SUPPORTING ASYNCHRONOUS TELEMEDICINE

ELECTRONIC MAIL VS. THE WORLD WIDE WEB VS. REPLICATED DATABASES

RAHIM S. PIRA

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

SUPPORTING ASYNCHRONOUS TELEMEDICINE

ELECTRONIC MAIL VS. THE WORLD WIDE WEB VS. REPLICATED DATABASES

By

©Rahim S. Pira; B.A.Sc

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment
of the Requirements for the Degree of
Master of Engineering
in
Electrical Engineering

Faculty of Engineering and Applied Science
Memorial University of Newfoundland
August, 1998

St. John's, Newfoundland, Canada

ABSTRACT

Telemedicine has been presented as the "holy grail" for rural and remote regions yet many commercially available applications ignore the limitations faced by clinicians in these areas, in terms of the information carrying capacity of the communication mediums available, or have not considered the user model of the consultation process. This research investigates the development of a robust, efficient, low-data rate teleconsultation application that provides the "best fit" with users' models of the consultation process appropriate for rural and remote regions. The results indicate that teleconsultation applications should adopt an asynchronous operational model where the patient medical data closely resembles the traditional paper-based medical record.

ACKNOWLEDGEMENTS

I would like to thank Dr. John Robinson for his invaluable support, patience and guidance, crucial to bringing this thesis to fruition.

I would also like to thank Li-Te Cheng for his assistance in the development of the telemedicine applications.

I would like to express my appreciation to the Department of Telemedicine at the Memorial University of Newfoundland and Dr. Rod Elford for their advice and input during the course of this work.

Finally, I would like to thank my parents and my sister for their prayers, love and support.

The Road goes ever on and on

Down from the door where it began.

Now far ahead the Road has gone,

And I must follow it if I can,

Pursuing it with eager feet,

Until it joins some larger way

Where many paths and errands meet.

And whither this? I cannot say.

J.R.R. Tolkien,

The Lord of the Rings

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Problem Domain

The majority of us take for granted the wide variety of medical services we have at our fingertips. In rural and remote areas, and especially third-world countries, this is not the case. Access to even the most basic medical services is limited whereas access to specialists is near impossible. Advances in computing and telecommunications technologies have provided a wide range of solutions, known generally as telemedicine. A subset of this area is teleconsultation – medical consultations between physicians. Undoubtedly, the services provided by these technologies will alleviate some of the problems related to inadequate access to medical expertise.

Solving the problem of providing access to medical expertise in rural and remote areas, as well as the developing countries, does not stop at simply developing applications that enable teleconsultations. It is reasonable to assume that access to an advanced telecommunications infrastructure is quite unlikely. At best, the telecommunications infrastructure will consist of standard telephone lines in rural areas, or wireless services in the developing world, greatly limiting the amount and type of data that can be transmitted over these lines. For this reason, potential solutions must take into account these limitations, in addition to the work habits of physicians. It is exactly this problem that remains unsolved and its solution is the purpose of this study.

1.2 Problem Statement

The problem to be researched is the development of a low-data rate telemedicine application for the purpose of clinical teleconsultations to be utilized in remote locations, using a data model most similar to the current consultation process.

1.3 Research Objectives

The research objectives can be outlined as follows:

- develop a number of possible user models and their associated technological model for medical teleconsultations for remote areas

- determine which user model and technological model is most appropriate for teleconsultations in remote locales using the current methods employed as the basis of comparison using input from the target user group

- provide efficient and robust teleconsultation applications for use; these will be developed in-house

1.4 Thesis Organization

This thesis is organized as follows:

- ***Chapter 1 – Introduction***, this chapter, provides an overview of the problem and the research objectives.

- ***Chapter 2 – Background to Asynchronous Telemedicine***, provides a detailed background of telemedicine and its usage and methods, especially asynchronous telemedicine, and a typical scenario is provided. The status of and need for telemedicine is also discussed.

- ***Chapter 3 – Design Options and Issues***, outlines the problem, the constraints on possible solutions and the criteria used to judge if a solution is successful. The possible solutions – mail-based systems, videoconferencing systems, Web-based systems, and medical database systems – are outlined and discussed. The solutions we chose to pursue are also discussed as are the reasons for selecting them. As

database solutions are being looked at, the various kinds of databases are examined including the pros and cons of each alternative. Lastly, the user interface issues involved in designing a graphical application are discussed including the types of windows and menus available.

- ***Chapter 4 – Software Design***, presents the detailed design, including functional specification, input and output interfaces, high-level data objects and the interactions amongst them, for the three applications developed.

- ***Chapter 5 – Implementation Issues***, discusses the development environment, third-party tools used, the issues and problems faced during the implementation of the applications and the method of testing and verifying them.

- ***Chapter 6 – User Testing***, presents the results obtained during the user testing phase of the developed systems.

- ***Chapter 7 – Conclusions***, summarizes and discusses the major results of the research and presents suggestions for future work.

CHAPTER TWO

BACKGROUND TO ASYNCHRONOUS TELEMEDICINE

2.1 Introduction

The history of telemedicine is an interesting one – from its origins during the start of the space program to its near abandonment during the 1970s, to the renewed interest in telemedicine more recently. In this chapter, the history of telemedicine will be explored, beginning with an envisioned scenario of the future of health care, followed by the human and technical factors that need to be considered when developing systems for telemedicine. This is followed by a survey of the current uses of telemedicine, the various methods used to accomplish this, and a discussion of the need for asynchronous

telemedicine. Finally, the state of telemedicine in Newfoundland is discussed and the typical scenario for asynchronous telemedicine is presented.

Consider the following scenario:

Seated at his desk in his office, Dr. Jones logs into his computer. Displayed on the screen is a list of new patient cases received during the night, awaiting his attention. Clicking on the first patient case in the list, Mrs. Peel, a record browser appears showing all entries in the patient record; video, audio, pictures, and text. Clicking on a video item, playback begins of an examination that occurred yesterday. Mrs. Peel had undergone cosmetic surgery last week after being involved in a car accident; the examination was a routine one to monitor the skin graft and to ensure its proper healing – the graft was doing well and all sign of her injury would soon disappear. He quickly records some instructions for Mrs. Peel and assures her that all is going well. The second patient case is a troubling one; one for which he had needed to consult with a paediatrician. Young Derek Thompson has had a serious cough for over a week now and shows no signs of improving. The paediatrician has agreed that the situation appears to be serious and has recommended that Derek be immediately hospitalized; Dr. Jones agrees. He quickly retrieves Derek's address from the electronic record and informs Mrs. Thompson of the paediatrician's recommendation; as well, he forwards Derek's medical record to the hospital to which he will be admitted. He hopes that Derek will get well.

The preceding scenario is an example of what health care may be like in a few years. Driven by advances in computer hardware and networking and communications infrastructure, the health care of tomorrow will be a highly networked, distributed system of professionals working together to provide superior health care at a lower cost. Crucial to providing improved health care in the coming years will be systems that allow physicians to communicate effectively with one another, share expertise and resources, and rapidly organize and digest large amounts of patient data. While attempting to improve communication, it must be kept in mind that physicians will still need to spend their time providing patient care rather than focusing on the methods with which to communicate with other physicians assisting in the care. Simply put, the health care professional must spend his time and resources in healing, not overcoming obstacles in co-operating with others involved in the healing. But of what importance are such systems today? The introduction of computers and telecommunications into medicine produces systems relevant in the present to health care-givers in remote locales. These systems would allow physicians in even the most remote areas easy access to medical resources they would otherwise not have access to, or access to only at great expense.

Telemedicine means different things to different people: a database of medical data for the purpose of education; the consultation between two physicians through videoconferencing; the transfer of medical images for analysis. Whatever definition of telemedicine is used, it is agreed that at its most fundamental level, telemedicine uses

computing, telecommunications and information technologies to provide flexible, easy and rapid access to shared and remote medical expertise and resources, regardless of where the patient, the pertinent information or the resources are located [Alle95], [Gome94].

The explosion of interest in telemedicine over the past decade may make it appear that telemedicine is a relatively new use of telecommunications technology but telemedicine has been in use for over thirty years. The National Aeronautics and Space Administration (NASA) played an important part in the early development of telemedicine in the 1960s for monitoring astronauts during space missions. Physiological data were telemetered from both the spacecraft and the spacesuits during missions. These early forays into the field of telemedicine proved unsuccessful primarily because the effort was not sustained and failed to become an accepted mode of health care delivery [Bash97]. Alternative explanations exist for the failure of the widespread acceptance of telemedicine. Firstly, the telecommunications infrastructure was in the early stages of development. More importantly, information transmission capability was rather primitive by today's standards and few could predict the extent and capacity of future developments, especially in digital compression and computer miniaturization and portability [Bash77]. For instance, even dedicated telephone lines could transmit only slow-scan analog images, and the quality of audio transmission was poor with coaxial cables. Data compression, although developed earlier, was not

incorporated into first generation telemedicine applications, compounding the negative opinions against early telemedicine programs.

Secondly, the experimenters, especially the medical providers, were inexperienced in the efficient use of the technology, and were faced with puzzling questions regarding the capability and the problems the new technology presented. In fact, there was a general reluctance to accept telemedicine because it was mostly unknown [Fuch79], [Grig95].

Lastly, funding for many projects was cut-off prematurely before any of the demonstration projects could reach maturity or provide a valid experimental basis for reasonable policy in this area. Still, the first generation of telemedicine applications provided ample evidence of the feasibility of remote consultation, the clinical effectiveness of several clinical functions, training, and education. These findings, however, were ignored at the time because of a widespread fear that sophisticated technology would only increase the cost of health care.

