ANALYSIS OF USER INTERFACE IN MEDICAL REPORT GENERATION

By

RICHARD S. BEAN

A THESIS PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001
TO SARAH
ACKNOWLEDGMENTS

First, I would like to thank Dr. Doug Dankel for his excellent guidance and support with this thesis. His advice and time are both greatly appreciated and respected. In addition, I would also like to thank my other committee members Dr. Joe Wilson and Dr. Randy Chow. Moreover, I would also like to thank Dr. Sumi Helal for substituting at my defense.

I would also like to thank the people of the Department of Obstetrics and Gynecology for their help and assistance. Special thanks go to Dr. Stan Williams and Dr. Rodney Edwards for their help with my research. I'd also like to thank Rick Anthony for his infinite understanding of all my emergencies and his unwavering "excellent" nature.

In addition, I would like to give special thanks to my parents, Richard and Heina Bean. Without their support, I would have never made it this far.

Finally I'd like to thank my wife Sarah for all her love and support during this time.
# TABLE OF CONTENTS

ACKNOWLEDGMENTS........................................................................................................... iv

ABSTRACT............................................................................................................................ vii

1 INTRODUCTION ............................................................................................................. 1

1.1 Problem Domain......................................................................................................... 2
1.2 Different Implementation......................................................................................... 3
1.3 Document Structure................................................................................................ 4

2 MEDICAL MANAGER AND AUTHORWARE ................................................................. 5

2.1 Medical Manager’s Consultation Report Generator In-Depth................................. 5
2.2 Macromedia Authorware 5.2..................................................................................... 6
   2.2.1 Authorware Flowline......................................................................................... 6
   2.2.2 Delivering the Application.............................................................................. 8
   2.2.3 Windows Controls.......................................................................................... 8
2.3 Database and Authorware...................................................................................... 9
2.4 Interface Design Principles................................................................................... 10
   2.4.1 Interface Principles........................................................................................ 11
      2.4.1.1 Make sure the program is familiar to the users.................................... 11
      2.4.1.2 Do not make the user's job harder....................................................... 11
      2.4.1.3 Make the program responsive............................................................ 12
      2.4.1.4 Make sure the program is consistent.................................................. 12
   2.4.2 Interaction Design........................................................................................... 12
      2.4.2.1 Design for your users........................................................................... 12
      2.4.2.2 Design with goals in mind................................................................. 13
      2.4.2.3 Design to serve people...................................................................... 14
   2.4.3 Interface Design in Medicine.......................................................................... 14
2.5 Summary and What Is Next.................................................................................... 15

3 ReportGen USER INTERFACE INTERACTION.......................................................... 16

3.1 Visit Report.............................................................................................................. 17
3.2 History and Physical.............................................................................................. 26
3.3 Summary................................................................................................................ 30
Computers are playing a continually growing role in almost all professions. However, the software on these computers is quite often inadequate due to poorly designed user interfaces, which generate significant stress for the users. This added stress lowers the productivity of the workers, causing businesses to lose large amounts of time and, as a result, money. In some industries, it is even preventing the acceptance of software to replace traditional methods of achieving tasks.

One of these industries is health care. Physicians resist software solutions because this software often takes more time or is more difficult to use than the traditional paper methods. This thesis examines designing software for creating both a Visit Note and a History and Physical Note. The focus of this thesis is the user interface of this software. In particular, the software is designed with the attempt to make it easy to use, and also quicker than traditional methods for generating these reports.
The resulting software, called ReportGen, was developed using Macromedia Authorware. It provides physicians with an easy-to-learn interface. The software also helps generate the two reports by anticipating what the physician will want and obtaining that information from a database without a user request. This helps the software compete with the traditional pen and paper method.
CHAPTER 1
INTRODUCTION

In the new century, computers have revolutionized the way we live and interact with other people and our surroundings. Computers allow achievements that were unimaginable even 20 years ago. They have become prevalent in all areas of human life. Computers are in the ovens with which we cook, the cars we drive, and even our alarm clocks.

However, even with all the great services computers provide, they present their share of problems as well. One of the most troublesome areas in the computer genre is the user interface. The computer industry has been woefully inadequate in designing systems that users can use easily, without incurring additional stress and anxiety. This is by no means intentional but a product of programmers instead of interaction designers building the interface. Software is designed with ease of programming as the main priority and user interaction as a secondary distraction.

This lack of good interface design limits the acceptance of software systems in business. People do not like to work with computer systems due to the stress that they add to a workday. Companies can also lose large amounts of money in unproductive time due to bad interface design. It is necessary to look at user interface designing in a much more prominent light, allowing software to be accepted more readily.
1.1 Problem Domain

Like all fields in the business world, health care’s computer systems can benefit from well-designed user interfaces. Doctors need quick and easy software that does not present as many challenges as it solves. If this software is present, it has the potential to save the doctor and patients a large amount of time and money.

To show this problem, the user interface of a report generation tool, which generates reports for a patient's medical record, is examined. The goals are to design a working tool, and to take great care in designing an interaction that is easy to learn and use and which causes minimal user stress. For the greater medical community to accept these types of tools, the physicians and medical personnel that will use them must not see these software applications as a hindrance or "necessary evil," but rather as a time-saver and useful aid in their daily job.

The report generation tool is modeled after several existing systems. These systems like the consultation report generator of Medical Manager, by TriMed Technologies, Inc., provide a basic model for medical report generators. These systems have the medical professional select a series of symptoms and enter a diagnosis. Medical Manager’s report generator aims to make it easier for medical professionals to create patient visit reports. However, its interface design is not as effective as it could be. With this software as a basis, this thesis investigates the creation of another tool, having much of the same function, but which uses a different and less confusing interface. This document refers to this tool as ReportGen from this point forward.
1.2 Different Implementation

Macromedia Authorware was used in designing ReportGen, allowing for quick and easy interface modifications. ReportGen takes the physician through a series of pages, which gather information for specific parts of the report. The physician simply traverses each section, inserting any data observed. When the physician reaches the end, ReportGen generates the report, which it can save or print.

In addition to simply creating a better interface, the program design provides helpful hints or feedback in anticipation of the doctor's activities. This can be seen in the diagnosis tool which, based on the symptoms selected, presents a list of possible diseases or conditions to the physician. The physician can select any of these conditions or, if the physician disagrees with all of them, select a blank entry and manually enter the disease. This feature allows the program to anticipate treatment types and medication prescriptions. In addition, the program makes it easy to create reports: either a summary for the patient or a more technical entry for the patient's file.

These changes in the design provide for a more robust interaction without cluttering the software with too many unnecessary features. The added features are intended to be inserted intelligently in a way that does not overwhelm the user. This helps prevent the software from degenerating into a laundry list of features that simply get in the way. ReportGen maintains a balance between providing a pleasant and useful experience to the medical professionals, and providing the needed tools to speed up the report generation process (while being readily accepted and wanted).

