The ITB Journal

Volume 5 | Issue 2 Article 4

2004

Voice Activated Command and Control with Speech Recognition over Wireless Networks

Tony Ayres

Brian Nolan

Recommended Citation

Ayres, Tony and Nolan, Brian (2004) "Voice Activated Command and Control with Speech Recognition over Wireless Networks," *The ITB Journal*: Vol. 5: Iss. 2, Article 4.
doi:10.21427/D78F2X
Available at: https://arrow.tudublin.ie/itbj/vol5/iss2/4

Voice Activated Command and Control with Speech Recognition over Wireless Networks

Mr. Tony Ayres, Dr. Brian Nolan

Institute of Technology Blanchardstown

E-mail tony.ayres@itb.ie brian.nolan@itb.ie

Abstract

This paper presents work conducted to date on the development of a voice activated command and control framework specifically for the control of remote devices in a ubiquitous computing environment. The prototype device is a Java controlled Lego Mindstorm robot. The research considers three different scenario configurations. A recognition grammar for command and control of the robot has been created and implemented in Java, in part in the recognition engine and in part on the robot. The physical topology involves Java at each node endpoint, that is, at the handheld PC (iPaq), the PC workstation, the Linux server and onboard the robot (including its Java based Lejos OS). Network communications is primarily WLAN with an element of IR where the robot is concerned. The speech recognition software used includes Sphinx4, Microsoft SAPI and the Java Speech API. We compare these speech technologies and present their benefits in the context of this research. For each given scenario we present and discuss the implementation challenges encountered and their corresponding solutions. We outline our future plans to create additional grammars to extend the frameworks range of devices.

1. Introduction

This research project is concerned with building a framework with applications for command and control of a remote device by voice activation with speech recognition from a local control station over a wireless network in three different scenario configurations. In each scenario Java plays a critical role.

The first of these scenarios or configurations is based on a PC workstation under the Windows operating system connected via a wireless network to a PC-based server. The server issues commands to a remote device. The second scenario will involve developing a distributed speech recognition engine, between an iPaq pocket PC and a PC based server which will issue the commands to the remote device. The third scenario will involve a mobile device, specifically an iPaq Pocket PC that will connect over a wireless network to a PC-based server. The server will, again, issue commands to the remote device.

For purposes of this research project the remote device will be a Java controlled Lego MindStorms robot that will move and undertake certain actions under instructions relayed to it over a wireless interface. The robot could be replaced with practically any computing or electronic device which has a Java Virtual Machine installed. As part of this research it is our intention to develop a speech recognition engine using the Java programming language.

2. Technology Review

2.1 Applications of Speech Technology

The applications for speech recognition can be grouped into three distinct categories; these are command and control, dictation and authentication

Command and Control applications are concerned with providing the user of these systems the means to control items within their environment with voice commands appropriate to the domain. The appliance of command and control technology may manifest itself in the control of user interface menus in personal computing desktop applications or the control of large scale mechanical or electronic and computing devices.

Dictation applications allow the user to speak to the system and have it generate a transcript of what has been said. This is particularly useful in legal or medical arenas where information is recorded in real time and making written notes would be too slow. Specialized dictation grammars exist for application domains such as this. Furthermore dictation and command/control functionality can be combined in word processing applications such as Microsoft Word.

Speech technology can also be used for authentication purposes as part of a security system. The signal analysis algorithms employed as part of a speech recognition front end generate a feature vectors which can be matched to a pre recorded sample of a users voice. Given that each person has a unique voice print this can be used for authentication purposes.

2.2 Types of Speech Recognition Systems

Speech Recognition systems can be classified according to whether they are speaker dependent or speaker independent. Speaker dependent speech recognition engines require the user to train a profile of their voice for the engine to use when performing recognition. This process typically involves reading sample passages of text to the engine. Conversely speaker independent systems do not require the user to train them before achieving high recognition accuracy. In this instance a pre recorded corpus of words is compared to the input speech vectors to generate recognition result.

In general terms speaker dependent systems achieve greater recognition accuracy given that the engine will be customized for a specific users voice, however speaker independent systems can achieve comparable accuracy levels where the grammar is constrained and as such are ideally suited for applications which may have a large number of users.

2.3 Process of Speech Recognition

The components of speech recognition systems include a speech corpus (database), a frontend processing system and a speech decoding unit. The frontend is responsible for analyzing the speech input and extracting feature vectors which will be used in the decoding process (figure 1). The speech corpus, in the case of speaker independent recognition engines will contain acoustic information for all the words and phonemes which the corpus contains. The speech decoding process compares input features generated by the frontend with those in the corpus, the result is usually a probability score, representing the engines confidence in the accuracy of any match found.

