

**SMARTPAD : A MOBILE MULTIMODAL
PRESCRIPTION FILLING SYSTEM**

A Thesis in TCC 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Bachelor in Science in Computer Science

by

Stelios E. Lambros
Computer Science

March 25, 2003

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in TCC Courses.

Signed _____

Approved _____
Technical Advisor - Alfred C. Weaver

Approved _____
TCC Advisor - Patricia C. Click

Acknowledgements

There are many people I would like to thank for their contribution to this project over the past 16 months :

My parents, for everything they've done for me.

My advisor Dr. Alfred Weaver for his enthusiasm and support.

My internship supervisor and good friend Dr. Manolis Tsangaris, for the invaluable help and guidance.

Dr. Yiannis Christou and Dr. Spiros Potamianos for their technical help and for making the office a very fun(ny) place to be.

Dr. Alex Potamianos for his constructive comments.

My TCC professors, Dr. Belanger, Dr. Johnson and Dr. Click for helping me with my writing skills and for enriching my E-School experience with interesting and entertaining *non-technical* courses.

My U.Va friends for their moral support.

Ari, for his useless but much appreciated advise.

Marilena, for providing a philosopher's view on my project.

Valeria, for her enthusiasm and encouragement.

Kathy, for being the first person to help me with testing the project.

Table of Contents

<i>Table of Contents</i>	<i>iii</i>
<i>Glossary</i>	<i>v</i>
<i>Abstract</i>	<i>vi</i>
Chapter 1 Introduction	1
1.1 Statement of Purpose	1
1.2 Problem definition and background	1
1.3 Rationale and Scope	3
1.4 Overview of the rest of the report	4
Chapter 2 Literature Review	6
2.1 Prescription Errors	6
2.2 Multimodal User Interfaces	7
2.3 Conclusion	9
Chapter 3 System Design and Implementation	11
3.1 Requirements Specification	11
3.1.1 Basic Requirements	11
3.1.2 A Typical Transaction	13
3.2 Design and Implementation	17
Chapter 4 System Testing	21
4.1 Overview	21
4.2 Objectives	21
4.3 The Testing Sheet	22
4.4 Procedure	22
4.5 Results	23

<i>Chapter 5 Conclusion</i>	24
5.1 Summary	24
5.2 Interpretation	24
5.3 Recommendations	26
5.4 Final Remarks	27
<i>References</i>	28
<i>APPENDIX A Testing Sheet</i>	29
<i>APPENDIX B TCL/TK Source Code (Client)</i>	31
<i>APPENDIX C Recognition Grammar (GSL Syntax)</i>	41
<i>APPENDIX D Wire Protocol Server Code Sample</i>	43

Glossary

User Interface (UI) : The part of a software system that deals with communicating information from and to the user.

Graphical User Interface (GUI) : A User Interface that utilizes graphics for output and graphical methods (such as a mouse) and a keyboard for input.

Multimodal User Interface (MMUI) : A User Interface that concurrently utilizes multiple methods for input and output, such as speech recognition and graphical methods.

Personal Digital Assistant (PDA): A pocket sized personal computer.

Speech/Voice Recognition: The method employed by computer software to convert spoken language into text.

Wireless Network: A computer network, in which some or all of the inter-computer connections are physically implemented with wireless technologies (radio, infra-red, etc.)

TCL/TK: Tool Command Language / TK toolkit. An interpreted programming language, with rich GUI capabilities.

C++: A commonly used, general purpose programming language. C++ code is compiled into native computer code.

Interpreted Programming Language: A programming language, in which programs are not compiled into stand-alone executables (native code) but rather require an interpreter to read and execute them.

Abstract

The purpose of this thesis project was to develop a system that would show that automating the process of prescribing medications, without compromising the convenience of the traditional notepad, is possible. The use of such a system would also reduce the occurrence of prescription errors, since the element of handwriting is removed from the prescription process. Automation is much needed in this area, as many people are injured or killed every year because of prescription interpretation errors.

The system consists of a PDA which utilizes a Multimodal Interface for entering prescriptions. The PDA connects to a server via wireless connection so that speech recognition can be employed with the help from the faster computer. The interface was designed to provide maximum usability and efficiency.

The testing of the system showed that its efficiency was satisfactory. Prescriptions were entered and verified on the PDA quickly, as the average duration of the transactions was comparable to the time of scribbling prescriptions on paper. Also, the students who tested the system all agreed that it would be an effective and convenient replacement to the notepad for the specific task of prescribing medications.

Chapter 1

Introduction

1.1 *Statement of Purpose*

The purpose of this project was to develop a system for prescribing medications that can effectively replace the physician's traditional notepad. The system must be mobile, lightweight and easy and efficient to use, while at the same time the prescriptions it produces must reduce interpretation errors (compared to scribbled prescriptions). For this project, a Personal Digital Assistant application with a Multimodal User Interface was developed. The goal was to demonstrate and test the working system, to serve as a proof of concept of the idea that effective use of mobile computing technologies can reduce deaths and injuries that arise from prescription interpretation errors.

1.2 *Problem definition and background*

This project addresses two different, yet related, problems: a) interpretation errors in medication prescriptions and b) inefficient user interfaces for mobile computing devices. While at first glance these problems seem to be unrelated, this project shows that solving the second problem, and applying the solution to the medical field, can reduce the magnitude of the first problem.

The first problem, i.e. interpretation errors in prescriptions, has caused many deaths and injuries in the past decade and the problem is getting worse every year. Physicians are still filling prescriptions using the traditional notepad. Illegible prescriptions, misinterpretations of medication dosages and strengths, increasing

similarities between drug names and other factors have caused a significant percentage of prescriptions that are delivered in pharmacies to be erroneous. This percentage is increasing every year as more and more new medications are introduced into the market. Most of these errors are potentially dangerous, and official studies have documented large numbers of deaths and injuries that have occurred because of them. The documented evidence is illustrated in Chapter 2.

The second problem this project deals with is the inefficient user interfaces that current mobile computing devices have. A User Interface (UI) is the program and/or mechanism, with which a device is operated by its user. For example, on a modern Windows PC, the UI is the desktop with its menus and windows, which can be manipulated with the mouse and the keyboard. This is called a Graphical User Interface. Each program also has its own UI that consists of different arrangements of windows, menus, textboxes etc. In other words, a User Interface is the communication method between the user and the computer. While Graphical UIs are adequately efficient for desktop and laptop computers, the same is not true for mobile devices such as Personal Digital Assistants (PDA). PDAs have screens capable of displaying graphics, and pointing mechanisms similar to mice but they lack keyboards. This configuration greatly limits the interaction abilities for the user. Various adaptations, such as handwriting recognition software and emulated on-screen keyboards, partially solve this problem but are still very slow and inefficient compared to their desktop counterparts. Therefore, PDAs are still not sufficient for many computing tasks.

In the following section, the relationship between the two problems and their solutions will be discussed.

1.3 Rationale and Scope

Why do physicians still scribble their prescriptions? While most professionals in other industries, and even students, are required to produce clean printed documents, doctors are still using their pens and notepads. Is there anything special about the profession ?

The answer is yes. Physicians are rarely sitting at their desks as they examine patients. Usually they have nowhere to sit when they are writing prescriptions and they are probably going from one room to another (this is especially true for doctors working in hospitals). This fact makes it virtually impossible for them to use a desktop computer to write prescriptions. If they are to type their prescriptions, they need something mobile. Laptop computers are mobile but bulky, and they are still inconvenient if there is no place to sit, since they require both hands to be operated. Even if there are desks in every room, laptops are still too large to carry around constantly. Another solution could be the use of smaller mobile devices, such as Personal Digital Assistants. PDAs are small enough to fit in a pocket, and they can be operated without requiring the user to sit down. However, PDAs, as discussed in the previous section, are quite slow and inefficient for data input.

This brings us back to the problem of inefficient user interfaces for mobile devices. If PDAs were just as effective as desktop and laptop computers for data input, then quite possibly doctors would be able to use them for writing prescriptions without compromising the convenience they are accustomed to with notepads. In addition, the produced prescriptions would be printed or electronically transmitted and thus much more readable and less error-prone.

For this project, a medical prescription application for PDAs was developed that utilizes a new type of user interface. The interface simultaneously accepts both speech input (using speech recognition) and conventional graphical selection methods (using the pen pointer) to allow maximum efficiency of communication between the user and the device. This interface is a type of Multimodal User Interfaces (more information on Multimodal UI's can be found in Chapter 2). The combined use of speech and graphics makes data input much faster than the standard PDA input methods. The time it takes to complete a prescription using this system is comparable to the time a doctor spends to scribble one. Consequently, this application can be much more useful for physicians to fill prescriptions.

1.4 Overview of the rest of the report

Chapter 2 presents information from the relevant literature. The evolution of User Interfaces as well as the current research in the field of Multimodal User Interfaces is discussed. Also, documented data on the number of deaths and injuries due to prescription interpretation errors as well as other relevant information is presented.

Chapter 3 focuses on the Design and Implementation of the system. The system is much more complex than a single isolated application for the PDA, since some functionality is handled by a faster desktop computer, which communicates with the PDA via wireless connection.

Chapter 4 explains the testing procedure that followed the completion of the system and presents the results.