In addition to the patient’s visit report, ReportGen includes several other commonly used forms. The same overall design was also used in these forms, resulting in a custom report for each form. The program helps complete these forms by pulling
patient data from a database and placing them in the appropriate locations. The physician simply needs to examine the form and verify that the information is correct.

To perform the diagnosis and smart form generation, a database of information must be used. Since Authorware can use SQL, it can interface to any database. This thesis used Microsoft Access, which provided easy manipulation of the data so the focus could remain on the user interaction.

1.3 Document Structure

The following chapters of this document show the software and its interface. Advantages and disadvantages of the approach are also discussed. Chapter 2 discusses Macromedia's Authorware and why it was chosen for the design tool. Principles of user interface design that ReportGen follows are also discussed. Chapter 3 and 4 examine ReportGen's interface and discuss its design. Chapter 5 presents conclusions and examines extensions to this thesis.
CHAPTER 2
MEDICAL MANAGER AND AUTHORWARE

The consultation report generator of Medical Manager is the basic model for which this thesis is based. It is a commercially available report-generation tool from TriMed Technologies, Inc. that allows physicians to select symptoms and create a visit report. Macromedia Authorware 5.2 is a software development tool primarily used to produce electronic learning programs that people can use as lessons on various subjects. The report generator of Medical Manager is what the thesis is trying to improve and Authorware is the tool that was used to do it.

2.1 Medical Manager’s Consultation Report Generator In-Depth

Medical Manager’s report generation interface is essentially contained in one screen broken into two primary frames. Each frame is a sub-window containing different parts of the report’s information gathering, allowing the physician to enter the data and view the final report. The main window, where the report is generated, contains a field for the patient’s primary reason for the visit. A vertical panel on the left side presents a long list of symptoms. This list has a series of pluses and minus that can be checked for each symptom indicating whether the symptom is present or absent, respectively. Included in the main frame is a place for the physician to enter a diagnosis. Note that the program does not assist in determining the diagnosis. The physician simply enters the diagnosis by hand. When all is finished the system then produces a report that can be used later.
This software has some disadvantages that this thesis addresses. The interface for medical manager is clunky. Too much information is presented at once. The clutter that is created cancels out the convenience of having everything in one screen. This is especially true on systems with low resolution. The software does not take into account the principle of information hiding [Joh00]. Information that is used frequently should be prominent and lesser use information or tools should be hidden unless the user wants it. These problems in report generation are addressed using Macromedia’s Authorware.

2.2 Macromedia Authorware 5.2

Authorware was chosen because of its development ability. It is very easy to generate a prototype that is easily manipulated and redesigned using its “What you see is what you get” design interface. Authorware’s design purpose is the creation of academic software for teaching purposes. It contains many built in functions for providing tests and obtaining interactions from users. Many of these functions can easily be adapted for nontesting purposes. The software has a built-in ability to use external databases and network resources such as streaming media and networked databases. Lastly, Authorware programs can be packed into stand-alone applications for both Microsoft Windows and Macintosh computers, and also World Wide Web applications. In addition to Authorware’s flexibility in platforms, it uses a flowchart-type language that makes modifying the interface very easy.

2.2.1 Authorware Flowline

The flowline is the framework of any program constructed in Authorware [Rob97]. The flowline determines the flow of execution. It is analogous to a flowchart. A number of paths can be followed, depending on the interaction the user provides. The flowline is composed of a number of icons listed below [Rob97] and shown in Figure 2.1:
The display icon is used to display text and graphics on screen.
- The motion icon is for animating an object on the screen.
- The erase icon will erase an object in the display.
- The wait icon is used to insert a pause in the flow. The pause can either be a user click or a timed delay.
- The navigate icon is used to jump to different icons in the flow line. This can be used as a jump and return or a simple goto.
- The framework icon provides the framework for a series of pages. This icon contains the navigation structure of a set of pages on the display.
- The decision icon is used to present a set of paths from which to choose. This icon can iterate through the different paths or, based on a calculation, pick a path.
- The interaction icon is the basis for user interaction in Authorware. This icon allows a developer to select from a set of interaction options such as buttons or text entry.
- The calculation icon is where scripts can be inserted to perform specific functions. The scripting language is a simple Pascal-like language having normal programming constructs such as loop and assignments.
- The map icon is essential a holder. It allows you to place icons inside it to group related icons together and prevent the flowline from becoming cluttered.
- The movie, video, and sound icons allow the insertion of multimedia into the Authorware lesson being constructed.

In addition to these icons, Authorware 5.2 has several other items that can be inserted into the flowline to provide more specialized controls than the basic icons including ActiveX controls. The ActiveX controls allow you to plug in external program such as Microsoft Internet Explorer to perform specific tasks that Authorware cannot do on its own. Authorware also contains “Knowledge Objects” which are built in
Authorware controls that provide another set of specialized tools. These tools include built-in quiz frameworks and Standard windows controls. Each of these controls has a built-in wizard allowing the developer to construct the object’s properties.

The flowline, as shown in Figure 2.2, is edited by dragging whatever icon the developer needs into the flowline. The order of the icons determines the flow of the program. This property makes the interface design more flexible. As new ideas arrive, the interface can be molded to fit the needs of the users quickly. This is an important factor in the choice of Authorware for this thesis. The ability to design the core of the program quickly and then change the user interface as needed made Authorware very valuable.

2.2.2 Delivering the Application

Once an application is constructed, Authorware allows it to be packaged in several ways. The first is to package it as a stand-alone application. The program can be executed on a Microsoft Windows or Macintosh computer. This is beneficial for systems that are not networked or for applications that need to access more of the file system’s resources. The application can also be packaged for the Internet. Here, the application is broken into small files that are placed in a web page. This setup works well when delivering the application to a large institution.

2.2.3 Windows Controls

As mentioned earlier, the Knowledge Objects provide additional features to an application. For this thesis, the Windows Control knowledge object was used frequently to create standard windows objects such as checkboxes, comboboxes, memos, lists, etc. These are constructed with a wizard and are assigned a variable for identification. Authorware then provides two more Knowledge Objects to interact with the controls.
One of these objects sets the control’s properties. This allows you to manipulate the fonts and values of the object. The other allows you to obtain a value from the control. These objects are controlled with the same wizard as the control.

Figure 2.2. A section from ReportGen's flowline.

2.3 Database and Authorware

Authorware can also interact with a database. Using the calculation icons, you can create Structured Query Language (SQL) queries to execute on an external database. This provides access to a variety of databases. Communication between Authorware and
the database uses ODBC, which stands for Open Database Connectivity. It is a programming interface that enables applications to access data in databases that use SQL as a data access standard. An ODBC control is created on the computer and the Authorware application can be interfaced to a local or networked database. For this thesis, Microsoft Access 97 is used for the database. It is a flexible database without a large amount of complexity providing a good base to store the data being used. Access would not be the best choice in a production application because it would be difficult to maintain on a large scale. For the purposes of the thesis it is adequate and provides all needed functionality.

