A computerized tool for the collection and manipulation of physical therapy outcomes data

Doug Lawrence
The University of Montana

A COMPUTERIZED TOOL FOR THE COLLECTION AND MANIPULATION OF PHYSICAL THERAPY OUTCOMES DATA

By

Doug Lawrence

Presented in partial fulfillment of the requirements

for the degree of

Master of Science

The University of Montana

2000

Approved by:

[illegible]

Chairperson

[illegible]

Dean, Graduate School

5-30-2000

Date

Lawrence, Douglas C., M.S., May 2000 Computer Science

A Computerized Tool For The Collection And Manipulation Of Physical Therapy Outcomes Data (66 pp.)

Director: Dr. Alden Wright [illegible]

ABSTRACT

It is important in most any scientific discipline to review your work, to assess and evaluate any weak areas and fix them, as well as enhance those aspects that are strengths. Furthermore it is important that the practitioner have a method by which results can be tracked. This computer program was developed as an answer to this problem as presented by the Physical Therapy Department at the University of Montana as part of their Physical Therapy Outcomes program.

This paper presents in detail a project completed as part of the fulfillment for the University of Montana Computer Science Masters Degree. It details the steps taken to create a computerized tool for collecting patient data and responses and tracking these responses for future evaluation. Some of the development tools were the Visual Basic programming language, Universal Modeling Language (UML) and Structured Query Language (SQL). The following pages present the background of this endeavor, the methods used, and the final results of this project.

Table of Contents

BACKGROUND

In the Spring of 1999, the Physical Therapy Department of the University of Montana approached the Computer Science Department requesting help in completing a project on which two of their graduate students were working. These graduate students, Eric Mills and Sarah Pataya, were researching a new tool, called an Outcome Tool, the purpose of which was to "assess the efficiency and effectiveness of [physical therapy] treatment". This outcome tool is used to collect patient data in three key areas: clinical, financial and satisfaction outcomes. These results are then used to aid in the evaluation of the profession, develop expertise, and direct future financial decisions regarding the physical therapy program.

The problem they encountered was the storing and tracking of the results. They had developed several forms which were filled out by the patients containing data in the above three areas but realized that a stack of papers in a file drawer would not work well once they began to accumulate. To most computer science students it becomes a classic example of a database application.

Kathy Lockridge approached me about helping Eric and Sarah and in February a meeting was held to discuss their needs. They explained in general that what they wanted would essentially consist of a program that would 1) allow for entry of data collected from patient data and patient responses into a database and 2) allow for manipulation of this data such that teachers, educators and therapists could gauge how well their programs were working. For example, it would track the average costs of visits for an individual therapist. Different therapists could then be compared. It would also track satisfaction scores given by patients.

In March of 1999, during Spring Break, I wrote what is referred to as a “rapid prototype”. Because it only took a week and because there was no specification or design work, the program was functional but nowhere near perfect. After using the program for a while, several ‘bugs’ became evident. In the summer it was decided a better and more complete program was needed.

METHODOLOGY

One of the main reasons for choosing this project was to allow me to learn and apply the art of software design. Although this project was not specifically about the software engineering process it is important that the reader have a basic understanding of what it entails. The project further allowed for some practical experience in the use of what is known as Unified Modeling Language, or UML.

The software process is the way software is produced. The process begins by simply talking to the client (the person wanting the software) and determining what they want and/or need. This is called the Requirements stage and is in many respects the most important. What follows are several other stages including the Specifications Phase where it is determined what the product is supposed to do, the Design phase where it is determined how the product is to do it, and the Implementation phase where the ideas are coded into what is commonly referred to as 'code'.

The Unified Modeling Language (UML) is a notation (mainly diagrammatic) for modeling systems using object-oriented concepts. In the section that follows we will see how the diagrams created using UML became the final program.

In practice the Software Process and UML become intertwined. They are used in parallel in the development of the computer program. Whereas the software process is the art of designing a piece of software, UML is the method by which we can put these ideas on paper (or virtual paper, if you will).

The process of designing software programs can be likened to that of building a skyscraper: Just as a builder would never build a skyscraper without blueprints, a computer program shouldn't be written without blueprints. Now, although there are major differences in how a skyscraper is built and how a program is written, the basic analogy is true. A poorly designed building will collapse whereas a poorly designed software package will crash. Even now, as in the past, there are software engineers sitting down to write code without either the slightest idea of where to begin or worse, what the client needs. Developing a method for writing computer programs has been a major concern for the software industry and is supported by the government itself in the form of standards and regulations designers must meet.

There are several different ways in which the software process can be realized. These are called Software Lifecycle Models, which take the elements of the software process and (possibly) reorder them. For example, some programmer may decide to put the design phase directly after the specification phase with the understanding that there is "no going back" to change the specifications (The Waterfall Model). Others may choose to build the software incrementally where each build adds to what has been already created (The Incremental Model).

The model chosen for this project was a hybrid of several models. I used a little from the Rapid Prototype model, combined with the Incremental Model and an Object Oriented Analysis approach. The Rapid Prototype model allows for the collecting of requirements and the rapid development of a functional but not-quite-complete program that is subsequently given to the client to evaluate. Based on this evaluation a better and more complete program is developed. There are many advantages to using this approach, not the least of which is nailing down a functional interface that the client can use effectively. This would be likened to the architect building a small-scale model of the skyscraper. The Incremental Model allowed for working on pieces of the project in exclusion of the others.

An object-oriented analysis approach is an iterative process that allows for the modification of previously created documents and thus allows for change. It is not in the scope of this paper to explain Object Oriented Programming (OOP). In brief, however, OOP is a type of programming where nearly every element of the program can be identified as an object. That is, an entity in and of itself, having properties and methods or procedures that operate on those properties. "During object-oriented analysis, there is an emphasis on finding and describing the objects – or concepts – in the problem domain."

Hopefully with these brief descriptions we can proceed and not lose the reader completely.

THE PROCESS

Object Oriented Analysis consists of stages. These different stages combine to form what is termed a Development Cycle. As mentioned above, the idea is to iterate through this development cycle, until you have the product desired. As Larman puts it,

> "A strength of an iterative and incremental development process is that the results of a prior cycle can feed into the beginning of the next cycle. Thus subsequent analysis and design results are continually being refined and informed from prior implementation work." (Larman, pp. 297-8)

What follows is a breakdown of the phases involved in the creation of this program. It by no means contains all the phases of object oriented analysis and is therefore, not complete. Time constraints do not allow for a complete cycle, much less several iterations over this cycle. Furthermore, this paper is not meant to be a tutorial on how to develop software. However, as each phase is introduced a brief description of what that phase is and why it is used will be given.

The phases used were 1) Plan and Elaborate, 2) Analyze, 3) Design and 4) Test. As each phase is introduced and discussed, sample elements or artifacts of the design relative to that phase will be displayed or discussed. While some sample diagrams and forms are presented in the body of this paper and the appendices, a complete collection of all documents and diagrams can be found on the CD accompanying this paper in HTML format. To view them open the file entitled "Physical Therapy Project.html".

Plan and Elaborate:

In this phase we define requirements, implement a prototype, and develop a conceptual model.

Requirements: This portion was begun during the spring of 1999 when the first meeting was held with Eric and Sarah. A requirement meeting can be as formal or informal as desired since it consists of finding out what the client wants or needs. Several techniques can be utilized from simply asking questions to letting the client do all the talking. The designer needs to understand not only what is wanted but also what is currently being used and why it isn't working anymore. In the case of this project what was desired was a way to track data over time. The current method was via paper forms that, as noted above, could become unwieldy after a certain amount of time. A sample of the data collection forms can be seen in appendix I.

Most requirements documents are not developed in one meeting. In fact, three different meetings were held to discuss issues before the prototype was coded. The final requirements document can be seen in the appendix II. Although there is no absolute method for writing requirements, it was decided to use an outline approach for this program. A copy of the requirements document is in Appendix II. This document also contains the design glossary, the system attributes and some assumptions made about the requirements.

Requirements prepared correctly can be the cornerstone to a great piece of code. Persons skilled at finding out what is needed verses what is wanted can be a great asset to a software engineering firm. It is often the case that a bad program can be traced to misunderstood or unclear requirements.

Implementing the prototype: The prototype is a quick-and-dirty model of the final product. It is usually created hastily and contains many bugs. The prototype for this program was created in a period of five (5) days, with about forty (40) man-hours of work. After the prototype was coded more meetings were held to discuss problems and issues that needed to be addressed.

Rapid prototyping can be an integral part of the requirements phase. However, there are many issues involved with using a rapid prototype that need to be addressed, such as the fact that a prototype is just that, a prototype and not the real thing. Prototypes are usually shelved after the information needed from them is made known. In this case the prototype was not shelved but it wasn't used in its entirety either. Parts of it were used with most of it being significantly modified.

A working copy of the prototype is included on the CD at the back of this paper.

Conceptual Model: After the requirements are completed and (hopefully) understood, the programmer can start to make a list of concepts that are contained in the document. A concept is just what it sounds like: It isn't necessarily a real thing (i.e. – a portion of code) although it might be. It illustrates "meaningful [] concepts in a problem domain; it is the most important artifact to create during object-oriented analysis." (Larman, p. 85) It might include people that interact with the program as well as a database file containing people's names. A partial list of concepts for this program is as follows:

The User
Satisfaction database
Patient database
The data entry form
Reports generated by the program

Once a list of concepts is developed the idea is to figure out how these concepts interact. For example, how will the user use the database? Is there a direct interaction? Appendix I is the preliminary conceptual model for the physical therapy program. Much discussion should revolve around whether a concept should be included and whether more should be added.

In this design the conceptual model became the class diagram, which is discussed later, and can be viewed in appendix III, figure 2. Notice the addition of many (if not nearly all) procedures and attributes. In a real world situation the conceptual model would contain a nearly complete list of attributes, methods and procedures before coding begins. In the case of the Physical Therapy program many things were found wanting once coding started. In a perfect world what should happen when this occurs is that the coding should stop and a re-defining of the conceptual model be done.

Analyze:

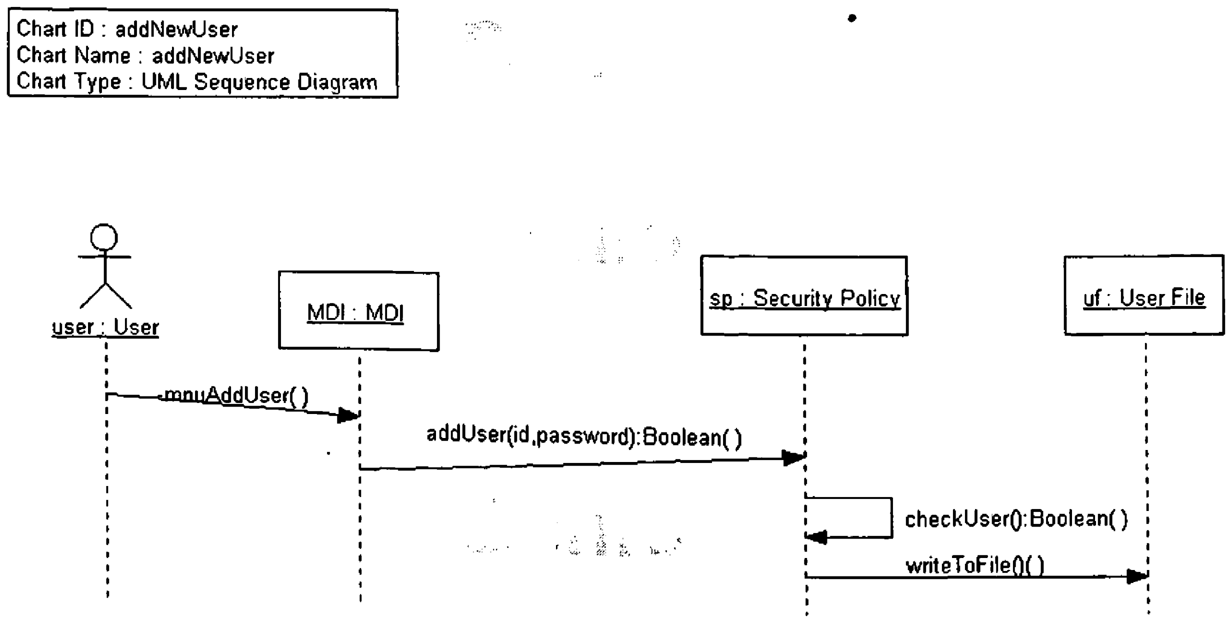
In this phase Use Cases and System Sequence Diagrams are created.

Use Cases: Use cases might be best described as 'how things might happen' during the use of the program. For example, in this program the user has to log in to gain access to the database. A use case tells us who is involved in this process as well as how the system reacts to the actor. The use cases can be read in Appendix IV.

Use cases can be ranked and ordered depending on the relative importance to the design. In our case, the use case involving data entry became very important and was thus a higher ranked use case.

System Sequence Diagrams: System Sequence diagrams illustrate events from users (actors) to the system (program). You can't really create System Sequence Diagrams without first creating Use Cases as they are created directly from them. "The UML includes System Sequence Diagrams as a notation that can illustrate actor interactions and the operations initiated by them." At this point we are not concerned with the implementation (the "how"), just that they will happen. This is often termed a "black box" approach.

Below is one of the System Sequence Diagrams developed for the Physical Therapy program.



The diagrams flow from top to bottom and left to right, just as you'd expect. The events are listed in the order they occur. System Sequence Diagrams consist of system events (those things generated by the user) and system operations (the action response from the system).

Again, System Sequence Diagrams are closely tied to the Use Cases and theoretically you would have a diagram for each Use Case.