Chapter 5 concludes this report, with an interpretation of the testing results and a discussion of the potential of this project.



Chapter 2 Literature Review

2.1 Prescription Errors

Countless advancements in Pharmacology during the last century have produced a vast amount of medications that successfully fight or cure many common illnesses or rare diseases. The great number of medications available today and the variety of side effects they cause has led authorities to regulate their distribution, in order to prevent unnecessary injuries or deaths, since most drugs can be dangerous if not taken properly. Some people might complain about having to visit the doctor every time they need medication for some common non-threatening illness. However, only a physician is capable of examining his or her patient's physical condition, reviewing the medical history and assessing the seriousness of his or her condition, in order to determine which medication is right and how much of it will not cause harm.

Yet, even though the process of obtaining medications has been successfully regulated, there is still a substantial problem in people taking the wrong medications.

The following facts illustrate the problem* :

- Medication errors in the U.S. result in about 7,000 deaths per year. [1]
- Outpatient deaths from prescription errors increased dramatically (about 800%) from 1983-1993. [1]

* These facts have been taken from Dr. Alfred Weaver's research proposal to Microsoft "*Reducing Prescription Errors via Mobile Technologies*". The original sources can be found in the bibliography section.

- Massachusetts estimates that 2.4 million prescriptions are filled improperly annually and that 88% of them could lead to errors [2]
- Massachusetts found in the same study, that 26% of the errors were due to illegible prescriptions, 29% due to similar-looking drug names, and 3% due to misinterpreting abbreviations. [2]
- A national study found that 12.5% of all outpatient prescriptions contained errors, and 1.6% were considered potentially dangerous. [3]
- The cost of prescription errors is as much as \$2 billion annually. [1]

In light of this evidence, it can be argued that the fact that almost all prescriptions are handwritten is one of the leading causes of errors. Pharmacists must process a large number of prescriptions every day and, given the ambiguity of handwritten text, they sometimes do not read them correctly. Many drug names are similar, and some that are spelled quite differently, look similar when handwritten. This problem gets worse every year because more and more drugs are introduced. It is therefore essential to remove the element of handwriting from the process of prescribing medications, in order to solve this problem.

2.2 Multimodal User Interfaces

A small screen, a pen, which exploits the screen's touch-sensitivity and a microphone characterize the PDA that will be used. Gibbon, Mertins and Moore state that, even though there is no clear agreement on the definition, Multimodal interfaces are defined by most as those that have the following features:

- “1.) The user communicates with the computer using several physical input devices (mouse, microphone etc.)
- 2.) In order to achieve this communication, several muscles are activated by the user (e.g. vocal cords, hand)
- 3.) The information sensed by the computer input devices can be processed at different levels of abstraction, providing different levels of understanding of the intention of the user. [...]” [5:103]

It is therefore clear that a user interface, such as the one in this projects application, which utilizes various different input devices, is a multimodal interface. Now, lets examine the benefits of multimodal interfaces, and whether these benefits apply to the project.

Landay says that Multimodal Interfaces can enable computers to be used in more situations, such as when the users hands are full, and in more places (e.g. walking) [6]. Both cases apply to the working condition of the typical doctor. Physicians are frequently on the move as they visit their patients. They need something mobile for prescription taking. Also, when on the move, one should be able to use the mobile computer without needing “extra” hands. In other words, a computer with a traditional keyboard/mouse interface requires both hands to be used. On the other hand, this project’s multimodal mobile device requires only one hand to be operated, leaving the other hand available for holding the device itself.

Other similar solutions to the prescription misinterpretation problem exist but they do not address the mobility issue. According to MD Net Guide, there are at least three commercial products that allow physicians to verbally communicate medical

information to their desktop computers. Dragon's NaturallySpeaking, IBM's ViaVoice and L&H's Voice Xpress for Medicine all support dictation of medical documents, such as prescriptions. The website even claims that these products can reach accuracy rates of 99% [7]. While this number is impressive, voice recognition accuracy alone is not enough to make these systems productive. Since they require desktop or laptop computers, it is neither convenient nor efficient for most physicians to write their prescriptions on this type of computer, as explained in Chapter One. However, these products show the maturity of voice-recognition technology today and this project depends heavily of this technology.

If the project's system reaches accuracy rates anywhere near that of these products, it will be a big success. The application's interface, being of the multimodal type, will be designed to exploit multiple input devices to provide maximum usability and efficiency. The user will be able to correct any error in the recognition result of the spoken prescription with two to five "pen-clicks". Therefore, even a recognition accuracy rate of only 80 or 90 percent should not be a problem, since errors can be corrected quickly. Only major recognition errors, i.e. when more than two of the individual elements of the prescription (drug name, dosage, frequency, duration) are flawed, would require the user to speak the whole prescription again, since in this case, repeating it is faster than correcting it manually.

2.3 Conclusion

To summarize, the relevant literature shows the extent of the prescription error problem and provides some guidelines for a possible solution. The idea of a wireless

device utilizing a Multimodal Interface to be used for prescribing medications seems to be a good one, since the benefits of such a system are evidently suitable for the working environment of a typical physician. Also, the documented maturity of voice-recognition technologies provides an encouraging indication of the possible success such a project if it were to be commercially developed.



Chapter 3

System Design and Implementation

3.1 Requirements Specification

Most projects start off with an idea. However, an idea is not enough to describe the final product's operation. A comprehensive description of the system's external behavior must be derived from the potential user's needs and be thoroughly documented. The external behavior is simply the behavior of the final product that its user will experience. The internals of the system are not described in this phase. In the software engineering process, this is the phase of Requirements Analysis and Specification. While most large projects require extremely detailed and exhaustive specifications, for this project the specifications were kept simple. This project will not be delivered to anyone, it is meant to be a proof of concept and therefore rigorous specifications would be superfluous. That is not to say that the Requirements Analysis phase was taken lightly. There was a lengthy period of communication between my technical advisor Dr. Weaver, Dr. Tsangaris (my internship supervisor), some physicians in the field of Telemedicine and myself before a single line of code was written.

3.1.1 Basic Requirements

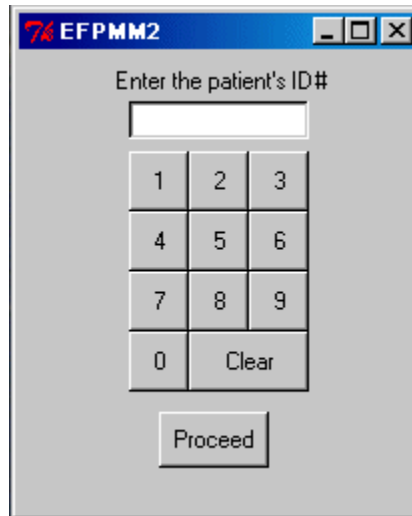
As mentioned in the Introduction, a computing device must be easy to use and efficient in order for physicians to use it as they work. This has been the main idea in the discussions with the physicians. The Multimodal Interface of the client software is characterized by various input methods which make each transaction as fast as possible.

One of the doctors mentioned that speech recognition accuracy is very important. He specifically said “if the recognition is below 90 percent, we turn into highly paid secretaries”. Recognition mistakes are expected, in fact they’re unavoidable. However, the system has been designed to enable very fast corrections, using only the pen pointer. For example, a wrong medication dosage can be corrected in as little as two clicks. If there are many mistakes in the prescription, the user can repeat it verbally instead of using graphical methods. While the speech recognition engine alone in this project can provide very good accuracy, these alternative methods of input reduce the impact of recognition errors, making the overall usability much better.

Also, probably the most important requirement for the system is to provide clear and **correct** prescriptions at the end of each transaction. For this reason, a verification step is essential. After the doctor has completed the prescription, the software summarizes all the information and waits for the physician to verify it before it is finalized.

3.1.2 A Typical Transaction

To further illustrate the operation of the system, a description of a typical transaction follows :



The image shows a software window titled "EFPMM2" with a blue header bar. Inside the window, the text "Enter the patient's ID#" is centered above a white text input field. Below the input field is a numerical keypad consisting of a 3x3 grid of buttons for digits 1 through 9, a button for 0, and a "Clear" button. Below the keypad is a "Proceed" button.

Figure 3.1 - Identification Form

A prescription filling transaction starts with the identification form. Obviously, the doctor must specify the client to which the prescription belongs. As shown in Figure 3.1, a simple identification number form has been used in this project. A numerical keypad has been included to eliminate the need for a separate keyboard. Once the number has been typed, the user clicks on the “Proceed” button to continue to the prescription form.

The image shows a software window titled "EFPMM2" with a blue header bar. Below the header, the text "Patient Number:" is displayed. The form contains four input fields: "Drug Name" and "Strength" are on the first row, and "Frequency" and "Duration" are on the second row. At the bottom of the form, there are four buttons arranged in two rows: "Start Speaking" and "Connect" in the first row, and "<< Back" and "Proceed >>" in the second row.

Figure 3.2 - Main Prescription Filling Form

The prescription filling form is the central part of the application. This part utilizes a full Multimodal Interface, as shown in Figure 3.2. There are two ways to input information, which can be simultaneously used. The first method is to speak the prescription after clicking on the “Start Speaking” button. Shortly after the speech has ended, all recognized information is inserted in the respective fields.