ReportGen uses the database to obtain the symptoms and diseases. It is a pseudo knowledge base. The information about the imaginary patients used to demonstrate the interface is also stored in this database.

2.4 Interface Design Principles

In addition to simply designing a new tool for report generation, ReportGen focuses on interaction design principles to make the software meet the needs of its users. These principles are what frame the development of ReportGen. The focus in interface design is the usability of your target audience [Joh00]. In addition, the software must do more than be just usable. It must provide what the customer desires. If software is desirable to its users, its users will remain loyal to it [Coo99]. Alan Cooper provides this illustration in his book *The Inmates are Running the Asylum*.

Desirability is easy to confuse with need, but they are dramatically different. I *desire* a six week vacation in Bermuda, but I don't *need* it. If I have gall stones I *need* gall bladder surgery, but it is not something I *desire*. [Coo99, pp. 74]
ReportGen is designed to be desirable to its users and at the same time meet their needs. This is accomplished by following some proven techniques in interface design.

2.4.1 Interface Principles

There has been much research done on interface design [Nai01, Shn92, Ras00], and several essential principles to user interface design have emerged. When followed, these principles lead to a well designed product.

2.4.1.1 Make sure the program is familiar to the users

The program should be written such that the interface represents concepts in the user’s domain [Chi93]. This means that when a user interacts with the software, they do not want to be confronted with a large amount of computer jargon that is irrelevant to their domain. Users do not care about file systems, RAM, or page faults. They do care about their reports, goals, and profession [Joh00]. The language and messages inserted into the program should reflect this. Error messages should be designed to give the users a message that they can understand [Jon00].

2.4.1.2 Do not make the user's job harder

This principle is simple to understand, however very important. If the program makes the job harder than it already is, users will not use it. If ReportGen is more difficult to use than simply creating a paper report, then it will fail as a real world application. To prevent this, ReportGen has features making the creation of reports easier, such as automatically filling in known histories, providing a list of possible diseases, and allowing the user to back out of any mistakes.

2.4.1.3 Make the program responsive

No matter how fast a computer or how quick an algorithm, if the user feels like the program is not responsive enough, they will not use it [Joh00]. Responsiveness is the reaction of the program to user’s actions. When a user performs a task that will take
awhile to complete, the computer should provide the user with constant and accurate feedback of the status of that task. If the program fails to do this, the user will think that the computer is slow or broken, resulting in the user performing one or more unwanted actions. For example, they may retry the task and get either no indication that the computer is doing anything new, or they may perform the task a second time wasting more computer resources. Alternatively, they might simply turn off the computer and try again.

This lack of feedback will frustrate users and forcing them to guess whether they even initiated the task. This adds to the stress related to using this product and results in the user not liking it or even using it.

2.4.1.4 Make sure the program is consistent

The software interface components should be used properly and in a consistent manner [Chi93]. This allows the user of the program to become acquainted with the way the software works. The components such as radio buttons, text boxes, and pull down lists need to be used correctly. If the components are used improperly it adds the level of frustration and confusion. Many of these misuses of these interface components are discussed in Jeff Johnson's book *GUI Bloopers Does and Don'ts for Software Designers and Web Designers*. If these misuses are understood and a careful effort is made to remove all of them from your software, the interface will be much more consistent and useable.

2.4.2 Interaction Design

2.4.2.1 Design for your users

When writing software a designer must focus on the needs of the end users (i.e., the users that will use the software). The developer should take care to determine what
these users want. To do this, the developer must perform three actions. First, know who
the intended users are. ReportGen’s intended users are physicians who need to write
patient reports. Secondly, the developer must investigate the users. The physicians that
will use ReportGen desire a system that simplifies their job rather than adding an
additional burden. Lastly, the developer must collaborate with the intended users
[Joh00]. In the development of this system, care was taken to talk with the physicians
concerning the design and structure of ReportGen.

In addition to know what users want, you must design the software to perform the
user’s tasks. For ReportGen this is quickly creating a visit report. Each visit by a patient
needs documentation, whether it is a simple visit report or a history and physical, and this
documentation needs to be created quickly and effectively.

2.4.2.2 Design with goals in mind

When creating new software the designer should focus on the goals of the people
who are going to use the software [Coo99]. To do this the designer must have an
intimate knowledge of the users who will use the software. The designer must be able to
determine the users goals. Cooper defines two sets of goals, practical goals and personal
goals. Practical goals are goals that involve the business of the user and their tasks that
need to be accomplished. These goals are centered around the job at hand and need to be
met for the tasks to be completed. Personal goals on the other hand are goals held by
each individual user. The goals could be to not be made to fell inadequate, or have their
time wasted. These goals are extremely important and must not be broken because they
will always lead to dissatisfaction with the product [Coo99].
2.4.2.3 Design to serve people

Far to often software is designed in such a way that the user must adapt to the computer. They are assaulted with cryptic error messages about RAM, file systems, and protocols. They are forced to check and double check the simple tasks the computer should do on its own [Der01]. People must work around the computers habits to achieve their tasks. Software should not be designed this way. When a user has the computer perform a task the computer should do it and do it well. In addition, the user should be able to perform that task without having the computer dictate how they do it. Software design must make the change to a more human centric mindset and away from the cryptic mindset of computer hardware. This will create much more acceptance of the computer systems people use.

2.4.3 Interface Design in Medicine

Software designed for the medical profession must follow the same general guidelines for software development as all other programs. However, some considerations need to be understood when designing for the medical professional. Much of the work done in clinical offices is still based on traditional pen and paper methods. However, many people have studied physicians using computer systems [Lan00, Mel98, Van98, Sit93, Sit99]. Many physicians are reluctant to move to computerized data entry because they are afraid they will be more difficult and require greater time. However, physicians will embrace computer systems if the systems are well integrated with a clinical record and are flexible enough to handle their preferences [Mel98].

Also most clinicians are "independent entrepreneurs" [Sit99, p. 400] who do not want their workflow interrupted. Any software designed for these people will need to accommodate the way they already do their business. This will encourage them to make
the switch. These systems must be responsive, but they must have an interface that optimizes the physician ability to perform their tasks.