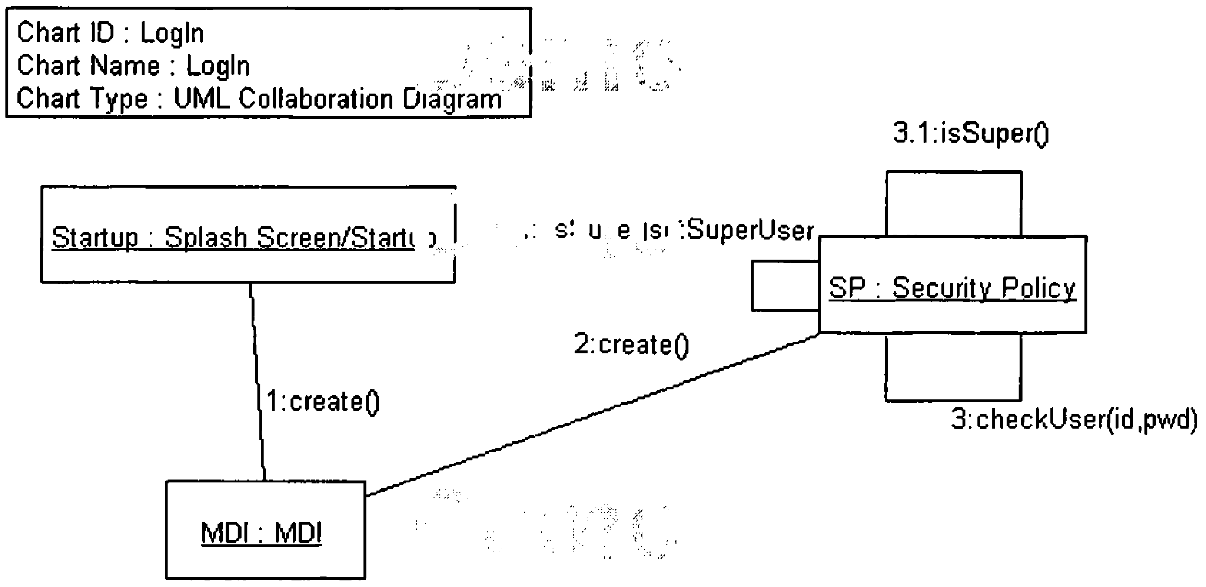
Design:

In this phase the Collaboration Diagrams are created and a Class Diagram is built. These objects are built simultaneously.

Collaboration Diagrams: Much can be shown with graphs and that is essentially what a Collaboration Diagram attempts to do. Collaboration Diagrams illustrate object interactions in a graph format. Theoretically at this point the programmer has a pretty good idea as to what classes will be included in the final project. At the least, the final list of concepts has been created and the programmer can now define exactly how one concept will collaborate with another.

Collaboration Diagrams illustrate the passing of messages between two concepts or classes. They are very much like the Sequence Diagrams discussed above. Collaboration Diagrams tend to be more expressive and can convey more information quickly than the Sequence Diagrams.

Below is one of the collaboration diagrams for this program.



As one might guess, the events occur in the order in which they are numbered.

Class Diagram: The Class Diagram is the logical extension of the conceptual model. Whereas the conceptual model represented ideas and concepts that may or may not be real, the Class Diagram is a definition of classes as software components. The Class Diagram contains all the important methods and procedures for a given class as well as its attributes. Furthermore it graphically conveys the flow of the program. Where does it start? What object creates another object? How does that object get information from that object?

The class diagram for the Physical Therapy project is quite large and can be viewed in appendix III, figure 2.

This completed the design portion of the project. Once again, in the real world, there is no end to this part of the job. A good programmer would be updating the Class and Conceptual Diagrams as more concepts became apparent. Additional Use Cases would be developed and Collaboration Diagrams written.

The point is that you are not bound to the paperwork. The beauty of the iterative, object-oriented approach is that the programmer can go back and change things, which, if done correctly and logically, will result in a well written, documented, robust piece of software.

THE PROGRAM

Ideally, the design generated during the software process will translate into a workable program. In this section I will explain what type of program this is and take the reader through a basic use of the Physical Therapy Program.

Computer software it is often classified by its architecture. That is, what are the component parts of the system? How does one part relate to the other? The Physical Therapy program here is an information system, meaning it needs to handle and store information. A 2- or 3- tier design would be a good choice.

In a 3-tier architectural design there are 3 layers – the Presentation (the things the user sees), the Application Logic (the rules that govern the process) and the Storage (such as a database or record file). In an object oriented world these three pieces can be written and developed separately, combining them at the end. A programmer may design the structure of the database, not really knowing how the user interface programmer will use it. All that needs to be handled up front is how these layers will communicate.

In a 2-tier design we erase the division between the Application Logic and Presentation layer, combining the two. A disadvantage of this design is the "inability to represent application logic in separate components". This can make using the same code later a more difficult task.

The Physical Therapy Program became a combination of the two types of design. This is possible because of the way Visual Basic, the language chosen for development, is itself designed. You can develop a completely object oriented program in Visual

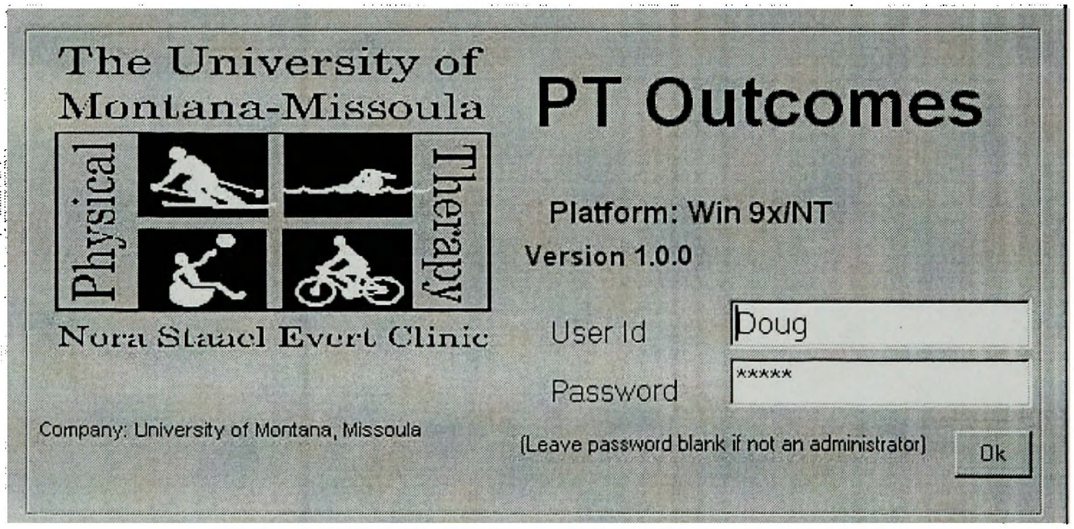
Basic. In fact, some authors claim that it is "intuitively object oriented. In fact, it's difficult to think of using Visual Basic [] without using one type of object or another." (Spenik, p. 387) This becomes most evident in the use of the Form objects, of which this program makes much use. A Form is certainly an object, which lends itself very well to user interface design, but it also can contain much of the application logic for the program. Not that it must contain the logic just that it lends itself to that task. In fact, it is the way most books teach Visual Basic, starting with the Form object and adding code to it. A good programmer will learn to separate the Forms (Presentation Layer) and the code (Application Logic) as he/she progresses.

Similarly, it was desired that we maintain what is called the Model-View Separation pattern. This pattern says that the logic of a program should be separated from the presentation portion. There are many good reasons for using this pattern of program design, and I tried to make use of it. But for some of the reasons noted above and due to programmer naivety, it was not strictly adhered to. One of the unpleasant results of not following the Model-View separation pattern is high coupling in code.

Sample Run-through

A basic run-through of the program consists of the following steps.

The user begins the program by executing the PTOutcomes executable. This brings up the combination Splash/Login screen. A Splash screen usually tells the user what program they're running, a version number, and essentially anything interesting the designer wants known. A login screen is just what it sounds like. Clicking on 'Ok' causes the users id and password to be checked. Administrators are allowed read/write access; everyone else only read access. Below is this Form.



PT Outcomes

Platform: Win 9x/NT
Version 1.0.0

User Id: Doug
Password: *****

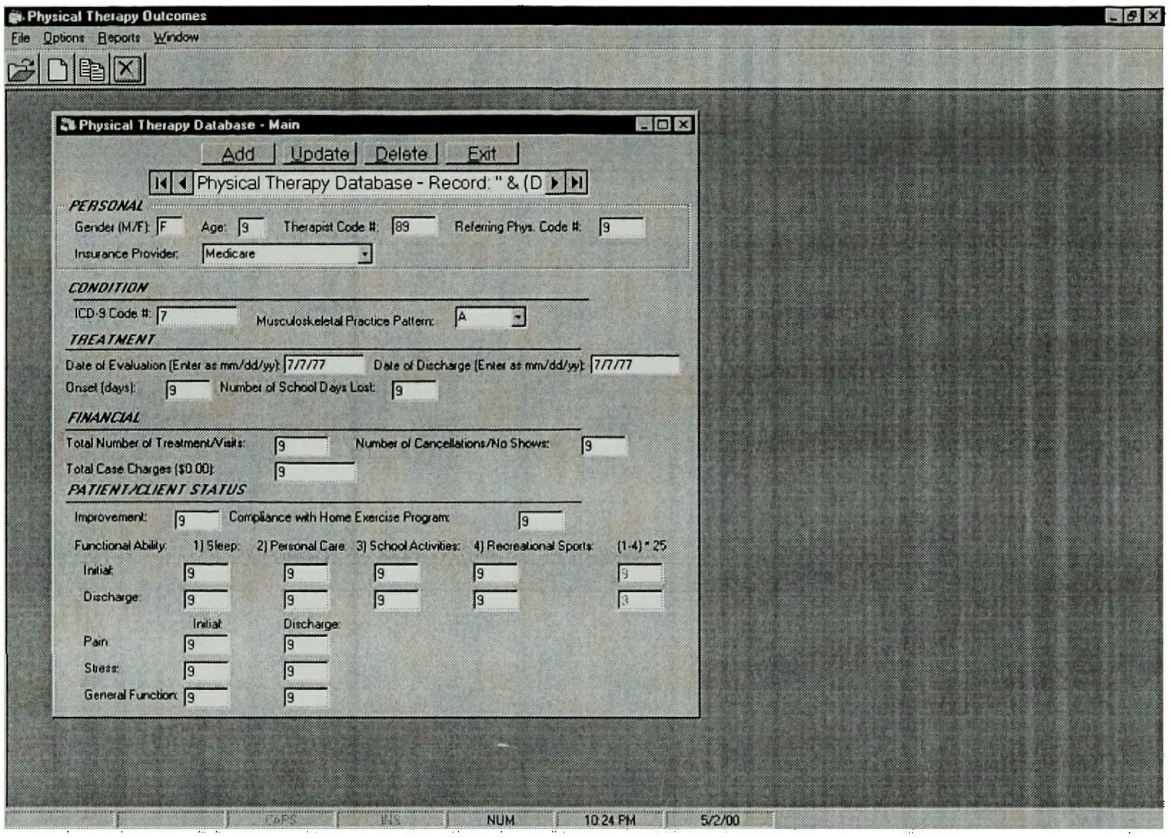
Company: University of Montana, Missoula

(Leave password blank if not an administrator)

Once the user has been verified, the main Form appears. This Form is what is called a Multiple Document Interface (MDI) Form, as opposed to a Single Document Interface (SDI). Most Windows applications use an MDI Form, which simply means you have one big Form in which all the others are displayed. In Visual Basic you can only have one MDI Form. This Form presents the user with the main menu consisting of the basic File, Options, and Reports options.

At this point the user can choose to enter new data, create a new database, or if an administrator, add other administrators. The most common task is data entry. If chosen the data entry Form is displayed. This Form allows for entry of all patient data from the paper forms as well as updating of current patient information.

Below is the MDI Form containing the Main Data Entry Form.


Physical Therapy Outcomes
File Options Reports Window
Physical Therapy Database - Main
Add
Update
Delete
Exit
Physical Therapy Database - Record: " & (D
CoPS
INS
NUM
10:24 PM
5/2/00

PERSONAL

Gender (M/F): F Age: 9 Therapist Code #: 89 Referring Phys. Code #: 9

Insurance Provider: Medicare

CONDITION

ICD-9 Code #: 7 Musculoskeletal Practice Pattern: A

TREATMENT

Date of Evaluation (Enter as mm/dd/yy): 7/7/77 Date of Discharge (Enter as mm/dd/yy): 7/7/77

Onset (days): 9 Number of School Days Lost: 9

FINANCIAL

Total Number of Treatment/Visits: 9 Number of Cancellations/No Shows: 9

Total Case Charges ($0.00): 9

PATIENT/CLIENT STATUS

Improvement: 9 Compliance with Home Exercise Program: 9

Functional Ability:	1) Sleep:	2) Personal Care	3) School Activities:	4) Recreational Sports:	(1-4) * 25
Initial:	9	9	9	9	9
Discharge:	9	9	9	9	3

	Initial:	Discharge:
Pain:	9	9
Stress:	9	9
General Function:	9	9

Of all the Forms in this program the data entry was the most difficult to create. The user interface was relatively easy but the data manipulation portion took some time. Visual Basic gives the option of using Microsoft's data control, which alleviates much of the dreariness of programming with databases. The flipside is that you must learn about the underlying concepts involved, namely Recordset objects, Structured Query Language (SQL) and a little Active Data Objects (ADO). As a side note, Microsoft utilizes these concepts throughout its Visual Studio programs including Visual J++ and C++.

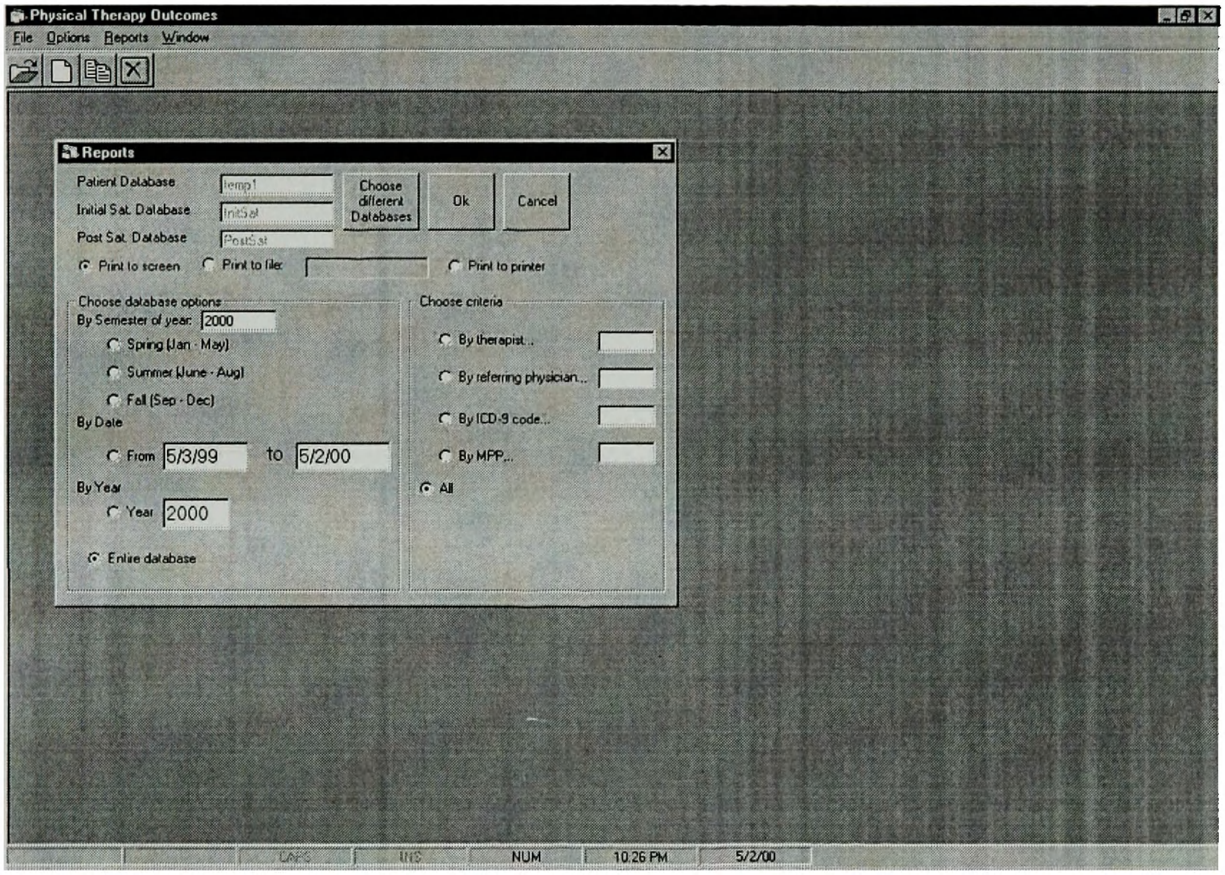
Once the user is done entering patient data, he may choose to enter satisfaction data. In the prototype version of the program the patient and satisfaction data was entered on the same Form. However, the client decided later to separate how these pieces of information were collected during treatment. Namely, it was decided that a patient's satisfaction results should be kept separate from their personal data for privacy reasons. In the new version of the program there are three databases – A patient database, an Initial Satisfaction database, and a Post Satisfaction database. The data entry person can choose which database to utilize depending on what data he needed to enter. The File menu gives options for entering data and creating databases for all three options.