The second method is to use the pen pointer to graphically select the information in each field. The following figure illustrates this procedure.

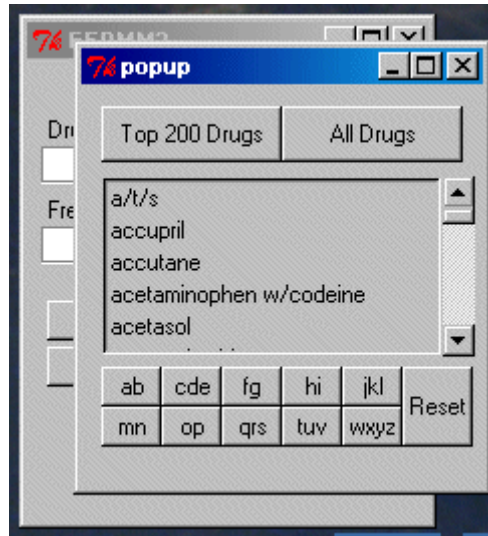


Figure 3.3 - Graphical Selection Methods for individual parts of prescription

In Figure 3.3 we can see the new window that appears when the user clicks on one of the fields. In this example, the “Drug” field was clicked. The goal is to make these selections as fast as possible using only the pointer. The “Drug” field is the most difficult one because of the immense number of available medications. In the top we see two buttons, “Top 200 Drugs” and “All Drugs”. If the physician knows that the drug he or she wants is a popular one, the list can be significantly reduced by clicking the first button. Otherwise, the entire list of drugs can be shown by clicking the second button.

After the list has been populated, even if the Top 200 Drugs were selected, it is still too large to efficiently find and select the desired drug. A special “T9” interface¹ (the bottom buttons) can be used to limit the number of displayed drugs. Each time a T9 button is clicked the list is dramatically reduced. An example of this is the best way to understand its operation :

¹ The name “T9” is taken from the interface that most cell phones use for entering text with minimal keystrokes. The operation of this interface is very similar to that of cell phones.

To select the drug Vioxx, the user can click the T9 buttons corresponding to the first letters of the drug name. In this case the user first clicks on the “TUV” button. This action cuts down the list, leaving only the drugs whose first letter is ‘T’, ‘U’ or ‘V’. A second click on the button “HI” will cut down the list even more, leaving the drugs whose first letter is ‘T’, ‘U’ or ‘V’ and whose second letter is ‘H’ or ‘I’. Clearly, there are a few drugs that can fit that description but what is important is that the list itself contains Vioxx and that it is small enough for the user to easily find the drug. Simple tests of this procedure have shown that with two to three button clicks, the list can be reduced to less than 10 elements.

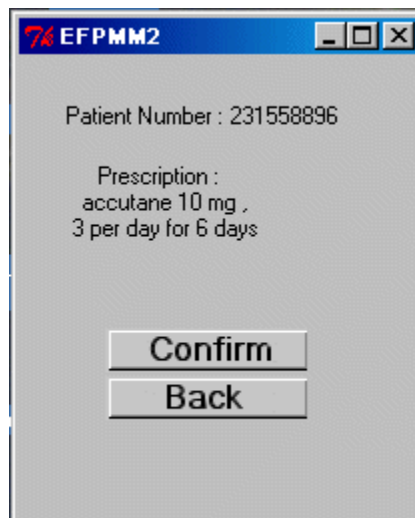


Figure 3.4 - Verification

Finally, once the prescription is complete, the user clicks the “Proceed” button and is taken to the verification step, as shown in Figure 3.4. The prescription is summarized and the user must verify its correctness by clicking on “Confirm”. Otherwise, the user can click on “Back” and correct the prescription.

3.2 Design and Implementation

The design is a detailed description of the internals of a system. For the purposes of this report only a high level description will be presented, as well as some implementation details. More details and the source code can be found in the appendix.

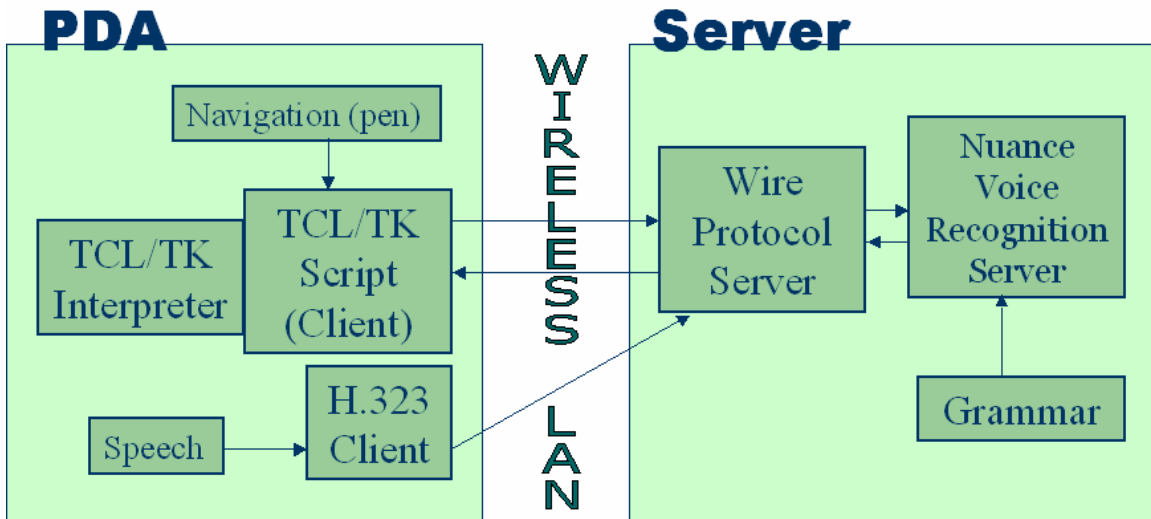


Figure 3.5 - System Architecture

The hardware that was used for this project was a Compaq iPaq PDA, a Windows 2000 desktop computer and a wireless network card for the PDA. The individual software components of the system as well as the flow of data between them is illustrated in the diagram in Figure 3.5. A description of each component follows:

TCL/TK Script and Interpreter

TCL/TK was chosen as the GUI programming language for the client program because of its portability and its reasonable learning curve. Both of these factors were important.

Portability was essential because it allowed the coding to be done on any desktop

computer with TCL/TK support and then the code could be transferred and executed on the PDA without any changes. The language's simplicity was a great time-saver, as only after a couple days of studying, the first throw-away prototypes were being produced. TCL/TK is an interpreted language, its code is not compiled into machine code but it is read and executed by a program called the interpreter. The interpreter must be present on a computer for a TCL/TK script to run on it. A stable version of a TCL/TK interpreter for the PDA was found online, and worked successfully for this project.

H.323 Client

H.323 is a protocol that defines methods for real time audio-visual communication over computer networks. The most famous example of a program that implements H.323 is Microsoft's Netmeeting. As discussed previously, in order for the client to support speech recognition, the recognition must be handled by stronger desktop computer. The H.323 Client is the program that runs on the PDA and is responsible for transferring the audio input to the desktop computer via the wireless connection. For this project, SJPhone was used, as it was the only available H.323 client that was compatible with the PDA. The H.323 Client runs concurrently with the TCL/TK script.

Nuance Voice Recognition Server

Nuance 8 was used for the voice recognition purposes. Its Voice Recognition Server was the main tool that was needed. The Server processes audio input and produces a text interpretation of the speech it recognizes. In order for it to operate, two components are

required, a ‘Grammar’ and an implementation of Nuance’s Application Program Interface (API).

Grammar

The Grammar is a precise description of what kind of utterances the recognition server should expect to “hear” and how to interpret them. Nuance 8 supports various grammar languages and for this project the Nuance Grammar Specification Language (GSL) was used.

The Grammar was designed to specify each field of a prescription (drug name, strength, dosage and frequency) independently as well as acceptable combinations of these fields. Also, the Grammar specifies the expected utterance of each element by describing its pronunciation, however the returned interpretation is slightly different so that abbreviations and other non-pronounceable parts of the textual form of a prescription can be supported.

To illustrate the differences between recognition and interpretation, a typical example is explained: For the prescription “Beconase AQ, 30 mg, 3 per day for 10 days” the Grammar specifies that the recognition server should expect to hear the utterance “beconase a q thirty milligrams three per day for ten days” (that is how it is pronounced). Once it hears it, it then returns the *interpretation* , that is “Beconase AQ” is the drug name, “30 mg” is the strength, “3 per day” is the dosage and “10 days” is the frequency.

Wire Protocol Server (WPS)

The Wire Protocol Server is the implementation of the Nuance API. It was written in C++. Its purpose is to control the recognition server and to forward data from and to the server. The WPS is the central controlling component that brings the rest of the components together. For the purposes of this project, the WPS specifies which grammar the recognition server must use, it establishes a connection with the TCL/TK client to accept commands and send recognition results, and it connects to the H.323 client to receive the audio stream and forward it to the recognition server.

Source code samples the components described above can be found in the appendices.

The implementation of the system was successful, even though many technical issues came up in the process. For example, shareware licenses expired, the wireless card drivers were incompatible with the University's network etc. However, these issues were dealt with in a timely manner.