### 2.5 Summary and What Is Next

This chapter examined the consultation report generator of Medical Manager as a baseline for ReportGen. Second, it provided an overview of Authorware and its development environment. The flowline and knowledge objects were described, and the Access database was described. In the following chapter, ReportGen’s interface is described by performing a walkthrough of the software.
CHAPTER 3
ReportGen USER INTERFACE INTERACTION

This chapter examines the user interface of ReportGen by performing a screen by screen walk-through of the reports and explaining the interface concepts. ReportGen has the ability to produce two report types. The first report is a patient visit report. It follows the Subjective, Objective, Assessment, and Plan (SOAP) method [FuL01]. SOAP is the standard format for a patient visit or progress report. The second report the software can generate is a History and Physical report. This report obtains information from a database and presents it to the physician to help facilitate the report creation processes. ReportGen attempts to simplify entering a report into the medical record system by improving the interface presented to doctors. To show this process, each screen of the document is displayed with an explanation of the interface design ideas used to create it.

The first screen that the physician encounters is the form select screen, shown in Figure 3-1. This screen allows the physician to select from the available forms and provides the information common to all forms, namely the patient’s name and medical record number.

The physician enters the patient name and medical record number. Then, the physician selects the desired form from the pull down list. Once all information is entered, the physician selects the OK button to begin creating the report. ReportGen provides only three buttons the physician can choose. Each button is labeled to make its use clear to the physician. Removing any confusion of available buttons is a good technique for making the software more usable [Joh00] and enjoyable to the physician.
The first button displays a README file. This file contains the information about the software and its features. It is positioned away from the main components because it is used very infrequently. The second button launches the program, and the last button quits the application. This opening screen is kept very simple to allow the interface to be clear. Once the physician selects the form to create and clicks OK, the program proceeds to the selected form. Each is examined in the following sections.

Figure 3-1. The form select screen presented to the physician.

3.1 Visit Report

When the physician selects the visit report form, the program displays the screen shown in Figure 3-2. This is the screen where the physician enters the patient’s reason for the visit, the primary complaint. In the SOAP format, this is the subjective part of the report. The patient describes their problem to the physician and the physician enters it on this screen.
Figure 3-2. Visit report form's first page. The physician enters the patient’s primary reason for the visit.

This screen has two regions, the main area where the physician enters the necessary data and the navigational region at the bottom. The main region contains the patient's name and medical record number. In addition, the title displays the report type that is being created. This helps to orient the physician. It provides extra information to help focus the physician [Coo99]. The program then instructs the physician to enter the reason for the patient's visit. A large text box is provided to allow the physician to enter as much information as needed. This region is kept simple to remove unnecessary complexity.

The bottom of the screen contains the navigational framework. It provides the physician with information on their status in the report generation process giving them an indication of how far along they are. This gives the physician an impression of responsiveness [Joh00]. The physicians can chose from two out of the four buttons. The
previous button is deactivated because this is the first page. The finish button is also
deactivated because the program does not have enough information to generate the
report. Deactivating the buttons prevents the need for error boxes by preventing errors
before they happen. The two buttons that can be used are the new report and next button.
Both of these are self-explanatory. Clicking the new report button resets the program for
a new report, and clicking the next button takes the program to the next screen shown in
Figure 3-3.

Figure 3-3. Screen two of the visit report. It presents the primary reason and allows the
physician to go back and make corrections.

This screen provides instructions informing the physician to review the entry from
the last page. The physician can review their work or, if the patient is present, can show
the patient the information as well. If it is incorrect the physician uses the previous
button in the navigational area to go back the last screen. Once everything is correct the
physician clicks the next button to go to the next screen, which is shown in Figure 3-4.
Figure 3-4. The symptom selection screen where the physician can select symptoms from various regions of the body.

This screen greets the physician with the now familiar navigational section, patient name, and Medical record number. While these features may seem negligible, they make the program very simple to learn and operate. This allows a greater range of experience levels to use this software [Shn00]. The physician is presented with a silhouette of a person. The instructions inform the physician to select a region of the body, then a list of symptoms are shown on the right portion of the screen as shown in Figure 3-4. This technique is used to improve one of the problems with Medical Manager’s report generator. By clicking on a region of the body and getting the symptoms for that region, the software can present a large number of symptoms without cluttering the available screen area. Medical Manager presents all the symptoms in a large list taking much of the screen and making it difficult to find symptoms. The silhouette is marked with a changing icon to show when the physician is in a click-able
region. The mouse changes from the default arrow to a small hand. This change indicates that there is something to do at that point. In addition, since the eye of the physician using the program is usually on the pointer the change is easily perceived [Joh00].

While in this screen the physician gathers the objective part of the visit note conforming to the SOAP format for the report. The physician selects several of these symptoms then presses next to get to the diagnosis screen.

The new screen, see Figure 3-5, allows the physician to select from a list of possible diagnoses. The order of the diseases that are shown is based on the number of the selected symptoms that match each disease. The disease with the most symptoms that match is listed at the top of the list. The physician can view information about each disease if they need some clarification. Hiding information on the disease behind these buttons is an example of information hiding. By hiding the information that most likely will not be needed (most doctors know the diseases in their domain), the physician has a much cleaner interface with the opportunity to get additional information if desired. The physician then selects the disease that is most appropriate. This provides ReportGen with the assessment of the SOAP model. If the physician feels that none of the produced diagnoses are appropriate then the "other" choice can be used. This allows the physician to enter the information on the disease manually.
Figure 3-5. Disease selection screen. The physician can view info about each disease and make a choice.

Having the program provide the physician with several likely diagnoses, allows the physician to save time by having the disease already supplied by the program, thus cutting down the amount of text entry. However, ReportGen does not force the physician to use the produced diseases. The physician still has the freedom to easily overrule the program and enter a different assessment. If the physician selects the other button, an additional screen is presented and two text boxes allow the physician to enter an alternative assessment. If the physician selects a disease, the program asks for confirmation. The physician then can click the next button to proceed to the next screen.

The screen shown in Figure 3-6 displays the treatment and medications for the selected disease. If the physician added a new disease then these are blank and the physician must added them manually. If the disease selected is one suggested by the program, then the physician must simply verify the treatment and medication. This screen is the plan portion of the SOAP model. The top field is an editable text box,
displaying the treatment. The treatment consists of general measures for treating the disease. The physician can edit this field to have it reflect the patient’s unique situation.

By providing a general treatment, the program anticipates what the doctor will want to enter thus speeding the process. The second field displays the medications for the disease as a list. The physician selects the appropriate medications for this patient.

![Treatment and medication screen](image)

Figure 3-6. Treatment and medication screen for the selected disease.

The Medication field shown in Figure 3-6 is not editable like the treatment field. A better design would be to allow the physician to edit the medication information. The physician should also be allowed to select the relevant information. Unfortunately, Authorware’s implementation of a list does not allow this. Not, being able to edit one field while the physician can edit the other will only cause confusion and should be corrected for a good interface design. Once satisfied with the treatment and medications the physician clicks the next button and proceeds to the screen shown in Figure 3-7.
Figure 3-7. Report selection screen. This screen is used to select the report to generate.