One of the more complicated issues to solve in the development of the program was data validation. There are a total of thirty-two (32) pieces of data in the patient form and 8 on each satisfaction form. In order to make sure data gets manipulated correctly and to make a robust piece of software, a design decision was made that data would not enter the database itself until it was in the correct form. This means that dates must be actual dates and numbers range-checked for validity. Almost every data item needed to be checked.

The problem was solved in this way: When the user moves from one text box to the next, the data in the box is checked for validity. If it is not valid a message appears and the offending box gets the focus. There is a double check on some boxes in that the data control has a validate event which double-checks the boxes whenever the user clicks on any of the move or update buttons.

Another issue was the structure of the database. In the prototype the database fields contained different types, meaning that a field might contain the date type while the next one a String type. This turned out to be a problem during validation so in the final version all fields contain either the String or Single type. Dates are cast into dates when date manipulation is desired.

After entry of any data the user might progress to the report generation screen.



Patient Database: temp1
Initial Sat. Database: InitSat
Post Sat. Database: PostSat

[✓ Print to screen] Print to file: Print to printer

Choose database options
By Semester of year: 2000
Spring (Jan - May)
Summer (June - Aug)
Fall (Sep - Dec)
By Date
From 5/3/99 to 5/2/00
By Year
Year 2000
[✓ Entire database]

Choose criteria
By therapist...
By referring physician...
By ICD-9 code...
By MPP...
[✓ All]

This screen allows for selection of the range of records to include in the outcome summary and how this summary should be created (File, Screen, Printer). The prototype made an attempt at giving the user a choice but the method seemed clumsy and confusing. After discussing this with the client a more exact requirements definition was developed.

Once ‘Ok’ is pressed an Outcome Summary object is instantiated. There are many properties that the Report Choice Form sets in this object, such as the type of output desired as well as the range of records. This object generates the appropriate recordset object, which is then manipulated. The report is generated according to the users preferences.

A sample of an Outcome Summary for the entire database is in Appendix III.

At this point the user can repeat any of the above steps or quit the program.

CONCLUSIONS

In this section I will make some conclusions about the process used to develop this software as well as some of the things I might do differently. Most of the elements that are left undone or that lack in completeness are due to a combination of a lack of time and lack of complete understanding of Visual Basic.

For the most part the written program translated very well into the written code. I was determined not to begin writing code until I had completed the paperwork. I discovered that I was running into mental blocks concerning the concepts and how they fit together. There was also the time issue to consider so even though I could have spent a considerable amount of time refining the diagrams and models, I decided to begin coding. I believe one could have made a complete project out of just the UML portion of this project.

Developing the conceptual model proved to be the most beneficial element of coding. It helped me see what classes or Forms needed to be created and how they needed to communicate. I believe very strongly that doing the up-front work created a better, more robust program. It also contributed to faster development.

One of the frustrating elements of the project was that often, after I had completed a section of code, I would discover a new and possibly better method of doing it. As I was doing the coding for the program I was also reading and doing some research on the Visual Basic programming language. Not just for this project but also for some of the students I had in a class I was teaching. I suppose that this happens to all beginners in any discipline similar to this one. I found myself fighting the desire to go back and do things differently.

One of the things I would do differently is to make the program even more object-oriented. For the most part it is centered around objects and manipulation of their properties but there are many aspects that could be developed further. For example, there are many properties and methods associated with the MDI Form that I did not use. For example the Forms collection allows the programmer to keep track of windows/forms that are open. It was while I was reading about collections that I further discovered Visual Basic does not handle garbage collection, as does Java. Using the collection object allows one to destroy all the Forms at once.

Another aspect of the program I might do differently is in the use of the Windows Registry. I found many bits of information on different web sites about how to utilize it but felt I didn't have the time to include it. (Maybe part of that was the fear of messing up my own computer's Registry!) There are several things I could use the Registry for, the most obvious being the opening of recently used files. You could program this using text files but most programs for which I have seen the code use the Registry to store a most-recently-used list of files.

By far the most interesting and maybe frustrating thing that happened while working on this project happened about two weeks after completion of the code. While helping a fellow employee with the Microsoft Access program, a program with which I had very little familiarity, I began to see how I could have written the whole program using Access. Access utilizes Visual Basic to do much of the coding but more importantly it allows one to develop Forms and manipulate databases with little effort. Although I learned quite a bit about using recordsets and databases using Visual Basic itself, I essentially wrote from scratch what was already being used in Access.

In developing reports I utilized three different sub procedures. Although this works as expected it isn't very elegant. All three procedures do the same thing, they just send their results to different output devices. It might be better to create the report as a text file and then use it for printing. If the user had chosen not to print the report the file could just be deleted.

I have also discovered how to use pop-up menus which, although not really necessary, would add a little completeness to the program.

FUTURE ENHANCEMENTS

In this section I will present some items that might be added to the program at a later date.

In the requirements meeting we discussed having the program work in a networked environment. I never implemented this into the code but it wouldn't take too much effort. Most of the work involves keeping databases open exclusively for reading or writing, much of which is handled by default in Visual Basic. Access to the server would also be needed.

The format of the reports seems very list-like. In looking at how a different school developed reports for a similar tool it is obvious that more could be done in this area. For example it would be nice to break up the results into more categories and do a little more analysis. This would require additional help from the Physical Therapy department as to what they would like to see.

The development of some graphs would be an excellent addition to the program, allowing the user to graphically view trends in certain areas. Visual Basic has at least two different ways by which a programmer could do this using built in controls.

Overall it would be nice to make the program adhere to the Windows standard of doing things. By this I mean making menus and any other visual items similar to what is seen in other Windows programs. I've tried to do this as much as possible but there is room for more work in this area.

References

Larman, C., (1998). Applying UML and Patterns. Prentice Hall (3,4,5,6,7)

Mill, E, Pataya, S. (1999) The University of Montana Outcome Tool, Mills, Pataya (1)

Mullet, Kevin, Sano, Darrell (1995) Designing Visual Interfaces. SunSoft Press, Prentice Hall publishing

Schach, S.R., (1998). Classical and Object Oriented Software Engineering. Vanderbilt University, Mcgraw Hill (2)

Spenik, M, Indovina, A, Jung, D, Boutquin, P (1999). Teach Yourself Visual Basic 6 Online. Sams publishing (8)

APPENDIX I

Below are the data collection forms used by the Physical Therapy Department to gather patient data.

The University of **Montana**

Nora Staael Evert Physical Therapy Clinic

Patient/Client Self Assessment (Discharge Visit)

Please take a moment to fill out the following form. Please answer **all** of the questions to the best of your ability.

________________________________ ________________
Name Date

School Days Lost: Approximately how many days were you unable to attend class due to your condition? ____________

Improvement:
On the scale below, please rate the amount of improvement in your condition since the beginning of your physical therapy treatment.
Please mark on the line with a slash (/).

No improvement ________________________________ Complete recovery

Home Program

On the scale below, please rate your adherence with the home program given to you by your physical therapist.

____
I never performed my home program I performed my home program regularly as instructed

Functional Ability

For each question, please rate your current condition on the scales below.

1) Do you have difficulty sleeping?

No difficulty ________________________ My sleep is completely disturbed

2) Do you have difficulty with personal care (dressing, washing, grooming, etc.)?

____
No difficulty I am completely dependent on others for my personal care

3) Do you have difficulty with school activities (walking to class, sitting, working at a computer, etc.)?

____
No difficulty I am unable to perform any school activities

4) Are you able to engage in recreational/sport activities as you did prior to your injury?

I am able to participate in my sport ________________ I am unable to participate fully

Pain: *On the scale below, please rate your worst pain in the last 48 hours.*

No pain ________________ Worst pain imaginable

Stress: *On the scale below, please rate the amount of stress currently in your life.*

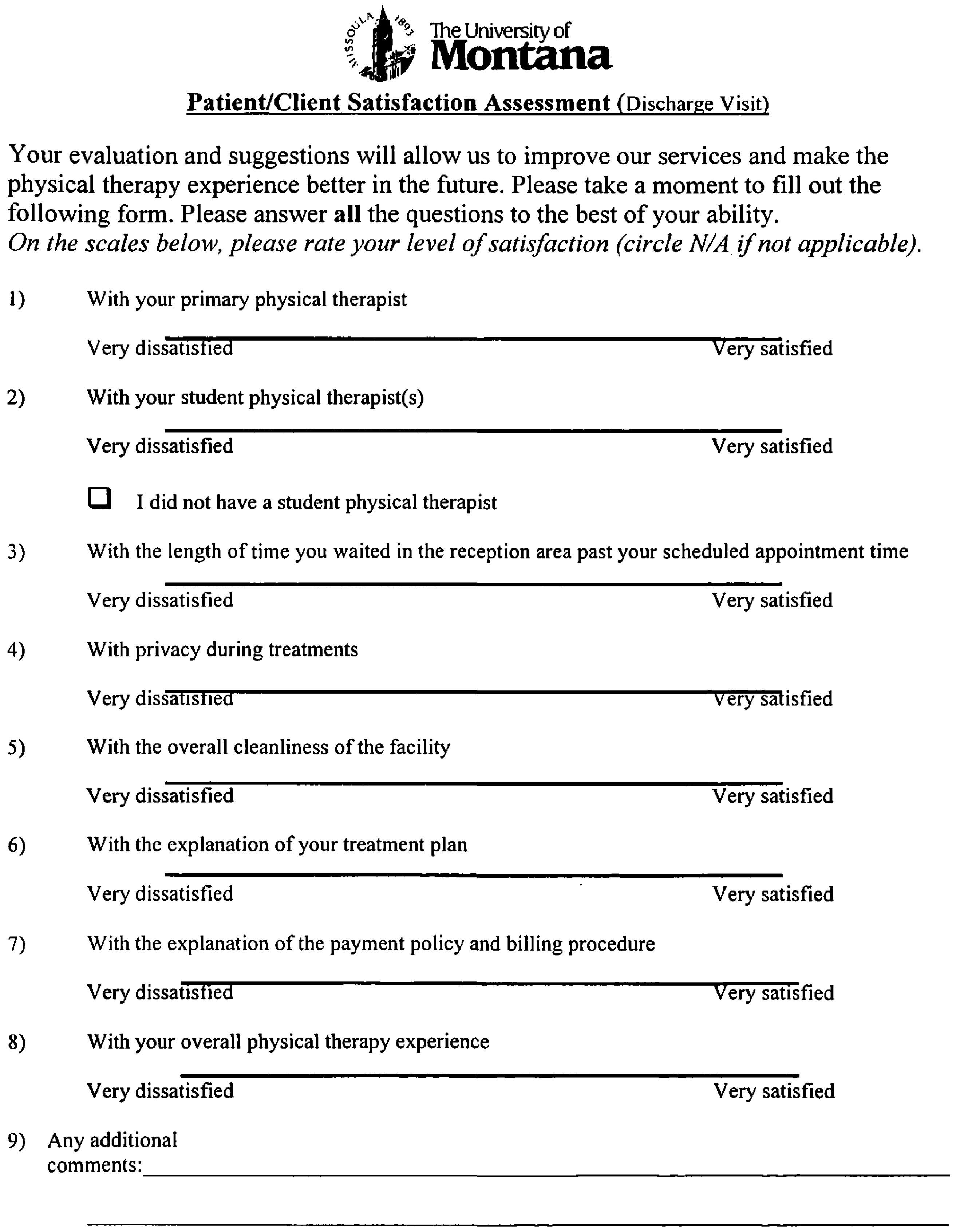
No stress ________________ Worst stress imaginable

General Function: *On the scale below, please rate your current level of overall functioning*

____

I am functioning I am at the worst level at my pre-injury level
of functioning since my injury

Figure 1



The University of Montana

Patient/Client Satisfaction Assessment (Discharge Visit)

Your evaluation and suggestions will allow us to improve our services and make the physical therapy experience better in the future. Please take a moment to fill out the following form. Please answer **all** the questions to the best of your ability.
On the scales below, please rate your level of satisfaction (circle N/A if not applicable).

1) With your primary physical therapist

Very dissatisfied ________________________________ Very satisfied

2) With your student physical therapist(s)

Very dissatisfied ________________________________ Very satisfied

☐ I did not have a student physical therapist

3) With the length of time you waited in the reception area past your scheduled appointment time

Very dissatisfied ________________________________ Very satisfied

4) With privacy during treatments

Very dissatisfied ________________________________ Very satisfied

5) With the overall cleanliness of the facility

Very dissatisfied ________________________________ Very satisfied

6) With the explanation of your treatment plan

Very dissatisfied ________________________________ Very satisfied

7) With the explanation of the payment policy and billing procedure

Very dissatisfied ________________________________ Very satisfied

8) With your overall physical therapy experience

Very dissatisfied ________________________________ Very satisfied

9) Any additional comments:____

____

Figure 2

APPENDIX II

Requirements Document

Overview Statement

The purpose of this program is to allow the client to enter data collected from paper forms filled out by Physical Therapy patients and subsequently manipulate this data. The results are used to evaluate the effectiveness of the physical therapy given to the patient as well some of the costs associated with providing Physical Therapy. The program will also be used to show clients (i.e. - physicians) the benefits of performing Physical Therapy.