Chapter 4 System Testing

4.1 Overview

In order to verify the usability of the system, some formal testing was performed. The testing was done with UVA students, where each of them was given a testing sheet with a list of prescriptions and they were asked to write them using the system. Each time a prescription was entered, the time it took to complete it was measured and recorded, and the relative accuracy of the voice recognition was marked on the testing sheet. Also, when the testers were done entering all of the prescriptions on the sheet, they were asked how convenient they found the system compared to using a notepad.

4.2 Objectives

Usability cannot be measured precisely. However, by measuring the time to complete a typical prescription on this system, the User Interface's efficiency could be compared to the traditional notepad. If the system was too slow and prescription writing took much longer on the PDA than it would to scribble it on paper, then the efficiency of the system would have been negative.

Also, asking the testers to judge the system's usability after they have used it, allowed a more diverse opinion to be formed, since the people that tested the system were mostly students in non-computer related fields.

4.3 The Testing Sheet

The Testing Sheet consisted of 30 prescriptions. Each prescription was of the form “*drug name, strength, dosage, frequency*”. This was all the information that the testers would need to specify the prescription. Under each prescription two metrics were listed to measure the system’s performance. The first one was the recognition accuracy, which could be marked as either “Poor”, “Partial” or “Correct”. The second one was the “Time to complete” entry, where the measured time to complete the prescription was recorded. For example, here is one of the typical entries in the testing sheet :

Amoxicillin 500mg, 2 times a day, for 4 days
Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

The complete Testing Sheet can be found in the Appendix.

4.4 Procedure

Four students agreed to test the system. Only one of them was a Computer Science major. Each tester was given a brief tutorial on how to use the system and then they were given at most 15 minutes to practice and get used to it. After that, they were instructed to enter the prescriptions from the Testing Sheet. The test conductor measured and noted the time for each prescription to be completed and ranked the recognition results based on the following rule: if one or two fields of the prescription were wrong after the speech recognition, it was marked “Partial”. If more than two were wrong, it was marked “Poor”.

A “Correct” mark was given if all fields were correct. In addition, if the recognition was marked “Partial” or “Poor”, the conductor also noted which fields were wrong.

Once the testers completed the whole testing sheet, they were asked to comment on the usability of the system. Specifically, they were asked if they found the system convenient enough to replace a notepad for the specific task of prescribing medications. Their comments were noted on their respective sheets.

4.5 Results

The results of the tests did not show any unexpected behavior on the system. The average time to complete a prescription was 13.4 seconds. This average includes all of the prescriptions, whether manual corrections were needed or not.

The average time to complete the prescriptions that did not require corrections, that is the prescriptions with “Correct” initial recognitions, was 7.1 seconds. The average time for the “Partial” prescriptions was 18.75 seconds. Surprisingly, there were no “Poor” initial recognitions, all prescriptions had two or less errors.

The major deviations from the main average occurred when the medication’s name needed correction. The average time for this type of prescription was 24 seconds. Also, the prescriptions that did not need correction of the medication name field, had a significantly lower-than-average time (10 seconds). It is important to mention, that the testing procedure did not include entering the patient identification number.

All of the testers agreed that the system was easy to use and that they thought that it was convenient enough to replace the notepad for the specific task of prescribing medications.

Chapter 5 Conclusion

5.1 Summary

This project started off as an idea and evolved into a working system, which was used as a proof of concept for that idea, that is it showed that physicians can effectively use mobile devices to prescribe medications more safely. The need for automation in this regular medical process is becoming greater since more and more people are getting hurt or are dying every year because of prescription interpretation errors. The goal was to design a system that would be easy to use and just as fast as using a notepad to scribble prescriptions. To achieve this goal, a Multimodal User Interface was designed and implemented, which maximized the system's usability, given the constraints of the PDA. In order to limit the development costs and to reduce the required workload to manageable levels, alternative tools were used, such as the TCL/TK programming language, and the features of the system were limited only to those needed to demonstrate the main idea. The construction of the system was mostly successful and the subsequent testing gave overall positive results.

5.2 Interpretation

The question that must now be asked is "Would doctors use such a mobile system and if yes, will its use reduce prescription errors?" To answer this question, an examination of the testing results is required.

First, let's look at the prospect of physicians being able and willing to use such a system. As discussed in Chapter 4, all of the testers agreed that the User Interface was easy and intuitive enough to learn how to use, and that overall it was convenient. Also, judging by the quantitative results, the average prescription was entered and verified in under than 14 seconds. Even though the average time for a person to scribble a prescription on a notepad was not measured, it is safe to say that the time to enter a prescription in the PDA is comparable (if not less) than the time to handwrite it on paper. It is important to mention that this measured average includes the prescriptions that needed manual corrections after the speech recognition. Also, another important aspect of these results is that the speed of the client program itself was responsible for the greater part of the delays in the manual corrections. Specifically, whenever the medication name needed manual correction, it would take a few seconds for the interface to load the entire list of medications and each list modification took some time as well. These delays would be eliminated if a compiled program was used instead of an interpreted one. This would theoretically improve the overall average, since the average time of the prescriptions that did not require correcting the medication name was 10 seconds and the average time of the prescriptions that did was 24 seconds.

Therefore, we can say that physicians should not have any trouble using this system since its easy to learn and use, and writing prescriptions with it is just as fast and convenient as scribbling them on a notepad.

The next thing we need to consider is if this system can reduce prescription errors. It is obvious that printed or electronically transmitted prescriptions are much easier to read than handwritten ones. However, this does not mean that using this system will

automatically solve the prescription error problem. Even if the prescription is printed, it could be erroneous. It is up to the physicians to use the software properly in order to ensure that they enter exactly what they intend to prescribe. The client software makes error correction very easy and tries to enforce the correctness of the final prescriptions with the verification step. If doctors ignore the verification step by simply clicking on the “Proceed” button without reviewing their prescriptions, it is quite likely that an erroneous prescription will eventually be printed. Even so, such errors should be quite less than the errors that occur with scribbled prescriptions because the doctors would be constantly looking at the prescription as they are enter and correct it through the main Multimodal Interface. We cannot be absolutely sure about this but the double redundancy (entering → verifying → printing) should make errors more likely to be caught. While it would be great to eliminate all of them, any significant decrease of prescription errors is welcome.

5.3 Recommendations

This project has great potential. The results illustrate that there should be no major problems in using this alternative method for prescribing medications. As a first step, future work on this platform should start off with some more testing, particularly with real doctors. The results were not negative but the testing would have been much more complete if physicians had tested it and given their opinion.

The possible improvements and enhancements are countless. The basic improvements should include implementing the feature of printing or transmitting of the prescriptions, as well as enabling the identification interface to retrieve the patients’ personal information. As discussed earlier, the client program should be implemented in

a language that can be compiled in the PDA's native code so that performance will not be an issue. The client program can also be linked to a database of patients' medical history so it can automatically warn the doctor about allergies and/or possible drug interactions. Another useful feature would be to make the server able to automatically update its list of medications by retrieving new medications from a disk or from the Internet.

This project can also become a part of larger medical applications. There are many commercial products for desktop computers in the market that help physicians with transcribing, maintaining medical records, looking up medication information and more. Adding Multimodal features to these programs can enable more functions on them and can make them much more usable so their transition to PDAs can be easier.

The long-term benefits of using such a system are important. Less prescription errors means less injuries and deaths. In general, the use of automated methods in the medical field can increase efficiency and therefore drive down medical costs.

5.4 Final Remarks

The goal of this project was to prove an idea, and to that extent it has been successful. Hopefully, the project's contribution will not stop there. It is now up to others to evaluate the results and decide if they should continue the research in this field by improving the system, using the ideas behind this system to develop their own, or by developing a commercially viable implementation of such a product. I believe that this project can be beneficial to both the medical field and the field of Computer Science.

References

- 1.) Institute of Medicine, “To Err Is Human: Building A Safer Health System”
- 2.) State of Massachusetts, Medication Error Study
(Summary at <http://www.state.ma.us/reg/boards/ph/phstudy/04phres.htm>)
- 3.) American Journal of Hospital Pharmacists, National Study
(Summary at http://www.nypirg.org/health/prescription_errors.html)
- 4.) Tan, Shi and Gao, “Advances in Multimodal Interfaces – ICMI 2000”
- 5.) Gibbon, Mertins and Moore, “Handbook of Multimodal and Spoken Dialogue Systems”
- 6.) Landay, James, “Invisible Computing Activities”
(<http://www.cs.washington.edu/mssi/tic/intros/Landay>)
- 7.) MD Net Guide, “Voice-Recognition Software: Deleting the Scribbled Prescription From Your Practice” (<http://www.mdnetguide.com/articles.shtml?issue=21&dept=4>)
- 8.) Bunt, Beun and Borghuis, “Multimodal Human-Computer Communication”
- 9.) Beasley, Farley, O’Reilly and Squire, “Voice Application Development with VoiceXML”