Work in the area of telemedicine continued, conducted primarily by NASA, the U.S. military and university researchers. The work by NASA and the U.S. military concentrated on the use of satellite technology for relaying information. Satellite technology, however, is not available to all and other researchers have concerned themselves with delivering practical, sufficiently advanced and cost-efficient telemedicine technologies. Examples abound in Canada and Australia [Dohr91],

[Hous77], [Hous91] showing that even a basic technology can produce good results. Videoconferencing was first used for telepsychiatry in 1959 using an interactive television system at the University of Nebraska [Grah96], [Pres92], [Bens65]. Research has also continued into other areas of telemedicine, for example, teleradiology and telepathology using a number of technologies such as radio communication and microwave networks.

2.2 Human and Technical Context

This section focuses on the human and technical aspects of telemedicine, including the factors one must take into account when developing telemedicine applications in general.

2.2.1 Human Factors

The medical community is extremely peer-influenced, more so than other communities. As clinicians will take their cues from colleagues, if a new technology is accepted by their peers and opinion leaders, this greatly enhances a clinician's receptivity to the new practice. In addition, the appearance of competence and one's physical appearance are of importance to clinicians. This is of concern as the use of electronic media may weaken a patient's trust in the clinician, affecting the proper treatment of a malady.

The health care sector has often been criticized as being too slow in adopting advanced communications and information technologies. Multiple reasons exist for this behaviour:

- the small market presented by the health care sector frequently translates into applications that are found lacking as developers aim to reach as wide a market as possible with their applications, failing to provide many of the tools required by the health care sector

- a lack of rigorous evaluation of new technologies, particularly telemedicine, discourages clinicians and administrators in using them as their benefits are unknown

- uncertainty regarding the regulation of medical software by federal health agencies

When interviewed, many clinicians believed a major deterrent to telemedicine to be the uncertainty regarding payment for teleconsultation services as these are not covered by federal health programs or other third-party payers [Fiel96].

2.2.2 Technical Context

Complexity and Variety of Technologies

Certain challenges arise in developing, evaluating and supporting information and telecommunications systems from:

1. the rapid pace of technological change of hardware and software
2. the multiplicity of hardware and software options and pricing
3. the scarcity of standards assuring that different hardware and software options will work together
4. the requirement of space adapted for the equipment, extensive user training and reinforcement, and sophisticated support staff
5. the diversity of needs and circumstances among potential users
6. the need to develop a variety of communication links with "outside" organizations and individuals who differ in their capacities and the configurations of their systems

Information Carrying Capacity

The capabilities of telemedicine applications are constrained by information carrying capacity – bandwidth – of the communications medium employed (see **Figure 2.1**). Higher bandwidth mediums tend to be more costly to install and maintain than lower bandwidth mediums. The cost and carrying capacity of different mediums are important because they affect the availability, quality and affordability of the information needed by clinicians to diagnose and manage health problems. Among the key issues relevant to clinicians are:

- sound fidelity
- image resolution (both spatial and contrast)
- range (completeness) of motion depicted
- transmission speed or the amount of information that can be transmitted in a defined period

Choosing among telemedicine technologies is an exercise in trade-offs involving the amount, quality, immediacy, and cost of different kinds of information, e.g. satisfactory voice communication requires less bandwidth than satisfactory video communication. The demand for information carrying capacity depends on user needs and resources and increases in carrying capacity can be achieved by improving the transmission media and by restructuring the data.

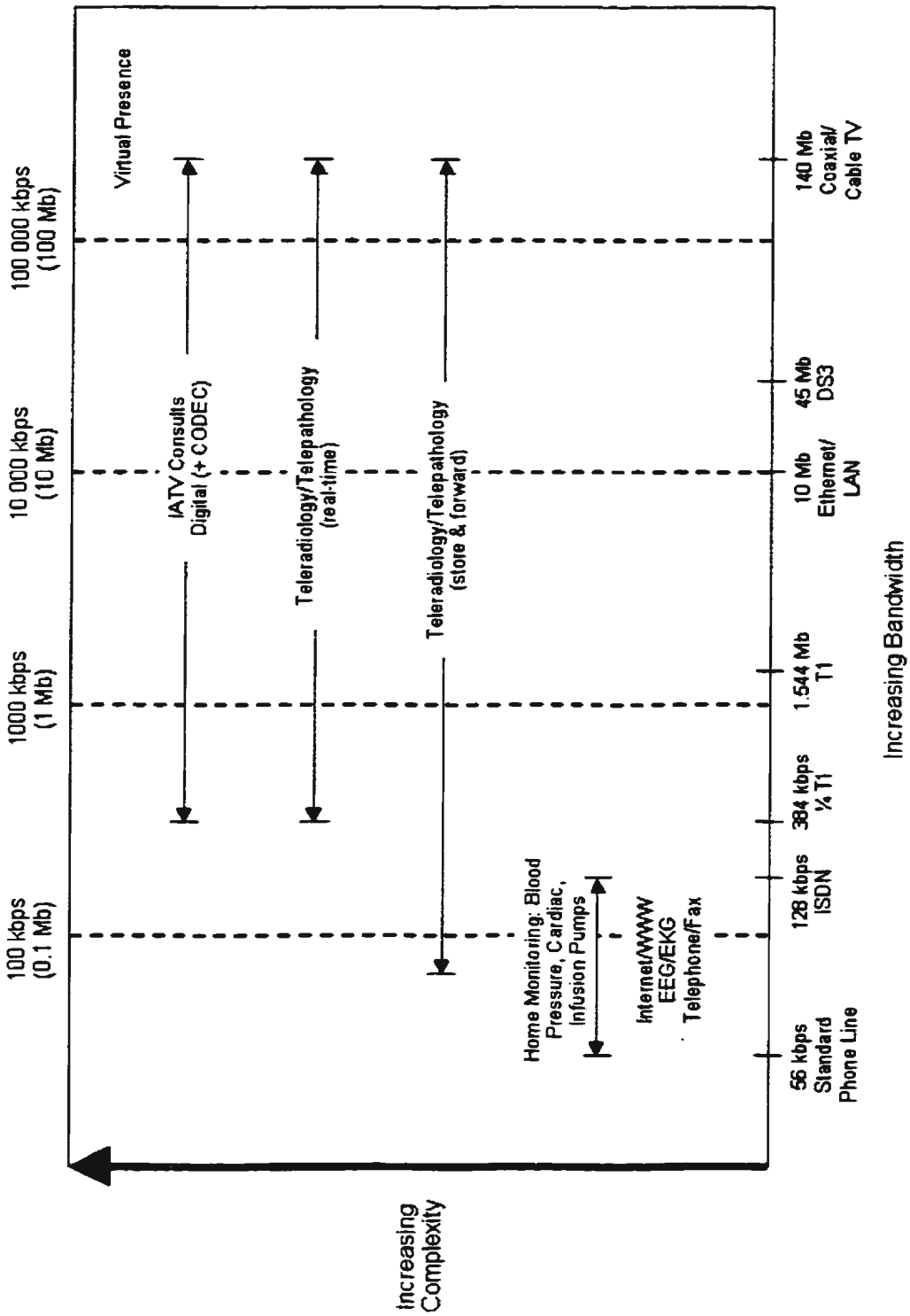

Figure 2–1. Relationship between complexity of telemedicine applications and bandwidth requirements. Source: Field, 1996 [Fiel96]

Privacy, Confidentiality and Security

Questions of privacy and confidentiality are not specific to telemedicine or information technologies, in general. Conventional health care practices and paper medical records present numerous opportunities for unintentional, careless or deliberate infringements of medical privacy. The electronic recording, storage, transmission, and retrieval of patient information has complicated the situation and increased the opportunities for infringement of the privacy and confidentiality of personal medical information which extends beyond the written word to include still images, audio records and videos of patients. These developments have required reassessments of the trade-offs between privacy concerns and other values such as the convenient and quick access to information and the practical realities of enforcing agreed-upon balances between competing objectives.

2.3 Survey of Usage and Methods

From the start, the raison d'être of telemedicine research has been to provide a sufficient level of health care to rural and remote areas. While applications have arisen for space and the military, the primary users of telemedicine will be clinicians in hospitals and clinics. For clinics and hospitals in urban areas, telemedicine provides convenient, time-saving services. For those clinicians in remote and rural areas, telemedicine offers access to services that are otherwise unavailable such as seeking specialty care for patients. While the technology for telemedicine has not quite reached

maturity yet, sufficient advancements have taken place for the deployment and use of telemedicine systems. In the U.S., correctional institutions have started to use telemedicine systems, reducing time and money expenditures for transporting inmates, at the same time increasing the safety of both health care personnel and the public. Usage of telemedicine is widespread across all specialities and is used for numerous applications including the transmission of radiological images, high-resolution photographs and audio, videoconferencing, and electronic medical record keeping. Just as the uses of telemedicine are varied, so are the methods through which this is accomplished. Medical data may be transferred using the Internet, Intranets, satellites, videoconferencing equipment, and telephones. The following applications and systems are representative of the telemedicine applications available today:

1. *TeleMed*,[1] Los Alamos National Laboratory and the National Jewish Medical and Research Centre

   The *TeleMed* system allows physicians to simultaneously see, edit and annotate a virtual patient record at remote locations. The patient record is comprised of multimedia items such as computed tomography (CT) images and audio annotations. A graphical interface is provided for the record, with icons representing the numerous types of medical data (laboratory tests, drug treatments, etc.) displayed. Where bandwidth permits, video teleconferencing is also

[1] *http://www.acl.lanl.gov/TeleMed/*

supported. Connectivity is achieved via the Internet using a "web browser"; a Java applet is executed to access and manipulate the patient record.

2. *VisiTran*®, MedVision[2]

The VisiTran® system is a store and forward telemedicine application that allows a healthcare provider to collaborate with other providers. Multimedia objects (audio annotations, video, still images) can be shared using the videoconferencing feature or may be forwarded to other healthcare providers via electronic mail. The still images can also be annotated using tools provided with the application to indicate areas of importance or in question.