Customers

Brenda Mahlum - Physical Therapy Instructor
Dave Levison - Physical Therapy Instructor
The University of Montana Physical Therapy Department

Goals

The goal of this program is to allow clients to view collected data in a manner that allows them to make critical evaluations of the Physical Therapy methods in use, provide better care to patients, and determine the costs involved in providing Physical Therapy.

System Functions

This section is broken down into four categories:

1. Input of data
2. Manipulation of data
3. Output of results
4. The user interface (GUI)

Note: The data collection form is the form from which the user will enter data into the database. The data collection form is a paper form created using 2 other forms. See attachment 1 and glossary.

1. Input of data
   - 1.1 Enter Patient Personal data from data collection form
     - 1.1.1 Gender (M or F)
     - 1.1.2 Age (Range from 0 to 150)
     - 1.1.3 Therapist Code Number (Range from 1 to infinity)
     - 1.1.4 Referring Physician Code Number (Range from 1 to infinity)
     - 1.1.5 Insurance Provider (One of these: Self Pay, Private Insurance, Workman's Comp., MVA, Medicaid)
   - 1.2 Enter Patient Condition data from data collection form
     - 1.2.1 ICD-9 Code Number (Positive number, possibly with decimal places)
     - 1.2.2 Musculoskeletal Practice Pattern (Range from (A to K) or Other)
   - 1.3 Enter Patient Treatment information from data collection form
     - 1.3.1 Date of Evaluation (mm/dd/yyyy format)
       - 1.3.1.1 Verify that value entered is in valid date format before allowing further input.
         - 1.3.1.1.1 mm/dd/yyyy
         - 1.3.1.1.2 mm/d/yyyy
         - 1.3.1.1.3 m/dd/yyyy
         - 1.3.1.1.4 m/d/yyyy
         - 1.3.1.1.5 mm/dd/yy
         - 1.3.1.1.6 mm/d/yy
         - 1.3.1.1.7 m/dd/yy
         - 1.3.1.1.8 m/d/yy
       - 1.3.1.2 Assume that '00' is equal to '2000' in reference to a year. (As opposed to 1900).
       - 1.3.1.3 A 2-digit date will be interpreted this way:
         - 1.3.1.3.1 40 to 99 will be 1940 to 1999
         - 1.3.1.3.2 00 to 39 will be 2000 to 2039
     - 1.3.2 Date of Discharge of Last visit (mm/dd/yyyy format, or other legal date format)
       - 1.3.2.1 See 1.3.1.3 regarding date issues.
     - 1.3.3 Onset of symptoms - days (Range from 0 to infinity)
     - 1.3.4 Number of School/Work Days Lost (Range from 0 to infinity)

1.4 Enter Patient Financial information from data collection form

1.4.1 Total Number of Treatments/Visits (Range from 0 to infinity)

1.4.2 Number of Cancellation/No-Shows - days (Range from 0 to infinity)

1.4.3 Total Case Charges (Dollar amount - currency format)

1.5 Enter Patient Status from data collection form
 1.5.1 Improvement (0-100)
 1.5.2 Compliance with Home Exercise Program (0-100)
 1.5.3 Functional Ability
  1.5.3.1 Initial (Average of the following four values)
   1.5.3.1.1 Sleep (0-100)
   1.5.3.1.2 Personal Care (0-100)
   1.5.3.1.3 School Activities (0-100)
   1.5.3.1.4 Recreational Sports (0-100)
   1.5.3.1.5 Verify all four values are between 0 and 100, compute average.
   1.5.3.2.1 Allow for 'no answer'. That is, the average is the average of only those entered.
  1.5.3.2 Discharge (Average of the following four values)
   1.5.3.2.1 Sleep (0-100)
   1.5.3.2.2 Personal Care (0-100)
   1.5.3.2.3 School Activities (0-100)
   1.5.3.2.4 Recreational Sports (0-100)
   1.5.3.2.5 Verify all four values are between 0 and 100, compute average.
   1.5.3.2.6 Allow for 'no answer'. That is, the average is the average of only those entered.
 1.5.4 Pain
  1.5.4.1 Initial (0-100)
  1.5.4.2 Discharge (0-100)
 1.5.5 Stress
  1.5.5.1 Initial (0-100)
  1.5.5.2 Discharge (0-100)
 1.5.6 General Function
  1.5.6.1 Initial (0-100)
  1.5.6.2 Discharge (0-100)

1.6 Enter Patient Satisfaction Scores from data collection form (Will be on a separate GUI)
 1.6.1 Eight (8) entries, each ranging from 0 - 100.
  1.6.1.1 Allow for 'no answers'. Need to keep track of these when averaging.

1.7 Write entered data to SQL database
 1.7.1 Write values to database

1.8 All data should be range-checked and corrected before allowing further input.
 1.8.1 The user cannot move to a new item until the current item has been entered correctly. For example, the date must be in the correct form before the functional ability can be entered.

1.9 There will be two types of users - Those that can enter data AND print reports and those that can only print reports. There should be some method of allowing/disallowing users to certain aspects of the program.

1.10 There needs to be some sort of help on the screen in regards to how to use the program. For example, how to input data, how to move through the screen, etc.

1.11 Each input should have a record number which can be associated with a paper copy of the data collection form.

1.12 Must be able to move through records.

1.13 Must be able to change/modify records that have already been entered.

1.14 Must be able to enter new records.

2. Manipulation of Data

2.1 Must be able to move through records.
2.2 Must be able to change/modify records that have already been entered.
2.3 Must be able to enter new records
2.4 Allow user to choose from more than one database (optional)
2.5 Create outcome summary
    2.5.1 Choose type of summary from list
        2.5.1.1 By semester
            2.5.1.1.1 By therapist
            2.5.1.1.2 By referring physician
            2.5.1.1.3 By ICD-9 code
            2.5.1.1.4 By MPP
        2.5.1.2 By Year
            2.5.1.2.1 By Therapist
            2.5.1.2.2 By referring physician
            2.5.1.2.3 By ICD-9 code
            2.5.1.2.4 By MPP
        2.5.1.3 By entire database
            2.5.1.3.1 By Therapist
            2.5.1.3.2 By referring physician
            2.5.1.3.3 By ICD-9 code
            2.5.1.3.4 By MPP
    2.5.2 Compute total number of visits
    2.5.3 Compute number of males/females
    2.5.4 Compute number of visits per therapist
    2.5.5 Compute number of new evaluations (total number of records)
    2.5.6 Compute number of new referrals per referring physician
    2.5.7 Compute average total case charges
    2.5.8 Compute average charge per visit (case charges/visits)
    2.5.9 Compute average charge per visit per therapist
    2.5.10 Compute number of cancellations/no shows
    2.5.11 Compute number of patients per type of insurance
    2.5.12 Compute average percent change in functional ability
    2.5.13 Compute average percent change in general function
    2.5.14 Compute average percent change in pain
    2.5.15 Compute average percent improvement
    2.5.16 Compute average number of school/work days lost
    2.5.17 Compute average days from onset
    2.5.18 Compute average satisfaction for each satisfaction question 1-8
    2.5.19 Compute average overall satisfaction
    2.5.20 Compute average number of days from evaluation to discharge

3. Output of Results

3.1 Screen

3.1.1 Print heading for report based on choice in 2.5.1

3.1.2 Print out computations from 2.5 to screen.

3.2 Printer

3.2.1 Determine printer options with common dialog box

3.2.2 Print heading for report based on choice in 2.5.1

3.2.3 Print out computations from 2.5 to printer.

3.3 File

3.4.1 Allow report to be output to a comma-delimited text file for import into other programs.

3.4.2 Create heading for report based on choice in 2.5.1

3.4.3 Ask user for name for file.

3.4.4 Write report to file, notifying user of name and location of file when done.

4. The user interface

4.1 Allow for the separation of the data entry GUI into two sections

4.1.1 Entry of data collected from data collection form.

4.1.1.1 Bring up separate GUI for data entry.

4.1.2 Entry of data collected from satisfaction questions.

4.1.2.1 Bring up separate GUI for data entry.

4.2 Each GUI should consist of areas to enter each of the data values collected on the data collection form.

4.2.1 Text boxes for numerical input.

4.2.2 Drop down or combo boxes for entries with limited choices. (i.e. - MPP)

4.3 Each data entry object should have some sort of explanation. (Label)

4.4 The satisfaction questions are not necessarily tied to the patient information.

4.5 The report screen should be separate from the data entry screen(s).

4.5.1 Allow for choice of report type. (See 2.5.1)

System Attributes

1. The program will run on Windows 95/98/NT machines.
2. The program will be a client/server application which means that the program itself will reside on individual computers while the main database will reside on a server. Write access to this database will be restricted. Only those users designated by Brenda or Dave will be allowed to modify the database.
3. The input screen should have 'helps' displayed that make it obvious how data is to be entered and how to maneuver through the screen.
4. Upon encountering invalid data, the program should inform the user in a clear way what the problem was. Although the data cannot be entered incorrectly (in theory) it may encounter bad data after it has been entered. (Power fluctuations, bad hard drives, etc..). The user should be allowed to view the offending data and correct it.
5. After completion of the program, the Physical Therapy Department should have access to support. That is, if a problem with the program arises, the writer of the program should be available to fix it. A reasonable amount of time would be assumed. For example, for 1 full year after delivery I would be available at no cost.
6. There should be some sort of written documentation. One set should consist of the source code for future students and the other should be a manual on how to use the program.

Glossary

Record - One complete entry in the SQL database. Includes all the information available from the data collection form.

SQL Database - A type of database that allows the programmer to perform SQL type queries. For example, it allows for selection of specific records, excluding others.

Onset - The number of days from when the malady was first detected.

MPP - Musculoskeletal Practice Pattern

Referring Physician - The number of the physician who referred this patient.

Functional Ability - A measure of how able the patient is to participate in a certain activity such as sleeping and school activities.

Therapist Code - A number assigned to a given therapist.

Comma-delimited file - A file in which each element is separated by a comma.

Graphical User Interface (GUI) - The collection of text boxes, buttons and graphical images that act as the interface between the user and the program and allow for interaction. For example, the text boxes that allow for input of data from the data collection forms are part of the GUI.

Assumptions

- We assume that the data on the data collection form is accurate to the extent that it is used for this program.

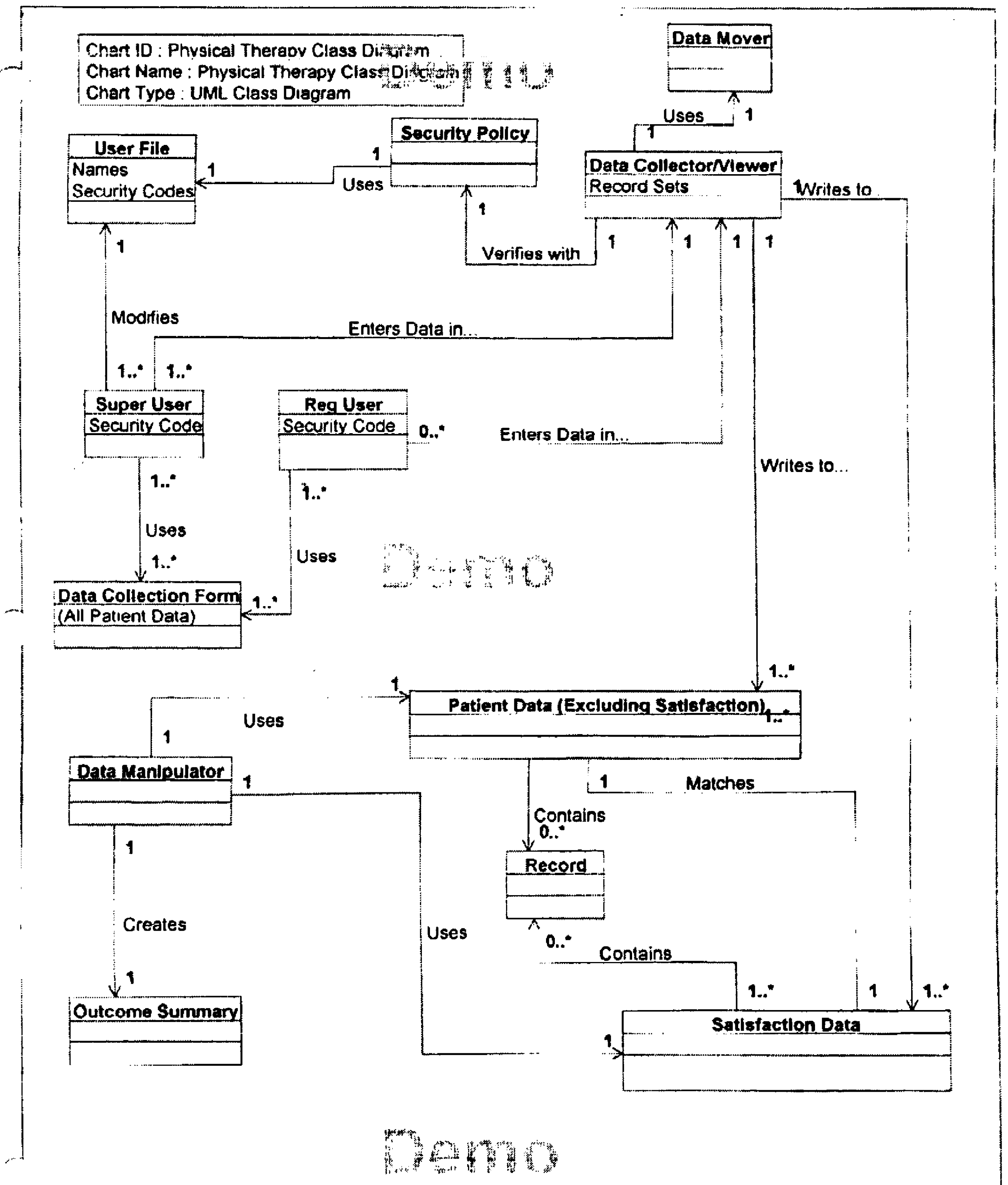
- We assume the users of this program are familiar with the Windows working environment. For example, how to use a mouse or close a window.

- We assume that the program will be run from the hard drive of the computer and not from the floppy drive.