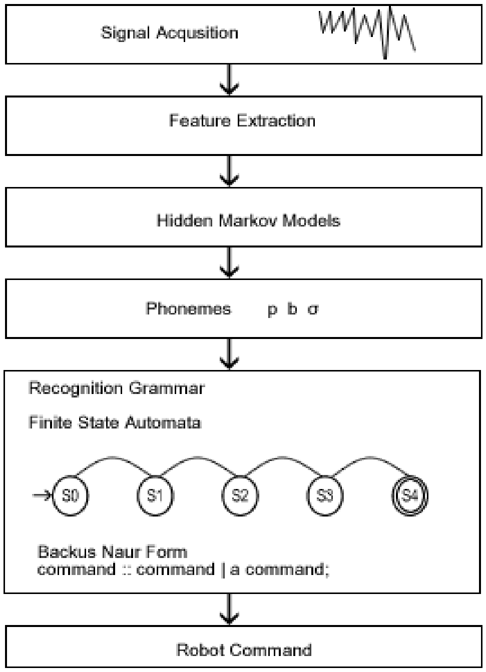

Figure 23 Speech Recognition Process

Modern speech recognition systems use stochastic techniques to model and decode speech signal data. Hidden Markov Models (HMM) have become the most successful statistical method for speech recognition. The HMM phase of speech recognition comes after an initial analysis and feature extraction process on the incoming speech signal. The feature extraction process generates feature vectors which are used as the input to the HMM.

A Markov model is specified by the states Q, the set of transition probabilities A, defined start and end states and a set of observation likelihood's B. A Hidden Markov Model formally differs from a Markov model by adding two other requirements. Firstly it has a set of observation symbols which is not drawn the from the same alphabet as the state Q. Secondly the observation likelihood function B is not limited to the values 1 and 0, in the HMM the probability can take any value between 0 and 1. The parameters needed to define a HMM are as follows:

- A set of states
- Transition probabilities
- Observation likelihood's
- Initial distribution
- Accepting States

In order to extract a suitable output for speech recognition we must parse the representation which the markov model contains.

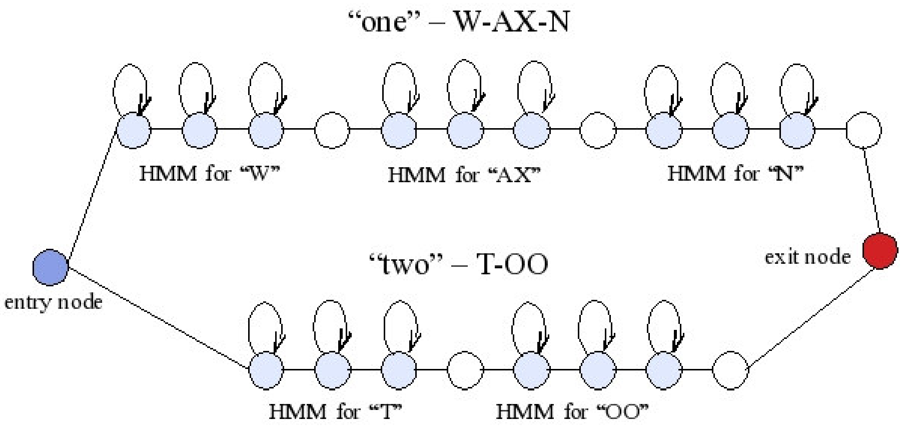

Figure 24 HMM Decoding of Phones (Taken From Sphinx4)

Figure 2 shows the HMM graph for the words "one" and "two" according to the Sphinx4 speech decoder. HMMs are generated for each phoneme that constitutes a word. Each HMM has a transition from to various nodes in the graph and to itself. The Viterbi [1] algorithm is used to find the best path through the graph based on the highest score of each transition.

2.4 Selecting a Speech Engine

The Java Speech API [2] is used to provide a platform independent speech recognition and synthesis interface for Java applications. Sun Microsystems supply a reference standard for the Java Speech API, but they do not provide an implementation. The Java Speech API implements no speech processing functionality of its own but allows Java applications to plug into functionality available on the host operating system.

Recent advances in signal processing algorithms coupled with the development of HMM based decoding techniques, has led to the development of many highly accurate speech recognition engines. The majority of these speech engines are commercial products which include text to speech capabilities in tandem with their recognition functionality. These products include Microsoft Speech API version 5 [3] (SAPI5), IBM Via Voice [4] and Dragon Naturally Speaking [5]. One defining characteristic of all these engines is that they are speaker dependent.