3. *MEDPAC*™ [3], The Bradford-Groupe Ltd.

MEDPAC™ is a digital storage and transmission system combined with teleconferencing capabilities for remote diagnostics and training. The system is mobile thus allowing bedside diagnostics and the teleconferencing feature allows physician to physician consultations.

4. *Apollo Image Management System* (IMS) *for Windows*[4], Apollo Software Telepathology

Operating on a Pentium computer, images from a video source (a camera on a microscope) are digitized and stored on disk. Users at remote sites are able to

[2] *http://www.medvis.com*
[3] *http://www.bradford-group.com/BBBMED.htm*

consult simultaneously, sending images over standard telephone lines. Real-time annotation is also possible, giving one pathologist access to the sample without forcing the other pathologist to give up possession of the sample.

2.4 Asynchronous Telemedicine

Due to the demands on a care-giver's time, setting aside time from patient care for teleconsultation is quite difficult. As well, the likelihood of consulting physicians being available at the same time is a rarity, exemplified by the difficulty of finding a group of physicians meeting over a patient case in the corridors of a hospital [Oroz92]. For these reasons, the "natural" operating mode for telemedicine systems is asynchronous where one physician sends medical data to another physician who is made aware of the newly received data. This is not to say that synchronous systems are infeasible for low-data rate telemedicine as a number of synchronous telemedicine applications are available. Synchronous systems are valuable for emergencies and in matters requiring clarification or out of a need for personal contact. We propose that telemedicine systems should operate asynchronously but have a synchronous mode available for emergencies.

[4] *http://www.apollotelemedicine.com/wims.htm*

2.5 Telemedicine in Newfoundland

The province of Newfoundland, consisting of the island of Newfoundland and Labrador on the mainland, is a prime subject for telemedicine. The province has a population of approximately one-half million with a landmass totalling just under one-half million square kilometres. However, about 50% of the population resides near St. John's, the capital of Newfoundland. This results in a very low population density for the remainder of the province. Consequently, much of the province is dotted with small communities, each with a population less than 3000. The large number of small towns poses a major obstacle in attempting to deliver a level of health care to remote areas equal to that available in large city centres. Telemedicine fulfils the role of the "great equalizer" by allowing residents of remote areas to receive the same level of care. However, the very reasons that make Newfoundland a perfect candidate for telemedicine also pose some problems for the implementation of telemedicine systems.

Given the distribution of small communities all over the province and the cost associated with laying high-data rate communication lines, a strong case on economic reasons can be made against their installation to these remote communities. As a result, remote communities have only low-data rate lines coming into their midst. The situation is so dire that some communities have insufficient lines to service their population, much less a telemedicine system.

The distribution and size of towns, especially in Labrador where air and boat are the primary means of transport, makes it costly to supply each town with a doctor. Instead, clinics are set up in these towns, operated by nurses, with one doctor assigned to several of these clinics. A central clinic is situated in Goose Bay where the physicians reside, each one responsible for a few remote nursing clinics. Currently, nurses at these remote stations have received sufficient training so that they are able to treat most cases. If a consultation with the doctor is required, the nurse simply telephones the doctor, describes the case and awaits his or her instructions. Using such a system allows for quick consultations and diagnoses but has its drawbacks as well. Describing a case verbally can often be time-consuming and the use of images (both still and moving) would surely shorten the time required to brief the doctor on the case, making it possible to arrive at a diagnosis even sooner and may provide additional information that would be otherwise difficult to convey.

All these issues surrounding health care in Newfoundland and Labrador, point to the deployment of telemedicine. Looking at existing solutions, one notices that many of these systems require high bandwidth lines in order to use them effectively. The conclusion that can be reached from this, that telemedicine systems require high-data rate communication lines, is erroneous. An interactive audio network was developed for the transmission of medical data and for teleconferencing in Newfoundland and Labrador, and succeeded in providing a model for the low-cost use of telemedicine

technology. This system uses the existing telephone lines and proves that telemedicine applications need not require high-data rate lines [Hous77].

In 1979, the Telemedicine Centre at Memorial University of Newfoundland began a teleconference system and by 1980, the three original teleconference circuits were operating at full capacity and expansion was necessary. This led to the creation of TETRA, the Telemedicine and Educational Technology Resources Agency, founded jointly between Telemedicine and the Division of Educational Technology. The teleconference network built is one of the largest and most sophisticated audiographic teleconference networks in Canada [Hous94], divided into 11 separate teleconferencing circuits with over 200 sites in approximately 150 communities throughout the province. In addition to the audioconferencing capabilities, telewriter workstations are available providing blackboarding functions to the users. The current capabilities of the network include:

1. Audioconferencing

   The current platform makes use of voice grade facilities and end equipment, supporting the real-time exchange of audio and graphic information. The end equipment consists of analog audioconference units, PC based stations, interface communications devices, and applicable software packages.

2. Medical Data Transfer

   Various services are provided for the transmission of medical data from rural and remote sites to the General Hospital in St. John's including remote nuclear medicine, tele-electroencephalogram, tele-ultrasound, and tele-radiology.

3. Interactive Audiographics

   The telewriter workstations or "electronic blackboards" available in over 140 sites allow for the interactive transmission and annotation of visual material during a teleconference. The telewriter can also be used for freehand writing or drawing in real-time or to display computer graphic pages.

4. Videoconferencing

   Videoconferencing services are provided over primary and basic rate services, allowing users to see each other and exchange visual materials in either a point-to-point or multi-point configuration. Access to this service is limited to areas supported by digital telecommunications systems.

2.6 Typical Scenario

As mentioned previously, the typical scenario for the telemedicine application developed is in the nursing stations situated in Labrador. The current situation is that small communities are each equipped with a medical clinic manned by nurses. The

training and area of expertise of these nurses is broad enough such that they are capable of handling most patient cases that they face. In the event that a nurse comes across a case that (s)he cannot immediately treat, (s)he consults with a physician at a central clinic in Goose Bay. At this central clinic, the physicians responsible for the nursing clinics are housed. Each physician at the central site is responsible for a number of these remote nursing clinics.

This basic steps of the teleconsultation between the nurse and doctor would occur as follows:

1. Medical data relating to the case is obtained by the attending nurse during the course of the day. This would include pictures, video and audio recordings of the patient.
2. The collected data would be sent electronically to the physician at some pre-determined time, e.g. in the middle of the night to save on long distance costs. This transmission of data can use any one of a number of possible methods – electronic mail, direct transmission to the physician's computer or the posting of the data to a central patient database situated on the World Wide Web. In cases requiring urgency, teleconsultation participants are able to send medical data immediately and not wait for the batch sending of the patient encounter.
3. Upon reviewing the patient's case, the physician would provide instructions as to treatment or request additional information from the attending nurse and transmit this to the nurse.

CHAPTER THREE

DESIGN OPTIONS AND ISSUES

3.1 Problem Statement

In an effort to increase the quality of care provided to residents of Labrador specifically and, in general, to residents of remote communities, telemedicine applications are being developed. However, economic considerations and the current rarity of consultation sessions, precludes the installation of high-speed transmission lines required by many available telemedicine solutions, assuming qualified medical expertise is available. Many of the commercially available teleconsultation applications are geared towards conference-style consultations which we believe to be counter productive, and do not provide a means for "batch-mode diagnosis" where multiple patient cases can be

sent/retrieved, to be looked at by the consultant physician at his/her convenience. In addition, the model used by these applications is often at odds with the model used by clinicians during conventional consultations. To ensure that remote communities gain access to these much-needed services, we have developed three asynchronous teleconsultation applications for the purpose of batch mode diagnosis, that attempt to fit the consultation model envisaged by physicians and nurses and that can be operated at low-data rates.

3.2 Design Constraints

A constraint involved in developing a solution to the proposed problem is that the application must function properly with only a basic telecommunications infrastructure in place. The typical operating environment of such a system is such that high-speed transmission lines will not be available, whether it be for economic considerations or otherwise. Low-data rate communication lines, on the other hand, will either already be in place or can be put in place rather easily (e.g. wireless services). As a result, a low-data rate solution must be developed.

Secondly, any developed solution must enhance the consultation experience without increasing its complexity. It must be kept in mind that care providers will need to spend their time providing patient care rather than focusing on the methods necessary to communicate with others assisting in the care. Simply put, the care-giver must be

able to spend his/her time and resources healing rather than overcoming obstacles in co-operating with others involved in the healing [Moor96].

The last constraint is with regards to the operating platform of any developed solution. Due to the widespread availability of personal computers (PCs) running the Microsoft Windows© 95 operating system, any developed solution must be compatible with this operating system.

3.3 Design Criteria

The successful teleconsultation system will allow health care-givers to devote much of their time to care-giving, requiring little or no training to use. Additional criteria of success are that the developed system will provide at least the same level of information regarding patients, will be as efficient as the consultation methods in use today, and to improve the consultation process. The cost of implementing such a system was not considered during the course of this research for the following reasons:

1. Currently, no cost has been associated with the developed teleconsultation systems. However, once a price tag has been applied to the developed systems, the cost of purchasing the application will become a factor in determining its success.
2. Due to the wide variety of computer equipment and peripherals available, the possible configurations for a teleconsultation system are too many to research. Doing so would also be out of the scope of the research as these systems were

developed in response to a problem in Newfoundland where hospitals and clinics, even remote ones, already had available to them the equipment used by the developed systems.

3.4 Alternative Solutions

Before alternative solutions can be developed, the issues involved in the design of such a system must be determined. These concern the mode of operation and the multimedia items generated and shared in a telemedicine system. It is worth noting that the issues concerning the multimedia items exist **not** because the items are of a multimedia nature. They arise from the mental model that users have when performing a consultation regarding the nature of the record items generated and shared. They include:

1. Status of components
   A. Part of permanent electronic patient record.
   B. Temporary messages independent of permanent patient record.
2. Storage of components
   A. In one or few central locations.
   B. At all relevant locations.
   C. While relevant, during exchange of messages only.