APPENDIX A Testing Sheet

Lipitor 20mg, 2 times a day, for 10 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Premarin 1.25mg, 3 times a day, for 15 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Synthroid 25mcg, 3 times a day, for 12 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Hydrocodone with APAP 30mg, 5 times a day, for 7 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Norvasc 5mg, 4 times a day, for 5 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Prilosec 20mg, 1 times a day, for 6 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Claritin 10mg, 2 times a day, for 10 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Glucophage 500mg, 2 times a day, for 10 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Celebrex 100mg, 5 times a day, for 14 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Prevacid 15mg, 4 times a day, for 7 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Zoloft 100mg, 1 times a day, for 10 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Albuterol, 1 times a day, for 14 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Paxil 20mg, 5 times a day, for 20 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Vioxx 25mg, 3 times a day, for 10 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Amoxicillin 500mg, 2 times a day, for 4 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Zocor 20mg, 3 times a day, for 8 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Zithromax Z-Pak, 2 times a day, for 10 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Prozac 10mg, 4 times a day, for 12 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Zesril 5mg, 2 times a day, for 3 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Zyrtec 5mg, 1 times a day, for 10 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Lanoxin 0.25mg, 1 times a day, for 11 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Celexa 20mg, 2 times a day, for 7 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Neurontin 300mg, 3 times a day, for 4 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Viagra 50mg, 2 times a day, for 20 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Glucotrol XL, 2 times a day, for 15 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Nasonex 30 mg, 4 times a day, for 10 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Allegra-D 60mg, 2 times a day, for 4 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

Alprazolam 30mg, 4 times a day, for 10 days

Initial Recognition: Poor ____ Partial ____ Correct ____ / Time to Complete : ____

APPENDIX B

TCL/TK Source Code (Client)

```
#!/usr/local/bin/tclsh7.5

# Receives data from socket and processes it
proc read_sock {sock} {

    set l [gets $sock]
    parseAndDisplay $l
}

# Sends 'string' to receiving end of socket
proc send {sock string} {

    global eventLoop
    puts $sock $string          ;# send the data to the server
}

# Takes two windows and "centers" them by equating their centers
proc PositionWindows {parent child} {

    update

    set x [wininfo x $parent]
    set y [wininfo y $parent]
    set x1 [wininfo width $parent]
    set y1 [wininfo height $parent]

    set w1 [wininfo width $child]
    set z1 [wininfo height $child]

    set w [expr $x + ($x1 - $w1)/2]
    set z [expr $y + ($y1 - $z1)/2]

    wm geometry $child +$w+$z
    wm withdraw $parent
}

# Takes an interpretation string, parses it and displays the data
# in each corresponding field
proc parseAndDisplay {recResult} {

    global drug
    global dosage
    global freq
    global duration

    set dummy ""
    set _drugname ""
    set _units ""
    set _amount ""
    set _duration ""
    set _freq ""

    # Example of interpretation format : "{<drug amoxicillin:dr> <units
mg:un>}"
}
```

```

regexp {<drug (.+):dr>} $recResult dummy _drugname
regexp {<units (.+):un>} $recResult dummy _units
regexp {<amount (.+):am>} $recResult dummy _amount
regexp {<duration (.+):du>} $recResult dummy _duration
regexp {<freq (.+):fr>} $recResult dummy _freq

if { $_drugname != "" } { set drug "$_drugname" }
if { $_amount != "" } { set dosage "$_amount$_units" }
if { $_freq != "" } { set freq "$_freq" }
if { $_duration != "" } { set duration "$_duration" }

}

proc establishConnection {host port} {
    global Sock
    set Sock [socket $host $port]
    fileevent $Sock readable [list read_sock $Sock]
    fconfigure $Sock -buffering none
}

# Initializes the main window (prescription form)
proc DisplayPrescriptionInterface {idnum} {

    label .drugnamelabel -text "Drug Name"
    label .patient -text "Patient Number: $idnum"

    label .strengthlabel -text "Strength"
    label .freqlabel -text "Frequency"
    label .durationlabel -text "Duration"

    entry .drug -textvariable drug -width 22 -relief
groove
    entry .dosage -textvariable dosage -width
7 -relief groove
    entry .freq -textvariable freq -width 14 -relief
groove
    entry .duration -textvariable duration -width 15 -
relief groove

    button .recognition -text "Start Speaking" -command
{send $Sock "rec.recognize"}
    button .review -text "Proceed >>" -
command {
set
temp "$drug $dosage ,\n $freq $duration"

destroy .patient .drugnamelabel .drug .strengthlabel .dosage \
.freq .freqlabel .durationlabel .duration .recognition .connect .back \
.review
review
$temp
}

    button .connect -text "Connect" -command
{establishConnection $host $port
after
500
}

    button .back -text "<< Back" -command {back}
}

```

```

place .patient                -x 30 -y 5
place .drugnamelabel         -x 7 -y 25

place .drug                   -x 7 -y 43

place .dosage                 -x 146 -y 43
place .strengthlabel         -x 146 -y 25

place .freq                   -x 7 -y 83
place .freqlabel             -x 7 -y 65

place .duration               -x 98 -y 83
place .durationlabel         -x 98 -y 65

place .recognition           -x 10 -y 120 -width 100 -height 20
place .review                 -x 110 -y 143 -width 80 -height
20
place .connect                -x 110 -y 120 -width 80 -height
20
place .back                   -x 10 -y 143 -width 100 -height 20

# Bind clicks on fields (entries) with corresponding manual selection
interfaces

bind .drug <Button-1> {
    DisplayManualSelectionInterface_drug2
}

bind .dosage <Button-1> {
    DisplayManualSelectionInterface_strength
}

bind .freq <Button-1> {
    DisplayManualSelectionInterface_freq
}

bind .duration <Button-1> {
    DisplayManualSelectionInterface_duration
}

}

# v.2 of Manual drug selection interface - Implements a T9 style keyboard
proc DisplayManualSelectionInterface_drug2 {} {

    global t9pattern
    set t9pattern ""

    if {1 == [wininfo exists .popup]} {destroy .popup}
    toplevel .popup
    #PositionWindows . .popup

    global drug_manual
    set drug_manual ""
    listbox .popup.drugslist -yscrollcommand ".popup.scroll1 set"
    scrollbar .popup.scroll1 -command ".popup.drugslist yview"

```

```

button          .popup.top200      -text "Top 200 Drugs"
button          .popup.all        -text "All Drugs"

label           .popup.selectdrug -text "Select Drug"

place           .popup.top200     -x 10 -y 10 -width 90
place           .popup.all        -x 100 -y 10 -width 90

place           .popup.drugslist   -x 10 -y 45 -width 170 -height 90
place           .popup.scroll1    -x 180 -y 45 -height 90

button          .popup.t9_1       -text "ab" -command
{UpdateList "[a b\]}
button          .popup.t9_2       -text "cde" -command
{UpdateList "[c d e\]}
button          .popup.t9_3       -text "fg" -command
{UpdateList "[f g\]}
button          .popup.t9_4       -text "hi" -command
{UpdateList "[h i\]}
button          .popup.t9_5       -text "jkl" -command
{UpdateList "[j k l\]}
button          .popup.t9_6       -text "mn" -command
{UpdateList "[m n\]}
button          .popup.t9_7       -text "op" -command
{UpdateList "[o p\]}
button          .popup.t9_8       -text "qrs" -command
{UpdateList "[q r s\]}
button          .popup.t9_9       -text "tuv" -command
{UpdateList "[t u v\]}
button          .popup.t9_10      -text "wxyz" -command
{UpdateList "[w x y z\]}
button          .popup.resetT9    -text "Reset" -command
{UpdateList "-reset-"}

place           .popup.t9_1       -x 10 -y 140 -width 30 -height 20
place           .popup.t9_2       -x 40 -y 140 -width 30 -height 20
place           .popup.t9_3       -x 70 -y 140 -width 30 -height 20
place           .popup.t9_4       -x 100 -y 140 -width 30 -height 20
place           .popup.t9_5       -x 130 -y 140 -width 30 -height 20
place           .popup.t9_6       -x 10 -y 160 -width 30 -height 20
place           .popup.t9_7       -x 40 -y 160 -width 30 -height 20
place           .popup.t9_8       -x 70 -y 160 -width 30 -height 20
place           .popup.t9_9       -x 100 -y 160 -width 30 -height 20
place           .popup.t9_10      -x 130 -y 160 -width 30 -height 20
place           .popup.resetT9    -x 160 -y 140 -width 35 -height

```

40

```

bind .popup.all <Button-1> {
    global DRUGS
    update
    update idletask
    set f [open dnames.txt]
    set DRUGS ""
    set i 0
    while {[gets $f line] >= 0} {
        incr i
        set line [string trim [string tolower $line]]
        lappend DRUGS $line
        .popup.drugslist insert end $line
        if {$i == 100} {update idletask; update; set i 0}
    }
}