In the open source domain the Sphinx project is the only suitably large candidate. The Sphinx project is concerned with developing HMM based speaker independent recognition systems. The project has 3 engines available as source code downloads, namely Sphinx2, Sphinx3 and Sphinx4. Sphinx2 is a real time speech decoder, its feature include continuous speech decoding, can provide a single best or several alternative recognition's, support for bigram, trigram, or finite-state grammar language models. Sphinx3 is a state of the art speech decoder written in C. While it has a slower decoding speed than Sphinx 2, it provides more accurate recognition. Initially it was developed to perform batch speech decoding from audio files but is now capable of live decoding.

Sphinx4 is the latest speech decoder to be released by the project, initially it started as a port of Sphinx3 to the Java programming language; however the engine evolved to become more flexible than Sphinx3. A defining characteristic of Sphinx4 is the configuration of the engine, which is achieved through an XML configuration file. Each Java object in the Sphinx4 system can be instantiated through this file, this helps keep application code which implement Sphinx4 clear of Sphinx4 code which makes for easier debugging. Sphinx4 also includes an implementation of the Java Speech API. Its object oriented structure and easy XML configuration make it an ideal choice for conducting research

This research project defines a number of application scenarios [section 3]. A different speech engine is implemented in each of the scenarios, Scenario 1 uses Microsoft SAPI5 with the Cloudgarden JSAPI [6] implementation; Scenario 2 uses the Sphinx4 speech recognition engine which includes its own JSAPI implementation, Scenario 3 which is still in development, will use a lightweight Java/C++ based speech decoder based on the Sphinx engine.

We tested the performance of the Java Speech API with Sphinx4 and SAPI5 in the context of developing this framework for command and control. The test provided an insight into the characteristics of speaker dependent and speaker independent recognition engine and presented and opportunity to compare the current state of the art of both approaches to speech recognition.

2.4.1 Performance Analysis under the Java Speech API

Under Windows XP, the setup time for a JSAPI recognizer under both speech engines is in a similar range, although SAPI5 is marginally faster. Most notable is the length of time Sphinx takes to process the first command; after this initial command has been processed the time drops back to just over 3 seconds for each subsequent command. The SAPI5 configuration is extremely quick for all commands; although SAPI could be prone to a high number of errors, this occurred under JDK1.4 where the average error rate was 1.5 (accuracy of 63%). The graph shown in Figure 2 highlights the speed of the SAPI engine in processing the commands.

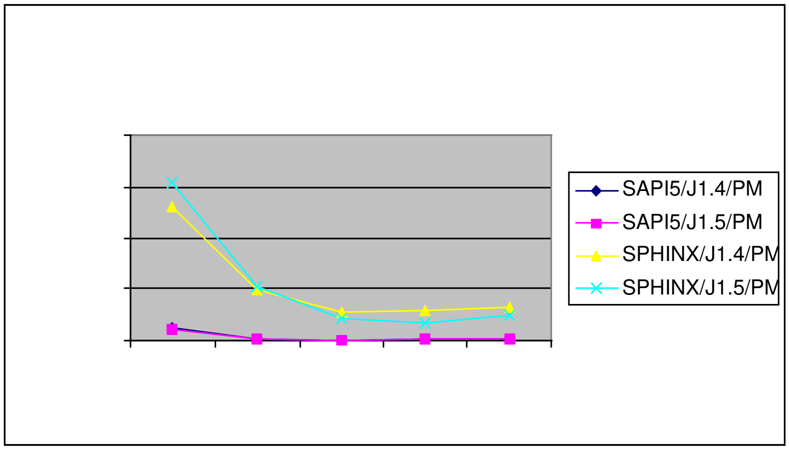

Figure 3 SAPI VS Sphinx Windows XP

Figure 3 also shows that Java 1.5 is faster than 1.4; this is most noticeable with the Sphinx4 test. While the setup times are almost identical, the time recognition time drops to around 2 seconds in comparison to 3 seconds with 1.4. The SAPI5 test also ran faster with JDK1.5, although the difference is marginal when compared to performance gain Sphinx4 achieves.

Sphinx encountered a trough in recognition accuracy with the Pentium IV/1.4 configuration, with at least one error occurring in the majority of tests, thus yielding an average of 1.4 errors which translates to an accuracy of 65%. In addition, the recognizer setup time was much longer than those on the Windows XP test machine. With JDK 1.5 Sphinx4 yielded a recognition accuracy of 93%, with only 0.2 errors. One noticeable difference is the recognition times under JDK 1.5, we encountered numerous spikes where decoding of commands took between 8 and 30 seconds, this occurred on 3 occasions while issuing the "Backward" command. As a result the average command time for JDK1.5 on the Windows 2000 machine is slower than JDK1.4. In figure 3 the graph shows the sharp peaks and troughs in the Sphinx performance under JDK1.5.

