University of Cape Town

Computer Science

Honours project report

Speech interfaces to Expert Systems on Android

Kevin Brenkel

Supervised by: Dr Audrey Mbogho

Student:	Kevin Brenkel
Semester:	Honours computer science
Student ID:	BRNKEV008
Birth date:	3 may 1988
E-Mail:	need.some.code@gmail.com

Abstract

Mobile speech recognition and speech synthesis technology has opened a possibility to bridge the digital divide. This is done via a speech driven interface. The system proposed, ADVISOR, provides a means for those who are not computer literate and/or are functionally illiterate to interact with a medical expert system. This research project investigates the possibility of a speech driven interface for users whose first language is anything other than English. Factors considered are the speech recognition technologies, specifically Google voice recognition (Google ASR) and Pocket-Sphinx. Also considered is how user friendly the users find the interface in terms of simplicity, satisfaction and understandability.

During the creation of the Interface it was necessary to 'port' Jess to Android using openbeans. Jess on Android is fully functional and works efficiently. It is capable of almost all the functionality of Jess version 6, except for some Jess.awt and Java.applet functions. It was also required to port Pocket-Sphinx to Android, and an Android library was produced.

Evaluation was carried out to determine the word accuracy of Google voice recognition and Pocket-Sphinx. Evaluation was also carried out to see if the system was user friendly. Both Qualitative and Quantitative data was analysed.

The results indicate that Google voice recognition has a Word Accuracy (WA) of 69.35% for all words and a WA of 82.69% when words that are difficult to recognize are removed. Pocket-Sphinx had a WA of 17% and therefore it is not practically usable for this purpose.

Even though participants were not happy that the speech recognition could not pick up some of their words in the experiment, they indicated that they were satisfied with the interface. The Interface is feasible if the user is not required to use words that are difficult to recognize.

	Category	Min	Max	Chosen
1	Software Engineering/System Analysis	0	20	10
2	Theoretical Analysis	0	25	0
3	Experiment Design and Execution	0	20	5
4	System Development and Implementation	0	15	15
5	Results, Findings and Conclusion	10	20	10
6	Aim Formulation and Background Work	10	15	15
7	Quality of Report Writing and Presentation	10		10
8	Adherence to Project Proposal and Quality of Deliverables	10		10
9	Overall General Project Evaluation	0	10	5
Total marks		80		80

Table of Contents

List of Figures

List of Tables

Appendices

List of Abbreviations and Symbols

GUI	Graphical User Interface
VM	Virtual Machine
OS	Operating System
NDK	Native Development Kit
JNI	Java Native Interface
GHz	Gigahertz
ms	Milliseconds
CPU	Central Processing Unit
RAM	Random Access Memory
3G	3rd generation of mobile communication technology
API	Application programming interface
SDK	Software development kit
Jess	Java Expert System Shell
OHA	Open Handset Alliance
CLIPS	C Language Integrated Production System
COOL	Clips also has the CLIPS Object Orientated Language
OO	Object Orientated
SWIG	Simple Wrapper and Interface Generator
WA	Word Accuracy
ASR	Automated Speech Recognition
UCD	User Centred Design

1 Introduction

Medical Android apps may never be able to replace a qualified Doctor and might not be able to assist in curing various diseases, but they could help manage the treatment of some diseases or even give advice to help prevent these conditions. With the ever increasing cost of medicine and medical treatment the main purpose of a medical advisor app, ADVISOR, is to assist patients in lifestyle management which could lead to a longer and healthier life and reduce the financial burden for the patient. The aim of this app is to give access to the above services to those who are not computer literate and/or are functionally illiterate. The cost of the middle to later stages of a disease can be financially crippling and stressful to the family.

The app to be designed in this project will use speech input and speech output as it's primary interface (easily replaceable with a GUI). The front-end will use text-to-speech to speak to users, and speech recognition with a limited vocabulary for user input. Two types of speech recognition will be used, Google voice and Pocket-Sphinx. The front-end will also check if the input is correct. The middle-end will be for communication between the interface and the expert system and the back end will be the Java Expert System Shell (Jess) running on the actual phone.

The App will be made with user friendliness in mind, so that users require little or no training to use the interface.

Good usability is defined as “the extent to which a product can be used by specified users to archive specified goals with effectiveness, efficiency and satisfaction in a specific context of use”[1]. To be deemed usable an interface has to be perceived as usable to it's target users, in this case users who are not computer literate and/or are functionally illiterate. As the target users is assumed not to be computer literate a interview style speech based interface will be used. This is because 'having a conversation' needs little explanation.

It is important that the interface is understandable so that users will not give incorrect answers resulting in incorrect output from the expert system. Accurate recognition is important so users are not irritated by the interface.

“A good user interface caters to end users and supports them in the tasks they wish to undertake”[1] thus in any question asked by the interface, the user must be sure what the expected input must be.

2 Background

2.1 Android and Android Applications

2.1.1 The android platform

Android refers to a complete set of software for a phone, including the Kernal, Libraries, runtime environment and key applications[2]. Android is designed in layers, the highest being the application layer and the lowest being the Kernel [3]. The layers are open source and available for any individual or industry. Android Development can be done in various development environments. For the purpose of this study Netbeans, with the NBandroid plug-in for Netbeans 7.2, was used (available from http://www.nbandroid.org/). At present Android runs on a Linux Kernel version 2.6. The Open Handset Alliance (OHA) develops open standards for phones, and allows anyone to develop Android for their own hardware. Android was designed with openness in mind, and is made to be as flexible as possible.

Users get to control their own Android experience as apps can be easily added to a phone (no restart is required). Users can choose their default apps and even replace the core apps with a third party application.

Developers do not need any special licence to begin programming. A phone can be changed to development mode in the Application settings of that phone. Developers have full control to "Integrate, extend and replace"[2].

An Android application runs on the Delvik virtual machine, and each application is a separate process.

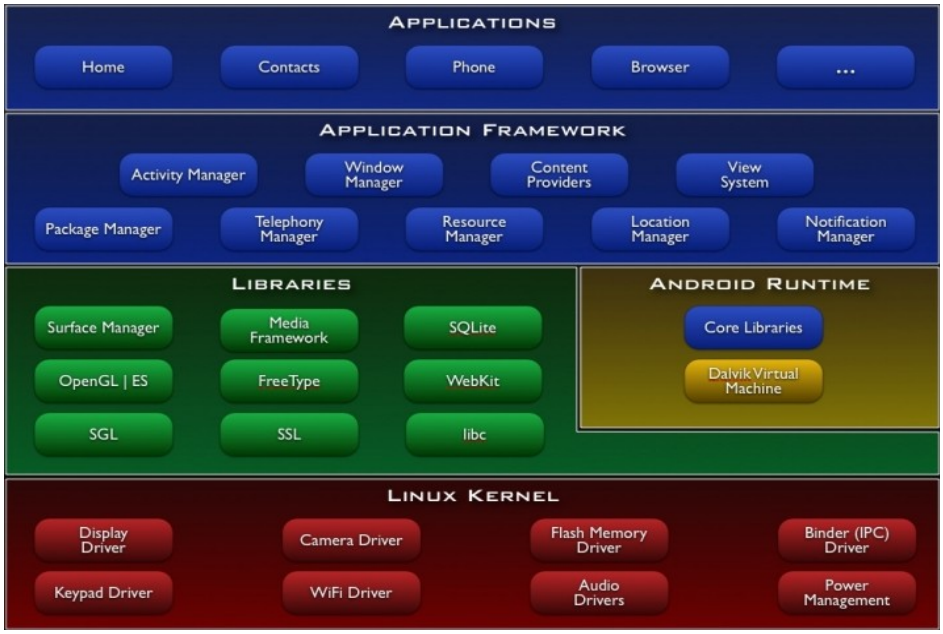


Fig. 1: Android Layers

Source: http://shareme.github.com/articles/2012/05/04/what-is-android/

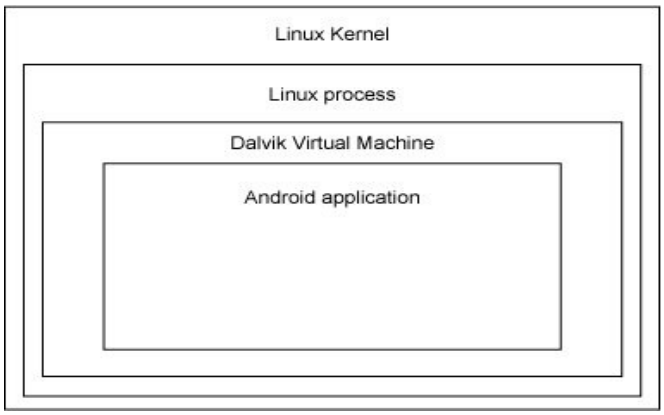


Fig. 2: An application on Android.
Source:An introduction to Android [2].

Table 1 shows that Android is an increasingly popular Operating System for phones[4], [5]. Table 2 shows is outperforming its opponents in it's functionality[4]. On 9 October 2012 there were 534544 apps for Android [6] .

Android provides a log for for collecting and viewing debug output called the logcat. Logs from various applications and portions of the system are collected in a series of circular buffers, which can then be viewed and filtered [7], Netbeans provides an interface to the logcat.

	2011	2015 (predicted)
Android	38.9	43.8
Blackberry OS	14.2	13.4
Symbian	20.6	0.1
iOS	18.2	16.9
Windows phone 7/Mobile	3.8	20.3
others	4.3	5.5
Total	100	100

***T*ab. 1: SMARTPHONE OPERATING SYSTEM MARKET SHARE ESTIMATIONS**
Source: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6028560

	Android	Symbian	Win Mobile
Portability	1	0.5	0
Reliability	1	1	1
Connectivity	1	1	1
Open system	1	0.5	0.5
Kernel size	0.5	1	0
Standards	1	0.5	0.5
Special features	1	0	0.5
Total	6.5	4.5	3.5

Tab. 2: RESULT TABLE FOR OS FEATURE COMPARISON
Source: International Journal on Computer Science and Engineering[8]

2.1.2 Android SDK and API

The Android SDK is publicly available for download, it includes samples, tools, an emulator and the necessary libraries [9]. There are various API levels and SDK platforms corresponding to the various operating systems. Android systems are forward compatible unless combined with private APIs.

Android applications made in Java are converted to .dex format for the Delvik VM to execute on the phone. This VM is optimised, using registers, for low memory use[2]. However not all Java libraries are included in android, specifically Java.bean and Java.awt.

All the libraries necessary for development are contained in android.jar. There is a different android.jar for each OS version. The SDK provides the necessary tools and libraries for a developer to write apps for an Android device[10] , the first SDK was released on 12 December 2007.

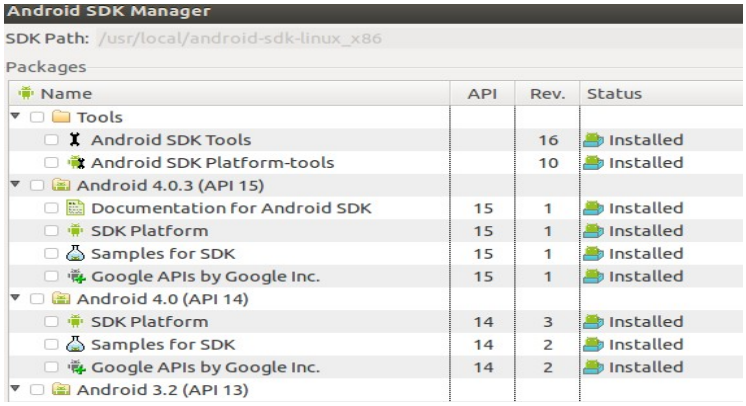


Fig. 3: Android SDK manager
Source: Own creation

2.1.3 Android NDK and JNI

The Native Development Kit will be referred to as NDK. Using the NDK a developer can program in c/c++ for the creation of their apps[11]. Android has a JNI (Java Native Interface) for the inclusion of native code. The NDK is not as simple to use as the SDK, but all the necessary

tools are provided. The NDK allows developers to embed components that make use of native code in their apps [10]. Benchmarking on a 1GHz processor(single core) revealed that native code runs on average 34.2% faster than Delvik with some tests running as high as 99.98% faster, and as low as -158.39% slower. Most tests gave a positive speed-up, with the exception of only 3[10].