Appendix III

Below is the preliminary conceptual model for the Physical Therapy Program:

Class Diagram: Physical Therapy Class Diagram



Date: Feb 23, 2000 *Page: 1 of 1* *Time: 8:46:04 AM*

Figure 1

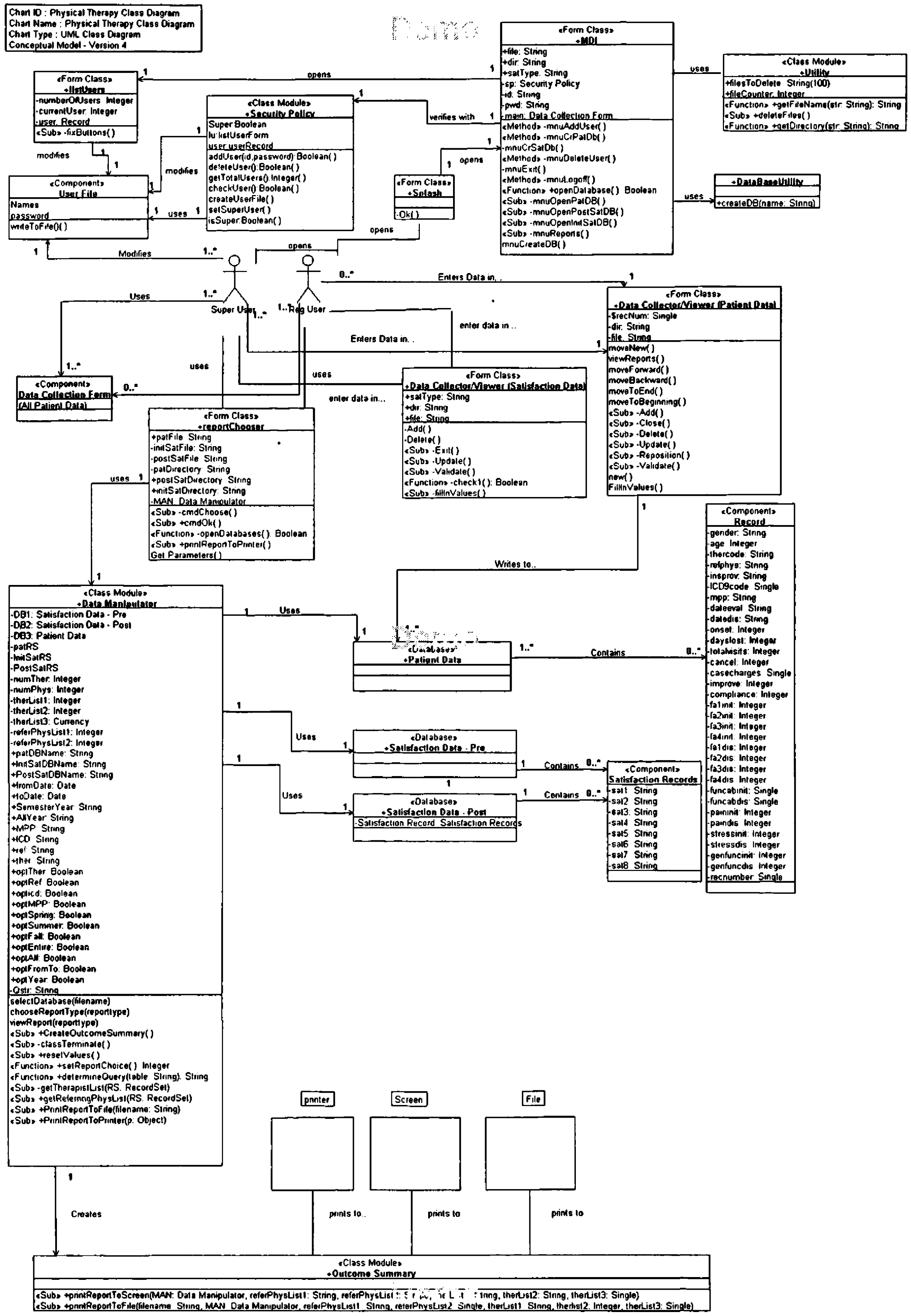


Chart ID : Physical Therapy Class Diagram
Chart Name : Physical Therapy Class Diagram
Chart Type : UML Class Diagram
Conceptual Model - Version 4
«Form Class» +MDI
+file: String
+dir: String
+satType: String
-sp: Security Policy
-id: String
-pwd: String
-main: Data Collection Form
«Method» -mnuAddUser( )
«Method» -mnuCrPatDb( )
-mnuCrSatDb( )
«Method» -mnuDeleteUser( )
-mnuExit( )
«Method» -mnuLogoff( )
«Function» +openDatabase( ) Boolean
«Sub» -mnuOpenPatDB( )
«Sub» -mnuOpenPostSatDB( )
«Sub» -mnuOpenInitSatDB( )
«Sub» -mnuReports( )
mnuCreateDB( )
uses
«Class Module» +Utility
+filesToDelete String(100)
+fileCounter: Integer
«Function» +getFileName(str: String): String
«Sub» +deleteFiles( )
«Function» +getDirectory(str: String): String
uses
+DataBaseUtility
+createDB(name: String)
opens
«Form Class» +listUsers
-numberOfUsers Integer
-currentUser: Integer
-user: Record
«Sub» -fixButtons( )
verifies with
«Class Module» +Security Policy
Super Boolean
lu listUserForm
user userRecord
addUser(id,password) Boolean( )
deleteUser():Boolean( )
getTotalUsers() Integer( )
checkUser() Boolean( )
createUserFile( )
setSuperUser( )
isSuper Boolean( )
modifies
modifies
«Component» User File
Names
password
writeToFile()( )
uses
opens
«Form Class» +Splash
-Ok( )
opens
opens
Modifies
Super User
Reg User
Enters Data in, .
Uses
enter data in ..
Enters Data in. .
uses
uses
«Component» Data Collection Form (All Patient Data)
enter data in...
«Form Class» +Data Collector/Viewer (Patient Data)
-$recNum: Single
-dir: String
-file: String
moveNew( )
viewReports( )
moveForward( )
moveBackward( )
moveToEnd( )
moveToBeginning( )
«Sub» -Add( )
«Sub» -Close( )
«Sub» -Delete( )
«Sub» -Update( )
«Sub» -Reposition( )
«Sub» -Validate( )
new( )
FillInValues( )
«Form Class» +Data Collector/Viewer (Satisfaction Data)
+satType: String
+dir: String
+file: String
-Add( )
-Delete( )
«Sub» -Exit( )
«Sub» -Update( )
«Sub» -Validate( )
«Function» -check1( ): Boolean
«Sub» -fillInValues( )
«Form Class» +reportChooser
+patFile String
-initSatFile: String
-postSatFile String
-patDirectory String
+postSatDirectory String
+initSatDirectory: String
-MAN Data Manipulator
«Sub» -cmdChoose( )
«Sub» +cmdOk( )
«Function» -openDatabases( ) Boolean
«Sub» +printReportToPrinter( )
Get Parameters( )
uses
Writes to..
«Class Module» +Data Manipulator
-DB1: Satisfaction Data - Pre
-DB2: Satisfaction Data - Post
-DB3: Patient Data
-patRS
-InitSatRS
-PostSatRS
-numTher: Integer
-numPhys: Integer
-therList1: Integer
-therList2: Integer
-therList3: Currency
-referPhysList1: Integer
-referPhysList2: Integer
+patDBName: String
+InitSatDBName: String
+PostSatDBName: String
+fromDate: Date
+toDate: Date
+SemesterYear String
+AllYear String
+MPP String
+ICD String
+ref String
+ther String
+optTher Boolean
+optRef Boolean
+opticd: Boolean
+optMPP Boolean
+optSpring: Boolean
+optSummer: Boolean
+optFall: Boolean
+optEntire: Boolean
+optAll: Boolean
+optFromTo: Boolean
+optYear Boolean
-Qstr: String
selectDatabase(filename)
chooseReportType(reporttype)
viewReport(reporttype)
«Sub» +CreateOutcomeSummary( )
«Sub» -classTerminate( )
«Sub» +resetValues( )
«Function» +setReportChoice( ) Integer
«Function» +determineQuery(table String). String
«Sub» -getTherapistList(RS. RecordSet)
«Sub» +getReferringPhysList(RS. RecordSet)
«Sub» +PrintReportToFile(filename: String)
«Sub» +PrintReportToPrinter(p: Object)
Uses
«Database» +Patient Data
Contains
«Component» Record
-gender: String
-age Integer
-thercode: String
-refphys: String
-insprov: String
-ICD9code Single
-mpp: String
-dateeval String
-datedis: String
-onset: Integer
-dayslost: Integer
-totalvisits: Integer
-cancel: Integer
-casecharges Single
-improve: Integer
-compliance: Integer
-fa1init: Integer
-fa2init: Integer
-fa3init: Integer
-fa4init: Integer
-fa1dis: Integer
-fa2dis Integer
-fa3dis: Integer
-fa4dis Integer
-funcabinit: Single
-funcabdis: Single
-paininit: Integer
-paindis Integer
-stressinit: Integer
-stressdis Integer
-genfuncinit: Integer
-genfuncdis Integer
-recnumber Single
Uses
«Database» +Satisfaction Data - Pre
Contains
Uses
«Database» +Satisfaction Data - Post
-Satisfaction Record Satisfaction Records
Contains
«Component» Satisfaction Records
-sat1 String
-sat2 String
-sat3: String
-sat4 String
-sat5 String
-sat6 String
-sat7 String
-sat8 String
printer
Screen
File
Creates
prints to..
prints to
prints to
«Class Module» +Outcome Summary
«Sub» +printReportToScreen(MAN: Data Manipulator, referPhysList1: String, referPhysList2: Single, therList1: String, therList2: String, therList3: Single)
«Sub» +printReportToFile(filename String, MAN Data Manipulator, referPhysList1 String, referPhysList2 Single, therList1 String, therlist2: Integer, therList3: Single)
1
1..*
0..*
Figure 2

```
                    Outcome Summary for 01/01/1900 to 12/31/2050

General Summary

Total number of visits:                                          30
Total number of new evaluations:                                 4
Number of new referrals per referring physician:
   Physician # 9 had 1 referral(s).
   Physician # 8 had 1 referral(s).
Total males in this selection:                                   2
Total females in this selection:                                 2
Number of visits by Therapist and average charge per visit:
  (For this selection only)
   Therapist # 89 had: 9 visit(s).  Average charge:              $1.00
   Therapist # 6 had: 21 visit(s).  Average charge:              $0.57
   Therapist # 0 had: 0 visit(s).  Average charge:               $0.00
Average case charges for this selection:                         $5.25
Average charge per visit for all therapists:                     $0.70
Number of cancellations/no-shows:                                21
Percent of patients by insurance type:
   Self Pay: 0.00%
   MVA:0.00%
   Private Insurance: 0.00%
   Medicare: 75.00%
   Workman's Compensation: 25.00%
   Other: 0.00%

Average functional ability at initial evaluation:                7.50
Average functional ability at discharge:                         7.00
  Average percent improvement in functional ability:             -7.14%

Average general function score at initial evaluation:            3.00
Average general function score at discharge:                     3.00
  Average percent improvement in general function:               0.00%

Average pain score at initial evaluation:                        7
Average pain score at discharge:                                 7.00
  Average percent decrease in pain:                              0.00%

Average stress score at initial evaluation:                      3.25
Average stress score at discharge:                               15.75
  Average percent decrease in stress:                            -384.62%
```

Average number of school/work days lost:	7.50
Average number of days from onset:	5.25
Average number of days from evaluation to discharge:	7.25

Overall home program compliance:	7.00%
Overall improvement:	7.00%

Satisfaction information

INITIAL SATISFACTION

Average satisfaction for questions 1 through 8:	
Physical Therapist:	1.00
SPT:	1.00
Waiting time:	1.00
Privacy:	1.00
Cleanliness:	1.00
Explanation Rx:	1.00
Explanation billing:	1.00
Overall Experience:	1.00
Average overall satisfaction:	-0.17

POST SATISFACTION

Average satisfaction for questions 1 through 8:	
Physical Therapist:	100.00
SPT:	100.00
Waiting time:	100.00
Privacy:	100.00
Cleanliness:	100.00
Explanation Rx:	100.00
Explanation billing:	100.00
Overall Experience:	100.00
Average overall satisfaction:	100.00

Appendix IV

Use Cases:

Note: There is only one actor in this system, the user of the program.

Use Case: Input Data
Type: Primary
Cross Reference: R1.1 to R1.7
Description: The user opens the program and chooses to enter new data. Using the data collection form for reference, each item is entered and checked for validity. When finished, the data is written to the SQL database.

Use Case: Log in
Type: Primary
Cross Reference: R1.9
Description: This use case begins when the user opens the program. A name and password is entered. The program then allows access to the appropriate elements of the program.

Use Case: Move through data
Type: Primary
Cross Reference: R1.11 to R1.14 and R2.1 to R2.3
Description: The user has opened a database and desires to display records. The user chooses to move
forward or backward through the database. Depending on the choice, the program responds by displaying the data for the previous or next record. The user might also choose to enter a new record at which point the program allows it. See use case "Input Data".

Use Case: Manipulate Data
Type: Primary
Cross Reference: R2.4 to R2.5
Description: This use case begins when the user has decided to use the data entered in previous steps.
The user decides how he wants to group the data. Based on this choice an outcome summary is generated.

Use Case: Output Results
Type: Primary
Cross Reference: R3.1 to R3.3
Description: The user desires to see the outcome summary. He chooses to display this summary on the screen, send it to the printer or send it to a file. If it is a file the user is asked for a name for the file. If it is sent to a printer a printer dialog box comes up.