3. Access to components

   A. Single user at send or receive site.

   B. Any number of users at send and receive sites.

   C. Any number of users anywhere.

Regarding the mode of operation, an asynchronous application, as opposed to a synchronous one, is preferred; i.e. store and forward versus real-time. The reason for this is not primarily because of the technological requirements, as a low-data rate synchronous teleconsultation system is feasible, but as a result of the human interaction involved in the consultation process [Alle95]. Consequently, the co-ordination of patient care is typically carried out in an asynchronous manner via the patient record.

While asynchronous teleconsultation is natural, instances will arise where a synchronous mode of operation is desirable and will be reserved for urgent cases, matters that require clarification or out of a need for personal contact. Consultations on a synchronous basis alone will reduce efficiency, as all participants will have to commit time to the connection while one is reviewing the materials [Alle95].

In developing a teleconsultation system, it is preferable to take advantage of standards where possible. Hence, standard file formats for the multimedia items of a patient record were used, i.e. JPEG, WAV, AVI. The one area where this principle was not followed was in the design of the patient record/encounter viewer. Work in the development of standards for electronic patient records is currently taking place and we

did not wish to develop a structure for the record that would be incompatible with standards chosen in the future. It was decided that a generic structure for the record would allow us to provide organization in the records and when a standard for the electronic medical record is chosen, existing patient records can be easily converted to conform to the standard.

Since the system under consideration is to be used for medical consultations, several possibilities exist: mail-based systems, videoconferencing systems, Web-based systems, and medical database systems. Each of these possible solutions will be investigated and the merits of each will be identified.

3.4.1 Mail-based Systems

Mail-based systems are the current state-of-the-art in telemedicine. The concept is quite simple – medical data is transferred using electronic mail. In the context of teleconsultation, each e-mail message would correspond to a patient encounter or record.

The advantages of such a system are:

- It is asynchronous as the user would create a mail message, send it, and it is sent to the recipients at a later point in time. As such, it is a natural choice for consultations.
- Electronic mail makes communication on a one-to-one or one-to-many basis an easy one.

- Using electronic mail allows for the inclusion of multimedia items into the message.
- Electronic mail is analogous to existing methods used for consultations – courier, mailing, etc.

The disadvantages of such a system are:

- As is the case with electronic mail, there is no inherent structure between subsequent mail messages. The end result is that organizing the separate messages (patient encounters) into some structure or hierarchy is a daunting task for the user, as this cannot be performed automatically by the system.
- Although electronic mail makes communication possible on a one-to-one or one-to-many basis, collaboration between groups of people is not a simple process, involving the sending and receiving of numerous messages.

3.4.2 Videoconferencing Systems

Videoconferencing systems are becoming quite popular and may soon supplant electronic mail systems as the state-of-the-art. Videoconferencing uses computing and telecommunications technologies to provide conferencing services with full audio and video capabilities between a number of users. Many systems also provide text-based entry and whiteboard capabilities as well. It is of no doubt that the ability to see and hear someone during a conference greatly adds to the experience.

However, several drawbacks exist with such systems.

- The first is that they require the users to be present at the same time to participate in the conference. Physicians typically are unable to set aside time for videoconferences and, consequently, view such systems as having little or no value to teleconsultations.

- The second drawback to such systems is that the video data, the main feature of such systems, requires a great deal of bandwidth and computer processing power to use. The high bandwidth requirements of such systems eliminates the possibility of their use in remote locales where bandwidth is a costly resource where the CPU requirements usually results in choppy or sluggish video sequences. The somewhat jerky motion of video not only presents a hindrance to consultations rather than a benefit but is ineffectual where full-motion video is required, e.g. for assessing gait or other physical signs or in other applications where smooth video is necessary.

3.4.3 Web-based Systems

The increasing popularity of the World Wide Web has led to the emergence of Web-based or client-server telemedicine applications. The Web browser is used to view the patient record (e.g. *TeleMed* – see **Section 2.1**), presuming the user is authorized to access the data. The centralization of medical data at one or few sites allows for the simultaneous viewing of a patient's medical information by a number of people, enriching the collaboration experience. The existence of standards for the displaying of multimedia information will serve to lessen the burden on low bandwidth communication lines when viewing patient record items. The issues that need to be addressed with such systems involve controlling access to patient data and how adding to or editing a patient's record would be accomplished. The natural method of editing or adding a patient record would use a form with data fields for the patient information and the data items of the encounter. User interface issues arise from this as considerations of how to handle the addition of the data items (insertion by direct capture or by specifying a file to which the data has been saved) and the display of the form itself (spawning a new window may complicate the desktop), among others, must be resolved. Additionally, issues arise concerning the integrity of the patient data. As the Web allows multiple, simultaneous access to the data, it is possible for simultaneous editing of a patient record to occur. A related concern is how to display a patient record to users after it has been changed by another user.

3.4.4 Medical Database Systems

Certainly not new by any means, these systems utilize databases containing patient medical data. As with any database system, authorized access allows a user to select and view a patient's medical information. As with Web-based systems, the unresolved issues concern data security and integrity. Since the systems considered here are to be used for remote medical consultations, how data retrieval and patient record editing are to occur must be determined.

In developing a medical database system, the primary design consideration is the type of database to implement – standalone or networked database. Each has their own advantages and disadvantages, with different implications for a teleconsultation system. In a standalone system, only one site is involved and all data operations involve that site alone. Networked database systems involve a number of standalone systems connected by some communications medium for the transfer of updates made to the database. The data stored can be either centrally located or distributed between the participant sites. Networked databases can also be either single-user systems or multi-user systems. In a single-user system, only one user can access a record at a time whereas a multi-user system allows multiple users concurrent access to the database. The single-user system avoids problems of data integrity but introduces other problems, such as two users wishing to access a record simultaneously but only one is allowed access. Data integrity problems arise with multi-user systems and affect the validity of data when multiple users attempt to edit a record [Date90].

The implications of a standalone system for a teleconsultation system are that all patient data is located on a machine at the clinic site. If patient data is requested at another site, the entire patient record must be sent to the requesting site. If the patient's record is altered, the updated record must be sent to all sites that contain that record. This introduces its own set of problems as now multiple copies of a patient record exist on different systems and sorting which record is valid and which is not becomes an issue. Networked systems get around this problem as the system updates the patient records after changes are made but, as mentioned above, concurrency issues arise with multiple users accessing a record. Access of the patient records themselves in a networked system may require high bandwidth lines for temporary local storage of the record during access, if a centralized database is used. Once the record is closed, the local copy is removed and updates are made to the "master" copy. More on the types of networked databases available will be discussed next in **Section 3.5**.

3.5 Solution Set

The scope of the project is to develop possible solutions for use and investigate which of the alternatives provides the best "fit" with the mental model of the consultation process held by the users.

For the investigation, two proposed solutions were initially chosen: the electronic mail-based solution and the Web-based solution. The mail-based approach was chosen for

the simple reason that mail-based solutions are the current state-of-the-art. The Web-based approach was also chosen due to the increased interest in the Internet and the collaborative advantages offered over the mail-based solution. During the course of development of these alternatives, it was felt that a third solution using the database model could be of benefit to users, providing the advantages of both these systems. In the envisioned usage model, a networked database model is appropriate and three alternatives are available: the centralized database, distributed database and replicated database. The pros and cons of each will be discussed.

3.5.1 Centralized Database

A centralized database is like a standalone database in that only one copy of the database exists but allows multiple users to access it. Making use of such a database avoids the pitfalls of ensuring the integrity and validity of the data contained within. However, the disadvantages of such a system become apparent when users from remote sites attempt to access the database. The size of the database record and the capacity of the communications medium affect access by increasing the latency for data access.

3.5.2 Distributed Database

In a distributed database, the database is separated into pieces with these pieces being distributed across a number of systems. All the systems involved are interconnected and each has autonomous processing capability for serving local applications. In such

systems, data access involves retrieving data from other sites and is hidden from the user. Distributed databases have a number of advantages over centralized databases:

- As the number of sites involved grow, each can be added with little or no upheaval to the database management system. Compare this to a centralized system where growth requires upgrading with changes in the hardware and software that affect the entire database.

- Efficiency in distributed databases is achieved by locating the databases close to the anticipated point of use, minimizing latency for data access. Flexibility is achieved by being able to dynamically move or replicate the data if usage patterns change.

- In a distributed database, if one node goes down, the overall system is still available and the remaining sites continue to function. Greater reliability and accessibility can be achieved by replicating the data so that all the data is still available.

The problems with distributed databases are their cost and complexity when compared to a centralized system. The additional complexity is a direct result of hiding the database's distributed nature from users. The increased complexity also means that the acquisition and maintenance costs of a distributed system are greater than a centralized system. Additionally, a large communications overhead is

associated with distributed databases for the co-ordinating messages between the different sites.

3.5.3 Replicated Database

The replicated database is a variant of the distributed database where duplicate versions of the database are made and distributed among the participating sites. One important feature of replicated databases is the synchronization of replicas so that changes made to one replica are reflected in all the others. Replicated databases share the same advantages of distributed databases with the added advantage of low-latency access to the data as it resides locally. If implemented correctly, the replication should be hidden from the user creating the illusion that the user is accessing a centralized database residing on his/her machine. The disadvantages of replicated databases arise from the synchronization process for sites connected by low-data rate links and from the possibility of conflicts between database replicas. As updates to a record are reflected in all replicas, this could prove inefficient for a large number of changes if a low-data rate link is used.