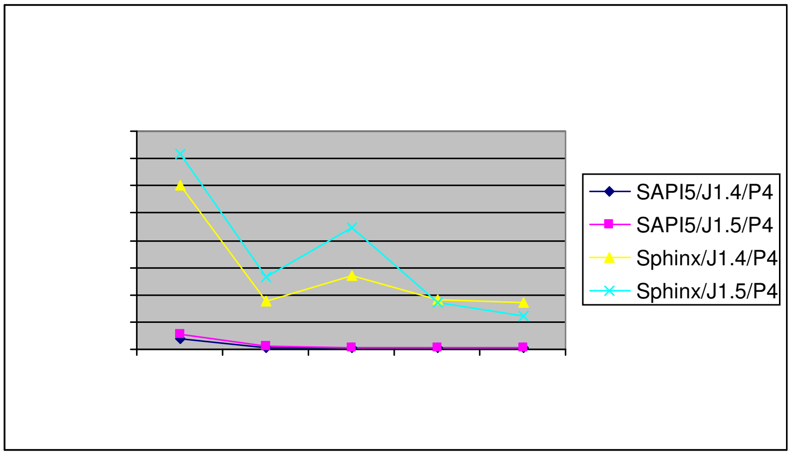

Figure 4 SAPI VS Sphinx on Windows 2000

The SAPI5 results on this machine were inline with the previous test, recognition time averages below 500ms and the setup time averages around 1 second. As with the Sphinx tests on this machine, the SAPI performance under JDK1.5 was not faster than JDK1.4.2.

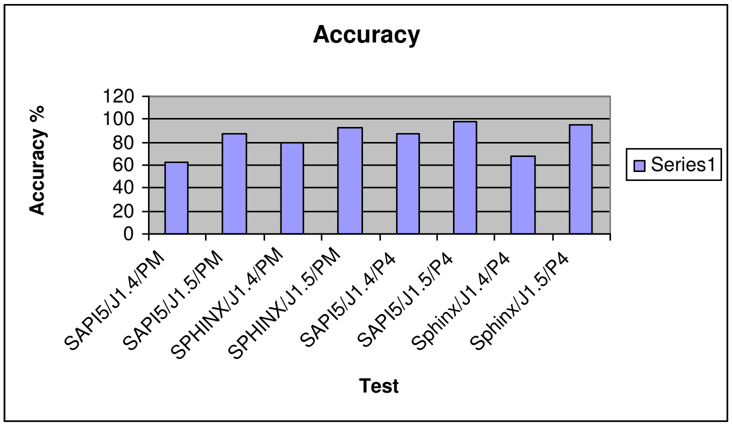

Figure 5 Accuracy Percentage

Figure 5 presents a bar graph indicating the recognition accuracy of test configuration. SAPI5 under Windows 2000 under JDK1.5 was the most accurate with a score of 98%, Sphinx4 on the same system and the same JDK received the second highest score with 95%. SAPI also holds the lowest score of 63% on Windows XP with JDK1.4. The average accuracy for both engines was 84%. Both engines had one trough of poor performance where the accuracy dropped considerably and as a result both engines received the same average accuracy percentage. If we ignore the trough values, then SAPI5 achieves the highest average score of 91% with Sphinx scoring 89%.

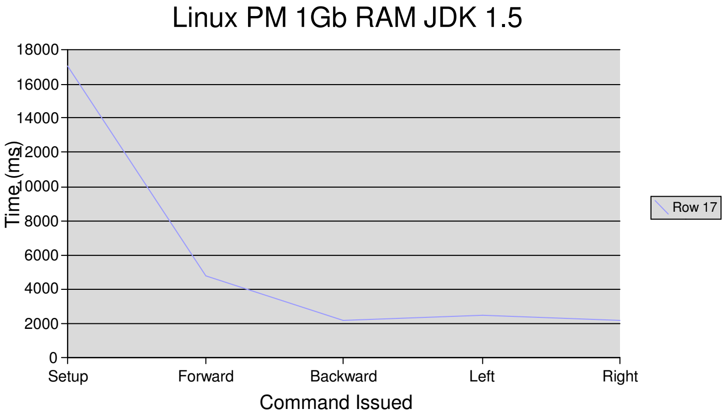

Figure 6 Linux Performance Results

We tested the Sphinx4 engine under Mandrake Linux 10.1 running the release version of JDK 1.5. This allows us to compare the performance of the Java runtime environment on a second operating environment for the purposes of speech recognition. Furthermore, Sphinx4 is one of the few large scale speech recognition engines available for the Linux operating system. In comparison to the Windows based tests the Linux test achieved the greatest accuracy with a score of 97%. Again the initial engine setup time is longer than

SAPI5/Windows, however it does perform better than Sphinx4 on Windows in both recognition processing time and accuracy.