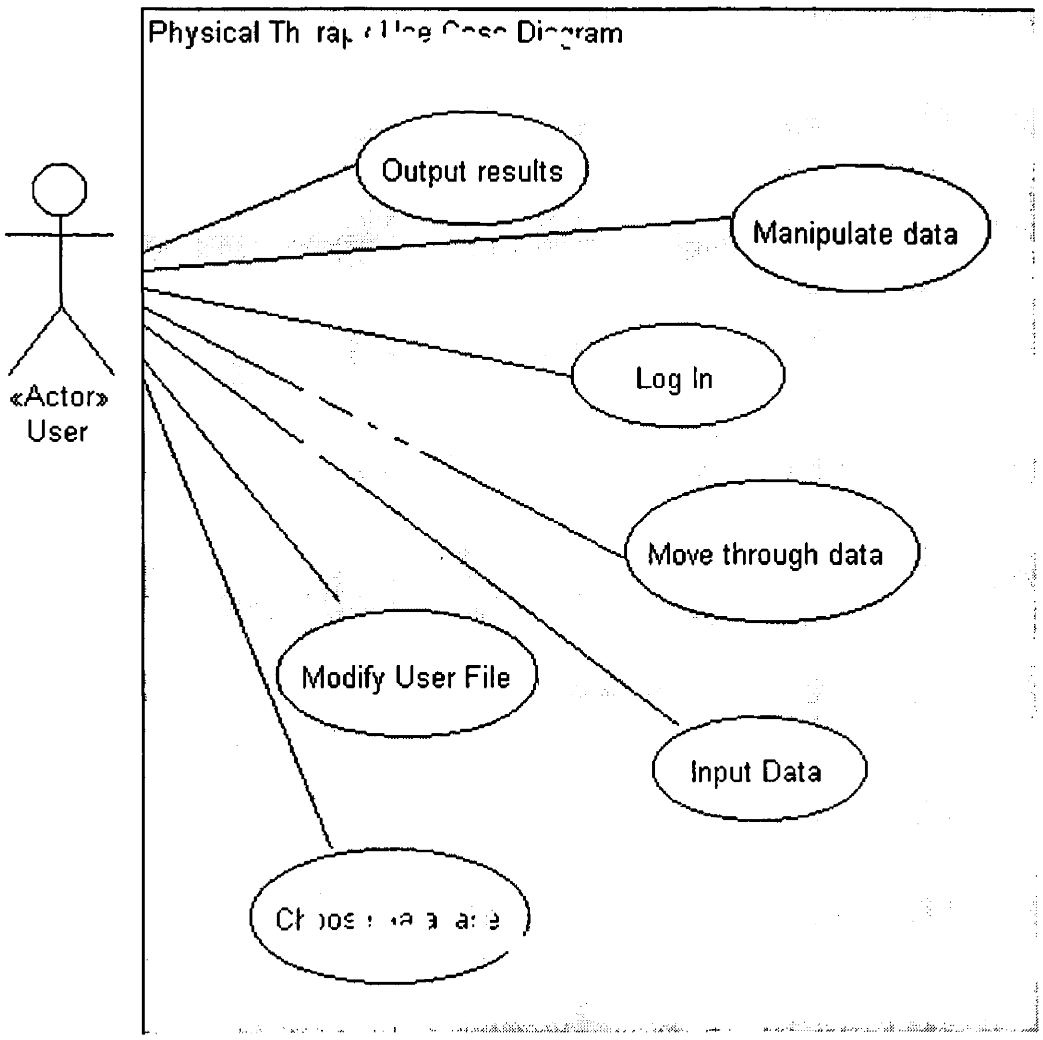
Use Case: Choose database
Type: Primary
Cross Reference: R2.4
Description: Upon choosing to enter data, the user must choose which database in which to enter the data or whether to create a new database.

Use Case: Modify User File
Type: Secondary
Cross Reference:
Description: Upon opening the program, a super user decides to add or delete records of those people allowed to change the database.

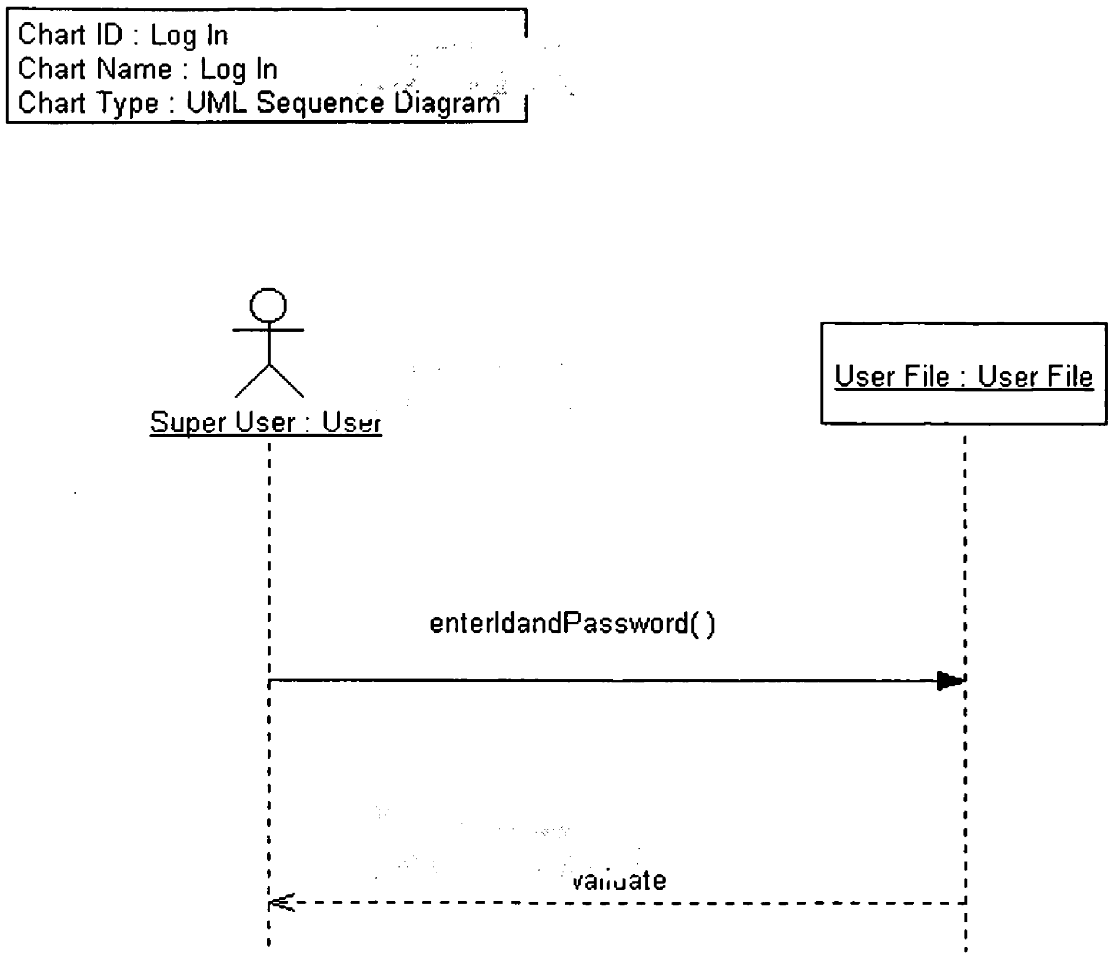
Appendix V

The following pages contain all the diagrams pertaining to this project.

Use Case Diagram



System Sequence Diagrams



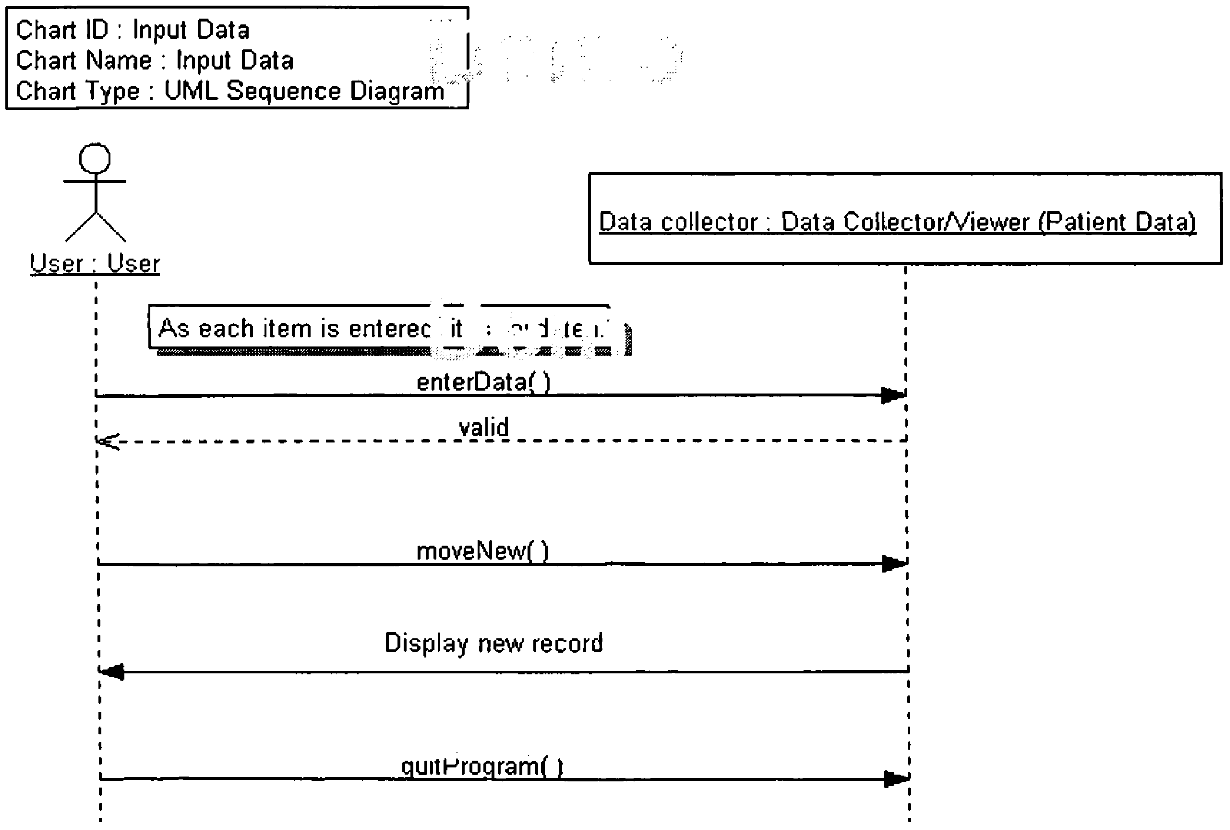


Chart ID : Input Data
Chart Name : Input Data
Chart Type : UML Sequence Diagram
User : User
Data collector : Data Collector/Viewer (Patient Data)
As each item is entered it [illegible]
enterData( )
valid
moveNew( )
Display new record
quitProgram( )

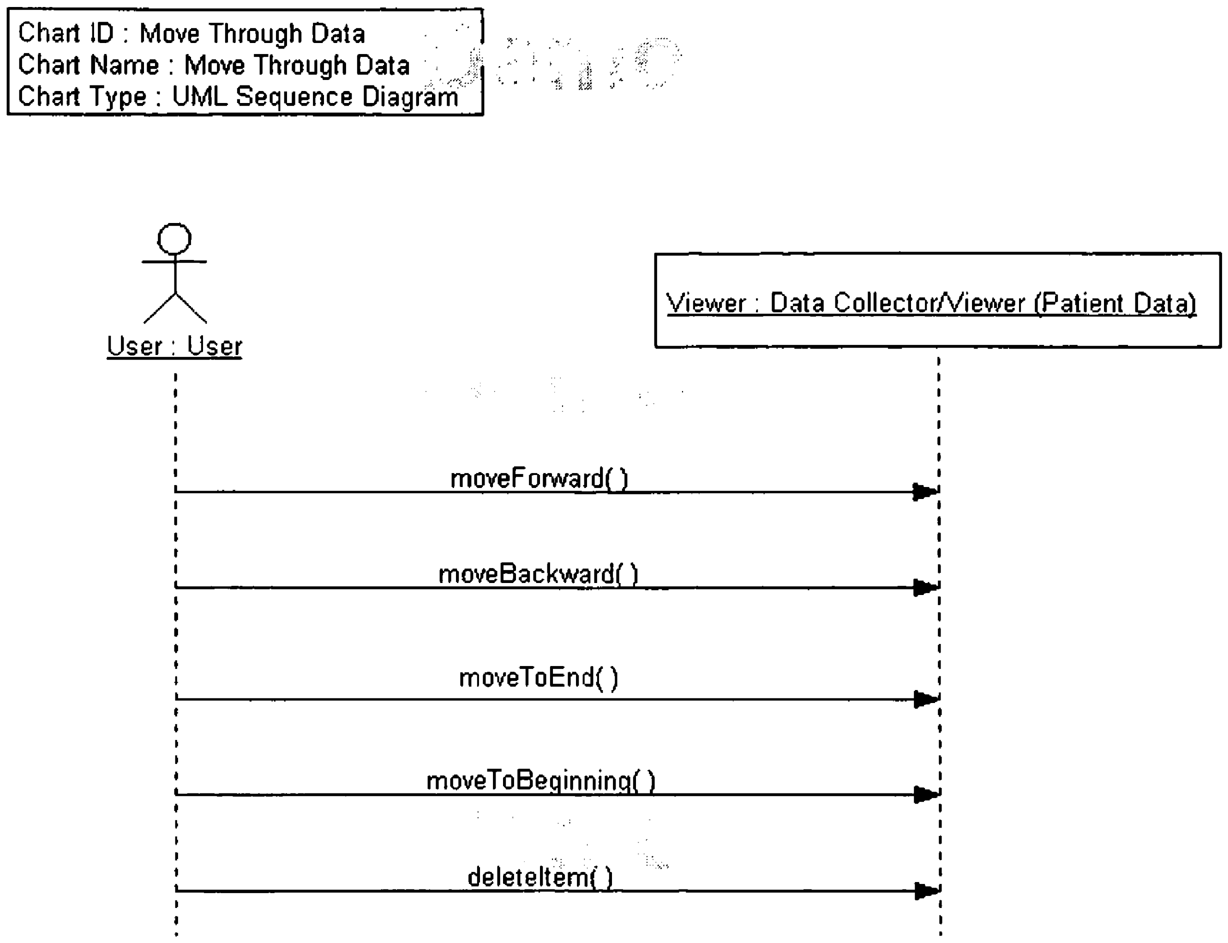


Chart ID : Move Through Data
Chart Name : Move Through Data
Chart Type : UML Sequence Diagram
User : User
Viewer : Data Collector/Viewer (Patient Data)
moveForward( )
moveBackward( )
moveToEnd( )
moveToBeginning( )
deleteItem( )

Chart ID : Output Results
Chart Name : Output Results
Chart Type : UML Sequence Diagram

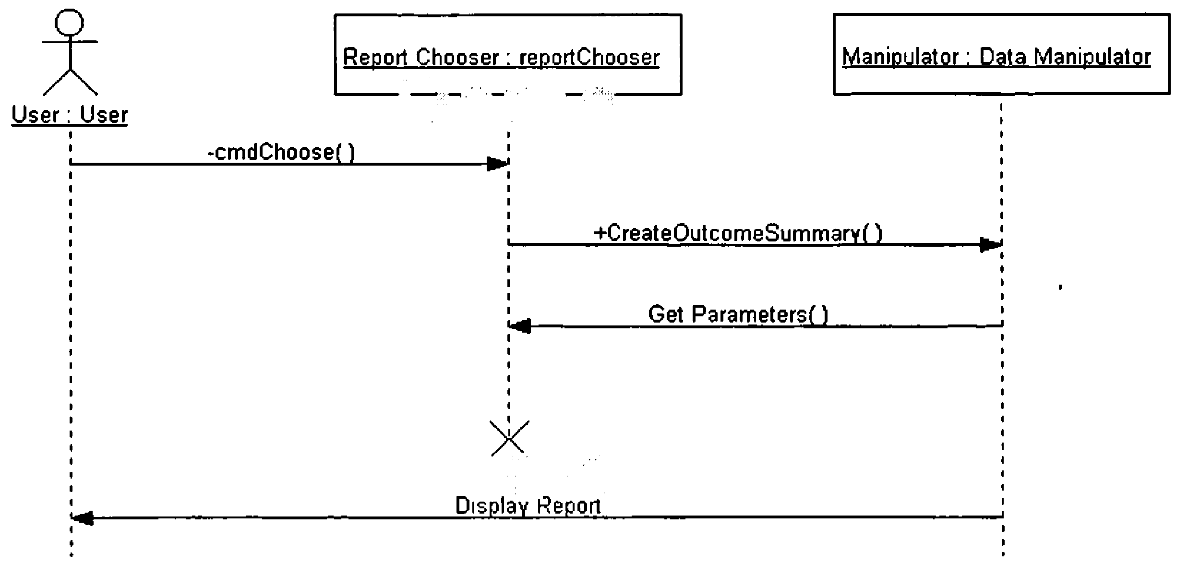


Chart ID : Choose Database
Chart Name : Choose Database
Chart Type : UML Sequence Diagram