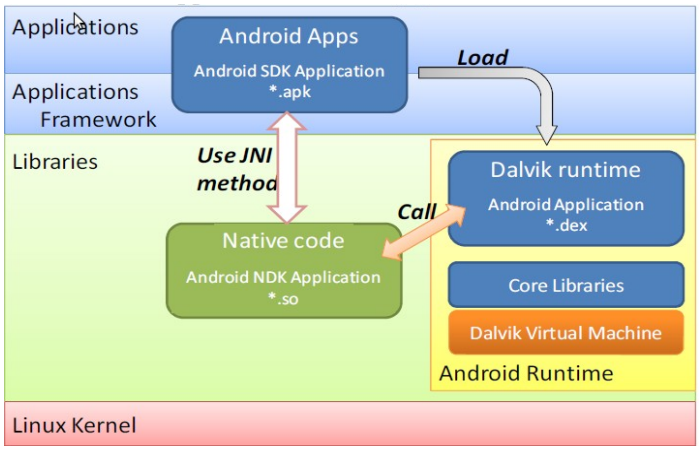


Fig. 4: Android NDK overview

Source:Benchmark Dalvik and Native Code for Android System

2.2 Expert Systems

‘‘An expert system is an interactive computer-based decision tool that uses both facts and heuristics to solve difficult decision problems based on knowledge acquired from an expert.”[12]. Expert systems can reason similarly to a human and can provide the rationale for their decisions. Expert systems are easy to scale and update.

A typical expert system consists of an inference engine, a rule base, working memory and a user interface. The inference engine consists of a pattern matcher, rule prioritiser and instructions to execute when a rule is fired. A well-known early medical expert system shell was EMYCIN, which is based on MYCIN, an expert system for the diagnosis of blood disorders. [13]. Research into expert systems began in the 1960s. These systems are programmed with an expert's knowledge and they allow non-experts to perform better but cannot replace a human expert. Thus making expert knowledge available where it may not have been[12]. Examples of modern expert systems include: javax.rules, Jess, CLIPS and Prolog [14]. The advantages of expert systems is that they offer conversational interfaces, can use a large amount of knowledge and rules, are reliable and are easy to change.

Expert systems have been used for medical diagnosis [15], decision support[16], market analysis and forecasting [17], sheet metal die on CAD[18], weight management[19], DNA analysis, nutrition diagnosis [12] and many more. The first expert system to be used in clinical medical practice was probably PUFF, which was used for pulmonary function testing [20].

Creating an expert system involves:

- determining an exact and specified problem,
- gathering accurate knowledge from experts or text-books,

- converting the knowledge into a form that a computer can understand (e.g. rules),
- selecting or creating an expert system language and development environment (e.g. Jess on Android),
- creating a knowledge base in the selected language and
- testing [12].

2.3 Expert System Interfaces

2.3.1 Question-Answer and Menu

When expert systems were first used the interface was modelled after interactions with human experts (the question-answer model).However this came with a ready made set of interaction problems.[21]

Various studies have been performed between users and advisers to record the interaction between the parties. In a study preformed by M. J. Coombs and J. L. Alty [21] users were given queries to ask the advisor, and then had to rate that experience. It was found that the expert users were much more satisfied with the advice given than novice users. Novice users found the advice insensitive, unhelpful, not meaningful and seeing an advisor was seen as something painful. The advisers claimed that the queries were ambiguous 42.9% of the time, and had to be made clearer. It was also found that the conversation between the advisor and the user was very complex. The content of the conversation benefited both the user and the advisor- the advisor tried to clarify his/her thoughts and the user judged whether the advice was relevant. When the advisor was giving a solution, it was not always clear what the solution was or how the advisor got to that solution and only 32% of users would get a justification for the solution.[21]

“Menu one way” (Socratic/tutorial style), was found to be too complex and involved for the average user, and too simplistic for an expert. With one way interaction there is no way of knowing whether the user understood the output, or if the output was even meaningful. Two way dialogue was judged to be better by the users. Providing advisory commentary is one suggested way to improve communication- that is telling the user why the next step needs to be made or how the expert system arrived at a decision[22].

The question-answer and menu one way interfaces were used as a front-end for the MYCIN expert system. MYCIN is a medical expert system that allows users (physicians) to interface via a menu/question-answer system. In MYCIN most of the branching is done via “yes” or “no” questions and users cannot easily go back to change an answer. A user must also go though a great deal of dialogue before MYCIN can give a diagnosis. MYCIN can tell the user the rule numbers that it used to make the diagnosis, but cannot explain why in understandable English. A natural language interface could reduce the amount of time that the user must spend interacting with MYCIN. A menu system has the weakness that the user cannot provide data which s/he feels is important to the diagnosis but has not been included in the menu path.[15]

2.3.2 Data interfaces

In the EMERGE Medical expert system (which diagnoses chest pain) it was important in the interface design to make sure that users could access the system rapidly and easily. There were three types of interfaces to EMERGE, a selection interface, a question interface and a data interface. The question interface was designed with training new users in mind. Novice users could simply answer a series of questions with “yes” or ”no” and in doing so learn the terms in the questions. Data mode was intended for more advanced users, where they could simply and quickly enter the terms in text format and the expert system would then only ask a question if it needed specific data. However, it was found that entering a long list of terms was too cumbersome. [23]

2.3.3 Moving forward from the basic interfaces

“One of the clearest lessons learned from early pioneering expert systems is that excellent decision making performance in itself will not guarantee user acceptability”[24]. The designer needs to involve users in the process of design to get a clear understanding of user needs and requirements. This increases the user acceptance of interfaces. Involving users early in the creation of an interface can make major adaptations to the interface possible, but some people believe that user models could solve many of the interface problems. Regardless of how this is done, the communication between the expert system and the user has to match the communication style of the user.

The menu interface typically has few options and mostly all the users can input is “yes” or ”no”. In this type of interface users may have to answer a large number of questions, some of which may seem irrelevant. It was noticed that users wanted to volunteer information rapidly in an order which suited them. A graphical interface was suggested so that users could interface directly with the system. The menu system also differs from the interactions of a human expert, who would engage in cooperative problem solving whereby users would take an active role in solving the problem and would also provide constraints to the solution. The human expert would often give a heuristic solution very early in the diagnosis process and later refine the solution. [24]

Another proposed interface model was the mixed initiative dialogue interface, where the system would try to take whatever possible knowledge from a database instead of asking it. Most organizations found it complicated to use (as novice users needed to be guided more), while safety critical systems did not want any assumptions to be made. However, mixed initiative dialogue did work for doctors, because when doctors perform a critical procedure they do not have time to answer many questions when a patient database could be consulted. This allowed the expert system to give real-time answers to problems. The mixed initiative dialogue was still limited, as it could not answer questions about its workings such as “why does the remedy work?”. To put it simply the user and the expert system could disagree, due to assumptions, on what the problem was.

It is very important that a system have good explanation facilities. Most commonly, systems simply output the rule, this is because the explanation system is used as a debugging tool instead of a tool to interact with users- that is it is used for the benefit of the programmer and not the user. Thus they rarely provide an acceptable explanation to the user.[24]

To address the problems mentioned above, a new interface was designed for EMERGE. The new interface printed a numbered list of terms on the screen and asked the user to simply input the appropriate numbers separated by a comma. This interface was fast and simple to use. [23]

2.3.4 Graphical interfaces

In classical (menu) interfaces to expert systems the user's role was to reply to prompts and accept the final output. This only allowed very simplistic interactions. A graphical interface is capable of significantly increasing the complexity of interacting with an expert system, allowing much more complex problems to be presented to the system. An average person is used to the spatial representation of a problem for example finding a book in a library or finding a location on a map. One factor in making graphical interfaces more efficient is that in humans perceptual processing (graphical) is an order of magnitude faster than cognitive processing (speech/menu) [25]. A graphical “hypermap” interface was built for the ChEM expert system, exploiting the direct manipulation design principal. This method allowed the user to focus on the problem at hand and not the interface. This method also increased the accessibility of the system. About 40% of the classical interface can be efficiently mapped to a spatial environment.[25]

It was argued that a good cognitive model would enhance the human-computer interaction and a graphical interface and explanation was better than a textual one. A cognitive mental model could help the user to understand the rationale of the system as well as how the results were inferred. Providing the user with explanations helped the user to trust the system. There were significant differences between using a text based interface and a graphical interface. Users with a graphical model had better performances than those without it.[26]

2.3.5 Natural Language

Natural language can solve many problems with expert system interfaces, but it comes with it's own set of issues.

One of the reasons behind implementing natural language interfaces was that users found it easier to communicate with natural language than with an artificial language. But implementing a natural language interface was much more complicated than other types of interfaces. The reason being that this interface could not simply communicate with the expert system materialistically, but a far broader information transfer was needed. Any expert system that has a natural language interface must be designed with this in mind. Implementing a natural language interface requires that the expert system should have natural language support capabilities.[24]

When a user needed help, they often found that the help messages were unhelpful and insensitive. The interfaces needed to use the expert system to produce advice-giving capabilities

that effectively supported the needs of human users. A user model is what the system knows about the person interacting with it. This is useful to ascertain the needs of the user, in order to create the interface to serve the user and not merely the programmer. Natural language is acknowledged as important to natural language interfaces- but there could be consequences when the user goes away from the template and gets garbage from the advisory system, some have said that it causes more problems than it solves. A user models, such as hiding advanced features from new users is one suggestion. In addition one needs to take into account the vocabulary and the goals of a user. (The user may not know what the problem is, or what the problem is called)[22]

Friendly interfaces which require minimal user training and do not place too high cognitive demands on users, allow them to focus on the task at hand. XCALIBUR is a Natural language interface to XCEL which is a system that provides automated assistance to salesmen, which uses DYPAR-II as a natural language parser. XCALIBUR can handle a variety of natural language commands, but it struggles with ellipsis. XCALIBUR then gives natural language output to give the user the answer to a query. It is a flexible and robust expert system interface[13]

A natural language interface was successfully implemented into SOPHIE I, an expert system dealing with electronic circuits. The interface translated constrained natural language commands into instructions for the SOPHIE I back-end. One interesting point was that when students saw that SOPHIE I could understand natural language commands, they immediately assumed it had conversational abilities! Students would make use of a variety of spoken language constructs that would not typically be used in written language such as ellipsis (even though it was text based). The designers then had to adapt the interface to deal with these types of queries. The designers would use input that the interfaces could not understand as test cases for the next version. The natural language interface took four years to implement. Changes had to be made to SOPHIE I to accommodate the interface. The interface could correctly answer 90% of queries taking less than 150 ms each, and allowed users to use conversational language input with context sensitive commands.[28]. One suggestion is for the interface to guide the users in what they should say.

2.3.6 Recent uses of Natural Language.

More recently natural language interfaces have been used to help manage patients with hypertension. This system requires patients to dial a toll free number and a recorded voice is used to ask the patient questions and the user's language is then interpreted.[29] This system is being used in three Italian hospitals.[29]

Expert systems have recently been used in natural language processing [12]

2.3.7 Critical comparison.

When expert systems were first introduced they used a question-answer interface. However this interface could not be scaled up to be used by a wider range of users. Question-answer interfaces were based on the way users would interact with human experts and came with pre-

existing interaction problems. However the question-answer model is understood by novice users.

Data interfaces were thought to be more usable than question-answer interfaces and it was thought that advanced users would want to use this interface. However this required a large amount of typing and memorizing terms.

A selection based interface is significantly faster than data interfaces. Both data and selection interfaces are made for advanced users.

Natural language allowed users to interface with the expert system in their normal language, which had major benefits for novice users. Users tended to assume that the interface had full conversational capabilities and communicated with it like they would communicate with another human. It is important for a natural language interface to understand a wide range of inputs as well as conversational constructs such as ellipsis. The natural language interface can reduce unexpected inputs by guiding users in what they should say.

Graphical interfaces have great benefits for all types of users and allow the designer to broaden the scope of the problems that can be presented to the expert system. Graphical interfaces have shown significantly faster interactions between the users and the expert systems. These interfaces also allow users to interact directly with the underlying expert system.

2.4 Android Medical Apps

There are many available android applications for various needs. On the 8 October 2012 there were 5608 medical apps and 12388 health and fitness apps available in the Android Play Store[30] In March 2010 there were only 586 medical apps available [4].