After considering the alternatives available for a network database, it was decided that a replicated database system would be used for the efficiency it offered over the choices available. To overcome the problems with replicated databases, primarily the avoidance of transferring large amounts of data between sites, some algorithm would

need to be developed to achieve this. The algorithm developed is explained in **Section 4.4.2.**

3.6 General User Interface Issues

In considering the issues involved in the development of applications, the design of the user interface is an important aspect. A difficult to use interface has two implications for an application; the application will not be used, affecting sales, and productivity of users will drop [Eber94]. Consequently, the design of the interface can be considered to be as important as the functionality of the application. The interface of an application can be improved in a number of ways: the use of appropriate metaphors and analogies and the consideration of the window and menu styles used.

3.6.1 Metaphors and Analogies

One common metaphor used widely in computers today is the desktop metaphor and is one of the most successful. In order to incorporate a metaphor, the user must be able to apply, old, familiar knowledge to a new situation and must provide a reasonable match to the new knowledge. Simply ensuring a reasonable match between the metaphor and the activity is insufficient to make an interface good; the metaphor must be used consistently. For example, the Macintosh interface makes use of a desktop metaphor, using windows to represent folders that would be obtained from a filing cabinet. The contents of each folder, when opened, will appear in a separate window. These windows can be overlapped so that only the topmost

window can be seen (see **Figure 3–1**). This is akin to laying folders on a desk and seeing only the topmost folder(s) completely. Using the desktop metaphor, the user would expect that any folders stacked on a desk will keep their position unless otherwise manipulated by the user. This, however, does not always occur with the Macintosh operating system. In particular, if the user has several folders stacked on the desktop and then begins some application, once the application is finished the folders should have the same stacking arrangement as before. This is not the case if the folders are from different storage devices. When an application is finished, the Macintosh operating system restacks the folders as follows: folders off the internal hard disk, folders off any external hard disks, and then folders off any diskette. If a folder from the internal hard disk was originally on the top of the stack before the application was started, this folder will now be on the bottom, below the external disk and diskette folders (see **Figure 3–2**) [Eber94]. This behaviour can be predicted by the user but is different from what is expected if the desktop metaphor is used correctly.

One of the problems with using metaphors is that for some operations no match may be found between the known task and the target task or the match will be clumsy. As an example, how would a user eject a disk from the computer using the desktop metaphor? In the Macintosh interface, ejecting the disk can be performed by dragging the disk icon to the trash can and then releasing the mouse. As the trash can is used

for the deleting of files, a novice would be understandably reluctant to "trash" his/her disk, thinking that the files would be deleted.

With greater amounts of available memory in today's computers, operating systems are able to incorporate a large amount of graphics into the interface to enhance usability. One of the most popular uses for graphics is in icons representing some object or function on the screen. There are many reasons for why an icon representation of an object or function may be appropriate. Research has shown that people can recall and recognize very large numbers of pictures [Paiv71], [Stan70], far better than words.

In the context of the telemedicine applications developed, the metaphors used for the interface are inbox/outbox for the listing patient records and encounters and a record browser for the encounters. Icons will be utilized for buttons where possible to indicate the operation performed. Alternatively, text buttons will be used. Icons will also be used to represent the data items of the patient encounter.

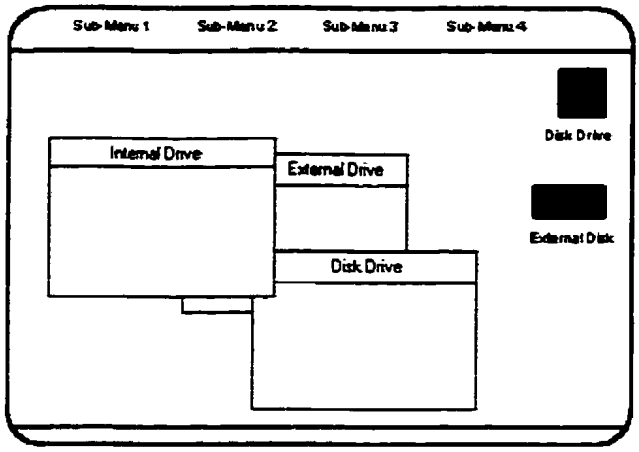

Figure 3–1. Stacking of folders on desktop before entering application.

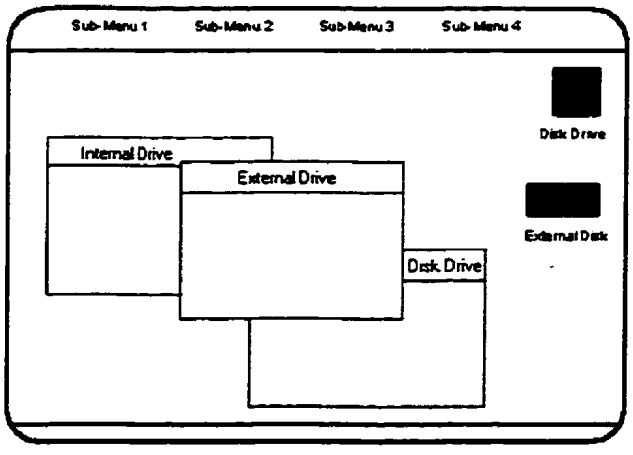

Figure 3–2. Stacking of folders on desktop after leaving application.

3.6.2 Window Styles

As a greater number of applications produced today utilize a graphical user interface, the use of windows is quite common. It is thus appropriate to discuss the types of windows available and the uses for each.

Single vs. Multiple Windows

In developing a windowed application, the first option is whether to use a single window or to use multiple windows. A single window avoids cluttering the screen and presents all the information in a single window. However, if different types of information are to be presented (e.g. a list of patients and the data for each patient) then multiple windows are a good idea, each presenting different types of information. Multiple windows also enable metaphors to take root (e.g. an index card of patients on record and a folder of the patient data). However it is quite easy to get carried away and clutter the desktop with too many windows.

Tiled Windows

Tiled windows take their name from floor or ceiling tiles and appear in one plane on the screen, expanding or contracting to fill up the entire screen (see **Figure 3–3**). Tiled windows are felt to have the following advantages [Gali97]:

- The system usually allocates and positions windows for the user, eliminating the necessity to make positioning decisions.

- Open windows are always visible, eliminating the possibility of them being lost or forgotten.

- Every window is always completely visible, eliminating the possibility of information being hidden.

Perceived disadvantages include:

- Only a limited number can be displayed in the screen area available.

- As windows are opened or closed, existing windows change in size. This can be annoying to users.

- As the user changes window sizes or positions, the movement of the other windows can be distressing.

- As the number of windows displayed increases, each window can get very small.

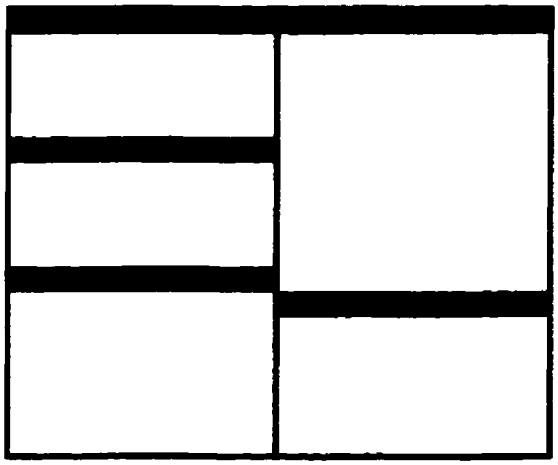

Figure 3–3. Tiled Windows

- The changes in sizes and locations made by the system are difficult to predict.

- The configuration of windows provided by the system may not meet the user's needs.

- They are perceived as crowded and more visually complex because window borders are flush against one another and they fill up the whole screen. Crowding is accentuated if borders contain scroll bars and/or control icons. Viewer attention may be drawn to the border, not the data.

- They permit less user control because the system actively manages the windows.

Overlapping Windows

Overlapping windows (see **Figure 3–4**) may be placed on top of one another like papers on a desk. They possess a three-dimensional quality, appearing to lie on different planes. Users can control the location of these windows, as well as the plane in which they appear and the size of some types of windows may also be changed. This type of window is most commonly used by applications. Overlapping windows have the following advantages:

- Visually, their look is three-dimensional, resembling the desktop that is familiar to the person.

- Greater control allows the user to organize the window to meet his/her needs.

- Windows can maintain larger as well as consistent sizes.

- Windows can maintain consistent positions.

- Conserving screen space is not a problem as windows can be placed on top of one another.

- There is less pressure to close or minimize windows no longer needed.

- The possibility exists for less visual crowding and complexity. Larger borders can be maintained around window information, and the window is more clearly set off against its background. Windows can also be expanded to fill the entire display.

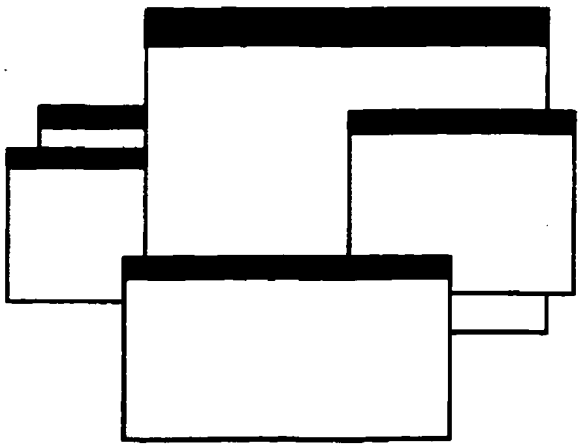

Figure 3–4. Overlapping Windows

Disadvantages include the following:

- They are operationally more complex than tiled windows. More control functions require greater user attention and manipulation.
- Information in windows can be obscured behind other windows.
- Windows themselves can be lost behind other windows and be presumed not to exist.
- Control freedom increases the possibility for greater visual complexity and crowding. Too many windows or improper set-off can be visually overwhelming.