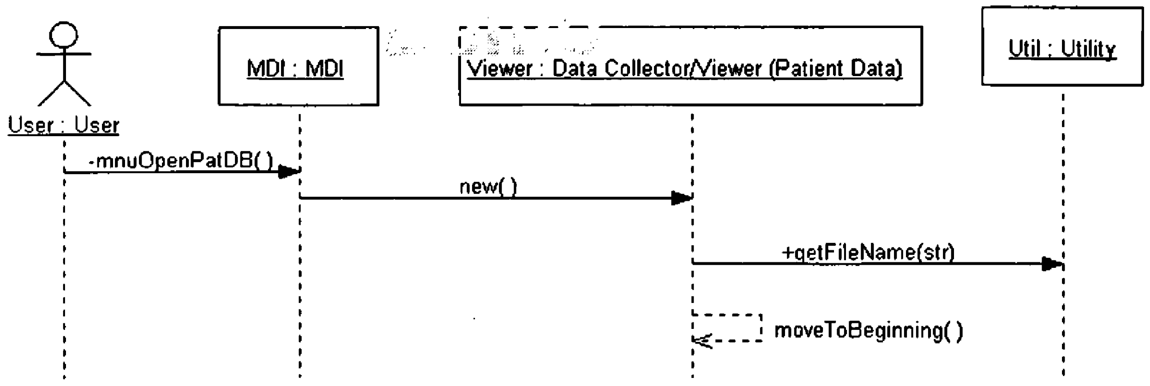


Chart ID : AddRecord
Chart Name : AddRecord
Chart Type : UML Sequence Diagram

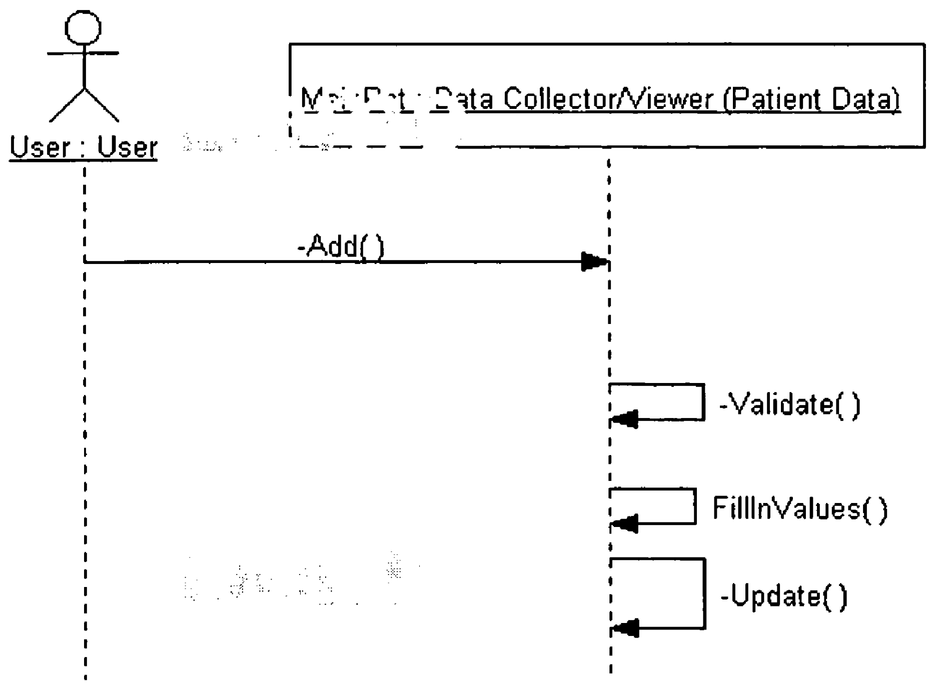


Chart ID : Delete
Chart Name : Delete
Chart Type : UML Sequence Diagram

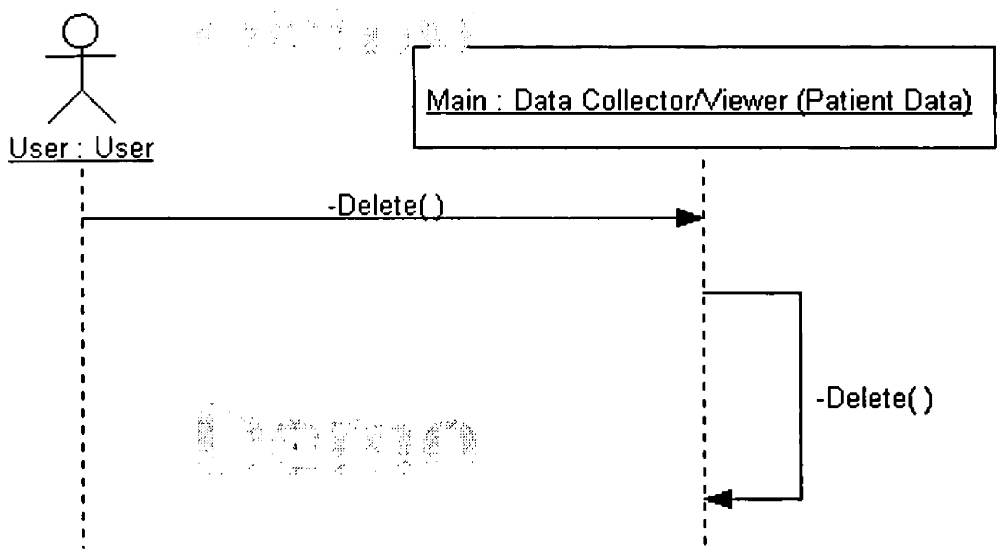


Chart ID : createNewDatabas
Chart Name : createNewDat[illegible]
Chart Type : UML Sequence Diagram

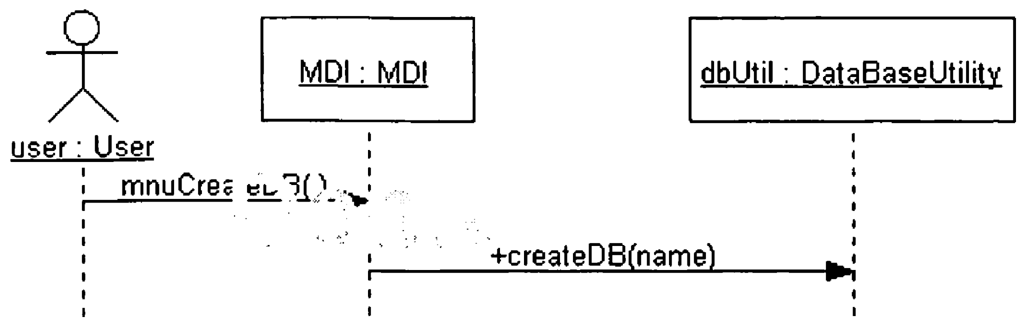
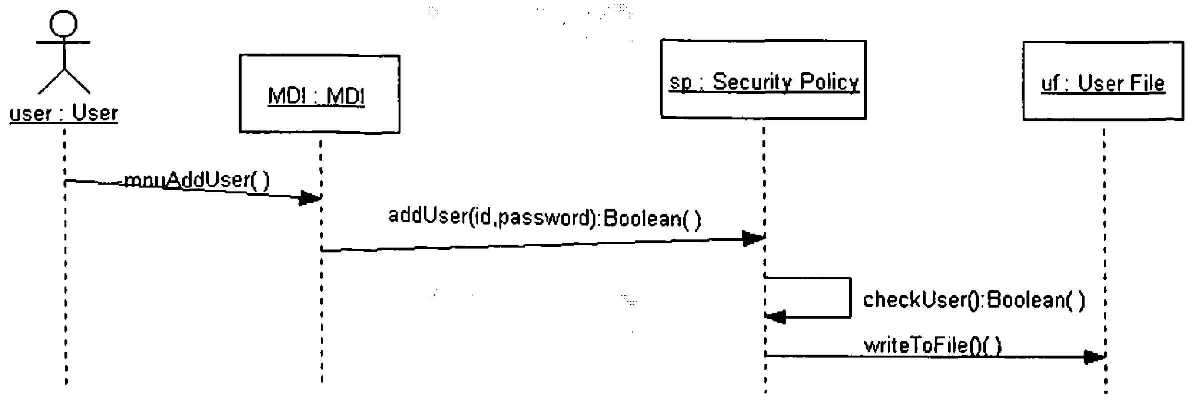


Chart ID : addNewUser
Chart Name : addNewUser
Chart Type : UML Sequence Diagram



Collaboration Diagrams

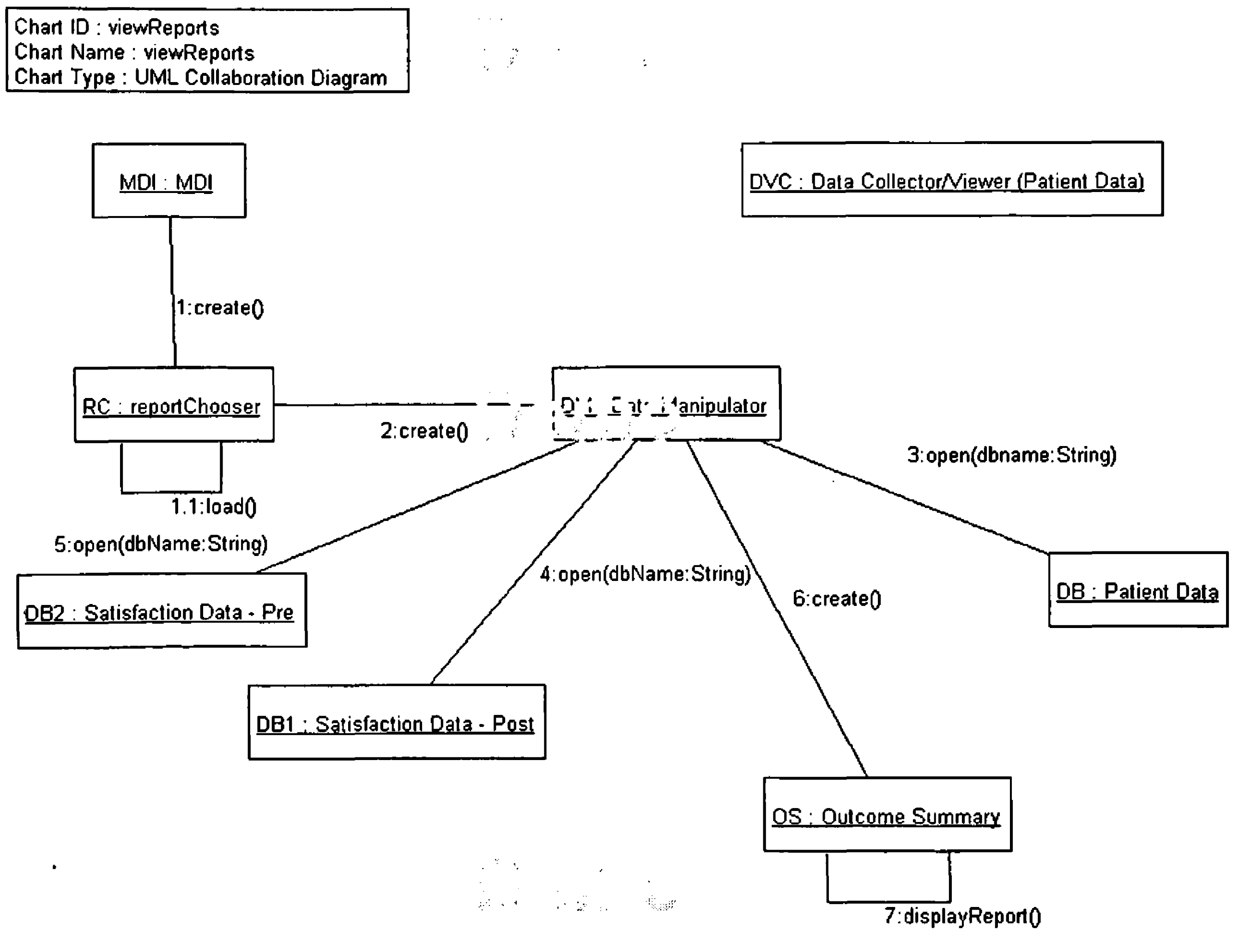


Chart ID : OpenExistingDB
Chart Name : OpenExistingDB
Chart Type : UML Collaboration Diagram