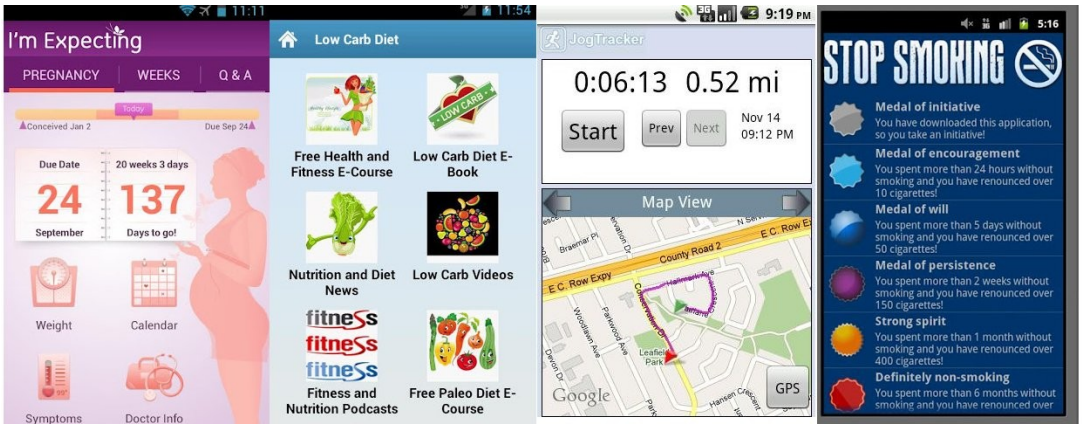


Fig. 5: Android Health and Fitness Apps

Source: Android Play Store

2.5 Google Speech Recognition

Google's Speech recognition service records what the user has to say on the phone and offloads it's processing to the cloud, it then reflects accurately what the user said. It can recognize any length of speech and rare words even with an accent [31]. The service uses

artificial intelligence to recognize speech. Google stores millions of speech utterances from the users in order to improve their service.

2.6 Pocket-Sphinx

Pocket-Sphinx was originally designed for use on devices with low CPU speed, lack of support for floating point operations, no virtual memory, limited RAM memory, limited storage capacity and low bandwidth [32]. Pocket-Sphinx could run on hardware with a 206Mhz processor, 64MB RAM and 16MB of flash memory[32] and possibly less of any of the above specifications.

To train Pocket-Sphinx with the English acoustic model can take 211 hours of data[33]. Pocket-Sphinx also has words which are likely to be inaccurately recognized[33]. It can have a word error rate of 55%[33], [34] and recognition can degrade when on a mobile device[32] . In general results can vary from 55% to 99% with the highest recognition rate coming from the people who trained the speech recognition software[34].

Sphinx works by taking the audio in waveform and splitting it up into utterances separated by silences and then trying to recognize what each utterance is. It then takes all possible words and tries to do a best match with the audio. Sphinx divides up the speech into frames, usually about 10ms and represents the frame numerically. A mathematical model of speech is also used and finally the model is compared to the frames to get the best fit[35]

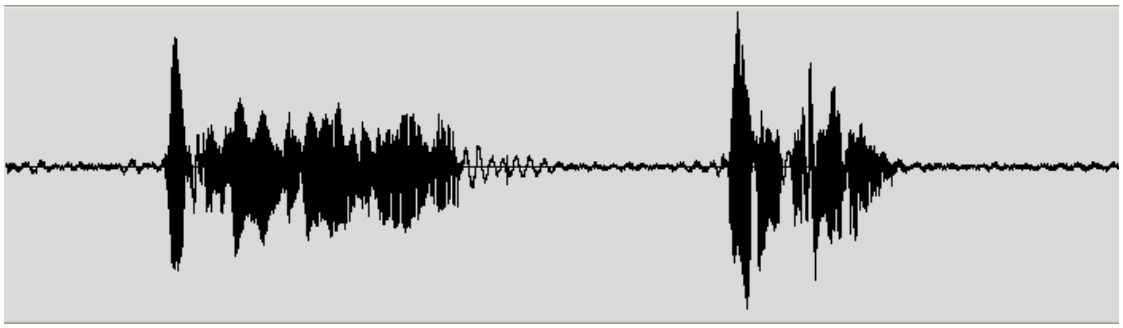


Fig. 6: A waveform
Source: http://cmusphinx.sourceforge.net/wiki/tutorialconcepts

2.7 Java Expert System Shell

2.7.1 Jess

Jess stands for the Java Expert System Shell. Jess is a rule based system. Rule based systems are commonly used commercially. They can function efficiently where normal algorithms struggle and they can even handle incomplete information [13]. Rule based systems excel over normal procedural algorithms where the 'program' has many branches, context-sensitive decisions, many loops and where there are so many scenarios that creating a procedural program is nearly impossible. Jess uses declarative programming, which describes what must be done but the runtime engine can use the declarations to solve problems. As the

programmer does not control the flow of execution, declarative programs can handle partial or disordered input better [13]. In other words normal programming involves the 'how' and not the 'what', but declarative programming describes the 'what' and not necessarily the 'how'.

Jess uses the Rete algorithm, which speeds up pattern matching by eliminating redundant rule checks.[36] The Rete algorithm has been used in many expert systems and reduces the complexity from exponential to linear. This algorithm is implemented by building a network of nodes based on the left hand side of the rules. The network is then used to process new facts. [37]. The Rete algorithm was originally designed for production system interpreters, containing more than a thousand patterns and objects, like OPS5[36].

There are 7 potential methods of creating an expert system with Jess and Java, namely:

- Pure Jess language scripts. No Java code at all.
- Mostly Java code, which loads Jess language scripts at runtime.
- All Java code, which manipulates Jess entirely through its Java API.
- Pure Jess language scripts, but the scripts access Java APIs.
- Half Jess language scripts, with a substantial amount of Java code providing custom commands and APIs; main() provided by you or an application server.
- Mostly Jess language scripts, but some custom Java code in the form of new Jess commands written in Java.
- Half Jess language scripts, with a substantial amount of Java code providing custom commands and APIs; main() provided by Jess.[38]

2.7.2 Jess Vs Clips

Jess was originally inspired by CLIPS, but it does not provide all the functionality of CLIPS and it also has some functionality CLIPS does not have. CLIPS is free to download and use. It comes with a easy to understand manual, an interface guide, an architecture manual and an advanced programming guide. The Clips package also comes with a convenient editor to assist easy creation of expert systems. Nothing like this comes with Jess. Clips also has the CLIPS Object Orientated Language(COOL), with all benefits of an OO language. Variants of CLIPS can also support fuzzy logic. However CLIPS is hard to debug. [39]

Jess is written Java, and one can obtain an academic licence to use for a period of time. The Jess manual explicitly assumes that users have had some previous experience with CLIPS or Lisp.[40] Jess provides a subset of CLIPS' functionality. For example CLIPS has seven strategies for rule execution while Jess only has two. Jess lacks COOL but COOL may be unnecessary as Jess can make use of Java Objects via Java.bean. This can make CLIPS code hard to port to Jess.[39]. Jess has many advantages over CLIPS, because of its access to the Java API, such as networking, threading, database access and so on. Jess also has it's own awt component, Jess.awt [13]. Jess, of course, can easily be embedded into Java programs and there is API documentation to support this [41].

2.7.3 Jess Interfaces

Jess can be used in a terminal, GUI applications, servlets and applets [38]. Jess also has the ability to be integrated into a Java program, thus giving it a large number of possible interfaces. It is possible to write a Jess application entirely in Java. There is no prescribed way to make an interface and this is left up to the designers. Figures 7 and 8 show examples of such interfaces.

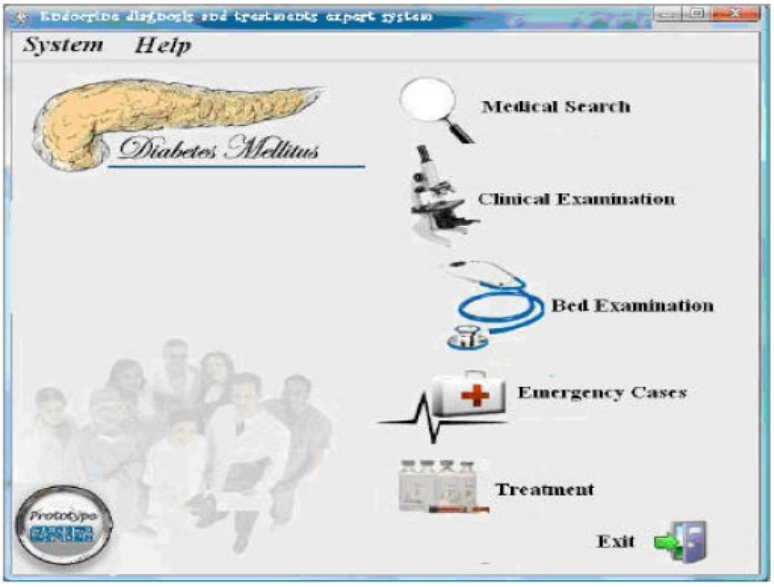


Fig. 7: Endocrine
Source: http://scialert.net/fulltext/?doi=jai.2010.239.251&org=11

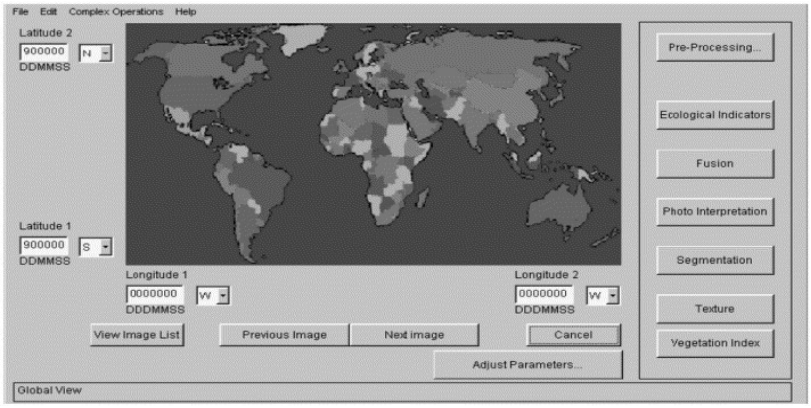


Fig. 8: Intelligent image analysis for environment monitoring
Source: http://www.sciencedirect.com/science/article/pii/S1093019101000831

2.8 Jess on Android

Searching online to see if any tutorials existed for Jess on android, gave very negative results. “There is no way currently to implement Jess in Android. There is no fix”.[42], [43] . Jess is said to be too reliant on the Java.bean and Java.awt libraries, which are not implemented on Android [44], [45]. One suggestion is to run the application on a server, avoiding running Jess on the actual device [46] this may cause the app work slower as every question needs to go via

the internet. However on searching for options, an implementation of Java.bean called openbeans[47] solved the problems with Java.bean reliance, but reworking the Java.awt dependencies was necessary. Jess version 6 was used over Jess version 7 as it had less dependencies on Java.awt.

3 Design and Implementation

3.1 General Design

This chapter will explore the design of the interface, middle-end and Jess back-end on Android. An iterative design method was used involving users early in the development as users play an integral role in the design of any interface.

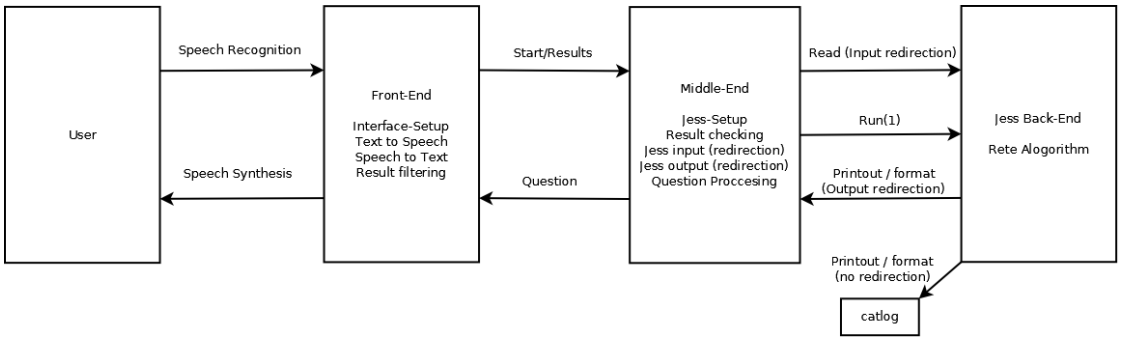


Fig. 9: System overview
Source: Own Creation

In figure 9 one can see the overall design of the system. Even though a graphical interface would be easier to make and possibly simpler for users to interact with, the target users were assumed not to be able to use a normal graphical interface. Thus the front-end is simply a blank screen, with a speech icon appearing when the user is indicated to start speaking. However the modules were designed with repeatability in mind, so the front-end can easily be replaced with a graphical one or perhaps a form of terminal.

The iterative design approach involved getting feedback from users and improving the interface based on the feedback. Figure 10 shows how this was done.