```

```

        }
        close $f
    }

    bind .popup.drugslist <ButtonRelease> {
        set drug [.popup.drugslist get [.popup.drugslist curselection] ];
        wm withdraw .popup
        wm deiconify .
    }

}
# Used by DisplayManualSelectionInterface_drug2() to update the list given a
new regexp
proc UpdateList { pattern } {

    global DRUGS
    global .popup.drugslist
    global t9pattern

    if {$t9pattern == ""} {set t9pattern "^"}

    set t9pattern "$t9pattern$pattern"
    if {$pattern=="-reset-"} {set t9pattern ""}

    .popup.drugslist delete 0 end

    set i 0

    foreach dr $DRUGS {
        incr i
        if { $i == 10 } {update idletask; update; set i 10}
        if {[regexp "$t9pattern" $dr] } {
            .popup.drugslist insert end $dr
        }
    }
}

}

proc back {} {

destroy .drugnamelabel .drug .strengthlabel .dosage .freqlabel .freq .duration
.recognition .review .connect
destroy .back .manual .patient .durationlabel
DisplayIdentificationInterface

}

proc DisplayManualSelectionInterface_drug {} {

    if {1 == [winfo exists .popup]} {destroy .popup}
    toplevel .popup
    #PositionWindows . .popup

    global drug_manual
    set drug_manual ""
    listbox .popup.drugslist -yscrollcommand ".popup.scroll1 set"
    scrollbar .popup.scroll1 -command ".popup.drugslist yview"

```

```

button          .popup.top200          -text "Top 200 Drugs"
button          .popup.all            -text "All Drugs"

label          .popup.selectdrug      -text "Select Drug"
entry          .popup.firstletters    -width 5          -textvariable
letters
label          .popup.letterslabel     -text "Enter first few letters : "

place          .popup.top200          -x 10 -y 10      -width 90
place          .popup.all             -x 100 -y 10     -width 90
place          .popup.letterslabel     -x 10 -y 50
place          .popup.firstletters     -x 130 -y 50

place          .popup.drugslist        -x 10 -y 75      -width 170 -height 90
place          .popup.scroll1         -x 180 -y 75     -height 90

bind .popup.all <Button-1> {
    global DRUGS
    update
    update idletask
    set f [open dnames.txt]
    set DRUGS ""
    set i 0
    while {[gets $f line] >= 0} {
        incr i
        set line [string trim [string tolower $line]]
        lappend DRUGS $line
        .popup.drugslist insert end $line
        if {$i == 100} {update idletask; update; set i 0}
    }
    close $f
}

bind .popup.firstletters <KeyRelease> {
    global letters
    global DRUGS

    update
    update idletask

    .popup.drugslist delete 0 end
    set letters [string trim $letters]

    set i 0

    if { $letters == "" } {
        foreach dr $DRUGS {
            incr i
            .popup.drugslist insert end $dr
            if { $i == 100 } {update idletask; update; set i 100}
        }
    } else {
        set i 0

        foreach dr $DRUGS {
            incr i
            if { $i == 30 } {update idletask; update; set i 30}
            if {[regexp "^$letters" $dr]} {
                .popup.drugslist insert end $dr
            }
        }
    }
}

```

```

        }
    }
}

bind .popup.drugslist <ButtonRelease> {
    set drug [.popup.drugslist get [.popup.drugslist curselection] ];
    wm withdraw .popup
    wm deiconify .
}

}

# Manual Selection Interface for "strength" field
proc DisplayManualSelectionInterface_strength {} {

    if {1 == [wininfo exists .popup]} {destroy .popup}
    toplevel .popup
    #PositionWindows . .popup

    listbox          .popup.strengthlist -yscrollcommand ".popup.scroll12
set"
    scrollbar        .popup.scroll12     -command ".popup.strengthlist yview"

    label .popup.select_strength -text "Select Strength :"

    place            .popup.select_strength -x 10 -y 10
    place            .popup.strengthlist -x 10 -y 35 -width 70 -height 150
    place            .popup.scroll12      -x 78 -y 35 -height 150

    .popup.strengthlist insert end "0.5 mg" "1 mg" "2 mg" "2.5 mg" "3 mg" "4
mg" "5 mg" "6 mg" "10 mg" \
    "12 mg" "15 mg" "20 mg" "25 mg" "30 mg" "40 mg" "50 mg" "60 mg"
"100 mg" "125 mg" "150 mg" \
    "200 mg" "250 mg" "300 mg" "500 mg"

    bind .popup.strengthlist <ButtonRelease> {
        set dosage [.popup.strengthlist get [.popup.strengthlist
curselection] ];
        wm withdraw .popup
        wm deiconify .
    }

}

}

# Manual Selection Interface for "frequency" field
proc DisplayManualSelectionInterface_freq {} {

    if {1 == [wininfo exists .popup]} {destroy .popup}
    toplevel .popup
    #PositionWindows . .popup

    listbox          .popup.freqlist      -yscrollcommand ".popup.scroll13 set"
scrollbar        .popup.scroll13     -command ".popup.freqlist yview"

    global complete_prescription

```

```

label .popup.selectF -text "Select Frequency"
place .popup.selectF -x 10 -y 20

place .popup.freqlist -x 10 -y 50 -width 50 -height 120
place .popup.scroll3 -x 59 -y 50 -height 119
label .popup.times_per_day -text "per day"
place .popup.times_per_day -x 90 -y 65

.popup.freqlist configure -selectmode single

.popup.freqlist insert end 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

bind .popup.freqlist <ButtonRelease> {
    set freq "[.popup.freqlist get [.popup.freqlist curselection]] per
day";
    wm withdraw .popup
    wm deiconify .
}

}

# Manual Selection Interface for "Duration" field
proc DisplayManualSelectionInterface_duration {} {

    if {1 == [wininfo exists .popup]} {destroy .popup}
    toplevel .popup
    #PositionWindows . .popup

    listbox .popup.durationlist -yscrollcommand ".popup.scroll4 set"
    scrollbar .popup.scroll4 -command ".popup.durationlist yview"

    label .popup.selectD -text "Select Duration"
    place .popup.selectD -x 10 -y 20

    place .popup.durationlist -x 30 -y 50 -width 50 -height 100
    place .popup.scroll4 -x 80 -y 50 -height 99
    label .popup._days -text "days"
    place .popup._days -x 115 -y 100

    .popup.durationlist configure -selectmode single

    .popup.durationlist insert end 1 2 3 4 5 6 7 8 9 10 12 14 15 20 25 30
45

    bind .popup.durationlist <ButtonRelease> {
        set duration "for [.popup.durationlist get [.popup.durationlist
curselection]] days"
        wm withdraw .popup
        wm deiconify .
    }

}

# Final display of complete prescription
proc review {prescr} {

global ID

```

```

label .id_final -text "Patient Number : $ID"
label .prescr_final -text "Prescription : \n $prescr"

place .id_final -x 20 -y 20
place .prescr_final -x 20 -y 50

}

proc enter digit {

    global ID
    set ID "$ID$digit"

}

proc clear { } {

global ID
set ID ""

}

# Numerical Keypad for entering patient's ID number
proc DisplayIdentificationInterface {} {

    label .ident -text "Enter the patient's ID#"
    entry .id -textvariable ID -relief sunken

    button .one -text "1" -command {enter "1"}
    button .two -text "2" -command {enter "2"}
    button .three -text "3" -command {enter "3"}
    button .four -text "4" -command {enter "4"}
    button .five -text "5" -command {enter "5"}
    button .six -text "6" -command {enter "6"}
    button .seven -text "7" -command {enter "7"}
    button .eight -text "8" -command {enter "8"}
    button .nine -text "9" -command {enter "9"}
    button .zero -text "0" -command {enter "0"}

    button .clear -text "Clear" -command clear
    button .proceed -text "Proceed" -command {proceed $ID}

    place .one -x 55 -y 50 -width 30 -height 30
    place .two -x 85 -y 50 -width 30 -height 30
    place .three -x 115 -y 50 -width 30 -height 30

    place .four -x 55 -y 80 -width 30 -height 30
    place .five -x 85 -y 80 -width 30 -height 30
    place .six -x 115 -y 80 -width 30 -height 30

    place .seven -x 55 -y 110 -width 30 -height 30
    place .eight -x 85 -y 110 -width 30 -height 30
    place .nine -x 115 -y 110 -width 30 -height 30

    place .zero -x 55 -y 140 -width 30 -height 30
    place .clear -x 85 -y 140 -width 60 -height 30

    place .proceed -x 70 -y 180

    place .id -x 55 -y 25 -width 90
    place .ident -x 45 -y 5

```

```
}  
  
proc proceed {idnum} {  
    destroy .one .two .three .four .five .six .seven  
    destroy .eight .nine .zero .clear .proceed .ident .id  
    DisplayPrescriptionInterface $idnum  
}  
  
set host 10.0.0.1  
set port 1200  
  
wm geometry . 200x230  
  
DisplayIdentificationInterface  
  
vwait eventLoop
```

APPENDIX C

Recognition Grammar (GSL Syntax)