This screen presents the physician with the selection of a report type and a description of the report types to remind the physician what each type entails. The physician selects the report type from a pull down list. The reports that can be generated are a Patient Visit Summary and a Technical Summary. The Patient Visit summary is a report targeted to the patient or a referring physician. It structures the report in the SOAP format, using natural language sentences and easy to read nomenclature. The Technical Report also uses the SOAP format, but is targeted for the patient’s medical record. The report is tabular in structure and is designed for quick review by the physician. Once the report type is selected, the physician clicks the next button to proceed to the final screen shown in Figure 3-8.

This screen displays the generated report. The physician can edit this report and make any modifications to the information already entered. This screen provides two buttons in the main part of the screen allowing the physician to save the report and to
print the report. ReportGen disables the print button until the physician saves the
document. The document is saved into a text file wherever the physician would like.
After it is saved, the physician can select the finish button, which is now activated in the
navigation area or can print the document. Pressing the finish button causes ReportGen
to return to the opening screen shown in Figure 3-1.

![Figure 3-8. The report screen where the physician can view and modify the completed report.](image)

The visit note report separates the sections of the generation process into a series
of linear steps. This process decreases the time to create a report and removes
complexity and clutter that would irritate a physician using the system. The linear
method of report generation minimizes any confusion and allows the physician to quickly
become accustomed to the navigation of the program. This allows the navigation to fall
into the background, enabling the physician to focus on the report and not the software
itself.
3.2 History and Physical

The second report that can be created is the History and Physical. This report computerizes the form used by the Florida Surgical Center. To make this form usable by physicians, it is linked with a database to gather information about patients. The physician selects the report from the opening screen in Figure 3-1. Once the physician selects the OK button, the computer displays the first screen of the History and Physical shown in Figure 3-9.

![Presentation Window](image)

Figure 3-9. This screen shows the patient demographic screen from the History and Physical part of ReportGen.

Like the last report, this has the screen divided into two sections. The upper section contains the fields and information that need to be gathered, while the lower section contains the same navigational information allowing for a familiar interface. The first screen presents a series of fields. The physician completes the information required. The physician can select the person performing the procedure from the pull down list.
shown near the top of Figure 3-9. This pull down list is generated from a list of doctors in the database that can be updated as local doctors join or leave this practice.

The ASA class entry, which is the anesthesia class of the patient's surgical risk, is depicted as radio buttons since they are mutually exclusive. The last three fields are all text boxes where the physician enters textual information. Once all the data has been entered the physician the clicks the next button to go to the next screen shown in Figure 3-10.

![Figure 3-10. The second screen where the physician can enter the patient's history.](image)

The physician enters the patients history on this screen. According to Dr. Rodeny Edwards, a physician for the University of Florida’s Department of Obstetrics and Gynecology, having to tab through many fields to enter data makes a computerized version of this form to time consuming to use. To solve this problem ReportGen takes the medical record number entered in the previous screen and pulls the patients history information from a database. By providing the physician with the patient's information
from a database or medical record, the physician does not have to reenter this information and can move quickly through the screens. In addition to trying to save time by providing information, ReportGen tries to save time by minimizing errors. An example of this is the Adverse Drug reaction fields. The physician is presented with a yes or no choice using radio buttons. When the button is set to no, the text field for the description is disabled. When the physician selects yes, the text field is activated and the physician can enter a description. This prevents confusion by the physician when completing that question.

The studies consist of a group of pull down menus containing the status of each of the study categories. The physician verifies the information and clicks next to move onto the next screen shown in Figure 3-11.

Figure 3-11. This screen presents the Review of Systems to the physician.

This screen is where the physician enters the review of systems. It contains a list of bodily systems where each system has a pull down menu with three choices: normal,
abnormal, or not applicable (N/A). The systems default to normal and the physician only has to change those systems that are not normal. The physician then selects the next button and ReportGen takes one of two paths.

If the physician marked any of the systems as abnormal, ReportGen goes to the screen shown in Figure 3-12. This screen allows the physician to enter an abnormality description in the text box provided. Once the abnormalities are addressed the physician selects the next button and proceeds to the final screen to review the report shown in Figure 3.13. If no abnormalities are selected, ReportGen goes directly to the last screen to review the report. Displaying the abnormalities screen when no systems are abnormal would only get in the way of the physician and slow the process.

Figure 3-12. This screen allows the physician to enter the abnormality description for any systems that were marked as abnormal.
Figure 3-13. This screen allows the physician to review the final History and Physical report and make any modifications.

The final screen is identical to the layout of the report screen from the visit note section. The screen consists of the editable text field of modifications to the report. Also the physician can save or print the report from here. When the physician finishes the project, clicking the finish button brings them back to the start screen shown in Figure 3-1.

3.3 Summary

The two reports created by ReportGen use a simple and identical interface for straightforward navigation. This allows the use of the program to become second nature permitting the physician to focus on the task at hand [Coo99]. The interface is designed to minimize errors and speed entry, providing the physician benefits over the conventional paper methods. The next chapter looks into the details of how ReportGen
works underneath the interface. It examines the Database used and how the information is presented to the program and physician.
CHAPTER 4
INSIDE ReportGen

To have ReportGen do the necessary tasks and make it useable from the physician's point of view, the software must be able to provide the physician with the information wanted. To accomplish this, ReportGen stores its data in a Microsoft Access database. Authorware uses this database as a knowledge base for the visit report and a patient database for the history and physical. To communicate with the database the Authorware component of ReportGen uses SQL. ReportGen initializes the system by installing an ODBC conduit so that ReportGen can make the SQL queries. This chapter examines the Access database and the Authorware components of ReportGen and explains how they work together to provide the interface with the necessary information.

4.1 ReportGen database

The database used by ReportGen is a Microsoft Access 97 database. ReportGen uses it for two purposes. The first is as a simple knowledge base for the patient report and the second is to store patient information to provide the information needed in the history and physical form. The database is simple in design and acts more as a proxy to more sophisticated methods of getting the data. With the focus of this thesis being to design a good interface and not a perfect knowledge base or an online medical record system, this database simulates these functions for ReportGen. The database has five tables as shown in Figure 4-1.
Figure 4-1. A collection of the tables used in ReportGen’s database showing all the relations that are present.

The database contains two groups of tables. The first group consists of three tables. The first table contains all the available symptoms for the ReportGen program and their mapped body region shown in Figure 3-4. The second table contains all the possible diseases and information concerning treatment and descriptions. These two tables are related to a middle table that maps the symptoms to the diseases, making the diagnosis process possible. To add more diseases or symptoms to the program these three tables need to be modified. Currently, ReportGen contains information from the field of obstetrics and gynecology, but if this system were ported to another medical domain, repopulating these tables would be necessary.
The second group of tables is for the history and physical form. These two tables simulate an online medical record system capable of providing information in a tabular format for ReportGen. These tables populate the fields in the report to reduce the amount of typing required by the physician. More information would need to be provided to these tables as additional forms are added to ReportGen in the future.