The results of these tests show the relative strengths of both speech engines, performance wise there is little to separate them. There are significant differences between them, SAPI5 as a speaker dependent engine using a speech profile to decode speech input has the ability to recognize any speech utterance from the user who trained the profile. Sphinx4 is limited to recognizing only those words which are contained within the language model. This limitation in Sphinx4 makes it less suitable for dictation applications, but its high accuracy and open source code make it excellent for conducting research and building command and control applications. SAPI5 is an excellent engine for both dictation and command and control, but it is less adaptable for research purposes as it a closed system.

3. Application Topology

3.1 Scenario One

To date we have been developing the overall system architecture and evaluating speech technologies which will meet our requirements. Figure 1, illustrates the architecture of scenario one. In scenario one, we propose to develop a speech recognition application in Java using the Cloudgarden implementation of the Java Speech API. Cloudgarden provides JSAPI functionality in conjunction with a SAPI 4/5 compliant speech engine on the host operating system, supported SAPI speech engines include Microsoft SAPI, IBM Via Voice and Dragon Naturally Speaking.

The client application accepts speech input via microphone and performs speech recognition using the JSAPI. A successful output from the speech recognition process is one or more string tokens. The application delivers the recognized speech strings to a Java based server application over TCP/IP via a wireless network. The server application is running on a Linux operating system. The server processes the recognized speech strings in order to determine if they match terms in the robot control grammar. If a match is found, that particular command is issued to the robot over a wireless network.

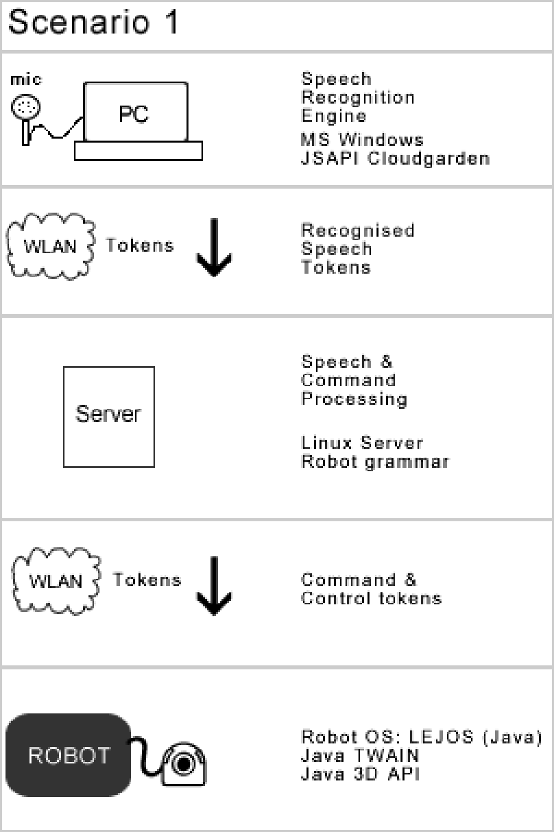

Figure 7 Scenario One

The robot will be fitted with a camera and we are investigating the possibility of using Java API's including the Java Twain and the Java 3D API to capture and process images returned to the client application from the robot.

3.2 Scenario Two

Scenario two offers an alternative approach. The PC workstation is replaced by a PDA with wireless networking capabilities, in this case an HP iPaq 5550 running Pocket PC 2003. The configuration of the HP iPaq Pocket PC 5550 is:

Processor	Intel XScale 400mhz
Memory	128mb RAM
ROM	48mb ROM
Networking	802.11b WLAN and Bluetooth
Operating System	Pocket PC 2003

The application on the PDA is concerned with capturing a speech input signal and sending it through the wireless interface to the server. The server hosts a processing engine for the robot grammar as in scenario one, however it also has an additional layer of functionality, namely, the speech recognition engine.

The PDA provides mobility but has limited processing power. In order to lessen the load on the PDA the speech recognition engine is distributed between the server and PDA. The PDA has the Jeode runtime environment installed on it. Jeode is a Personal Java compatible runtime environment.

Personal Java [7] is fully compatible with JDK 1.1, however JDK 1.1 does not have sufficient libraries to perform signal capture or speech recognition. To circumvent this obstacle, the Java Native Interface will be used to plug into the sound capture functionality of the PDA hardware; a digitized speech signal will be returned. This digitized speech will be sent to the server application where the remainder of the speech recognition process will take place.