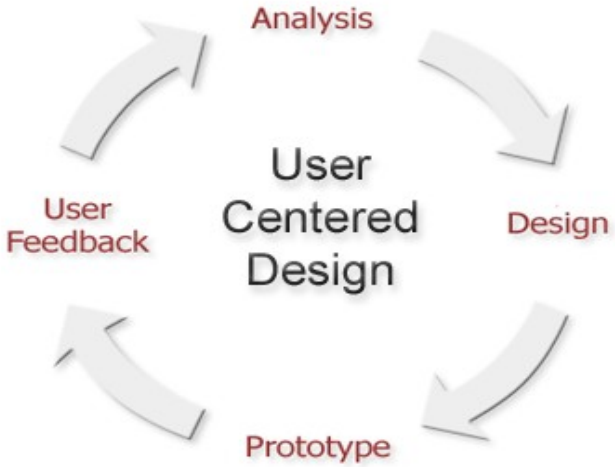


Fig. 10: User centred design
Source: http://kevinbury.com/images/process_circle.gif

The Back-end was also designed to run different rule files, and not restricted to diabetes diagnosis.

3.2 Jess On Android, The back-end

To get Jess to run on android required the source code, as parts needed to be edited. Most significantly:

java.beans for com.googlecode.openbeans

java.beans.beancontext for com.googlecode.openbeans.beancontext

java.beans.beaninfo for com.googlecode.openbeans.BeanInfo,

java.beans.IntrospectionException for com.googlecode.openbeans.IntrospectionException

java.beans.Introspector for com.googlecode.openbeans.Introspector

java.beans. PropertyDescriptor For com.googlecode.openbeans.PropertyDescriptor

All of jess.awt needed to be removed except for jess.awt.TextReader, which is used to “fake” terminal-like input from a non terminal-like app. All functionality that is used for java.applet needed to be removed.

Input and output redirection was necessary in order to receive text output from Jess and give text input to Jess. This will be discussed in the middle-end.

As it was believed that Jess will not run on Android, no tutorials were available and most coding was by trial and error. The result is a version of Jess that functions fully on Android. It is capable of preforming the same instructions that a normal Jess terminal could. On a Samsung's Galaxy S one, Galaxy S two and Galaxy Note Jess does not hang or crash, for any of various different tasks given.

3.3 Middle-end

The middle end firstly creates a Jess Rete engine. The middle end uses input redirection via a jess.awt.TextReader, the TextReader -named 'in'- is then added to the Jess as an input router. A TextReader has the method “appendText” which is used to give input to Jess, thus replacing terminal input. Subsequently in any Jess read statement, “(read in)” must be used for an Android app over “(read)” or “(read t)”. For output redirection a StringWriter and PrintWriter combination -called 'out'- was used. The PrintWriter was then added to Jess as a output router. Subsequently in any printout or format statement “(printout out)” must be used over “(printout t)” and similarly with format. It was also found that System.out prints to the logcat, so if “(printout t)” is used the statement can be found in the logcat. Various applications exist to monitor, filer and record the logcat, Catlog was used on the phone and Netbeans provides this functionality on a computer.

Next the Middle-end batches the .clp file(a file containing rules), and resets the Rete engine so it is ready for use. At this point Jess is set up for use.

During the programs execution the middle-end checks to see if there was one usable input from the speech recognition (eg a user could say “yes and no”), if the input is not acceptable then the middle-end returns the the last question, otherwise it conveys the input to Jess via 'in' and

the runs the expert system to receive the next question. The designer of the expert system needs to communicate what response the expert system is expecting from the user. This is done by pre pending the question with the question type. For example the question “Who moves first, Computer or Human? " expects the response to be either “computer” or “human”, thus it is prepended with “HC::” which indicates to the middle-end that the expected responses are 'computer' and 'human'. The middle-end then makes a Question object that includes the question and the responses as well as some other information to limit the response if the response is a number and this is then sent to the front-end. Adding a new type of question involves simply adding an if statement to the middle-end and front end, and a list of acceptable words to the front end.

3.4 Front-end design

The front end consists of two main parts: Speech recognition and a text-to-speech engine. The front end could easily be replaced with a GUI, but a speech interface was specifically chosen as the app is meant to be used by a user with physical or literacy challenges that make conventional interfaces difficult or impossible. The front end also makes sure that the screen does not go off when the app is functioning, it also prevents the app from being destroyed and recreated when the phone is tilted. The front-end also adds the .clp file from the apps resources to the Android file system so that the Middle-end can batch the file. The front end also checks if there is a text to speech engine installed on the device. Unfortunately some of the emulators do not posses a text-to-speech engine or speech recognition, some devices also lack Text-to-speech libraries[48]. Developers will need an actual device for there testing and deployment.

When the app is running the front end receives a question from the middle-end and processes the required answers of the question object and then uses text-to-speech to read the question to the user. However as the text-to-speech engine does not communicate with the speech recognition, if one was to let both of them run the speech recognition would simply record what the text-to-speech engine synthesises and not give the user a chance to speak. Thus the program needs to enter a spin lock while the text-to-speech engine is synthesising speech. It then attempts to recognize what the user's response is, filter the results according to the expected response from the question object and send the results to the middle end for processing.

3.4.1 Android Text-to-Speech engine

Firstly the Text-to-Speech engine is set to speak in English, the language need to be specified as a word like “Paris” is pronounced differently in French than in English [48]. The pitch and speech rate can also be set but these values translate to different pitches and speeds on different devices, for example setting the speed to 0.85 is fine for a Galaxy S, but painfully slow for a Galaxy S two. As described above the app needs to pause while Speech synthesis is taking place otherwise the speech recognition will record the speech synthesised, a spin lock is used for this reason, the TextToSpeech object provides a boolean function isSpeaking(), and while this returns true the app does not progress or, if possible, does some background tasks.

Unfortunately the Text-to-speech engine has a problem synthesising some words, in particular “Diabetes”, this can be solved by breaking the word up into syllables and feeding the synthesiser the syllables, for example “die a bee teas”. It can also read the names of punctuation instead of following the rules for that punctuation, for example “,” might be read as “comma” and “|” might be read as “vertical bar”.

Android also has an option where the voice can be changes from the default voice, allowing a device to synthesise with any accent. One can find many different from American English Romanian. The text-to-speech API is well developed and easy to use.

The Text-to-speech synthesis relies on a dedicated service that runs across all apps that use synthesis[48]] so when the app terminates it should call the shutdown() function from the TextToSpeech API.

3.4.2 Google voice, Speech recognition

In order to run the speech recognition from google one needs to make an “intent” object with the intent RecognizerIntent.ACTION_RECOGNIZE_SPEECH. The language model that was picked is the LANGUAGE_MODEL_FREE_FORM, This model is designed for recognition general speech. The alternative model is LANGUAGE_MODEL_WEB_SEARCH, which is optimized for searching the web. startActivityForResult() is then called with the intent, this then places a message on the message queue, and when the program pauses Android will go though the messages and when it reaches that message it will start a new activity and return the result of the activity. Figure 10 shows what the new activity window looks like on Android 2.3.

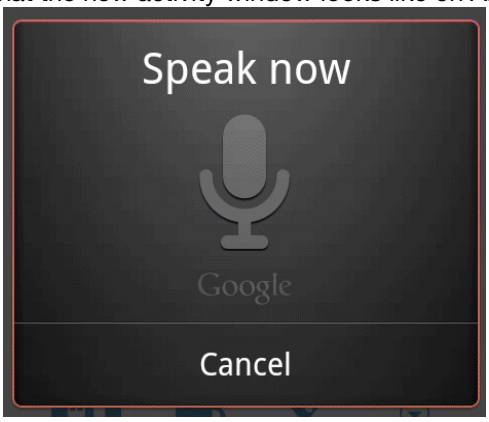


Fig. 11: Speech recognition
Source: Phandroid

When the speech recognition activity is complete, it will then proceed back to the ADVISOR app's onActivityResult() method. As ADVISOR is not a speech to text application, one does not need to determine the most accurate text to fit the speech, rather all possibilities are mined for the expected input from the question object. For example if the user said the expected word “mice”, the most likely result from the engine may be “nice”, and the second most likely “mice”, thus the correct response is found.

As the speech recognition happens in a different activity, and returns to a different part of the program, a simple loop cannot be used. Instead at the end of the result filtering the apps needs to return to the code immediately after the setup. Thus the app will loop infinitely unless the middle-end sends the front-end an “exit” question - one with the prefix rr:: - which will tell the app to preceded with exiting.

3.4.3 PocketSphinx, Speech recognition

At first glance having PocketShpinx run on android seems like a simple task from simple tutorial [49], however there were many complications. The first instructions were to run autogen.sh followed by ./configure, make and make install. The configure script checks if the necessary libraries used in the building of pocket-sphinx are installed, and on a developers version of Ubuntu 11.10 to many libraries were missing to be practical. A full install of Cygwin (9.9 gigabytes) was then used to build pocket-sphinx.

The Python script setup.py contained an error and the following change needed to be made: libraries=['pocketsphinx', 'sphinxbase'] to libraries=['pocketsphinx', 'sphinxbase', 'iconv']. It is also nessisary to use a snapshot version of pocket-sphinx as the file necessary for swig are not present in the other versions. The swig files also do not automatically generate with the makefile and invoking another makefile in the swig folder is necessary to generate the necessary files:

- config.java
- Hypothesis.java
- pocketsphinx.java
- pocketsphinxJNI.java
- SegmentIterator.java

The next step was to preform an NDK build, which again would result in many errors. Notably:

- The NDK cannot understand a space in a file's path
- The NDK needed the “-B” flag
- #import <stdlib.h> ; in android.mk
- Had to change the android.mk, line 163 from: LOCAL_STATIC_LIBRARIES := sphinxutil sphinxfe sphinxfeat sphinxlm pocketsphinx
  to: LOCAL_STATIC_LIBRARIES := pocketsphinx sphinxlm sphinxfeat sphinxfe sphinxuti

- Had to mark some sources for arm compilation in android.mk, in the sphinxfeat module the LOCAL_SRC_FILES must equal “agc.c \ cmn.c.arm \ cmn_prior.c.arm \ feat.c \ lda.c.arm \”
- Some static and shared libraries were not always generated so the ./configure command was used with the following flags: --cache-file=/config.cache --enable-static --enable-shared --enable-extra-encodings. The iconv library also needed recompilation under these flags.
- The was a library linking problem, “LDFLAGS=-no-undefined” had to be exported in the cygwin environment, and even hard-coded into some makefiles.
- A different version of the file amacod.c was needed, available from: http://www.speech.cs.cmu.edu/sphinx/doc/doxygen/pocketsphinx/acmod_8c-source.html
- To finally solve the library linking problem the following changes had to be made to the swig makefile:

  “CFLAGS := -g -Wall -DPIC -fPIC -I/usr/local/include

  CPPFLAGS := -I/usr/lib/jvm/default-java/include -L/usr/local/lib `pkg-config --cflags sphinxbase pocketsphinx`

  all: libpocketsphinx_jni.so

  libpocketsphinx_jni.so: pocketsphinx_wrap.o

  $(CC) -shared -o $@ pocketsphinx_wrap.o `pkg-config --libs sphinxbase pocketsphinx` -L/usr/local/lib -liconv

  pocketsphinx_wrap.c: pocketsphinx.i

  mkdir -p edu/cmu/pocketsphinx

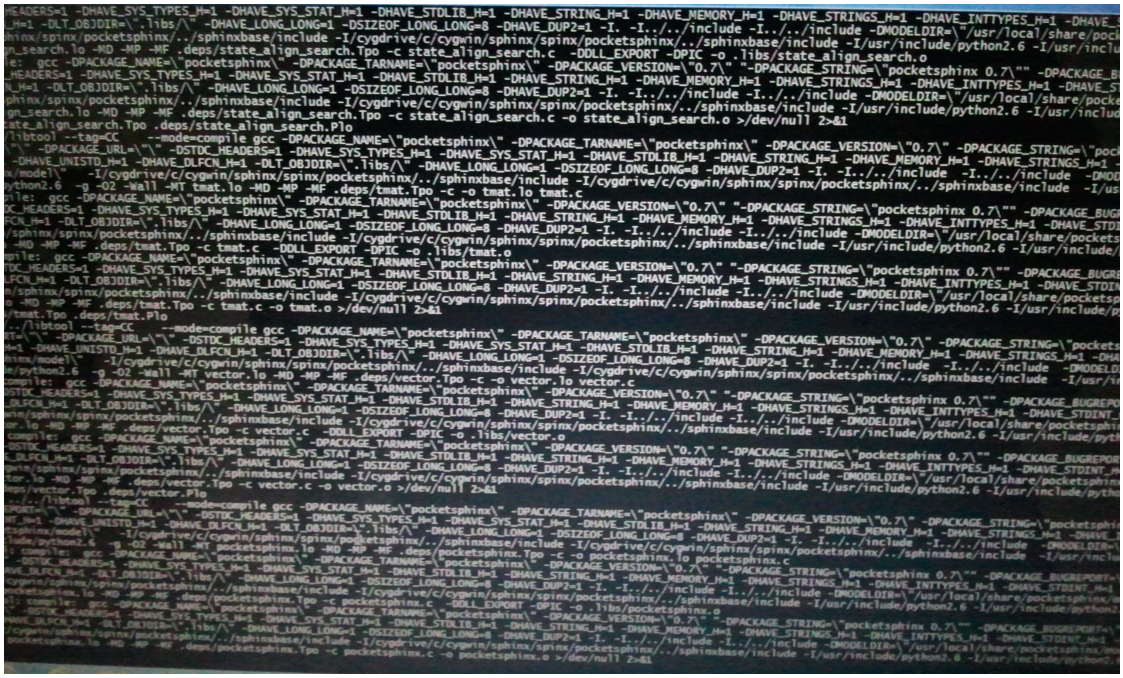
  swig -I../include -I../../sphinxbase/include \

  -java -package myappname \

  -I/usr/local/include \

  -outdir edu/cmu/pocketsphinx pocketsphinx.i”
- Smaller errors were incurred to do with folder and file location.

