling, the spelled word is added to the dictation grammar, so from next initialization, the new word can be recognized at dictation, and the user won't need to spell the word again. In this way the computer is learning each time a new word is spelled.

7. Action, App executor

Up to this point, the value can be taken from three alternatives; from first recognition, which is dictation, and the user confirms. Form candidates list, if the user wishes to select from it. Or from spelling, it user wishes to spell it.

Regardless of how this value is through intent, an action should be taken for it. The link assembler first assemblers the requested link and sends this link to application executor to execute the link in the web browser. Assembling link needs the search engine which was analyzed by domain analyzer and the value as parameters.

8. Grammar Model

The grammar model consists of different elements for different purposes. There are four main elements in the provided grammar model. In grammar modeling, these elements are referred to as rules that are identified via their unique rule id. The first element defines the initialize word and contains the two levels of choices linked to the initialize word. When it is said that levels of choices are linked to initialize, it means that in the grammar model (which is in XML format), the rule id of choices are used as references in initialize rule. That is, at recognition, it will be referred to choices. Notice that, this linking process is done in the grammar model, i.e. W3C Speech Recognition Grammar Specification. These elements together, return a string that contains the initialize word, i.e. “Computer”, the first level, i.e. action, and search engine. e.g. “Computer Search Google”. This text is what is manipulated to extract the domain and action. Sub-system grammar loader is responsible for loading these elements into recognizer at the time of initializations. Detecting a phrase from these elements causes the speech recognized event to be triggered.

Dictation element is a list of possible words to be dictated. This element is not linked to any other elements of grammar. Instead it is called via .net namespace during run-time. That is, after completing the first phase of recognition which is getting the domain and action, sub-system dictation loader, loads this rule to the recognizer. Dictation event is triggered when the recognizer detects a phrase from the rule. By recognition, the event holds the detected phrase as string. This string should be the value; therefore it needs to be confirmed.

Confirmation element is designed to handle intent and confirmation. As the initialize, confirmation is also linked to four other elements. These elements hold the definitions for each type of selections at confirmation, which are; yes, no-candidate, no-spell and cancel. Each one of these elements, on recognition returns a single semantic value indicating a response type. This semantic value is then manipulated to take the appropriate action.

The spelling element is for the purpose of assembling word from spelled letters. More than one item is assigned to each letter so the user can give more than one possibility to spell a letter. For instance, “A as in alpha”, “A as in apple” and “A as in Adam”; all return letter “A” as the semantic value of recognized phrase. This value is then sent to value assembler to assemble the value.

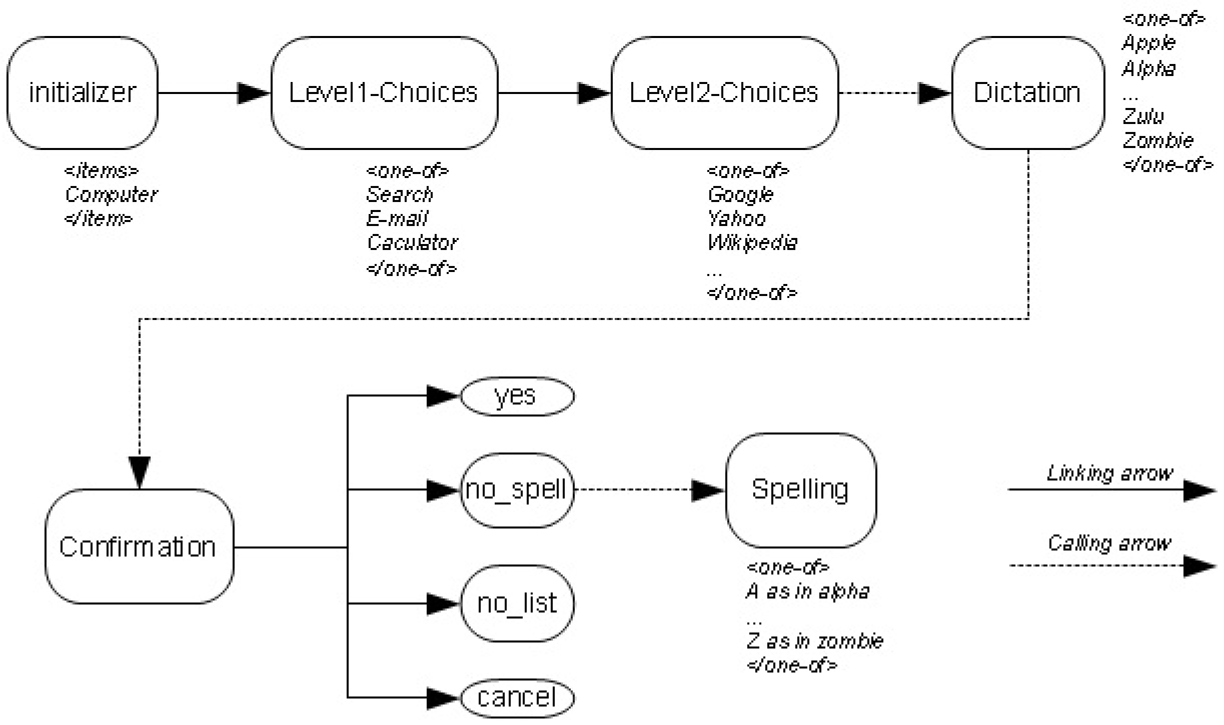
initializer
Level1-Choices
Level2-Choices
Dictation
<one-of> Apple Alpha ... Zulu Zombie </one-of>
<items> Computer </item>
<one-of> Search E-mail Caculator </one-of>
<one-of> Google Yahoo Wikipedia ... </one-of>
Confirmation
yes
no_spell
no_list
cancel
Spelling
<one-of> A as in alpha ... Z as in zombie </one-of>
Linking arrow
Calling arrow

Figure 7: Grammar Model

The diagram above illustrates the design of grammar model by indicating the relationship and connection between elements. As it is illustrated, the dashed lines indicate connections outside grammar file, which is using .net namespace. The continued lines indicate linking in grammar model.

Outcomes:

The explained design suggests that these factors should be considered*; Recognition Engine*, selecting an in-process engine results in better accuracy and performance compared to shared engine. *Choices*, enables the recognizer to detect phrases more accurate, that is linking levels of choices and reducing the number of possibilities. *Intent and confirmation* increases the accuracy of the application in overall, since the system learns each time the user confirms a phrase. By *spelling* and adding the item to grammar, the accuracy also increases, since the next time recognizer knows what to listen for.

Regardless of accuracy, the design also increases the usability of the application, since the user has many alternatives to input the value, that is; either by dictation, candidates list or spelling.