Cascading Windows

Cascading windows are a special type of overlapping window that has the windows arranged automatically in a regular procession. Each window is slightly offset from others (see **Figure 3–5**). While suffering the same disadvantages as overlapping windows, cascading windows offer these additional advantages:

- No window is ever completely hidden.
- Bringing any window to the front is easier.
- Simplicity in visual presentation and cleanness.

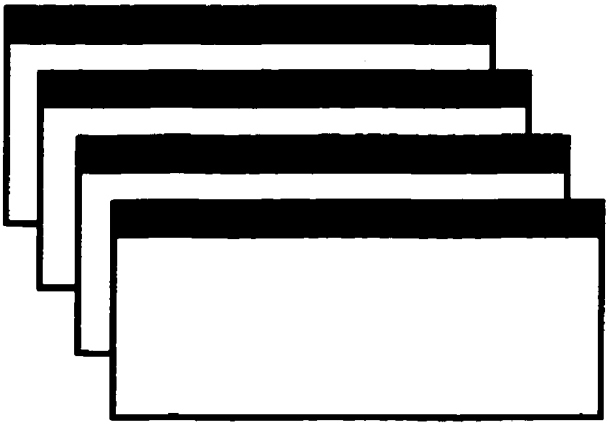

Figure 3–5. Cascading Windows

3.6.3 Menu Styles

Providing the proper kinds of menus to perform tasks is critical to system success. The best kind of menu in each situation depends on several factors. The following must be considered:

- The number of items to be presented in the menu.
- How often the menu is used.
- How often the menu contents may change.

Each kind of graphical menu will be described in terms of its purpose, advantages and disadvantages.

Menu Bar

A menu bar consists of a collection of action descriptions typically arrayed in a horizontal row at the top of a window. Menu bars are effectively used for presenting common or frequent actions to be used on many windows in a variety of circumstances. Menu bars often consist of a series of textual words as in **Figure 3–6**. Some applications place the choices within buttons as in **Figure 3–7**. The advantages of menu bars are that they are:

- Always visible, reminding the user of existence.
- Easy to browse through.
- Easy to locate consistently on the screen.
- Usually do not obscure the working area.
- Usually are not obscured by windows and dialog boxes.
- Allow for the use of keyboard equivalents.

The disadvantages are:

- They consume a full row of screen space.
- They require looking away from the main working area to find.

- They require moving the pointer from the main working area to select (unless keyboard equivalents are used).
- Menu options are smaller than full-size buttons, slowing selection time.
- Horizontal orientation is less efficient for scanning.
- Horizontal orientation limits the number of choices that can be displayed.

Figure 3–6. Menu Bar Comprised of Text

Figure 3–7. Menu Bar Comprised of Buttons

Pull-Down Menus

Selection of an alternative from the menu bar result in the display of the exact actions available to the user. These choices are displayed in a vertical listing that appears to pull down from the bar – hence the name "pull-down" menus (see **Figure 3–8**). Pull-down menus are used to provide access to common and frequently used application actions. They are most useful for a small number of rarely changing items, usually five to ten [Gali97]. A greater number of choices becomes awkward to use, best handled by cascaded menus (see next section). The advantages of pull-down menus are:

- Reminder of existence cued by menu bar.
- May be located relatively consistently on the screen.
- No window space consumed when not used.
- Easy to browse through.
- Vertical orientation most efficient for scanning.
- Vertical orientation most efficient for grouping.
- Vertical orientation permits more choices to be displayed.
- Allows for both keyboard equivalents and accelerators.

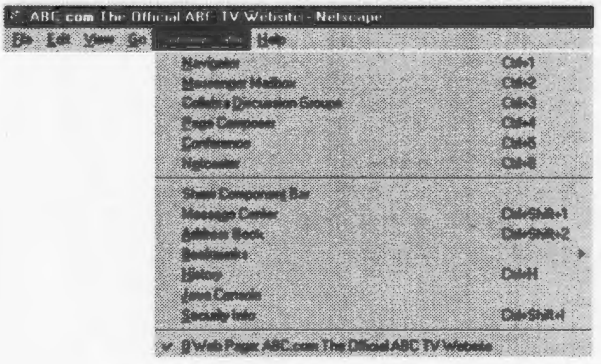

Figure 3–8. Pull-down Menu

The disadvantages are:

- Requires searching and selecting from another menu before seeing the options.
- Requires looking away from the main working area to read.
- Requires moving the pointer out of the working area to select (unless keyboard equivalents are used).
- Items are smaller than full-size buttons, slowing selection time.
- May obscure screen working area.

Cascading Menus

A cascading menu is a submenu derived from a higher-level menu, typically a pull-down but can be also attached to other cascading menus or pop-up menus. Cascading menus are located to the right of the menu item on the previous menu to which they are related (see **Figure 3–9**). Menu items that lead to cascading menus are typically indicated by a right-pointing triangle. These menus are used to simplify menus by reducing the number of choices that appear on one menu. Cascading menus can be used when many alternatives exist that can be grouped meaningfully. The advantages of cascading menus are:

- The top-level menus are simplified because some choices are hidden.

- More first-letter keyboard accelerators are available as menus possess fewer alternatives.

- High-level command browsing is easier because sub-topics are hidden.

The disadvantages are:

- Access to sub-menu items requires more steps.

- Access to sub-menu items requires a change in pointer movement direction.

- Exhaustive browsing is more difficult; some alternatives remain hidden as pull-down menus become visible.

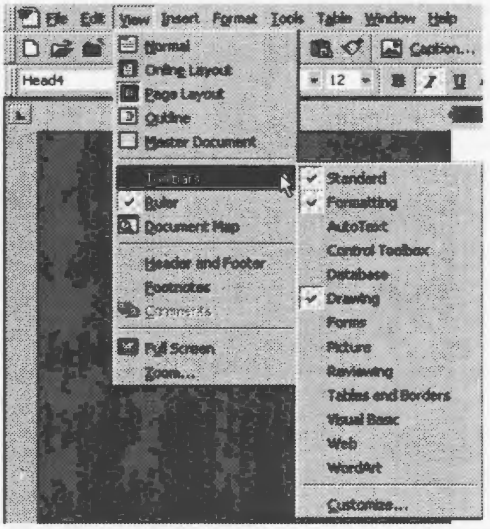

Figure 3–9. Cascading Menu

Pop-Up Menus

Choices may also be presented to the user on the screen through pop-up menus, vertical listings that only appear when specifically requested, for example by clicking the right mouse button (see **Figure 3–10**). The choices available in pop-up menus are context-sensitive, depending upon where the pointer is located when the request is made. They are useful for providing the user with options regarding the immediate task. For instance, when positioned over text, a pop-up menu would provide text-specific commands. Advantages of pop-up menus are:

- It appears in the working area.
- It does not consume window space when not displayed.
- No pointer movement is necessary if requested by a button.
- Items are vertically oriented for efficient scanning.
- The vertical orientation provided efficient grouping.
- Vertical orientation allows for choices to be displayed.
- It may be forced to remain showing ("pinned").
- It allows for the display of both keyboard equivalents and accelerators.

The disadvantages of pop-up menus are:

- Existence must be learned and remembered.
- Means for selecting items must also be learned and remembered.
- It requires a special action to see the menu (mouse click).
- It may obscure the screen working area.
- Display locations may not be consistent.

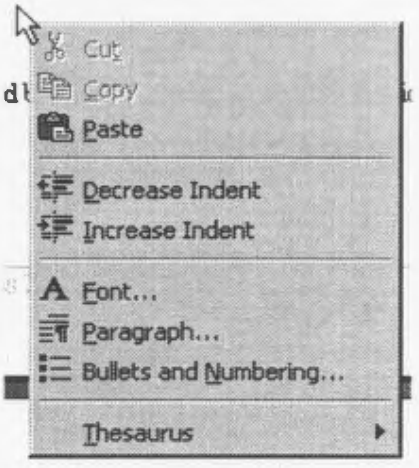

Figure 3–10. Pop-up Menu

Tear-off Menus

A tear-off menu is a pull-down menu that has been positioned on the screen for constant referral and, as such, contains all the characteristics of a pull-down menu. It is sometimes referred to as a *push-pin* or *detachable* menu. Its purpose is to provide the user with options that are needed infrequently at some times and heavily at other times.

The advantages of the tear-off menu are that it consumes no space on the screen when not needed and can remain continuously on the screen when required. The disadvantages are that it requires extra steps to retrieve and may obscure the working area.

Iconic Menus

An iconic menu is the representation of menu items or objects in a graphical or pictorial form (see **Figure 3–11**). The purpose of an iconic menu is to remind users of the functions, commands, attributes, or application choices available.

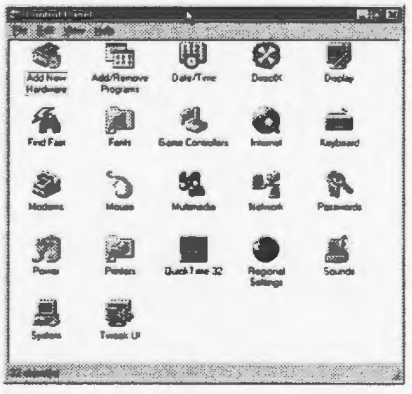

Figure 3–11. Iconic Menu

The advantages of iconic menus are that they help facilitate memory through the use of pictures and their larger size increases the speed of selection. The pictures, however, do consume more screen space than text and are difficult to organize for scanning efficiency. The creation of meaningful icons requires special skills and an extended amount of time.