Fig. 12: Cygwin
Source: Own creation/nightmares



Using Netbeans, the Native code was then wrapped into an Android library for use in an Android app. This library could be used by any application, and is not limited for use in ADVISOR. Te following three lines of code are all that is necessary to setup and app for pocket-Sphinx use:

```
FileManager fm=new FileManager(this);
lib l=new lib(fm);
System.loadLibrary("pocketsphinx_jni");
```

The FileManager is simply an object that assists with with file creation. lib is the Android pocket-Sphinx library, libpocketsphinx_jni.so is stored in the libraries resources. The library needs to be placed under /data/data/myappname/ on the phone, lib then checks if the library exists, writing it to the location if it does not. At this point the library is in position and needs to be loaded into the app. This segment of code needs to be the first code run in the onCreate() method in an app. An app needs a few more files to work, pocket-Sphinx uses these files in it's recognition process:

"/sdcard/Android/data/myappname/pocketsphinx.log"
"/sdcard/Android/data/myappname/hmm/en_US/hub4wsj_sc_8k/feat.params”
"/sdcard/Android/data/myappname/hmm/en_US/hub4wsj_sc_8k/mdef”
"/sdcard/Android/data/myappname/hmm/en_US/hub4wsj_sc_8k/means”
"/sdcard/Android/data/myappname/hmm/en_US/hub4wsj_sc_8k/noisedict"
"/sdcard/Android/data/myappname/hmm/en_US/hub4wsj_sc_8k/sendump"
"/sdcard/Android/data/myappname/hmm/en_US/hub4wsj_sc_8k/transition_matrices"
"/sdcard/Android/data/myappname/hmm/en_US/hub4wsj_sc_8k/variances"
"/sdcard/Android/data/myappname/lm/en_US/hub4.5000.dic”
"/sdcard/Android/data/myappname/lm/en_US/hub4.5000"

The pocketsphinxJNI class is a wrapper for native library functions. This class was automatically generated by SWIG, methods are declared as: public final static native {return type} methodName (Parameters);

Pocket-Sphinx then needs to be configured to use those files, The speech recognizer is then run in a Second thread. A third Thread is then used for audio recording, blocks of audio are then recorded and placed on a queue. While the audio recording is taking place the recognizer thread attempts to get partial results from the some of the audio blocks. The number of partial results depends on the thread scheduler. When the user indicates that the user is finished speaking, the Third Thread is then joined and the remaining audio blocks are then removed from the queue and the final speech recognition takes place.

3.5 Early user involvement

During the creation of the application users needed to be involved. This is necessary to design an application for users and not for programmers. This project involved the creation of the interface and not the expert systems questions. Computer science students and even lecturers were asked to use the system and there interactions were noted for use of improving the app. Weather the speech recognition was working was also crucial thus users with different accents were asked to evaluate the system.

4 Experimentation

4.1 Overview

Usability testing was carried out to see if users could use the system without any training, to see if the speech recognition was able to recognize the accents of users who do not speak English as there first language, and to see if the system could run in acceptable running time. All users were given consent form. All users had the purpose of the experiment clearly explained to them and had a chance to ask questions. All users did the same test, and filled out an Interface evaluation after the test for efficiency, effectiveness and satisfaction of the app.

As the interface was made for illiterate or blind people, the actual layout of the interface is irrelevant, and ADVISOR's “blank screen” interface would probably fail when compared to graphical interfaces.

4.2 Goal

The aim is to establish weather the system can be used by an untrained user, that is can the system give the user easy access to knowledge that the user would not have had before. Firstly two different expert systems were tested with the Google voice for interface usability, this also included recognition ability, system speed. With the results one can ascertain weather the system can be used by an illiterate person. Secondly the Pocket-Sphinx's recognition was tested, with the results once can see if Pocket-Sphinx can recognize different accents.

4.3 Participants

Participants in the experiment were all UCT students whom did not speak English as there home language. Since no necessary background knowledge or skills were needed, there was no selection process. All data collected was anatomised. Since it was necessary to test on an actual device to be similar to a real world scenario, A Samsung S Plus was used. The testing was preformed at UCT Obz Square residence, were the users stayed.

Participants were all informed of the purpose of the experiment and told that the interface and speech recognition is being evaluated and not the users themselves. They were also told that the questions that the expert system asks are not being evaluated. This is because the expert systems question design was not part of this project.

It needs to be noted that ethics clearance was obtained from UCT before any recorded experiment took place. Users were reimbursed a small amount for there time.

4.4 Experiment Design

During the experiment, users were asked to preform three tasks, two involving Google's speech recognition, and the last one Pocket-Sphinx. Since the interfaces to the Google speech recognition were identical, there was no need to mix-up the order of the experiments. Although

the purpose of the experiment were explained, Instructions were kept to a minimum. This is to simulate a real life scenario, were the potential user may not be able to understand. 'Having a conversation' also needs little explanation.

At the end of the experiment users were asked to fill out a standard interface evaluation questionnaire [50]. It was also observed if the participants had any trouble understand what to do.

4.4.1 Task one

The first task used the Advisor interface with the animal expert system. The interface was instructed to batch animal.clp, which contained the rules to the animal expert system, (see appendix I). Participants were simply told to think of an animal and the phone will play a game of 'twenty questions' to guess the animal. They were also instructed that all interaction will take place via speech and that they must wait for the speech icon to appear on the screen before answering any question. The animal expert system only uses 'yes' and 'no' input in order to come to a conclusion as to what animal the participant thought of. Words that the speech recognition could not understand were noted.

4.4.2 Task two

Task two was similar to the first but used the sticks game expert system. The interface was instructed to batch sticks.clp (see appendix II). The sticks game was chosen to be second as instructions were needed on how to play the game. The sticks game works with a pile of virtual sticks, the computer and human take turns picking a number of sticks out the pile. The player who takes the last stick looses. The sticks game expert system uses the numbers one, two, three eight and nine as input as well as the words 'computer' and 'human'. Again the words that the speech recognition could not understand were noted. It was assumed that the users would have learnt the interface at the start of the sticks game.

4.4.3 Task Three

Task three involved testing the speech recognition of pocket-sphinx, participants were asked to say the numbers one to ten, followed by the words 'yes', 'no', 'computer' and 'human'. It was then observed which words pocket-sphinx could recognize.

4.5 The Questionnaire

The Questionnaire used is called QUIZ 5.0 (Questionnaire for User Interface Satisfaction) [50] It was originally tested with that was liked, and was not liked. It The overall reaction ratings yielded significantly higher ratings for liked software over the disliked software.

There are 27 questions in the questionnaire all scaled from 0 to 9, with a not applicable box. Users have a chance to comment on every question, as well as on the positive and negative aspects of the interface.

5 Results and discussion

This chapter discusses the results from the experiments outlined in the above chapter. The quantitative result is speech recognition of both Pocket-Sphinx and Google speech recognition. The qualitative results are the user feed back and the researcher observations.

5.1 Early results

In early testing of the Advisor interface, it was noticed that users would speak before the program started recording speech. Unfortunately there seemed no way of correcting this on a software side as if the speech recognition was earlier in the program it would record the speech synthesis as described in chapter 3. To attempt to resolve this problem the statement “please wait for the speech icon” was added to the startup speech of the app. This seemed to reduce the occurrences of speaking before the appropriate time.

Google speech recognition, with mining all returned possibilities, seemed to recognize speech efficiency. Two problems were noticed, firstly the phone could not send the recorded speech though the UCT internet proxies. The solution was to use 3G internet or other wifi, this solved this problem but it caused an occasional speed problem. When using 3G, if the 3G connection was slow the app would appear to be slow as it was waiting for the file to send. The app may need to run on dedicated wifi internet. The second problem, which only happened rarely, was if the Google could not recognize any word in the speech, instead of sending back to the app that blank results, it sends a “speak again” (see figure 13) screen to the user.

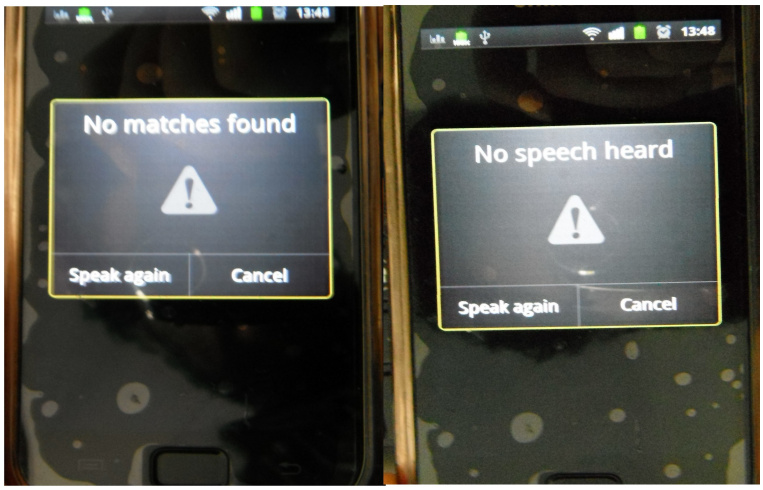


Fig. 13: Speak again
Source: Own creation

Pocket-Sphinx seemed to only be able to pick up a few words, and seemed not to be able to recognize many words. Pocket-Sphinx's recognition was even less when dealing with a non-English speakers accent.

5.2 Recognition: Google

Google speech recognition preformed very well when dealing with “yes”, ”no”, “human” and “computer”. It could not pick up the numbers “two” or “eight”. Word Accuracy is a “widely accepted evaluation measure for word recognizers” [51]. Word Accuracy is calculated as

$WA=100x(1-(W_s+W_i+W_d)/W)\%$

Where W_s, $W_i$and W_d are the words that needed substitution, insertion and deletion respectively. W is the total number of words in the reference. Table 3 shows the WAs of each word.

	WA
Yes	90%
No	90%
One	100%
Two	0%
Three	75%
Eight	0%
Nine	50%
Computer	100%
Human	90%
Average	$66.\overline{1}\%$

Tab. 3: RESULT TABLE FOR GOOGLE SPEECH RECOGNITION
Source: own creation

For all words spoken the WA is approximately 69.35%. The discrepancies between the average of the individual WA's and the WA across all words is because not all words were said an equal number of times. If one was to exclude the deletions from “two” and “eight” the WA for the remaining words is approximately 82.69%.

These WA's are for people who do not speak English as there home language, in the early testing of the interface people who spoke English first language could archive a WA of 100%. If one were to deploy a speech based interface similar to this project, it would be important to find words that can be recognized easily. That changes the WA from 69.35 to 82.69%, a increase of 13.34%.

5.3 Recognition: Pocket-Sphinx

Pocket-Sphinx preformed terribly compared to Google speech recognition. Out of 65 words Pocket-Sphinx was only able to recognize 11, as well as inserting incorrect words or not picking up some words. Thus the WA is approximately 17%. Pocket-Sphinx is known to have bad recognition for people with non-English accents[52]. Figure 14 shows some of the results.

With a WA of 17%, using Pocket-Sphinx is not practical for real world use. One possible reason is that the standard hub4.5000.dic for United States English speakers was used. It may be possible to obtain a higher accuracy when Pocket-Sphinx is trained by users from it's target audience[34], or use a limited vocabulary.