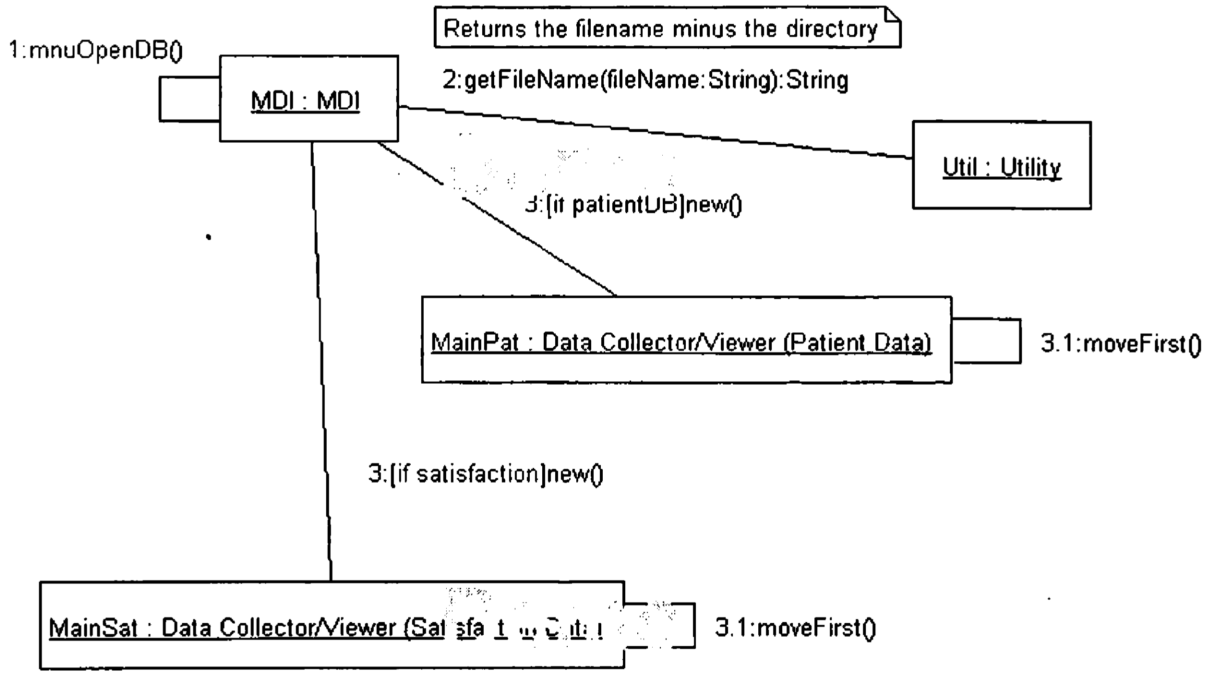
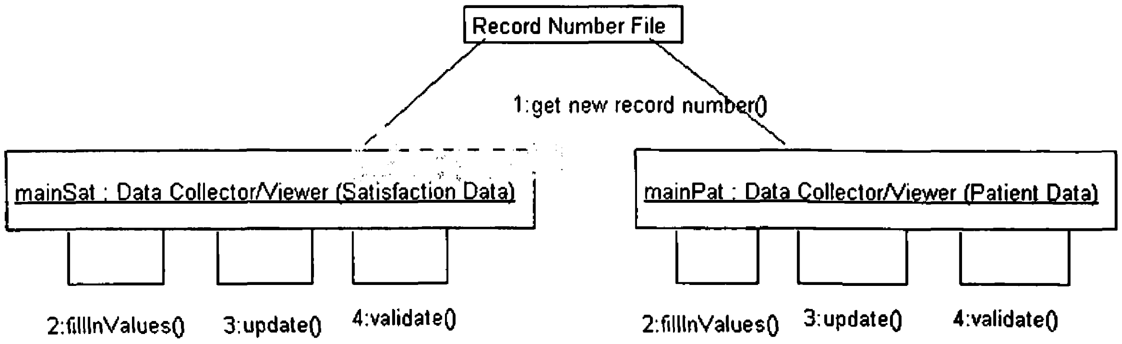

Returns the filename minus the directory
1:mnuOpenDB()
MDI : MDI
2:getFileName(fileName:String):String
Util : Utility
3:[if patientDB]new()
MainPat : Data Collector/Viewer (Patient Data)
3.1:moveFirst()
3:[if satisfaction]new()
MainSat : Data Collector/Viewer (Satisfaction Data)
3.1:moveFirst()

Chart ID : AddNewRecord
Chart Name : AddNewRecord
Chart Type : UML Collaboration Diagram


Record Number File
1:get new record number()
mainSat : Data Collector/Viewer (Satisfaction Data)
mainPat : Data Collector/Viewer (Patient Data)
2:fillInValues()
3:update()
4:validate()
2:fillInValues()
3:update()
4:validate()

After user has entered the values actions 2, 3 and 4 occur.

The actions occur in both the patient and satisfaction viewers depending on which one is being used.

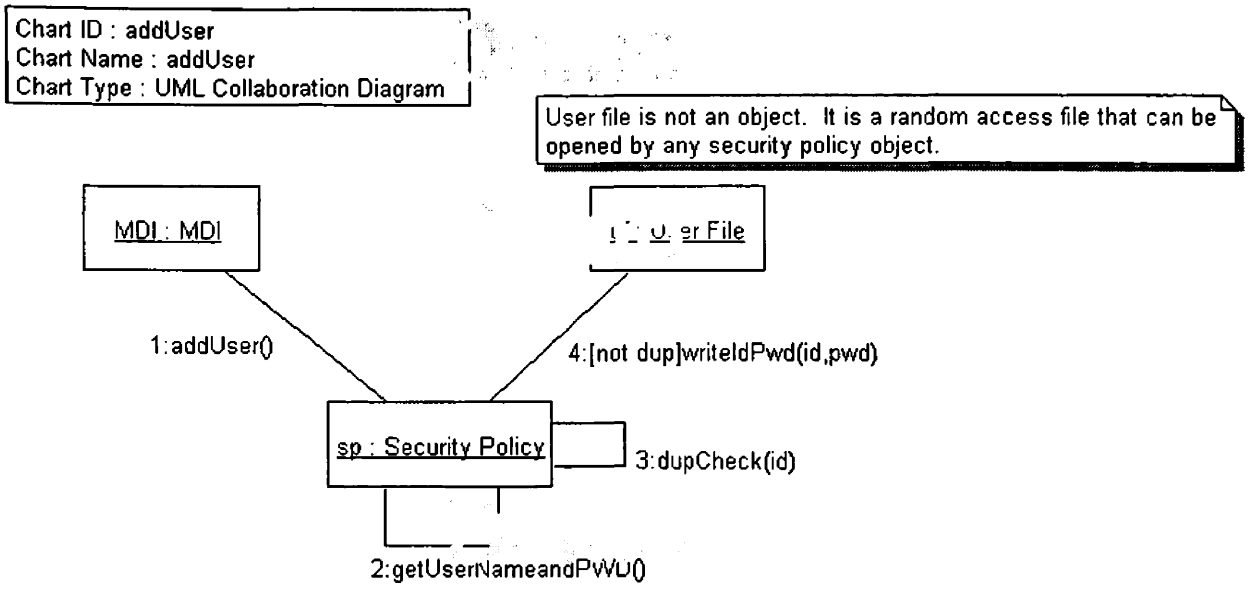


Chart ID : addUser
Chart Name : addUser
Chart Type : UML Collaboration Diagram
User file is not an object. It is a random access file that can be opened by any security policy object.
MDI : MDI
User File
1:addUser()
4:[not dup]writeIdPwd(id,pwd)
sp : Security Policy
3:dupCheck(id)
2:getUserNameandPWD()

Chart ID : deleteUser
Chart Name : deleteUser
Chart Type : UML Collaboration Diagram

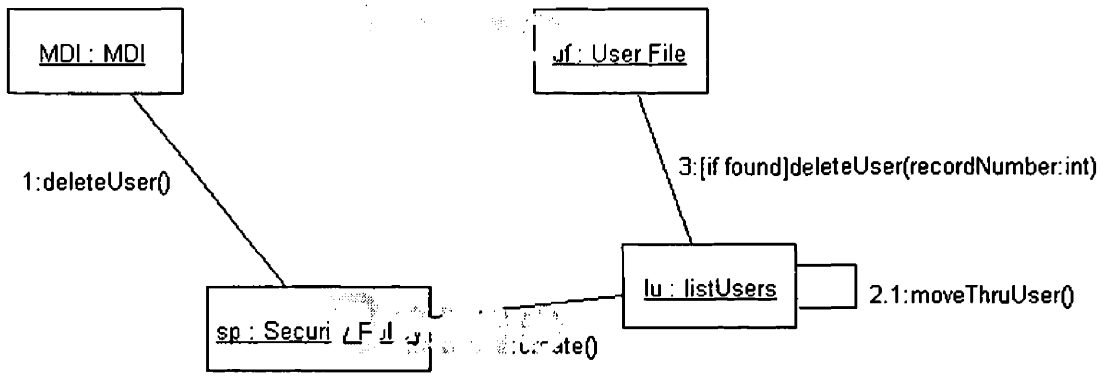
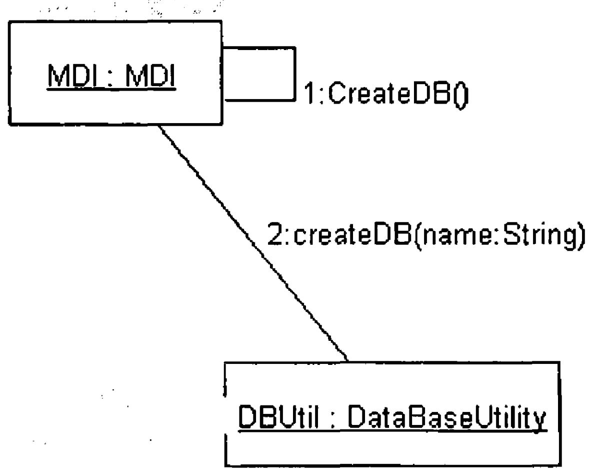


Chart ID : CreateNewDB
Chart Name : CreateNewDB
Chart Type : UML Collaboration Diagram



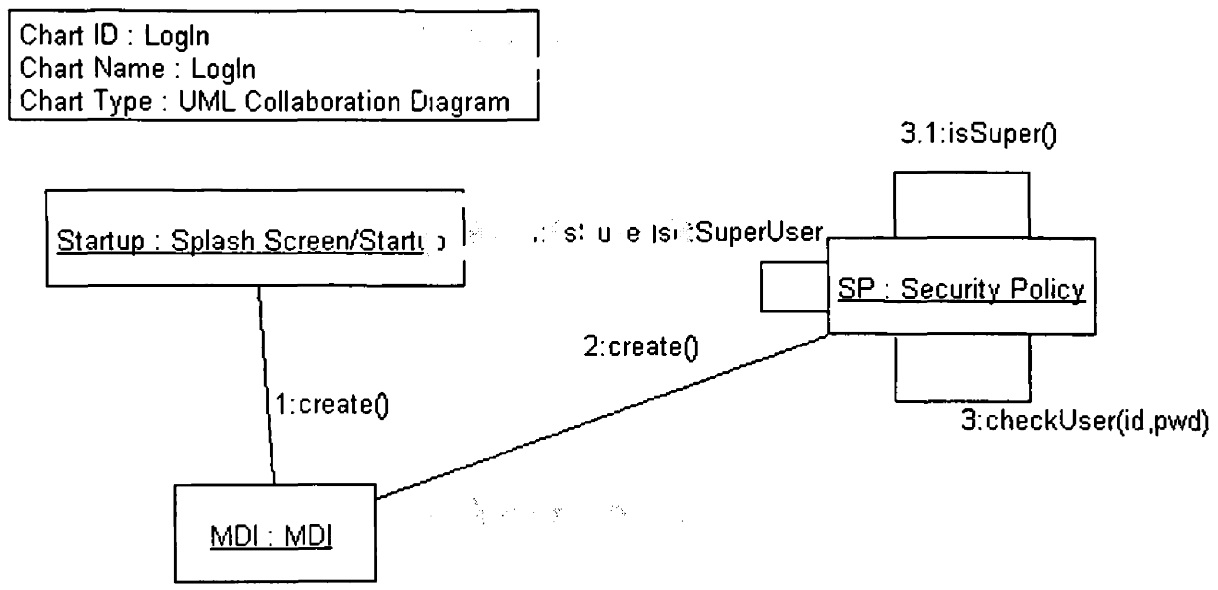


Chart ID : LogIn
Chart Name : LogIn
Chart Type : UML Collaboration Diagram
3.1:isSuper()
Startup : Splash Screen/Startup
SuperUser
SP : Security Policy
2:create()
1:create()
3:checkUser(id,pwd)
MDI : MDI

Chart ID : er[illegible]ata()
Chart Name : er[illegible]Data()
Chart Type : UML Collaboration Diagram

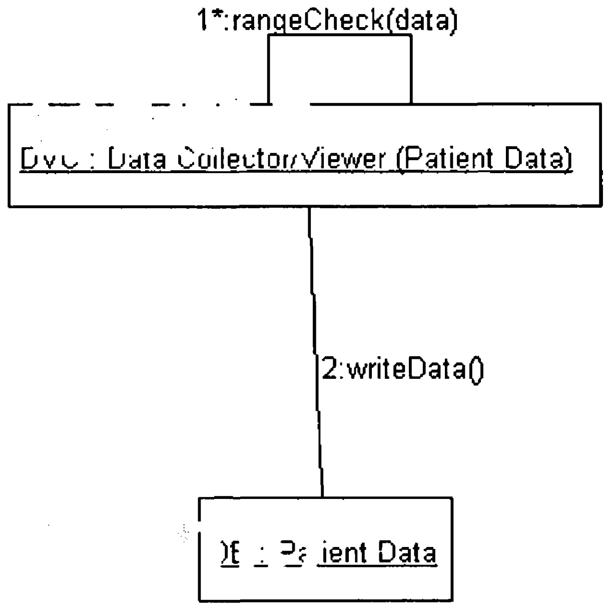

1*:rangeCheck(data)
DVC : Data Collector/Viewer (Patient Data)
2:writeData()
[illegible] : Patient Data

Chart ID : enterData()
Chart Name : enterData()
Chart Type : UML Collaboration Diagram

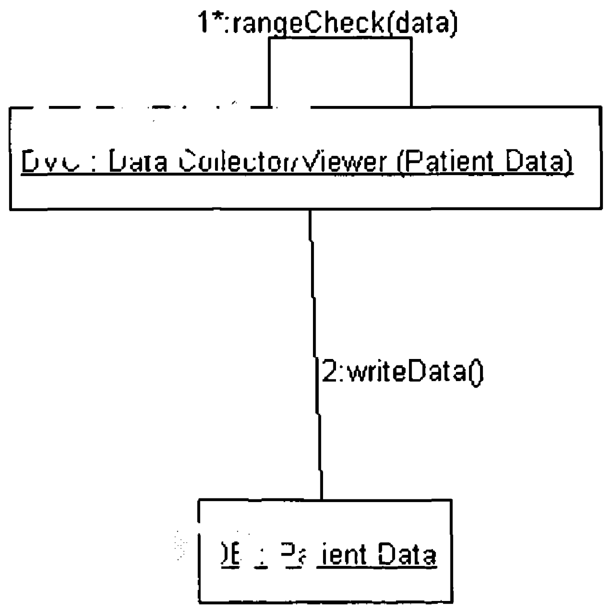
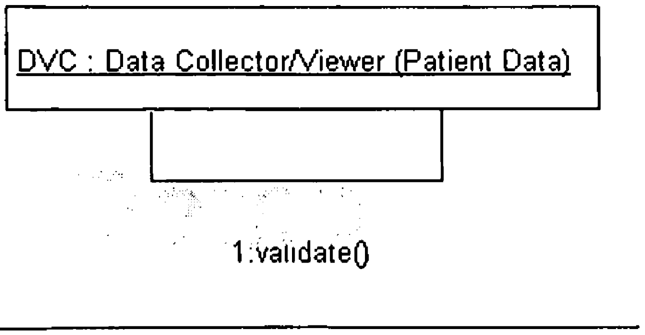


Chart ID : moveNew
Chart Name : moveNew
Chart Type : UML Collaboration Diagram


DVC : Data Collector/Viewer (Patient Data)
1:validate()