NOTE : The list of Drugs has been cut down to reduce the document's size.
There are more than 1200 drugs in the actual code.

```
#include "number.grammar"

.Prescription [
[ (Drug:d Amount:a Units:u ?Frequency:f ?Duration:dur) ]
]

Amount [
    [      (Number:n)      ]      { <amount strcat($n ":am")> }
]

Units [
    [ milligrams milligram (m g)      ]      {<units "mg:un">}
    [ grams gram (g r)                ]      {<units "gr:un">}
]

Frequency [

[(Number:f [(?times [per a] day))] )] {<freq strcat($f " per day,:fr")>}
[(Number:f [ (every Number:h hours) ] )] {<freq strcat($f strcat(" every "
strcat($h " hrs,:fr"))>}

]

Duration [

[(for Number:dur days)]      {<duration strcat("for " strcat($dur " days:du"))>}
[(for Number:dur weeks)]    {<duration strcat("for " strcat($dur " weeks:du"))>}
[(for Number:dur months)]   {<duration strcat("for " strcat($dur " months:du"))>}

]

; --- generated from dnames.txt ---
Drug [

[( a t s )]                  {<drug "a/t/s:dr">}
[(accupril )]                {<drug "accupril:dr">}
[(accutane )]                {<drug "accutane:dr">}
[(acetaminophen with codeine )] {<drug "acetaminophen w/codeine:dr">}
[(acetazol )]                {<drug "acetazol:dr">}
[(acetazolamide )]          {<drug "acetazolamide:dr">}
[(acetic acid )]            {<drug "acetic acid:dr">}
[(acetylcysteine )]         {<drug "acetylcysteine:dr">}
[(achromycin v )]           {<drug "achromycin v:dr">}
[(aciphex )]                 {<drug "aciphex:dr">}
[(aclovate )] {<drug "aclovate:dr">}
]
}
```

```
[(actigall )] {<drug "actigall:dr"> }
[(actos )] {<drug "actos:dr"> }
[(acular )] {<drug "acular:dr"> }
[(adalat c c )] {<drug "adalat cc:dr"> }
```

```
.
. // Cut down list
.
```

```
[(zoloft )] {<drug "zoloft:dr"> }
[(zonalon )] {<drug "zonalon:dr"> }
[(zovia )] {<drug "zovia:dr"> }
[(zovirax )] {<drug "zovirax:dr"> }
[(zyloprim )] {<drug "zyloprim:dr"> }
[(zymase )] {<drug "zymase:dr"> }
[(zyprexa )] {<drug "zyprexa:dr"> }
[(zyrtec )] {<drug "zyrtec:dr"> }
```

```
]
```

APPENDIX D

Wire Protocol Server Code Sample*

```
#ifdef _MSC_VER
#pragma warning (disable: 4786)
#endif

#include "WSWorker.h"
#include "SystemException.h"
#include "Debug.h"
#include <fstream>

using namespace std;

#define DEFAULT_SLEEP_PERIOD 100

WSWorker::WSWorker(SOCKET id, NuanceConfig* cP, Dispatcher* dP) :
    _sockId(id), _configP(cP), _dispatcherP(dP), _initializationDone(false),
    _isReadyToDispatch(false),
    _recognitionRunning(false),
    _engineP(NULL),
    _sleepPeriod(DEFAULT_SLEEP_PERIOD)
{
    if (_configP==NULL || _dispatcherP == NULL) {
        cerr << "WSWorker constructor: uninitialized objects?\n";
        throw new SystemException("WSWorker(): Exception\n");
    }
    if (Debug::debug(WSWORKER_BIT))
        cerr << "WSWorker constructor done\n";
}

WSWorker::~WSWorker()
{
    if (isAlive()) {
        setStop(1);
        while (!isDone()) Sleep(_sleepPeriod);
    }
    _actionsMap.clear();
    if (_engineP) delete _engineP;
    _engineP = NULL;
}

void WSWorker::init()
{
    _actionsMap.clear();
    _actionsMap["rec.start"] = startRecognition;
    _actionsMap["rec.stop"] = stopRecognition;
    _actionsMap["rec.recognize"] = endPointRecognition;
    _actionsMap["rec.grammar.get"] = getGrammar;
    _actionsMap["rec.grammar.set"] = setGrammar;
    _actionsMap["rec.grammar.setcontext"] = setGrammarContext;
    _actionsMap["quit"] = shutDown;
    NuanceStatus status;
    // #if 0
    WarnForUnhandledNotifications();
    _engineP = new RCEngine(_configP, *_dispatcherP, *this, status);
    if (status != NUANCE_OK || _engineP == NULL) {
```

* This code was provided in part by Nuance's sample programs. My co-worker Yiannis Christou completed most of the remaining coding and I just made some minor modifications. Dr. Christou has permitted me to use this code for the project.

```

        cerr << "WSWorker constructor: failed to initialize RCEngine\n";
        throw new SystemException("WSWorker(): Exception\n");
    }

    lock();
    _isReadyToDispatch = true;
    unlock();

    if (Debug::debug(WORKER_BIT))
        cerr << "WSWorker::init(): RCEngine built."
            << " Will wait for notification of completion";
    while (!initializationCompleted()) {
        Sleep(_sleepPeriod);
        if (Debug::debug(WORKER_BIT)) cerr << ".";
    }

    m_nl_result = NLInitializeResult(&status);
    if (NUANCE_OK != status) return;

    if (Debug::debug(WORKER_BIT)) cerr << endl;
// #endif
    if (Debug::debug(WORKER_BIT))
        cerr << "WSWorker::init(): Done\n";
    // ok, we're ready to run
}

void WSWorker::run()
{
    // read a line from socket, and act accordingly
    for (int i=0; i<MAXLINELEN; i++) _readbuf[i] = 0;
    _readbuf[MAXLINELEN-1] = '\0';

    int pos = 0;
    bool readline = false;
    int nb = 0;
    while (!readline && pos<MAXLINELEN) {
        Sleep(_sleepPeriod);
        nb = recv(_sockId, &(_readbuf[pos]), MAXLINELEN-pos, 0);
        if (nb == SOCKET_ERROR) {
            if (WSAGetLastError()==WSAEWOULDBLOCK) {
                WSAGetLastError(0);
                return;
            }
            std::cerr << "WSWorker::run(): id=" << getId() << ": recv() returns
with error "
                << WSAGetLastError()
                << " Exiting thread\n";
            closesocket(_sockId);
            setStop(1);
            return;
        }
        for (int j=pos; j<pos+nb; j++) {
            if (_readbuf[pos]=='\n' || _readbuf[pos]=='\r' ||
_readbuf[pos]=='\0') {
                readline = true;
                break;
            }
        }
        pos += nb;
    }
    if (nb==0 || _readbuf[0] == '\0' || _readbuf[0] == '\r' || _readbuf[0] == '\n')
        return;

    if (Debug::debug(WORKER_BIT))
        std::cerr << "WSWorker::run(): received " << _readbuf << std::endl;

    // parse line
    istringstream is(_readbuf);
    string word;
    is >> word;

```

```

        if (_actionsMap.find(word)!=_actionsMap.end()) {
            (_actionsMap[word])(this, is);
        }
        else { // command not understood
            if (!sendError(word + string(" REASON cmd not understood"))) {
                return;
            }
        }
    }
}

bool WWorker::initializationCompleted()
{
    lock();
    bool res = _initializationDone;
    unlock();
    return res;
}

bool WWorker::isReadyToDispatch()
{
    lock();
    bool res = _isReadyToDispatch;
    unlock();
    return res;
}

bool WWorker::sendEvent(const string& msg)
{
    string event = string("EVENT ") + msg + "\r\n";

    if (Debug::debug(WSWORKER_BIT))
        cerr << "WWorker::sendEvent(): will send: " << event << endl;

    int sb = send(_sockId, event.c_str(), event.size(), MSG_DONTROUTE);
    if (sb == SOCKET_ERROR) {
        cerr << "WWorker::run(): id=" << getId() << ": socket closed."
            << " Exiting thread\n";
        // closesocket(_sockId);
        setStop(1);
        return false;
    }
    return true;
}

bool WWorker::sendString(const string& msg)
{
    string event = msg + "\r\n";

    if (Debug::debug(WSWORKER_BIT))
        cerr << "WWorker::sendEvent(): will send: " << event << endl;

    int sb = send(_sockId, event.c_str(), event.size(), MSG_DONTROUTE);
    if (sb == SOCKET_ERROR) {
        cerr << "WWorker::run(): id=" << getId() << ": socket closed."
            << " Exiting thread\n";
        // closesocket(_sockId);
        setStop(1);
        return false;
    }
    return true;
}

bool WWorker::sendError(const string& msg)
{
    string err_cmd = string("ERROR ") + msg + "\r\n";

```

```

    if (Debug::debug(WORKER_BIT))
        cerr << "Worker::sendError(): will send: " << err_cmd << endl;

    int sb = send(_sockId, err_cmd.c_str(), err_cmd.size(), MSG_DONTROUTE);
    if (sb == SOCKET_ERROR) {
        cerr << "Worker::run(): id=" << getId() << ": socket closed."
            << " Exiting thread\n";
        // closesocket(_sockId);
        setStop(1);
        return false;
    }
    return true;
}

void shutDown(void* datP, istrstream& is)
{
    Worker* wP = (Worker*) datP;
    wP->sendEvent("OK\n");
    closesocket(wP->_sockId);
    wP->setStop(1);
}

void startRecognition(void* datP, istrstream& is)
{
    Worker* wP = (Worker*) datP;
    // if recognition state is running signal error, else start recognition
    if (wP->_recognitionRunning) wP->sendError("recognition already started");
    else {
        wP->_recognitionRunning = true;
        if (wP->_engineP) {
            unsigned int uid = wP->_engineP->GetUniqueID();
            wP->_activeIds.push_back(uid);
            wP->_engineP->StartRecognizing(wP->_grammar.c_str(), uid);
        }
        else wP->sendError("RCEngine is NULL");
    }
}

void stopRecognition(void* datP, istrstream& is)
{
    Worker* wP = (Worker*) datP;
    // if recognition is not active, signal error
    if (!wP->_recognitionRunning) wP->sendError("recognition not active");
    else {
        wP->_recognitionRunning = false;
        if (wP->_engineP) {
            wP->_engineP->StopRecognizing(wP->_activeIds[wP->_activeIds.size()-
1]);
            wP->_activeIds.pop_back();
        }
        else wP->sendError("RCEngine is NULL");
    }
}

void endPointRecognition(void* datP, istrstream& is)
{
    Worker* wP = (Worker*) datP;
    // if recognition state is running signal error, else start recognition
    if (wP->_recognitionRunning) wP->sendError("recognition already started\n");
    else {
        wP->_recognitionRunning = true;
        if (wP->_engineP) {
            int uid = wP->_engineP->GetUniqueID();
            wP->_activeIds.push_back(uid);
            wP->_engineP->RecognizeUtterance(".Prescription", uid);
        }
    }
}