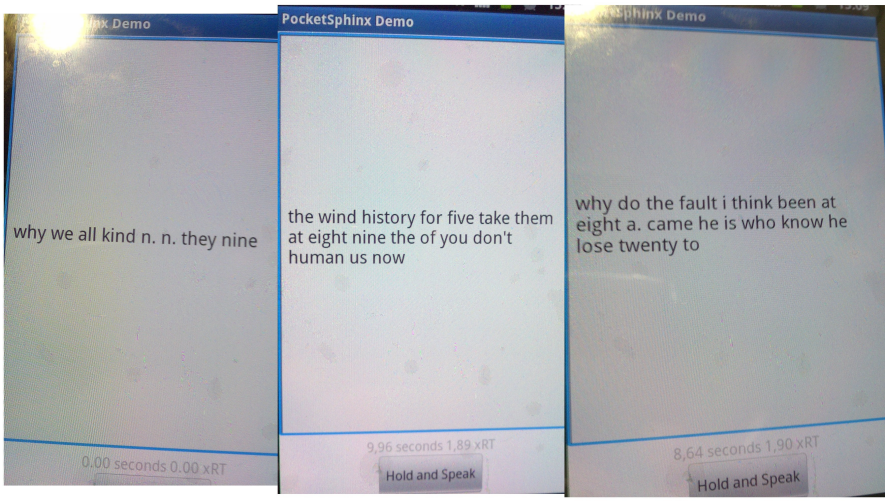


Fig. 14: Pocket-Sphinx
Source: Own creation

5.4 Satisfaction: User feedback

Almost all participants indicated that they were satisfied with using the system. Only one participant thought the interface was a bit hard to learn.

Participants commented mainly about the recognition of the numbers, “the voice recognition of the numbers was not accurate at all”. Another user thought “the google voice recognition was clear ”. One person rated the reliability 1/9 as well as “the voice recognition is in one language only (English) ”

All participants agreed on is that they understood what to do, and they were not confused as to what to do. They all found the system easy, simple and designed for all levels of users.

Two participants admitted that they had tried the speech recognition on there phones (google) and given up on it. One of them even demonstrated it's poor results.

It was also noticed that if a participant had to give an input that the system had trouble recognizing, they would change their input on the third or fourth try.

5.5 Discussion

The results from Pocket-Sphinx show that it is impractical as it is to use for a real world application. Google voice recognition, on the other hand, is suitable for use but with chosen words as Google has trouble recognizing certain words. Two participants admitted that they had tried to use the Google voice recognition for normal dictation and one demonstrated the poor result. One possible reason for the implemented systems 'higher' results could be because all possibilities are returned from Google and then mined for useful words, instead of just the most likely hypothesis. Another reason could be that some words are easier to recognize even with an accent.

Participants felt satisfied using the system, and were only troubled by the systems inability to recognize certain numbers. Before any speech recognition soft can practically be used, it needs to be established which words are easier to recognize. A user should not have to attempt to give the same answer more than twice as this may result in the user changing the answer and thus providing the back-end with false information. It also needs to be noted that the questions/instructions must be simple and short. The expert system works fast and efficiently, however bad connection to the internet could slow down (wall clock time) the application making the app appear slow.

On the positive side for Google speech recognition, it was capable of recognizing some words with a very heavy accent (for example one user's “three”). However to fully exploit speech recognition technology language models for non-English African languages need to be established. Doing such could drastically increase recognition, user satisfaction as well as usability.

6 Conclusion

This research concludes with the successfully implementation of a speech based interface, a successfully 'port' of the Java Expert System Shell to Android and a Pocket-Sphinx implementation. All applications were designed for an Android phone using the Android API and SDK, as well as the JNI and NDK. The Openbeans library was used as a replacement for Java.beans, thus Jess version 6 was possible to run on an Android phone. Most Jess.awt components had to be removed as well as the applet functionality. Lots of problems were encountered in the implementation of Pocket-Sphinx, and tutorials were not accurate.

The app, ADVISOR, gives users who are not computer literate and/or are functionally illiterate the access to knowledge-based systems that they previously would not had access to. User testing was done to ascertain weather the interface was user friendly and simple to learn. Two different speech-to-text packages were implemented and tested. Participants in the testing were asked to use and evaluate the interface.

The results leave one to conclude that the interface is in fact satisfactory, when the speech recognition works correctly. Participants indicated that they understood what they needed to do and were not confused by the interface. It seemed users trusted the interface.

It was also noticed that Google voice recognition was far better at recognizing the speech of participants with non-English accents. Google voice recognition struggled with the words “two” and “eight” but managed fine with other words. It had a WA of 69.35% for all words and a WA of 82.69% when ignoring words that are difficult to recognize. It managed to recognize some difficult words, which may not have been expected. More work needs to be done to create a usable system, but the interface is conceivable.

Jess is fully functional on an Android device and runs without any lag or delays. Jess on Android functions just the same as Jess on a terminal, although all functions have not been tested. Jess on Android can be used for a variety of functions and not only ADVISOR.

Pocket-Sphinx was only able to archive a WA of 17%, this could be due to the fact that a model trained for an American accent was used. It is not practically usable at its current state. To get enough training data to build an accurate model is time consuming. An android library was created so that Pocket-Sphinx can be used by any application without any trouble.

7 Future work

One possibility is to implement pocket-sphinx so that it can recognize accents, that is the user must say some phrase, and from there recognize what accent the user has and insert the appropriate trained model for that user, thus increasing the recognition ability and making Sphinx usable.

Due to time constraints, it was not possible to train Pocket-Sphinx with the target audiences accent and a limited vocabulary. The risk of limiting a vocabulary is that if the user does not respond with one of the words in the vocabulary, it will try to give a 'best fit' word from the vocabulary, which could give garbage output.

The words that Google voice recognition can recognize well can be recorded to further increase the reliability of the application. Perhaps another cloud based recognition tool can be investigated such as Bing.

One of the most relevant things that could be done is to create a language model for South-Africa languages (other than English) for use in speech recognition. This could allow many users who are not first language English a more natural and comfortable experience. However this is a time consuming job that cannot be automated.

As Jess now works on Android, more applications using expert systems can be made or 'refitted' for android. The modular design of the program makes replacing the speech interface with a GUI easy.