3.7 Input and Output Modalities

The input of media into the telemedicine systems can take on a number of forms: text, audio, still images, and moving images. Obviously, textual data will be entered via the keyboard. As for audio, images and videos, standard computer peripherals (microphone, videoconferencing camera) may be used to capture the media. Specialized medical equipment also exists that can be connected to the computer for capture of the media: electronic stethoscopes, episcopes for dermatological exams, etc. Standard Windows drivers are used for the media capture allowing the use of any hardware, provided it can be connected to the computer either directly or via a capture card. The downside of using generic drivers is that the capture may be slowed as the driver is not optimized for the device and specialized functionality in the device may not be accessible through the driver.

In addition to the standard input media, annotated images are possible. This medium consists simply of graphics overlaid on still images. The annotation can be performed using a number of devices – graphics tablets, electronic pen-based technologies and computer mice.

CHAPTER FOUR

SOFTWARE DESIGN

4.1 Introduction

In this section, the software designs of the three teleconsultation applications are presented including the functional specification, input and output interfaces, and a high-level view of the data objects used and their interactions. In this section, the term "user" is frequently used and refers to the targeted users of the developed systems; namely, physicians, nurses, and clinicians who would participate in teleconsultations. Typically, the nurse would create teleconsultation cases and the physician or clinician would receive them and generate a response to be sent to the nurse.

4.1.1 User Interface Design Decisions

In response to the user interface issues presented in the previous chapter, the following design decisions concerning the user interface were made, impacting the design of each system:

1. An inbox/outbox analogy will be used to present the available patient cases. The specifics of the information displayed will depend upon the specific system. More detail concerning this is provided in the **Input and Output Interfaces** section of the software design for each application.
2. To display the patient cases or encounters, a "browser" will be used containing patient related information and a complete iconic listing of all the components of the patient encounter.
3. Since multiple windows exist, the overlapping window style will be used to display the windows which will initially be positioned in the centre of the display. The user will be free to move the windows and a button for each open window will be available on the Windows© taskbar to provide easy and rapid access to each window.
4. To minimize the amount of textual information presented to the user in the interface, iconic pushbuttons will be used in place of menus. The appropriate selection of icons for the actions will serve to speed up menu selection and lessen the amount of training required.

5. In some instances, pop-up menus will be used providing advanced options available to the user.
6. To display system information and messages, modal dialogs will be made use of, ensuring that the information is brought to the user's immediate attention.

4.1.2 Use Case Model

Using Jacobson's "use case" approach, a model was developed to define the objects that interact with the teleconsultation system and the functions performed by the system. Using this approach, the use case model shown in **Figure 4–1** was obtained.

Actors

"Actors" in a use case model represent those objects that interact with the system under consideration and have a requirement to exchange information with the system. An actor is different from a user in that a user is an actual person using the system while an actor represents a certain role that a user can play. Two actors were identified for the teleconsultation system:

1. Consultant: Performs the role of consultant, receiving patient cases and creates responses.
2. Clinician: Conducts patient examinations, creating patient cases to be sent to the consultant.

Use Cases

The "use cases" represent a specific way of using the system and are a sequence of transactions performed by the user in a dialogue with the system. In the teleconsultation system developed, two primary transactions exist that can be performed by a user:

1. Create new patient case: This transaction involves the creation of a patient case resulting from a patient encounter (visit) and consists of case items obtained during the course of the examination. The created patient case is then sent to the consulting physician.
2. Respond to consultation case: In this transaction, the consulting physician, reviews the received patient case, viewing the items of the case and then generates a response to the case originator.

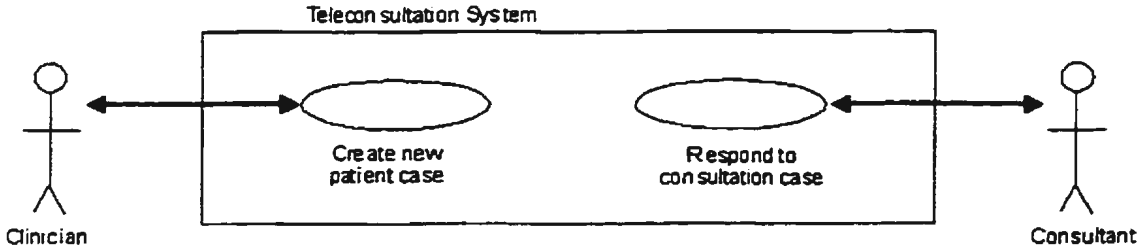

Figure 4–1. Use Case Model

4.2 *MailTeleMed* – Mail-Based Model

4.2.1 Overview

The first system developed was an electronic mail-based system as such systems are the current state-of-the-art. In these systems, each mail message corresponds to a separate patient consultation and is comprised solely of attachments representing the items obtained during the course of the consultation. Performing a consultation using such a system involves conducting an examination of the patient to obtain data, packaging the collected data into a message and sending the message to the other participants in the consultation. When the mail message is received, the patient data are viewed and any response to the message is sent back to the original sender (and to the other recipients of the message). This latter step may also be performed using some other means such as videoconferencing or a telephone call or existing standard mail applications (e.g. Microsoft Outlook, Netscape Messenger, Eudora).

4.2.2 Functional Specification

The mail-based system, enabling medical teleconsultation using electronic mail, will have the following functionality:

- Retrieval and sending of patient encounters (mail messages).
- Parsing of the encounter items from a message.
- Amalgamation of created encounter items into a mail message.

- Display/playback of encounter items upon their selection.
- Storage of patient encounters to disk.

4.2.3 Input and Output Interfaces

The input and output interfaces of the *MailTeleMed* system result from the nature of the data being presented, i.e. patient encounters, and from the user model of the consultation process. In the model corresponding to the electronic mail-based system, the patient cases are generated at the remote station on a needs basis and sent to the consulting physician. The physician then creates a response which is received at the remote station the next day. With this model, no opportunity for synchronous communication exists and only asynchronous communication can occur. As a result, the physician or nurse is presented with a list, each entry corresponding to an encounter of a particular patient (see **Figure 4–2**). The end result of using such a model is that this list of encounters soon becomes very large with numerous entries for each patient, each one for a different encounter with that patient. Consequently, management of the patient encounters becomes a time-consuming and daunting task. As for the encounters themselves, a browser is used to display the encounter and its items (see **Figure 4–3**). An iconic list on the browser is used to display the contents of the encounter using icons to represent the various types of items. Selection of a particular item will display the item. Picture buttons are provided in the browser corresponding to the possible actions the user can pursue, the insertion of a new item in the encounter, mailing the encounter to the intended recipients and for saving the

encounter to disk. User input will take on a multitude of forms – from simple point and click operations using the mouse to the entry of encounter data. Depending upon the encounter data to be entered, the method of input will be one of keyboard entry for text data, a video capture device for still and moving images and a microphone or other audio device for audio data. Keyboard entry will also be required for entry of key data fields, e.g. patient name. System output to the user will be displayed in message dialogs requiring acknowledgement or a decision from the user.