4.2 ReportGen's Authorware Component

To use the tables described in the previous section ReportGen must initialize and install the database to allow the SQL queries to work. The code for this is shown in the Appendix. Adding and configuring an ODBC entry in the computer’s ODBC table performs this action. This entry contains the information about the location of the database and the access privileges. This entry is also given a name, which is used to refer to the database throughout the rest of the program.

After the database is setup, ReportGen obtains all the symptoms available by making an SQL query to the database. This is performed during initialization because it is a lengthy query, which takes a noticeable amount of time. By obtaining this information here, the system is more responsive to the user when interacting with the system. To make the SQL query, a string containing the query is saved into a variable called **DB_SQLString**. Once ReportGen has this SQL string, it calls a subroutine that sends the query to the database and waits for the response. The response is saved in the variable **DB_ODBCData**. This variable’s value is returned and is parsed by the program. See the Appendix for this code.

After setting up the initial components and installing the database, ReportGen displays the form select screen and waits for the physician’s response. The form the
physician selects determines the path that ReportGen takes and the information that is loaded.

4.2.1 Visit Report

When the visit report is chosen, ReportGen initializes all the variables in the section. The segments of the report are stored in a variable used to build the final report. The fields that are shown, such as text boxes and radio buttons, are part of a set of Authorware Knowledge Objects called Windows Controls. These controls are used throughout the thesis whenever a field for data entry is needed. The controls can be fed a pre-entered value from an Authorware variable and the value from that control can be saved after the user changes or enters new information.

ReportGen gathers the primary reason for the visit and stores it in a variable. When the physician selects the symptoms, another SQL query is executed to obtain the symptoms for the region of the body selected. This is done identical to all other SQL queries in ReportGen. ReportGen takes the results from the query and feeds them into the List box Windows Control. When the physician selects the symptoms that are present, ReportGen uses them to build a list of all the selected symptoms. This list is structured as an array, with the name of one symptom at each position. After all the symptoms are stored in an Authorware variable, ReportGen can select the diseases that fit.

Iterating through the list of symptoms, an ORed SQL query is generated. This process of generating the query is shown in the icon titled Get Disease, which is in the Appendix. This query is sent to the database and the list of diseases returned is presented to the physician who selects the appropriate disease. As mentioned in Chapter 3 this listed is sorted based on the number of the selected symptoms each disease matches. The
disease that matches the most selected symptoms is placed at the top and the rest are placed in descending order.

Once all the data has been collected, ReportGen creates the report by manipulating the values stored in the variables and adding additional report text to format the material. This report is saved as a large string, which is displayed to the physician for use and editing.

4.2.2 History and Physical Report

The history and physical report section of ReportGen uses the database differently than the previous section. The database is used to store information about the patient's history. The report first gathers information about the patient's identity and the procedure being performed. The physician enters the data into the form created with Windows Controls. These controls then save the data into a set of variables that were initialized when the screen was displayed. When the physician moves to the next screen, ReportGen takes the medical record number and queries the database. If the patent exists, her information is saved in the variables corresponding to the fields of the tables. If the patient does not exist, these variables are initialized to blanks.

These variables are fed to the Windows Control so the physician does not have to edit those controls unless they are incorrect or out of date. Like the other report, as the physician enters data the information is stored in variables that are later used to create the report. The report is saved as a string containing the formatting in addition to the data entered by the physician.
4.3 Summary

This chapter examines how ReportGen obtains and processes the data needed to provide a useful interface to the physician. The way the database is used and formatted for the application was examined. In the next chapter, this thesis discusses conclusions about this research. It also discusses the work that remains to be done on ReportGen to make it a more robust tool for physicians.
CHAPTER 5
CONCLUSIONS AND FUTURE WORK

This thesis examined the user interface design for a medical report generator named ReportGen. The focus on the programs interface design was intended to create a piece of software that would be usable and accepted by the medical community. Chapter 1 provides an introduction to the problem and introduces ReportGen. Chapter 2 describes interface principles and introduces Authorware. Chapter 3 looks at the interface of ReportGen by performing a walkthrough of the software, and Chapter 4 examines ReportGen's inner workings. The rest of this chapter summarizes ReportGen as a whole and examine how it can be improved in the future.

5.1 ReportGen as a Whole

In the thesis, ReportGen was described in terms of its interface considerations and its system functionality. However, it is important to note that in a well designed piece of software these processes must be closely linked. If the design focuses only on the system implementation, the interface will be invariable linked to that particular implementation and not the user’s goals. If the interface is designed with no thought about the implementation then a shell of a program is built with lots of useless or under implemented features [Joh00]. ReportGen is designed so both aspects of the programs work together. The flowline allows both the interface and underlying system to be managed together as a single entity.
ReportGen's design is effective at removing a large amount of complexity from the report generation process. It allows the quick and easy creation of the reports. Unfortunately, it also contains some interface imperfections that should be removed for the interface to be truly well designed. In the visit report the physician cannot edit the medications until the next screen. In addition the program requires both the use of mouse and keyboard. Authorware does not implement buttons so they can be easily clicked by tabbing to them and pressing the enter key. This makes it difficult for people familiar with conventional computer programs to interact with this program. Both of these are limitations of Authorware. For Authorware's many pluses there proved to be a few drawbacks as well. Whenever a development environment provides an interface toolkit, there are always some limitations in their functionality [Joh00]. By making this point clear it also shows how ReportGen's interface can be further improved with improvements in its development language.

ReportGen does provide enhancements for physicians. The ability to help predict diagnoses and provide information about these diagnoses would decrease the amount of work for the physician when they are unsure of a condition. This is accomplished by presenting all the relevant disorders up front, for the physician to browse. In addition, by providing the ability to connect with an online medical record, ReportGen minimizes the amount of background work physicians must perform to get the patient's history. This information is downloaded directly into the report saving the physicians time and allowing them to focus on the procedure to come. The interface fades into habit when used, allowing the physician to focus on the unique reports not how to manipulate the program. This reduces frustration and improves the quality of the reports.
5.2 ReportGen Extensions

ReportGen has some aspects that could be extended. These additions would possibly make ReportGen a more effective tool. These fall into three general categories. First, the interface can be further enhanced. Second, the diagnosis ability can be implemented as an expert system. Lastly, the database can be linked to an online medical record.

5.2.1 Interface Extensions

The interface can be recreated in a design environment allowing the same design but overcoming the shortcomings of Authorware. This would improve the overall appeal of ReportGen by closing some of the user interface imperfections that are a by-product of the way Authorware is designed. In addition, ReportGen can be modified to handle a wider variety of report types than currently exist.