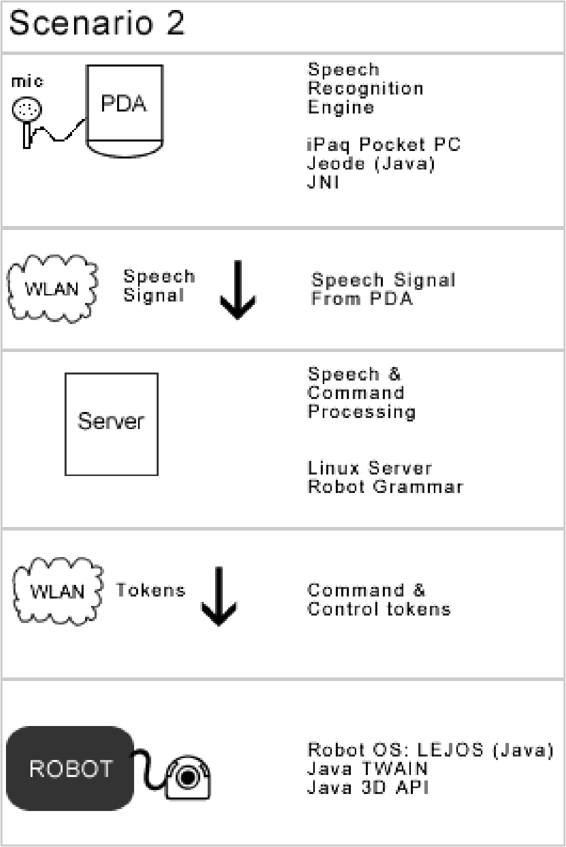

Figure 8 Scenario Two

Processing of the speech signal is the most processor intensive activity in the recognition process, therefore this function is assigned to the server, thus memory and processing power on the mobile device become of less importance. This approach presents greater complexity in acquiring the signal, sending it over the network and reading it back into the speech processing engine.

3.3 Scenario Three

In scenario three we port the speech recognition engine from the PC workstation in scenario one to the PDA. Given the resources of the PDA, a simple copy of the speech engine from scenario one is not feasible. The Cloudgarden JSAPI implementation requires a SAPI compliant engine and Java Development Kit 1.3 or better, as these are not available on the PDA an alternative is needed. The open source Sphinx 2 can be used to provide speech recognition functionality on the device. Sphinx2 is written in C, therefore the Java client application will use the Java Native Interface to plug into the methods which are provided by Sphinx.

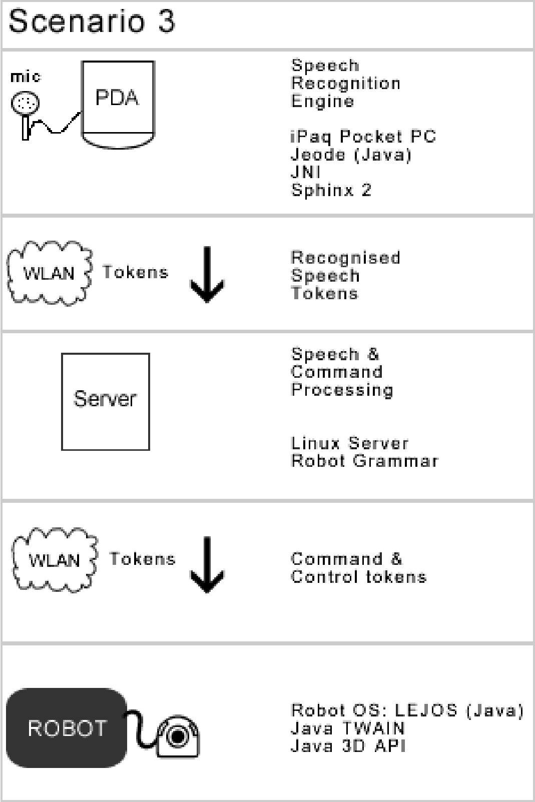

Figure 9 Scenario Three

The recognized speech will be sent over a WLAN link (TCP/IP) to the server application. As with Scenario 1 the server processes the recognized speech strings in order to determine if they match terms in the robot control grammar. If a match is found, the command is issued to the robot over a wireless network.

4. Design

4.1 Distributed Speech Recognition under Scenario 2

Scenario 2 requires a distributed speech recognition application topology. The architecture of this design is shown in figure 10.

The process begins with the client application installed on the iPaq, presently the client is written in C#, but will be converted to Java in due course. The application records and streams speech audio data from the client over the iPaqs wireless network interface to the server application.