8 Appendices

Appendix I: Edited Animal.clp

```
;;;======================================================
    ;;;   Animal Identification Expert System
    ;;;
    ;;;     A simple expert system which attempts to identify
    ;;;     an animal based on its characteristics.
    ;;;     The knowledge base in this example is a
    ;;;     collection of facts which represent backward
    ;;;     chaining rules. CLIPS forward chaining rules are
    ;;;     then used to simulate a backward chaining inference
    ;;;     engine.
    ;;;
    ;;;     CLIPS Version 6.0 Example
    ;;;
    ;;;     To execute, merely load, reset, and run.
    ;;;     Answer questions yes or no.
    ;;;======================================================

    ;;;(printout t "YN::please wait for the speech icon" )
    (printout t "begin of animal.clp " )
    ;;;***************************
    ;;;* DEFTEMPLATE DEFINITIONS *
    ;;;***************************

    (deftemplate rule
      (multislot if)
      (multislot then))

    ;;;***************************
    ;;;* INFERENCE ENGINE RULES *
    ;;;***************************

    (defrule propagate-goal ""
      (goal is ?goal)
      (rule (if ?variable $?)
            (then ?goal ? ?value))
      =>
      (assert (goal is ?variable)))
```

```
(defrule goal-satified ""
   (declare (salience 30))
   ?f <- (goal is ?goal)
   (variable ?goal ?value)
   (answer ? ?text ?goal)
   =>
   (retract ?f)
   (format t "%s%s%n" ?text ?value)
   (format out "%s%s%n" ?text ?value))

(defrule remove-rule-no-match ""
   (declare (salience 20))
   (variable ?variable ?value)
   ?f <- (rule (if ?variable ? ~?value $?))
   =>
   (retract ?f))

(defrule modify-rule-match ""
   (declare (salience 20))
   (variable ?variable ?value)
   ?f <- (rule (if ?variable ? ?value and $?rest))
   =>
   (modify ?f (if ?rest)))

(defrule rule-satisfied ""
   (declare (salience 20))
   (variable ?variable ?value)
   ?f <- (rule (if ?variable ? ?value)
               (then ?goal ? ?goal-value))
   =>
   (retract ?f)
   (assert (variable ?goal ?goal-value)))

(defrule ask-question-no-legalvalues ""
   (declare (salience 10))
   (not (legalanswers $?))
   ?f1 <- (goal is ?variable)
   ?f2 <- (question ?variable ? ?text)
   =>
   (retract ?f1 ?f2)
   (format t "%s " ?text)
```

```
   (format out "%s " ?text)
   (assert (variable ?variable (read in))))

(defrule ask-question-legalvalues ""
   (declare (salience 10))
   (legalanswers ? $?answers)
   ?f1 <- (goal is ?variable)
   ?f2 <- (question ?variable ? ?text)
   =>
   (retract ?f1)
   ;;;(printout t "YN:: " ?answers " ")
   (format t "YN:: yes or no %s " ?text)
   (printout out "YN:: yes or no " ?text)
   (bind ?reply (read in))
   (if (member$ ?reply ?answers)
     then (assert (variable ?variable ?reply))
          (retract ?f2)
     else (assert (goal is ?variable))))

;;;**************************
;;;* DEFFACTS KNOWLEDGE BASE *
;;;**************************

(deffacts knowledge-base
   (goal is type.animal)
   (legalanswers are yes no)
   (rule (if backbone is yes)
         (then superphylum is backbone))
   (rule (if backbone is no)
         (then superphylum is jellyback))
   (question backbone is "Does your animal have a backbone?")
   (rule (if superphylum is backbone and
          warm.blooded is yes)
         (then phylum is warm))
   (rule (if superphylum is backbone and
          warm.blooded is no)
         (then phylum is cold))
   (question warm.blooded is "Is the animal warm blooded?")
   (rule (if superphylum is jellyback and
          live.prime.in.soil is yes)
         (then phylum is soil))
   (rule (if superphylum is jellyback and
```

```
        live.prime.in.soil is no)
       (then phylum is elsewhere))
   (question live.prime.in.soil is "Does your animal live primarily in soil?")
   (rule (if phylum is warm and
        has.breasts is yes)
       (then class is breasts))
   (rule (if phylum is warm and
        has.breasts is no)
       (then type.animal is bird/penguin))
   (question has.breasts is "Normally, does the female of your animal nurse its young with
milk?")
   (rule (if phylum is cold and
        always.in.water is yes)
       (then class is water))
   (rule (if phylum is cold and
        always.in.water is no)
       (then class is dry))
   (question always.in.water is "Is your animal always in water?")
   (rule (if phylum is soil and
        flat.bodied is yes)
       (then type.animal is flatworm))
   (rule (if phylum is soil and
        flat.bodied is no)
       (then type.animal is worm/leech))
   (question flat.bodied is "Does your animal have a flat body?")
   (rule (if phylum is elsewhere and
        body.in.segments is yes)
       (then class is segments))
   (rule (if phylum is elsewhere and
        body.in.segments is no)
       (then class is unified))
   (question body.in.segments is "Is the animals body in segments?")
   (rule (if class is breasts and
        can.eat.meat is yes)
       (then order is meat))
   (rule (if class is breasts and
        can.eat.meat is no)
       (then order is vegy))
   (question can.eat.meat is "Does your animal eat red meat?")
   (rule (if class is water and
        boney is yes)
       (then type.animal is fish))
```

```
(rule (if class is water and
      boney is no)
     (then type.animal is shark/ray))
(question boney is "Does your animal have a boney skeleton?")
(rule (if class is dry and
      scally is yes)
     (then order is scales))
(rule (if class is dry and
      scally is no)
     (then order is soft))
(question scally is "Is your animal covered with scaled skin?")
(rule (if class is segments and
      shell is yes)
     (then order is shell))
(rule (if class is segments and
      shell is no)
     (then type.animal is centipede/millipede/insect))
(question shell is "Does your animal have a shell?")
(rule (if class is unified and
      digest.cells is yes)
     (then order is cells))
(rule (if class is unified and
      digest.cells is no)
     (then order is stomach))
(question digest.cells is "Does your animal use many cells to digest it's food instead of a
stomach?")
(rule (if order is meat and
      fly is yes)
     (then type.animal is bat))
(rule (if order is meat and
      fly is no)
     (then family is nowings))
(question fly is "Can your animal fly?")
(rule (if order is vegy and
      hooves is yes)
     (then family is hooves))
(rule (if order is vegy and
      hooves is no)
     (then family is feet))
(question hooves is "Does your animal have hooves?")
(rule (if order is scales and
      rounded.shell is yes)
```

```
     (then type.animal is turtle))
(rule (if order is scales and
      rounded.shell is no)
     (then family is noshell))
(question rounded.shell is "Does the animal have a rounded shell?")
(rule (if order is soft and
      jump is yes)
     (then type.animal is frog))
(rule (if order is soft and
      jump is no)
     (then type.animal is salamander))
(question jump is "Does your animal jump?")
(rule (if order is shell and
      tail is yes)
     (then type.animal is lobster))
(rule (if order is shell and
      tail is no)
     (then type.animal is crab))
(question tail is "Does your animal have a tail?")
(rule (if order is cells and
      stationary is yes)
     (then family is stationary))
(rule (if order is cells and
      stationary is no)
     (then type.animal is jellyfish))
(question stationary is "Is your animal attached permanently to an object?")
(rule (if order is stomach and
      multicelled is yes)
     (then family is multicelled))
(rule (if order is stomach and
      multicelled is no)
     (then type.animal is protozoa))
(question multicelled is "Is your animal made up of more than one cell?")
(rule (if family is nowings and
      opposing.thumb is yes)
     (then genus is thumb))
(rule (if family is nowings and
      opposing.thumb is no)
     (then genus is nothumb))
(question opposing.thumb is "Does your animal have an opposing thumb?")
(rule (if family is hooves and
      two.toes is yes)
```

```
     (then genus is twotoes))
(rule (if family is hooves and
      two.toes is no)
     (then genus is onetoe))
(question two.toes is "Does your animal stand on two toes/hooves per foot?")
(rule (if family is feet and
      live.in.water is yes)
     (then genus is water))
(rule (if family is feet and
      live.in.water is no)
     (then genus is dry))
(question live.in.water is "Does your animal live in water?")
(rule (if family is noshell and
      limbs is yes)
     (then type.animal is crocodile/alligator))
(rule (if family is noshell and
      limbs is no)
     (then type.animal is snake))
(question limbs is "Does your animal have limbs?")
(rule (if family is stationary and
      spikes is yes)
     (then type.animal is sea.anemone))
(rule (if family is stationary and
      spikes is no)
     (then type.animal is coral/sponge))
(question spikes is "Does your animal normally have spikes radiating from it's body?")
(rule (if family is multicelled and
      spiral.shell is yes)
     (then type.animal is snail))
(rule (if family is multicelled and
      spiral.shell is no)
     (then genus is noshell))
(question spiral.shell is "Does your animal have a spiral-shaped shell?")
(rule (if genus is thumb and
      prehensile.tail is yes)
     (then type.animal is monkey))
(rule (if genus is thumb and
      prehensile.tail is no)
     (then species is notail))
(question prehensile.tail is "Does your animal have a prehensile tail?")
(rule (if genus is nothumb and
      over.400 is yes)
```

```
     (then species is 400))
(rule (if genus is nothumb and
      over.400 is no)
     (then species is under400))
(question over.400 is "Does an adult normally weigh over 400 pounds?")
(rule (if genus is twotoes and
      horns is yes)
     (then species is horns))
(rule (if genus is twotoes and
      horns is no)
     (then species is nohorns))
(question horns is "Does your animal have horns?")
(rule (if genus is onetoe and
      plating is yes)
     (then type.animal is rhinoceros))
(rule (if genus is onetoe and
      plating is no)
     (then type.animal is horse/zebra))
(question plating is "Is your animal covered with a protective plating?")
(rule (if genus is water and
      hunted is yes)
     (then type.animal is whale))
(rule (if genus is water and
      hunted is no)
     (then type.animal is dolphin/porpoise))
(question hunted is "Is your animal, unfortunately, commercially hunted?")
(rule (if genus is dry and
      front.teeth is yes)
     (then species is teeth))
(rule (if genus is dry and
      front.teeth is no)
     (then species is noteeth))
(question front.teeth is "Does your animal have large front teeth?")
(rule (if genus is noshell and
      bivalve is yes)
     (then type.animal is clam/oyster))
(rule (if genus is noshell and
      bivalve is no)
     (then type.animal is squid/octopus))
(question bivalve is "Is your animal protected by two half-shells?")
(rule (if species is notail and
      nearly.hairless is yes)
```

```
     (then type.animal is man))
(rule (if species is notail and
      nearly.hairless is no)
     (then subspecies is hair))
(question nearly.hairless is "Is your animal nearly hairless?")
(rule (if species is 400 and
      land.based is yes)
     (then type.animal is bear/tiger/lion))
(rule (if species is 400 and
      land.based is no)
     (then type.animal is walrus))
(question land.based is "Is your animal land based?")
(rule (if species is under400 and
      thintail is yes)
     (then type.animal is cat))
(rule (if species is under400 and
      thintail is no)
     (then type.animal is coyote/wolf/fox/dog))
(question thintail is "Does your animal have a thin tail?")
(rule (if species is horns and
      one.horn is yes)
     (then type.animal is hippopotamus))
(rule (if species is horns and
      one.horn is no)
     (then subspecies is nohorn))
(question one.horn is "Does your animal have one horn?")
(rule (if species is nohorns and
      lives.in.desert is yes)
     (then type.animal is camel))
(rule (if species is nohorns and
      lives.in.desert is no)
     (then type.animal is giraffe))
(question lives.in.desert is "Does your animal normally live in the desert?")
(rule (if species is teeth and
      large.ears is yes)
     (then type.animal is rabbit))
(rule (if species is teeth and
      large.ears is no the type.animal is rat/mouse/squirrel/beaver/porcupine))
(question large.ears is "Does your animal have large ears?")
(rule (if species is noteeth and
      pouch is yes)
     (then type.animal is "kangaroo/koala bear"))
```

```
(rule (if species is noteeth and
       pouch is no)
      (then type.animal is mole/shrew/elephant))
(question pouch is "Does your animal have a pouch?")
(rule (if subspecies is hair and
       long.powerful.arms is yes)
      (then type.animal is orangutan/gorilla/chimpanzie))
(rule (if subspecies is hair and
       long.powerful.arms is no)
      (then type.animal is baboon))
(question long.powerful.arms is "Does your animal have long, powerful arms?")
(rule (if subspecies is nohorn and
       fleece is yes)
      (then type.animal is sheep/goat))
(rule (if subspecies is nohorn and
       fleece is no)
      (then subsubspecies is nofleece))
(question fleece is "Does your animal have fleece?")
(rule (if subsubspecies is nofleece and
       domesticated is yes)
      (then type.animal is cow))
(rule (if subsubspecies is nofleece and
       domesticated is no)
      (then type.animal is deer/moose/antelope))
(question domesticated is "Is your animal domesticated?")
(answer is "r::I think your animal is a " type.animal))

(printout t "end of animal.clp " )
```

Appendix II: Edited Sticks.clp

```
;;;======================================================
;;;   Sticks Program
;;;
;;;     This program was introduced in Chapter 9.
;;;
;;;     CLIPS Version 6.0 Example
;;;
;;;     To execute, merely load, reset and run.
;;;======================================================

; **************
```

```
; FACT TEMPLATES
; **************

; The phase fact indicates the current action to be
; undertaken before the game can begin

; (phase
;    <action>)      ; Either choose-player or
                    ; select-pile-size

; The player-select fact contains the human's response to
; the "Who moves first?" question.

; (player-select
;    <choice>)      ; Valid responses are c (for computer)
                    ; and h (for human).

; The pile-select fact contains the human's response to
; the "How many sticks in the pile?" question.

; (pile-select
;    <choice>)      ; Valid responses are integers
                    ; greater than zero.

; The pile-size fact contains the current number
; of sticks in the stack.

; (pile-size
;    <sticks>)      ; An integer greater than zero.

; The player-move fact indicates whose turn it is.

; (player-move
;    <player)       ; Either c (for computer)
                    ; or h (for human).

; The human-takes fact contains the human's response to
; the "How many sticks do you wish to take?" question.


; (human-takes
;    <choice>)      ; Valid responses are 1, 2, and 3.
```

```
; The take-sticks facts indicate how many sticks the
; computer should take based on the remainder when the
; stack size is divided by 4.

(deftemplate take-sticks
   (slot how-many)         ; Number of sticks to take.
   (slot for-remainder))   ; Remainder when stack is
                           ; divided by 4.

; ********
; DEFFACTS
; ********

(deffacts initial-phase
   (phase choose-player))

(deffacts take-sticks-information
   (take-sticks (how-many 1) (for-remainder 1))
   (take-sticks (how-many 1) (for-remainder 2))
   (take-sticks (how-many 2) (for-remainder 3))
   (take-sticks (how-many 3) (for-remainder 0)))

; ********
; DEFFUNCTIONS
; ********

(deffunction ask-start-again ()
  (printout out "rr::thank you for playing ")
  (if (eq (read in) yes) then
    (assert (phase choose-player))))


; *****
; RULES
; *****

; RULE player-select
; IF
;   The phase is to choose the first player
; THEN
;   Ask who should move first, and
```

```
;   Get the human's response

(defrule player-select
   (phase choose-player)
   =>
   (printout out "HC::Who moves first (Computer "
            "Human)? ")
   (assert (player-select (read in))))

; RULE good-player-choice
; IF
;   The phase is to choose the first player, and
;   The human has given a valid response
; THEN
;   Remove unneeded information, and
;   Indicate whose turn it is, and
;   Indicate that the pile size should be chosen

(defrule good-player-choice
   ?phase <- (phase choose-player)
   ?choice <- (player-select ?player&:(or (eq ?player computer) (eq ?player human)))
   =>
   (retract ?phase ?choice)
   (assert (player-move ?player))
   (assert (phase select-pile-size)))

; RULE bad-player-choice
; IF
;   The phase is to choose the first player, and
;   The human has given a invalid response
; THEN
;   Remove unneeded information, and
;   Indicate that the first player should be chosen again,
;   and Print the valid choices

(defrule bad-player-choice
   ?phase <- (phase choose-player)
   ?choice <- (player-select ?player&~computer&~human)
   =>
   (retract ?phase ?choice)
   (assert (phase choose-player))
   (printout t "ss::Choose computer or human." ))
```

```
; RULE pile-select
; IF
;   The phase is to choose the pile size
; THEN
;   Ask what the pile size should be, and
;   Get the human's response


(defrule pile-select
   (phase select-pile-size)
   =>
   (printout out "19::1 to 9, How many sticks in the pile? ")
   (assert (pile-select (read in))))

; RULE good-pile-choice
; IF
;   The phase is to choose the pile size, and
;   The human has given a valid response
; THEN
;   Remove unneeded information, and
;   Store the pile size

(defrule good-pile-choice
   ?phase <- (phase select-pile-size)
   ?choice <- (pile-select ?size&:(integerp ?size)
                           &:(> ?size 0))
   =>
   (retract ?phase ?choice)
   (assert (pile-size ?size)))

; RULE bad-pile-choice
; IF
;   The phase is to choose the pile size, and
;   The human has given a invalid response
; THEN
;   Remove unneeded information, and
;   Indicate that the pile size should be chosen again,
;   and Print the valid choices

(defrule bad-pile-choice
   ?phase <- (phase select-pile-size)
```

```
   ?choice <- (pile-select ?size&:(or (not (integerp ?size))
                                      (<= ?size 0)))
   =>
   (retract ?phase ?choice)
   (assert (phase select-pile-size))
   (printout t "ss::Choose an integer greater than zero."
             ))




; RULE computer-loses
; IF
;   The pile size is 1, and
;   It is the computer's move
; THEN
;   Print that the computer has lost the game

(defrule computer-loses
  ?pile <- (pile-size 1)
  ?move <- (player-move computer)
   =>
   (printout out "ss::Computer must take the last stick!"   "I lose!" )
   (retract ?pile)
   (retract ?move)
   (ask-start-again))

; RULE human-loses
; IF
;   The pile size is 1, and
;   It is the human's move
; THEN
;   Print that the human has lost the game

(defrule human-loses
  ?pile <- (pile-size 1)
  ?move <- (player-move human)
   =>
   (printout out ss::"You must take the last stick!"   "You lose!" )
   (retract ?pile)
   (retract ?move)
   (ask-start-again))
```

```
; RULE get-human-move
; IF
;   The pile size is greater than 1, and
;   It is the human's move
; THEN
;   Ask how many sticks to take, and
;   Get the human's response

(defrule get-human-move
   (pile-size ?size&:(> ?size 1))
   (player-move human)
   =>
   (printout out "13::1 to 3, How many sticks do you wish to take? ")
   (assert (human-takes (read in))))


; RULE good-human-move
; IF
;   There is a pile of sticks, and
;   The human has chosen how many sticks to take, and
;   It is the human's move, and
;   The human's choice is valid
; THEN
;   Remove unneeded information, and
;   Compute the new pile size, and
;   Update the pile size, and
;   Print the number of sticks left in the stack, and
;   Trigger the computer player's turn

(defrule good-human-move
   ?pile <- (pile-size ?size)
   ?move <- (human-takes ?choice)
   ?whose-turn <- (player-move human)
   (test (and (integerp ?choice)
              (>= ?choice 1)
              (<= ?choice 3)
              (< ?choice ?size)))
   =>
   (retract ?pile ?move ?whose-turn)
   (bind ?new-size (- ?size ?choice))
   (assert (pile-size ?new-size))
```

```
   (printout out "ss::"?new-size " sticks left in the pile."
             )
   (assert (player-move computer)))

; RULE bad-human-move
; IF
;   There is a pile of sticks, and
;   The human has chosen how many sticks to take, and
;   It is the human's move, and
;   The human's choice is invalid
; THEN
;   Print the valid choices, and
;   Remove unneeded information, and
;   Retrigger the human player's move




(defrule bad-human-move
   (pile-size ?size)
   ?move <- (human-takes ?choice)
   ?whose-turn <- (player-move human)
   (test (or (not (integerp ?choice))
             (< ?choice 1)
             (> ?choice 3)
             (>= ?choice ?size)))
   =>
   (printout t "ss::Number of sticks must be between 1 and 3,"

               "and you must be forced to take the last "
               "stick." )
   (retract ?move ?whose-turn)
   (assert (player-move human)))

; RULE computer-move
; IF
;   It is the computers's move, and
;   The pile size is greater than 1, and
;   The computer's response is available
; THEN
;   Remove unneeded information, and
```

```
;   Compute the new pile size, and
;   Print the number of sticks left in the stack, and
;   Update the pile size, and
;   Trigger the human players move

(defrule computer-move
   ?whose-turn <- (player-move computer)
   ?pile <- (pile-size ?size&:(> ?size 1))
   (take-sticks (how-many ?number)
              (for-remainder ?X&:(= ?X (mod ?size 4))))
   =>
   (retract ?whose-turn ?pile)
   (bind ?new-size (- ?size ?number))
   (printout out "ss::Computer takes " ?number " sticks."
              ?new-size " sticks left in the pile."
             )
   (assert (pile-size ?new-size))
   (assert (player-move human)))
```