```

```

        else wP->sendError("RCEngine is NULL");
    }
}

void getGrammar(void* datP, istrstream& s)
{
    WSWorker* wP = (WSWorker*) datP;
    string name;
    string contents;
    const int maxlinelen = 4096;
    char line[maxlinelen];
    for (int i=0; i<maxlinelen; i++)
        line[i] = 0;
    line[maxlinelen-1] = '\\0';
    s >> name;
    ifstream fs;
    fs.open(name.c_str());
    if (!fs.is_open()) {
        wP->sendError(name + string(" : file does not exist\\n"));
        return;
    }
    while (!fs.eof()) {
        fs.getline(line, maxlinelen);
        contents += string(line);
    }
    wP->sendEvent(string("grammar: ") + contents);
    // sendEvent(string("OK\\n"));
    return;
}

void setGrammar(void* datP, istrstream& s)
{
    WSWorker* wP = (WSWorker*) datP;
    string name;
    string contents;
    s >> name;
    ofstream fs;
    fs.open(name.c_str());
    if (!fs.is_open()) {
        wP->sendError(name + string(" : file could not be opened\\n"));
        return;
    }
    while (s >> contents) {
        fs << contents;
    }
    fs.close();
    // sendEvent(string("OK\\n"));
    wP->_grammar = name;
    return;
}

void setGrammarContext(void* datP, istrstream& s)
{
    WSWorker* wP = (WSWorker*) datP;
    string name;
    string contents;
    s >> name;

    // sendEvent(string("OK\\n"));
    wP->_grammar = name;
    return;
}

void NUANCE_MEMBER_FUNCTION
WSWorker::HandleInitializationCompleted(InitializationCompletedNotification const &icn)
{
    NuanceStatus status = icn.GetStatus();
    if (status != NUANCE_OK) {

```

```

        sendError(string("initialization of RCEngine failed\n"));
        cerr << "WSWorker: initialization of RCEngine failure. Thread will
stop\n";
        setStop(1);
    }
    else {
        cerr << "WSWorker::HandleInitializationCompleted(n): init. completed\n";
        cerr.flush();
        lock();
        _initializationDone = true;
        unlock();
    }
    return;
}

void NUANCE_MEMBER_FUNCTION
WSWorker::HandleAcknowledgment(AcknowledgmentNotification const &n)
{
    unsigned int request_id = n.GetRequestID();
    AcknowledgmentNotification::RequestType rt = n.GetRequestType();
    const char* rtstr = n.GetRequestTypeAsString(rt);
    // itc: HERE we could do some printing or other stuff but who cares
}

void NUANCE_MEMBER_FUNCTION WSWorker::HandleStartOfSpeech(StartOfSpeechNotification const
& n)
{
    cerr << "WSWorker::HandleStartOfSpeech(): speech started Notification received\n";
    sendEvent("speech started\n");
}

void NUANCE_MEMBER_FUNCTION WSWorker::HandleEndOfSpeech(EndOfSpeechNotification const &
n)
{
    cerr << "WSWorker::HandleEndOfSpeech(): speech ended Notification received\n";
    sendEvent("speech ended\n");
}

void NUANCE_MEMBER_FUNCTION WSWorker::HandlePartialResult(PartialResultNotification const
& n)
{
    const RecResult* rrP = n.GetRecResult();
    char result[4096];
    RecResultString((RecResult*) rrP, 0, result, sizeof(result));
    string event = string("rec.partial.result ") + string(result) + string("\n");
    sendEvent(event);
}

void NUANCE_MEMBER_FUNCTION
WSWorker::HandleRecognitionStopped(RecognitionStoppedNotification const &n)
{
    unsigned int id = n.GetUtteranceID();
    cerr << "WSWorker::HandleRecognitionStopped(): end of recognition because "
        << n.GetReasonAsString() << endl;
    switch (n.GetReason()) {
        case RecognitionStoppedNotification::COMPLETED: {
            const RecResult* rrP = n.GetRecResult();
            char result[4096];
            RecResultString((RecResult*) rrP, 0, result, sizeof(result));
            string event = string("rec.complete.result ") + string(result) +
string("\n");
            //sendEvent(event);

            char intrp[4096];

            NuanceStatus status ;

```

```

        m_nl_result = NLInitializeResult(&status);

        RecResultNLResult( (RecResult*)rrP, 0, m_nl_result);

        NLGetInterpretationString( (NLResult const*)m_nl_result, intpr,
4096);

        cerr << "Interpretation: " << intpr << endl;

        string interpretation = string(intpr);
        sendString(interpretation);

        char temp[2] = {0,0};
        istringstream is(temp);

        stopRecognition(this, is) ;
        //endPointRecognition(this, is);

        break;
    }
    default:
        sendEvent("Recognition aborted\n");
    }
}

//-----
// PrintFinalResult
//-----
void WSWorker::PrintFinalResult (RecResult * rr)
{
    NuanceStatus status;

    ofstream intfile("intfile.txt");

    intfile << "1" << endl;
    //if (m_verbose_mode_flag)
    // {
    //     print out the whole rec result
    //     PrintLn("*** RECOGNIZED");
    //     RecResultPrintToStdout(rr);
    // }
    // else
    // {
    //     print out only the text result and nl result
    int num_results;
    status = RecResultNumAnswers(rr, &num_results);
    // if (NUANCE_OK != status)
    // {
    //     PrintErrLn(status, "RecResultNumAnswers");
    //     return;
    // }

    //-----
    // Print each result.
    //-----

    intfile << "2" << endl;

    for (int index = 0; index < num_results; ++index)
    {
        char result_buf[1000];
        int confidence_score;

        status = RecResultString(rr, index, result_buf, sizeof(result_buf));
        if (NUANCE_OK != status)
        {
            PrintErrLn(status, "RecResultString");
            return;
        }
    }
}

```

```

        status = RecResultOverallConfidence(rr, index, &confidence_score);
        if (NUANCE_OK != status)
        {
//      PrintErrLn(status, "RecResultOverallConfidence");
            return;
        }

// Print("*** RECOGNIZED");
// if (num_results > 1)
//   Print(" %2d", index);

// if (0 == strlen(result_buf))
//   Print(": \"%s\" (Conf = %d)\n", "Rejected", confidence_score);
// else
//   Print(": \"%s\" (Conf = %d)\n", result_buf, confidence_score);

//-----
// Print the NL result, if any.
//-----

        intfile << "3" << endl;

        char nl_buffer[1000];
        intfile << "4" << endl;
        status = RecResultNLResult(rr, index, m_nl_result);
        intfile << "5" << endl;
        int num_nl_results = NLGetNumberOfInterpretations(m_nl_result, &status);
        intfile << num_nl_results << endl;
        if (NUANCE_OK != status)
        {
//      PrintErrLn(status, "NLGetNumberOfInterpretations");
            return;
        }

        for (int nl_index = 0; nl_index < num_nl_results; nl_index++)
        {
            intfile << "7" << endl;
            status = NLMakeIthInterpretationActive(m_nl_result, nl_index);
            if (NUANCE_OK != status)
            {
//          PrintErrLn(status, "NLMakeIthInterpretationActive");
                return;
            }
            status = NLGetInterpretationString(m_nl_result, nl_buffer, sizeof(nl_buffer));
            if (NUANCE_OK != status)
            {
//          PrintErrLn(status, "NLGetInterpretationString");
                return;
            }

//          PrintLn("*** INTERPRETATION %i: %s", nl_index, nl_buffer);

            intfile << "*** INTERPRETATION " << nl_index << " " << nl_buffer << endl;
        }

//      PrintLn();
    } // end for (for each result)
}

void NUANCE_MEMBER_FUNCTION
WSWorker::
HandleIncomingCall (IncomingCallNotification const & icn)
{
    // notification for incoming call (behavior)

    unsigned id = icn.GetCallID();
//   PrintLnInVerboseMode("incoming call ID %u from \"%s\"", id, icn.GetCallerID());

    int m_answer_call_id = id;
    _engineP->AnswerCall(m_answer_call_id);
}

```

```
    sendString("\nCall Answered\n");  
    //StateAnswerCall();  
}
```

```
void NUANCE_MEMBER_FUNCTION  
WWorker::  
HandleCallAnswered (CallAnsweredNotification const & can)  
{  
  
}
```