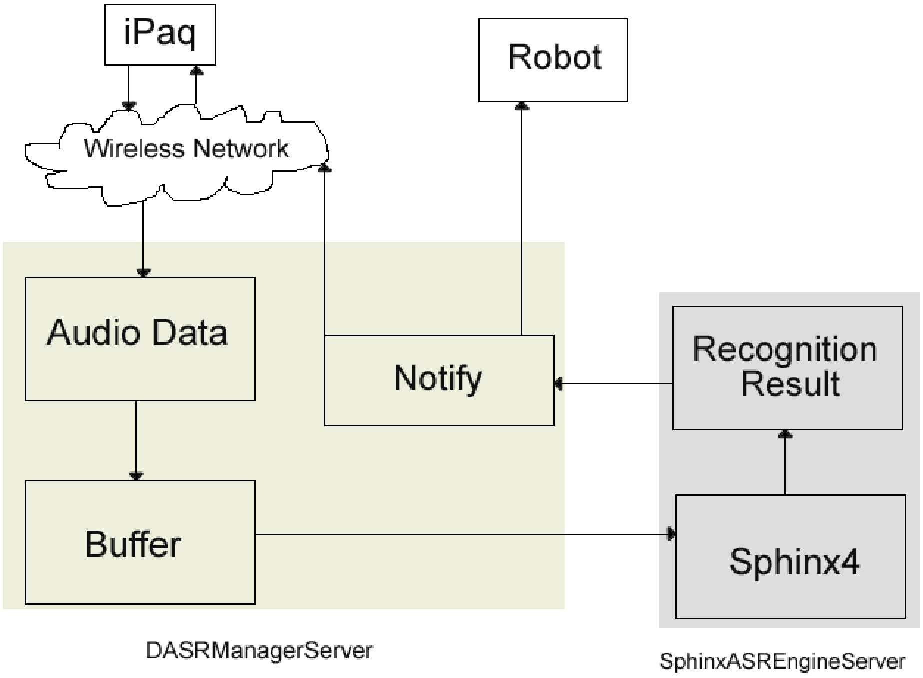

Figure 10 Distributed Speech Recognition under Scenario Two

The server application consists of two Java classes namely DASRManagerServer and SphinxASREngineServer. DASRManagerServer is responsible for receiving client connections and subsequently audio data send from the client. The class buffers the audio data before sending it to SphinxASEngineServer for speech recognition processing. The recognition result is send back to the Manager class and the notification process begins. If a command has been detected in the speech, this is sent to the robot (or other end device) a notification of the command is returned to the client.

4.2 Grammar Design

The commands appropriate to the target device are defined in a JSGF grammar file; this is used by the speech recognition engine to define what words the application can recognize.

```
#JSGF V1.0;
/**
 * JSGF Robot Grammar
 */
grammar robots;
public <forward> = forward | forward <distance>;
public <backward> = backward | reverse;
public <turn_left> = (turn left <distance>) | (left <distance>);
public <turn_right> = (turn right <distance>) | (right <distance>);
public <distance> = (five {five}) | (fifty {fifty}) | (ninety {ninety});

public interface RobotGrammarInterface {
    public int moveForward();
    public int moveBackward();
    public int moveForward(int distance);
    public int moveBackward(int distance);
    public int turnLeft();
    public int turnRight();
    public double turnLeft(int degrees);
    public double turnRight(int degrees);
    public void stop();
}
```

Code 1 JSGF Grammar and Java Interface

These commands are also built as methods into a Java interface. The interface is used by classes which wish to implement the real world functionality of a particular device.

The prototype device for this is a Lego Mindstorm [8] robot and we have devised a grammar which maps its functionality to software. The robots range of movements include forward, backward, left turn and right turn. We can enhance this further by specifying a distance or a number of degrees for the robot to move, the grammar is robust enough to cater for this. Code fragment 1 shows the JSGF grammar and the Java interface which defines the methods in code.

The implementation of the methods will vary depending on the application of the robot device, but the range of motions will be the same for the any application involving this robot.

5. Implementation Challenges

5.1 Ipaq Audio Capture

A suitable Java runtime environment must be installed on the Ipaq in order to run Java applications. The environment used in this research is Jeode, which provides JDK 1.1.8 compatibility and support for Java applets. The sand box security model implemented in Java prevents code from access to underlying system hardware, this has many security advantages but creates problems for application developers wishing to use low level device functionality, in our case access to the sound card. More recent versions of Java(1.3 and better) include the Java Sound API [9] which provides access to this functionality on more powerful desktop computers.

In order to gain access and record audio from the sound card the Java Media Framework (JMF) [10] was downloaded and a customized Java only version was built specifically for the device. The ability to customize the JMF is a relatively unknown and at this time, poorly documented feature of the software. Following numerous unsuccessfully attempts to record audio through the JMF, it was ascertained from other users that the sound functionality was not available to the JMF on the iPaq.