9 Bibliography

[1] D. Stone, C. Jarrett, M. Woodroffe, and S. Minocha, *User Interface Design and Evaluation*. Morgan Kaufmann, 2005.

[2] H. Xuguang, “An Introduction to Android,” *Dababase Lab. Inha Univeristy*, 2009.

[3] “What is Android?” [Online]. Available: http://www.academia.edu/1240698/What_is_Android. [Accessed: 07-Oct-2012].

[4] N. Gandhewar, “Google Android: An emerging software platform for mobile devices,” in *International Journal on Computer Science and Engineering*, 2010, p. 12.

[5] M. N. Cortimiglia, A. Ghezzi, and F. Renga, “Mobile Applications and Their Delivery Platforms,” *IT Professional*, vol. 13, no. 5, pp. 51 –56, Oct. 2011.

[6] “Number of available Android applications - AppBrain.” [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps. [Accessed: 21-Oct-2012].

[7] “logcat | Android Developers.” [Online]. Available: http://developer.android.com/tools/help/logcat.html. [Accessed: 05-Nov-2012].

[8] E. and T. W. A. of Science, *International journal of computer science and engineering*. Turkey: World Academy of Science, Engineering and Technology, 2007.

[9] A. K. Saha, “A Developer’s First Look At Android,” *Linux For You*, p. 3, 2008.

[10] C.-M. Lin, J.-H. Lin, C.-R. Dow, and C.-M. Wen, “Benchmark Dalvik and Native Code for Android System,” in *2011 Second International Conference on Innovations in Bio-inspired Computing and Applications (IBICA)*, 2011, pp. 320 –323.

[11] Y. Wu, J. Luo, and L. Luo, “Porting Mobile Web Application Engine to the Android Platform,” in *2010 IEEE 10th International Conference on Computer and Information Technology (CIT)*, 2010, pp. 2157 –2161.

[12] Y. Chen, C.-Y. Hsu, L. Liu, and S. Yang, “Constructing a nutrition diagnosis expert system,” *Expert Systems with Applications*, vol. 39, no. 2, pp. 2132–2156, Feb. 2012.

[13] E. Friedman-Hall, *Jess in action*. Greenwich, Conn.; London: Manning ; Pearson Education, 2003.

[14] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*. Berlin; New York: Springer-Verlag, 2003.

[15] G. D. Moerdler, “Menu Interfaces to Expert Systems: Overview and Evaluation.” Department of Computer Science, Columbia University, 1984.

[16] D. J. Power and R. Sharda, “Decision Support Systems,” in *Springer Handbook of Automation*, S. Y. Nof, Ed. Springer Berlin Heidelberg, 2009, pp. 1539–1548.

[17] M. H. Fazel Zarandi, B. Rezaee, I. B. Turksen, and E. Neshat, “A type-2 fuzzy rule-based expert system model for stock price analysis,” *Expert Systems with Applications*, vol. 36, no. 1, pp. 139–154, Jan. 2009.

[18] S. Kumar and R. Singh, “An expert system for design of progressive die for use in sheet metal industries,” *J. Sci. Ind. Res*, vol. 69, pp. 510–514, 2010.

[19] K. Rothert, V. J. Strecher, L. A. Doyle, W. M. Caplan, J. S. Joyce, H. B. Jimison, L. M. Karm, A. D. Mims, and M. A. Roth, “Web-based Weight Management Programs in an Integrated Health Care Setting: A Randomized, Controlled Trial,” *Obesity*, vol. 14, no. 2, pp. 266–272, 2006.

[20] S. S. Abu-Naser, H. E.- Hissi, M. A.- Rass, and N. E.- khozondar, “An Expert System for Endocrine Diagnosis and Treatments using JESS,” *Journal of Artificial Intelligence*, vol. 3, no. 4, pp. 239–251, Apr. 2010.
[21] M. J. Coombs and J. L. Alty, “Face-to-face guidance of university computer users: II. Characterizing advisory interactions,” *International Journal of Man-Machine Studies*, vol. 12, no. 4, pp. 407–429, 1980.
[22] J. M. Carroll and J. McKendree, “Interface design issues for advice-giving expert systems,” *Commun. ACM*, vol. 30, no. 1, pp. 14–32, Jan. 1987.
[23] D. L. Hudson and M. E. Cohen, “The Role of User-Interface in a Medical Expert System,” *Proc Annu Symp Comput Appl Med Care*, pp. 232–236, Nov. 1985.
[24] D. C. BERRY and D. E. BROADBENT, “Expert systems and the man-machine interface. Part Two: The user interface.,” *Expert Systems*, vol. 4, no. 1, pp. 18–27, 1987.
[25] N. M. Avouris and S. Finotti, “User interface design to expert systems based on hierarchical spatial representations,” *Expert Systems with Applications*, vol. 6, no. 2, pp. 109–118, Apr. 1993.
[26] F. W. Rook and M. L. Donnell, “Human cognition and the expert system interface: mental models and inference explanations,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 23, no. 6, pp. 1649 –1661, Dec. 1993.
[27] J. Carbonell, W. M. Boggs, M. Mauldin, and P. Anick, “THE XCALIBUR PROJECT: A Natural Language Interface To Expert Systems,” *Computer Science Department*, Jan. 1985.
[28] B. J. S, “Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II and III,” *Intelligent Tutoring Systems*, pp. 227–282, 1982.
[29] T. Giorgino, I. Azzini, C. Rognoni, S. Quaglini, M. Stefanelli, R. Gretter, and D. Falavigna, “Automated spoken dialogue system for hypertensive patient home management,” *International Journal of Medical Informatics*, vol. 74, no. 2–4, pp. 159–167, Mar. 2005.
[30] “Top categories on the Android market - AppBrain.” [Online]. Available: http://www.appbrain.com/stats/android-market-app-categories. [Accessed: 10-Oct-2012].
[31] F. Manjoo, “Now You’re Talking!,” *Slate*, 06-Apr-2011.
[32] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky, “Pocketsphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices,” in *2006 IEEE International Conference on Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings*, 2006, vol. 1, p. I.
[33] K. Vertanen and P. O. Kristensson, “Getting it right the second time: Recognition of spoken corrections,” in *Spoken Language Technology Workshop (SLT), 2010 IEEE*, 2010, pp. 289–294.
[34] S. A. Chowdhury, “Implementation of speech recognition system for Bangla,” Department of Computer Science and Engineering, BRAC University, 2010.
[35] “Basic concepts of speech - CMUSphinx Wiki.” [Online]. Available: http://cmusphinx.sourceforge.net/wiki/tutorialconcepts. [Accessed: 21-Oct-2012].
[36] C. L. Forgy, “Rete: a fast algorithm for the many pattern/many object pattern match problem,” *EXPERT SYSTEMS INTERNATIONAL JOURNAL OF KNOWLEDGE ENGINEERING*, p. 324, 1991.
[37] “Jess, the Java Expert System Shell - The Rete Algorithm.” [Online]. Available: http://www.jessrules.com/jess/docs/61/rete.html. [Accessed: 16-Oct-2012].

[38] “Jess, the Rule Engine for the Java Platform - Jess Application Design.” [Online]. Available: http://herzberg.ca.sandia.gov/docs/70/architecture.html. [Accessed: 21-Oct-2012].
[39] K. V. Laerhoven, “Comparison of the CLIPS and JESS expert system shells,” Project report for Industrial Applications of AI, 1999.
[40] “Jess, the Java Expert System Shell - Introduction.” [Online]. Available: http://www.csie.ntu.edu.tw/~sylee/courses/jess/docs/intro.html. [Accessed: 21-Oct-2012].
[41] “Jess, the Java Expert System Shell - Introduction to Programming with Jess in Java.” [Online]. Available: http://www.csie.ntu.edu.tw/~sylee/courses/jess/docs/library.html. [Accessed: 21-Oct-2012].
[42] “Old Nabble - JESS: Re: Jess.jar on Android.” [Online]. Available: http://old.nabble.com/JESS%3A-Re%3A-Jess.jar-on-Android-p30880037.html. [Accessed: 07-Oct-2012].
[43] “Jess.jar on Android - Google Groups.” [Online]. Available: https://groups.google.com/forum/?fromgroups=#!topic/android-developers/uUDtlr-pDuk. [Accessed: 07-Oct-2012].
[44] “Jess is a rule engine and scripting environment written entirely in Java.” [Online]. Available: http://comments.gmane.org/gmane.comp.java.jess/6165. [Accessed: 07-Oct-2012].
[45] Mindpipe, “Mind|Pipe - by Rolf Kulemann: Why Android does not support fully java.beans package,” *Mind|Pipe - by Rolf Kulemann*. 29-Nov-2011.
[46] “Re: RE: JESS: Re: JESS on android device.” [Online]. Available: http://www.mail-archive.com/jess-users@sandia.gov/msg11600.html. [Accessed: 07-Oct-2012].
[47] “openbeans - Repackaging of java.beans for android - Google Project Hosting.” [Online]. Available: http://code.google.com/p/openbeans/. [Accessed: 07-Oct-2012].
[48] J. Trivi, “Android Developers Blog: An introduction to Text-To-Speech in Android,” *Android Developers Blog*. 23-Sep-2009.
[49] “Building Pocketsphinx On Android | CMU Sphinx - Speech Recognition Toolkit.” [Online]. Available: http://cmusphinx.sourceforge.net/2011/05/building-pocketsphinx-on-android/. [Accessed: 05-Nov-2012].
[50] J. P. Chin, V. A. Diehl, and K. L. Norman, “Development of an instrument measuring user satisfaction of the human-computer interface,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 1988, pp. 213–218.
[51] M. Boros, W. Eckert, F. Gallwitz, G. Gorz, G. Hanrieder, and H. Niemann, “Towards understanding spontaneous speech: word accuracy vs. concept accuracy,” in , *Fourth International Conference on Spoken Language, 1996. ICSLP 96. Proceedings*, 1996, vol. 2, pp. 1009 –1012 vol.2.
[52] A. Harvey, R. McCrindle, K. Lundqvist, and P. Parslow, “Automatic speech recognition for assistive technology devices,” 2010.