5.2.2 Knowledge Base Extensions

The database used to provide possible diagnoses could be implemented as an expert system knowledge base. Medical professionals could be employed to create a much more sophisticated set of rules allowing the program to provide a better and more accurate list of diagnoses and allowing the system to contain more current and constantly changing material. Authorware's networking ability provides a means for connecting to the large knowledge base and could be connected and used in an enterprise setting.

5.2.3 Medical Record Extensions

One of the time saving features of ReportGen is its ability to obtain the patient history and partially create the report from its database. This could be modified to connect directly to an online medical record system. Many changes to the way it retrieves the data would need to be made to accommodate the various online medical record
systems that are in use. This, however, would prove to be extremely beneficial to physicians wanting to move to an electronic solution.

5.3 Summary

The design of software is increasingly important in all professions including health care. For software to be accepted by mainstream physicians and clinicians there must be a fundamental shift in the view of software design. The software must address the workflow and concerns of the physicians. ReportGen provides examples of how to accomplish this. As times change and more people turn to software, hopefully ReportGen's care for design will be replicated and the people will be satisfied.
APPENDIX
ReportGen CODE DIAGRAMS

The first figure shown here is the main framework of ReportGen. The flowline is displayed and the two report sections are clearly visible.

Figure A-1. The main flowline for ReportGen. It shows the structure of the program.
A.1 Main Sections

The main sections initialize the database and present the form select screen.

A.1.1 Opening Screen

A.1.1.1 Install database

DBPath:=FileLocation^"RepGenDB.mdb"

--   dbReqType Arguments
--1  Add data source
--2  Configure (edit) data source
--3  Remove data source
--4  add a system DSN
--5  Configure a system DSN
--6  remove a system DSN
--7  remove the default DSN
DB ReqType:=4

--      Database Type (Simple Enough)
DB Type:="Microsoft Access Driver (*.mdb)"

--      List of arguments to send to database. MUST be separated by ';' semicolon
--      You should be able to send ANY list item recognized for ANY database to
--      setup a database in the ODBC Control Panel.
DB List:="DSN=Thesis Database;"--<--Notice it's terminated by a semicolon ;
DB List:=DB List^"Description=DB for Rick's Thesis;"
DB List:=DB List^"Access;"
DB List:=DB List^"DBQ="^DBPath

--      Execute the function
ODBCDriverInstalled:=tMsDBRegister(DB ReqType, DB Type, DB List)
dwFileAttributes:="32"
lpFileName:="RepGenDB.mdb"
result:=SetFileAttributes(lpFileName, dwFileAttributes)

A.1.1.2 Get All Symptoms

--a Select statement retrieves data from the database
--refer to the documentation that came with your database software to learn about SQL
queries
DB_SQLString:= "SELECT [Symptom Table].Symptom, [Symptom Table].[Body Region] FROM [Symptom Table]"
--NOTE: You must first install the 32-bit Microsoft Access drivers and set up the
students.mdb file as a 32-bit Microsoft Access data source.
--Name the data source Students Database or change its value in the lines below.

-- Open a connection to an ODBC data source.
DB_DatabaseName:="Thesis Database"
--Clear any error messages that might be in the error variable
DB_ODBCError:=""
--Open a handle to the data source. Pass the Authorware window handle, the variable you
want to use for error messages, the data source
--name, the database user name, and the password if any.
DB_ODBCHandle :=
ODBCOpen(WindowHandle,"DB_ODBCError",DB_DatabaseName,"admin",""

-- Send our SQL string the ODBC driver. Any data returned is assigned to the
DB_ODBCData variable.
DB_ODBCData := ODBCExecute(DB_ODBCHandle, DB_SQLString)

--If our error tracking variable contains anything, we assign it to the variable that
normally holds the data.
--If you don't assign the error message to another variable, it will be lost when the data
source is closed in the
--next calculation (if it closes successfully, there is no error). You might want to display a
different message to
--your users.
if DB_ODBCError <>"" then
  DB_ODBCData := "An error occurred. The ODBC driver returned the following error
  message: "^DB_ODBCError
end if

--Closes the data source and initializes the handle variable.
ODBCClose(DB_ODBCHandle)
Initialize(DB_ODBCHandle)

TheSymp :=DB_ODBCData
TotalSymptoms:=LineCount(DB_ODBCData)

A.1.2 Form Select

The following figures show the format of the form select section.
A.2 Patient Visit Report

The following figure shows the code segments and flowline capture for the patient visit report.

Figure A-3. Navigational framework for the patient report.
A.2.1 Get Primary Reason For Visit

The following figure shows the code segment for the patients primary visit entry.

Figure A-4. The flowline to get the primary reason for the visit.

A.2.2 Symptom Selection

The following figure shows the code segment for the symptom selection section.
Figure A-5. The symptom selection flowline with the flowline from the Head body region.

A.2.3 Select Diagnosis

The following figure shows the code segment for the diagnosis selection.
Figure A-6. Flowline for the select diagnosis screen.

A.2.3.1 Get Disease

DB_SQLString:="SELECT [Disease Table].Disease FROM ([Disease Table] INNER JOIN [Disease Subtable] ON [Disease Table].ID = [Disease Subtable].[Related Disease]) INNER JOIN [Symptom Table] ON [Disease Subtable].ID = [Symptom Table].ID WHERE ( "

First := 0
i := 1
repeat while i <= TotalSymptoms
if SymptomSelected[i] <> ""
if First = 0 then
First := 1
else
DB_SQLString:=DB_SQLString^"OR "
end if
DB_SQLString:=DB_SQLString^"([\text{Symptom Table}].\text{Symptom}) = "^\text{SymptomSelected[i]}^"\) "

--Generate the Report Part for Selected Symptom
ReportSymptoms := ReportSymptoms ^ \text{SymptomSelected[i]}^"\r"
end if
i := i + 1
end repeat

if First = 0 then
DB_SQLString:=DB_SQLString^"(([\text{Symptom Table}].\text{ID}) = 77)"
end if
DB_SQLString:=DB_SQLString^"
"

Format Results

temp:=DB_ODBCData
i:=1
DiseaseArray := Array("",LineCount(temp))
repeat while i<=LineCount(temp)
  tempLine:=GetLine(temp, i)
  j:= 1
  check := 0
  repeat while j<= LineCount(temp)
    if tempLine = DiseaseArray[j] then
      check := 1
      end if
      j := j +1
    end repeat
  if check = 0 then
    DiseaseArray[i]:=tempLine
  end if
  i:= i+1
end repeat