A pure Java solution to this problem does not currently exist. Under Scenario 2 the client is developed using C# and the Microsoft .NET Compact Framework [11] which includes low level access to the iPaq audio capabilities. The C# code is similar to Java and the runtime characteristics are identical i.e. the code is interpreted not compiled as is the case with Java.

Scenario 3 which is currently being developed, will use C++ code to capture the audio and populate a Java object with the data, this will be returned to a Java program using the Java Native Interface.

5.2 Robot Movement

The robots movement is controlled by timers which activate the motors for a specified time before stopping. In order to move the robot accurately, the timings of the motor movements must be synchronized properly. The robot moves on two caterpillar tracks each controlled by a separate motor, therefore in order to move forward both motors must be engaged in the forward direction, the same is true for reversing.

Formula:

```
       (Degrees_To_Turn / 360) * Robot 360 Time
```

Code Implementation

```
      public double turnRight(int degrees)
      {
            return (degrees / 360) * FULL_CIRCLE_TIME;
      }
```

Equation 1 Calculating the turning of the robot to left or right

Turning is more complicated, in order to turn left, the left motor must be stopped while the right motor continues to move forward and vice versa when dealing with right turns. To allow for a greater degree of control, a distance to turn can also be specified, to execute such a command the time to pause the motor must be calculated. The equation to calculate this is show equation 1.

The time it takes the robot to complete a 360 degree turn was measured and found to be a 10000ms (10 seconds), this is set as a constant value in the code.

5.3 Robot Communication Protocol

The Java control program installed on the Lego Mindstorm robot acts a server which listens for incoming data packets from its control PC. The control program simply uses the built in data port functionality to do this. In order to send commands successfully to the robot a protocol for communication needed to be designed and implemented.

Specific byte values within the RCX computer are allocated to perform certain tasks related to motor and sensor control , therefore these values needed to be avoided to ensure interoperability between the RCX and the Java control program. The byte range of values from 70 to 79 were found to be free for external use.

Binary Value	Byte Value	Function
1000110	70	Turn Right
1000111	71	Turn Left
1001000	72	Move Forward
1001001	73	Reverse

Table 1Robot Communication Protocol Functions

In the case of each function an additional value can be follow the function value, this value indicates a distance or degree parameter for the function. If this value is not present the execution continues with the default values. This allows the protocol to comply with the application grammar.

6. Conclusions and Future Work

The successful implementation of scenario one indicates that a framework for the command and control of remote devices is both feasible and practical when considerable computing power is available. The distributed speech recognition and command and control model described in scenario two is almost complete, upon completion it will be clear how the framework fits within the ubiquitous computing paradigm, however the initial results during development are positive that the distributed model of speech recognition will be succeful in the context of the overall application. Scenario two will also be developed further to incorporate a Java client to augment the current C# client implementation.

The framework can be easily extended and moulded to suit a variety of devices or applications. We propose to conduct further development of the framework utilizing additional software and hardware end devices specifically the voice activated remote control of a web camera. Incorporating additional devices will demonstrate the flexibility of the command and control framework and extensibility of the grammar.

The development scenario three will move the speech recognition even closer to the ubiquitous computing paradigm. A usable and easily programmable speech recognition engine which can run efficiently on a mobile device with limited resources will spawn endless possibilities for the development of mobile command and control applications.

7. References

1. Jurafsky D. & Martin J.H. , Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2000, Prentice Hall, New Jersey.
2. Sun Microsystems Ltd, *Java Speech API*, [online at] http://java.sun.com/products/java-media/speech/
3. Microsoft Corporation, *Microsoft Speech and SAPI 5,* [online at] http://www.microsoft.com/speech/
4. IBM, *Via Voice* [online at] http://www-306.ibm.com/software/voice/viavoice/
5. ScanSoft, *Dragon Naturally Speaking*, [online at] http://www.scansoft.com/naturallyspeaking/
6. Kinnersley J, *Cloudgarden Java Speech Api Implementation*, [online at] http://www.cloudgarden.com
7. Sun Microsystems Ltd, *Java 2 Micro Edition: Personal Java* [online at] http://java.sun.com/products/cdc/index.jsp
8. Lego Mindstorm Robots
9. Sun Microsystems, Java Sound API [online at] http://java.sun.com/products/java-media/sound/
10. Sun Microsystems, Java Media Framework [online at] http://java.sun.com/products/java-media/jmf/
11. Wigley A., Sutton M., MacLeod R., Burbidge R., Wheelwright S., Microsoft .NET Compact Framework (Core Reference), 2003, Microsoft Press.