--Begin sorting additions
SFcountArray := Array(0,LineCount(temp))
i := 1
repeat while i <= LineCount(temp)
  SFcount := 0
  if DiseaseArray[i] <> "" then
    j=1
    repeat while j<= LineCount(temp)
      if DiseaseArray[i] = GetLine(temp, j) then
        SFcount := SFcount + 1
      end if
      j := j +1
    end repeat
  end if
  i:= i+1
end repeat
end if
j := j + 1
end repeat
end if
SFcountArray[i] := SFcount
i := i + 1
end repeat
--use selection sort
SFsize := LineCount(temp)
repeat while SFsize > 1
i := 1
SFmin := 1
repeat while i <= SFsize
if SFcountArray[i] < SFcountArray[SFmin] then
SFmin := i
end if
i := i + 1
end repeat
SFTemp := SFcountArray[SFsize]
SFtemp2 := DiseaseArray[SFsize]
SFcountArray[SFsize] := SFcountArray[SFmin]
DiseaseArray[SFsize] := DiseaseArray[SFmin]
SFcountArray[SFmin] := SFTemp
DiseaseArray[SFmin] := SFtemp2
SFsize := SFsize - 1
end repeat
--End sorting additions
DisplayResult := ""
i := 1
repeat while i <= LineCount(temp)
if DiseaseArray[i] <> "" then
DisplayResult := DisplayResult^DiseaseArray[i]^"r"
end if
i := i + 1
end repeat
DisplayResult := DisplayResult^"Other"

A.2.4 Treatment

The following figures shows the code segment for the treatment section.
A.2.5 Report

The following figure shows the code segment for the report section.

Figure A-8. Flowline for the report generation. Illustrating the patient report.
A.2.5.1 Get String to Save

ReportString := "Patient Report\r
Name: ^PatientName^\r
Medical Record Number: ^MedRec^\r
On ^FullDate^ ^PatientName^ visited complaining of ^Primary^\r
The symptoms that were indicated are listed below:\r
^ReportSymptoms^\r
The diagnosis determined was ^ReportDisease^\r
This condition is described as:\r
^ReportDescription^\r
The suggested treatment for this condition is as follows:\r
^ReportTreatment^\r

A.3 History and Physical

Code segments and flowline capture for the history and physical report.

![Figure A-9. A navigational framework for the history and physical.](image)

A.3.1 Demographics

The following figure shows the code segment for the demographics section.
Figure A-10. The flowline for the demographics section of the history and physical report.
A.3.2 History

The following figure shows the code segment for the history section.

Figure A-11. The flowline for the history segment of the report.
A.3.3 Review of Systems

The following figure shows the code segment for the review of systems section.

![Review of Systems](image)

Figure A-12. The flowline for the review of systems segment.

A.3.4 Abnormalities Description

The following figure shows the code segment for the abnormalities description.

![Abnormalities Description](image)

Figure A-13. The flowline segment for the abnormalities description.
A.3.5 Report

The following figure shows the code segment for the history and physical report.

![Diagram of the code segment for the history and physical report]

Figure A-14. The flowline from the report segment of the history and physical.

A.3.5.1 Get string to save

--For the back button
HPPage := "Report"

--Build report

HPReport := "Florida Surgical Center
History and Physical

Name: ^PatientName^ 
MR#: ^MedRec^ 

Procedure to be performed by: ^HPPerfBy^ Type of Anesthesia Planned: ^HPAnes
HPReport := HPReport^"ASA Class: ^HPASAClass
HPReport := HPReport^"Impression/Diagnosis: ^HPImpDiag
HPReport := HPReport^"Chief Complaint/Present Illness: ^HPChief
HPReport := HPReport^"History/Co-Morbid Conditions: ^HPHistory
HPReport := HPReport^"Relevant Past, Social, and Family Histories: ^HPPast
HPReport := HPReport^"Drug Allergies: ^HPDrugAller
HPReport := HPReport^"Previous Adverse Drug Reactions: 
if (HPAdvDrugRadioVal = 1) then HPReport := HPReport^"No"
else HPReport := HPReport^"Yes" ^HPAdvDrug

HPReport := HPReport^"Past Illness/Surgery: ^HPPast
HPReport := HPReport^"Relevant Diagnostic Studies: ^HPRelDiag
HPReport := HPReport^"Recreational Drugs/Alcohol/Tobacco: ^HPRAT
A.4 Subroutines

This section displays the important aspects of the subroutines. Namely how the SQL commands are sent to the database and the results received back by Authorware.

A.4.1 Execute SQL Command

![Execute SQL Command](image)

Figure A-15. The flowline called to execute the SQL query.

A.4.1.1 Access the database

--NOTE: You must first install the 32-bit Microsoft Access drivers and set up the students.mdb file as a 32-bit Microsoft Access data source.

--Name the data source Thesis Database or change its value in the lines below.
-- Open a connection to an ODBC data source.
DB_DatabaseName:="Thesis Database"
--Clear any error messages that might be in the error variable
DB_ODBCError:=""
--Open a handle to the data source. Pass the Authorware window handle, the variable you
--want to use for error messages, the data source
--name, the database user name, and the password if any.
DB_ODBCHandle :=
    ODBCOpen(WindowHandle,"DB_ODBCError",DB_DatabaseName,"admin",""
-- Send our SQL string the ODBC driver. Any data returned is assigned to the
DB_ODBCData := ODBCExecute(DB_ODBCHandle, DB_SQLString)

--If our error tracking variable contains anything, we assign it to the variable that
normally holds the data.
--If you don't assign the error message to another variable, it will be lost when the data
source is closed in the
--next calculation (if it closes successfully, there is no error). You might want to display a
different message to
--your users.
if DB_ODBCError <>"" then
    DB_ODBCData := "An error occurred. The ODBC driver returned the following error
message: "^DB_ODBCError
end if

--Closes the data source and initializes the handle variable.
ODBCClose(DB_ODBCHandle)
Initialize(DB_ODBCHandle)

A.4.2 Display Readme

The following figure shows the code segment for the readme display.

Figure A-16. The flowline segment for displaying the README file.
LIST OF REFERENCES


[Coo99] Cooper, A. *The Inmates are Running the Asylum*. Indianapolis, Indiana: Sams, 1999


[Ful01] Fu, L. Computer Applications to Medical Informatics. [http://www.cise.ufl.edu/~fu/Lecture/Medical/computer-medinfo.html](http://www.cise.ufl.edu/~fu/Lecture/Medical/computer-medinfo.html), 2001


BIOGRAPHICAL SKETCH

Rick Bean was born in Mount Kisko, New York on Aug 19, 1978. He lived in upstate New York for the first 5 years of his life. He then moved to Lake Mary, Florida. He graduated from Lake Mary High School with honors in 1996. He then attended the University of Florida where he received his bachelors degree in microbiology in 1999.

Rick currently works for the University of Florida department of Obstetrics and Gynecology, where he is a network administrator. He oversees much of the network used by the physicians. He enjoys working with the many networking components present in the department and at the remote sites.

Rick plans to move to Houston, Texas where he will take a position with ExxonMobil as a Windows NT network administrator.