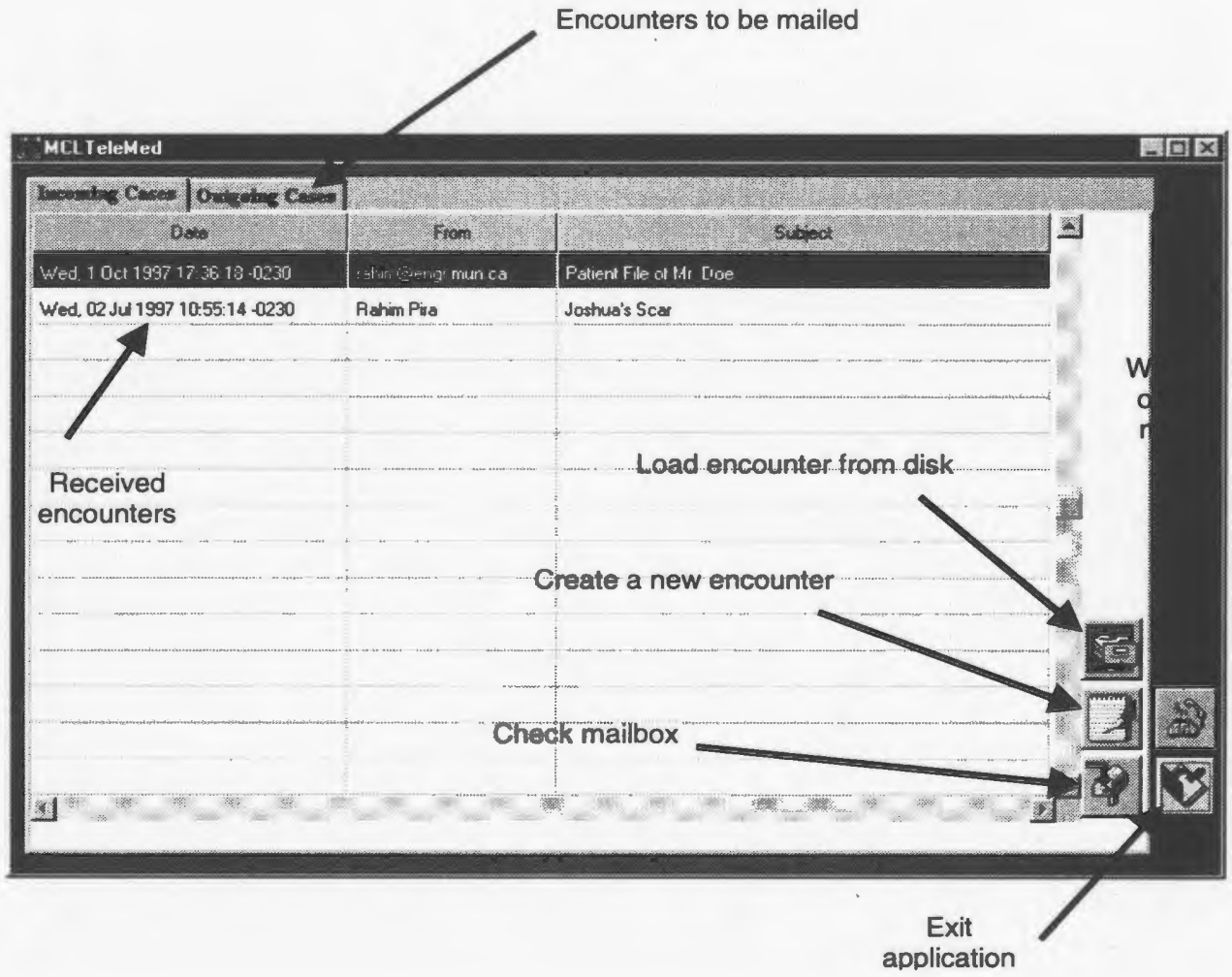

Figure 4–2. Listing of Patient Encounters

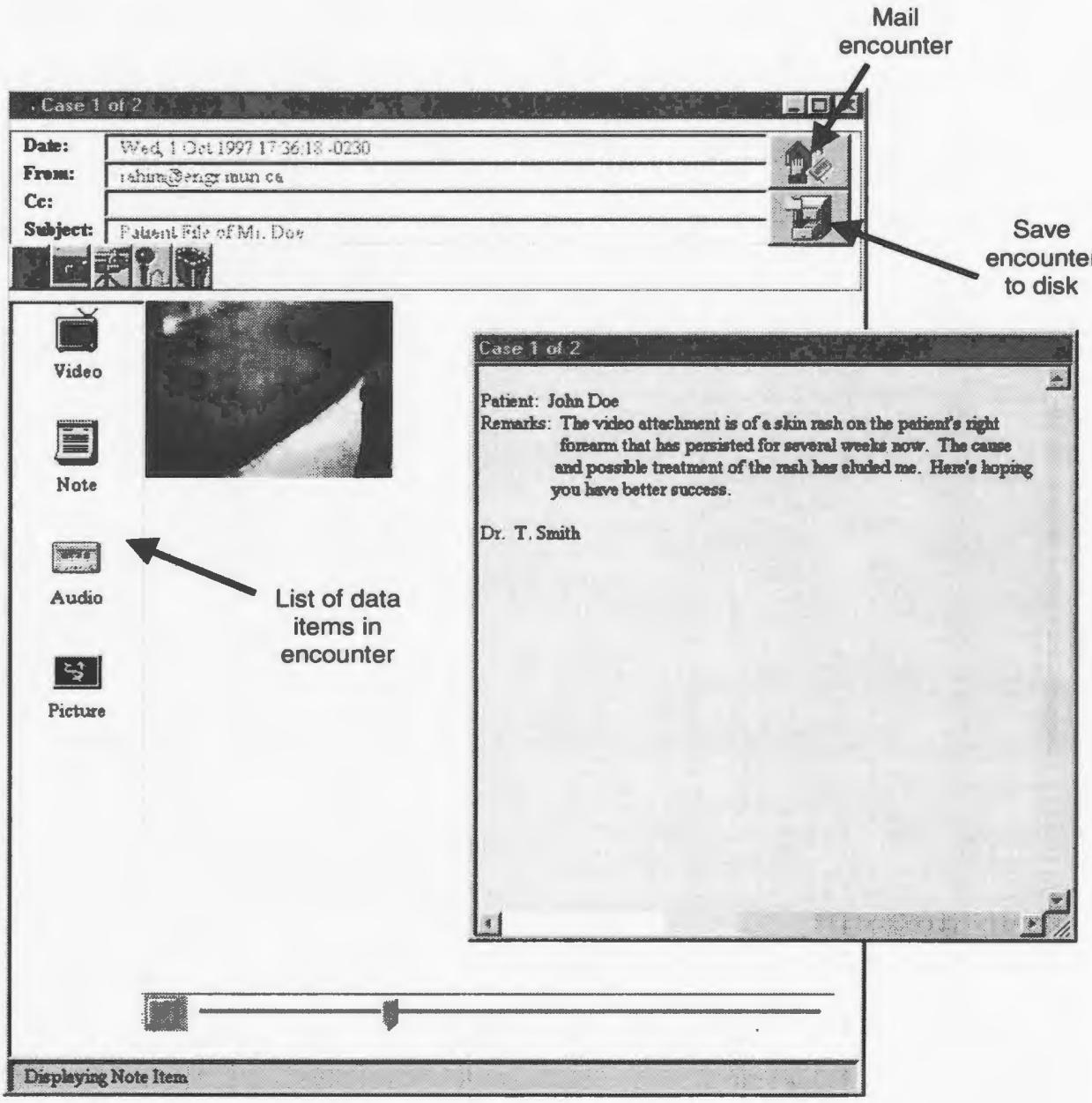

Figure 4–3. Display of Patient Encounter with Items Selected

4.2.4 High-Level Data Objects

Since *MailTeleMed* is a mail-based system, the obvious data object is a list of the messages themselves – *Patient_List*. Each entry in this data object corresponds to an individual mail message and holds the necessary information of that message; the subject header (the patient concerned), the sender and the date sent. This data object is created upon start-up of the application and filled with data when the user's mailbox is accessed.

The second data object relates to the information of the encounter itself – *Encounter_List* and is contained in each *Patient_List* item. This data object is a collection of the items in a patient encounter, the inline attachments to the mail message, and contains the filename of the encounter item and its data type.

After a patient encounter or a response to one is created, the user selects the encounter or response to be sent to the intended recipients. As each patient encounter is mailed, it is placed in a FIFO (first-in, first-out) queue, *Send_Queue*. This queue is traversed from top to bottom, mailing the encounter in the topmost slot (see **Figure 4–4**).

A listing of the classes in the *MailTeleMed* system can be found in **Appendix B**. The division of the *MailTeleMed* application into classes is primarily a result of the development environment used – *Powersoft's Power++*. In this environment, each window (or form) is represented by a class and is a perfectly reasonable way of doing things. Using this scheme, six classes are produced. Two additional classes were

created to encapsulate functionality/information that is required frequently – the conversion to and from MIME64 and for storing information related to the user (mail ID, mailbox password). The interactions between classes follows the state diagram depicted in **Figure 4-5**.

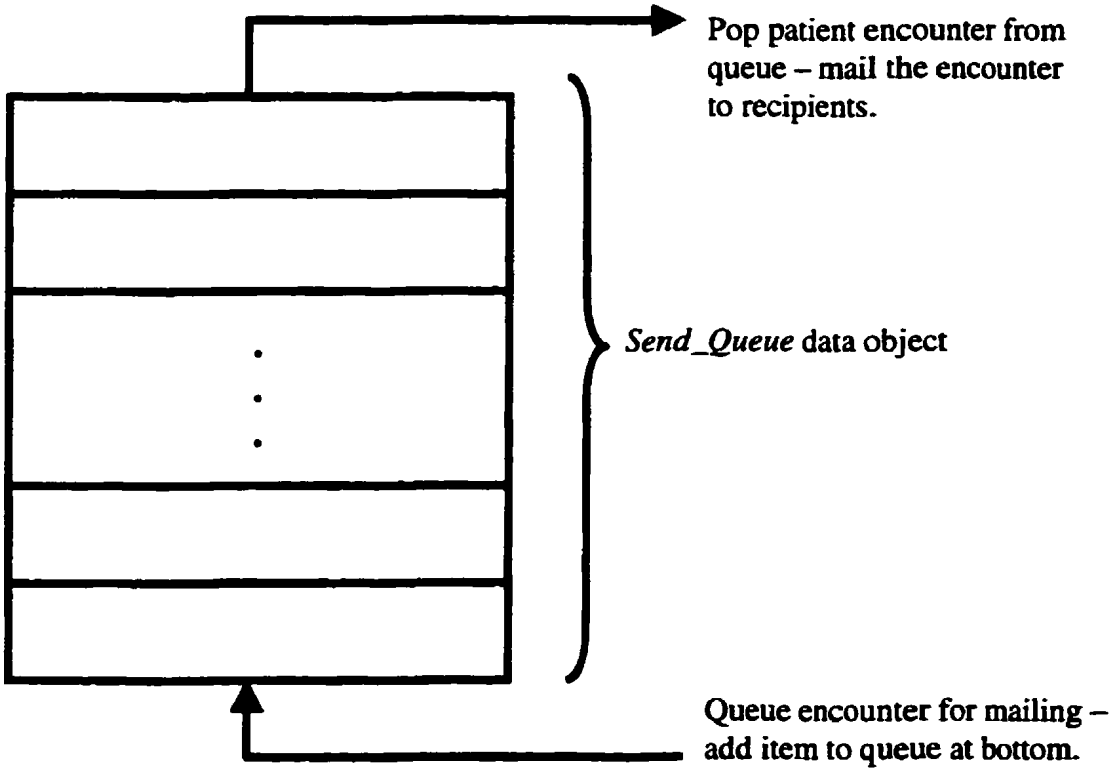

Figure 4–4. Queuing of Patient Encounters

4.2.5 Data Object Interactions

In keeping with the user model of the teleconsultation process using *MailTeleMed*, the following events occur during the process (see **Figure 4-5** for the state diagram):

- At login, the user's mailbox is accessed, using the provided login and password to retrieve the patient encounters and the subject and sender of each encounter. The encounters are then displayed in the patient encounter listing.
- As an encounter is selected, it is retrieved and parsed to obtain the encounter items. The encounter is then displayed in the encounter browser.
- As each item in the encounter is selected, it is played back or displayed.
- Creation of a new patient encounter results in the display of an empty encounter browser (see **Figure 4-6**).
- Insertion of a new item starts capture or allows the entry of the item.
- Mailing a composed encounter appends the encounter to the send queue. The queue is traversed from top to bottom, mailing the current encounter.

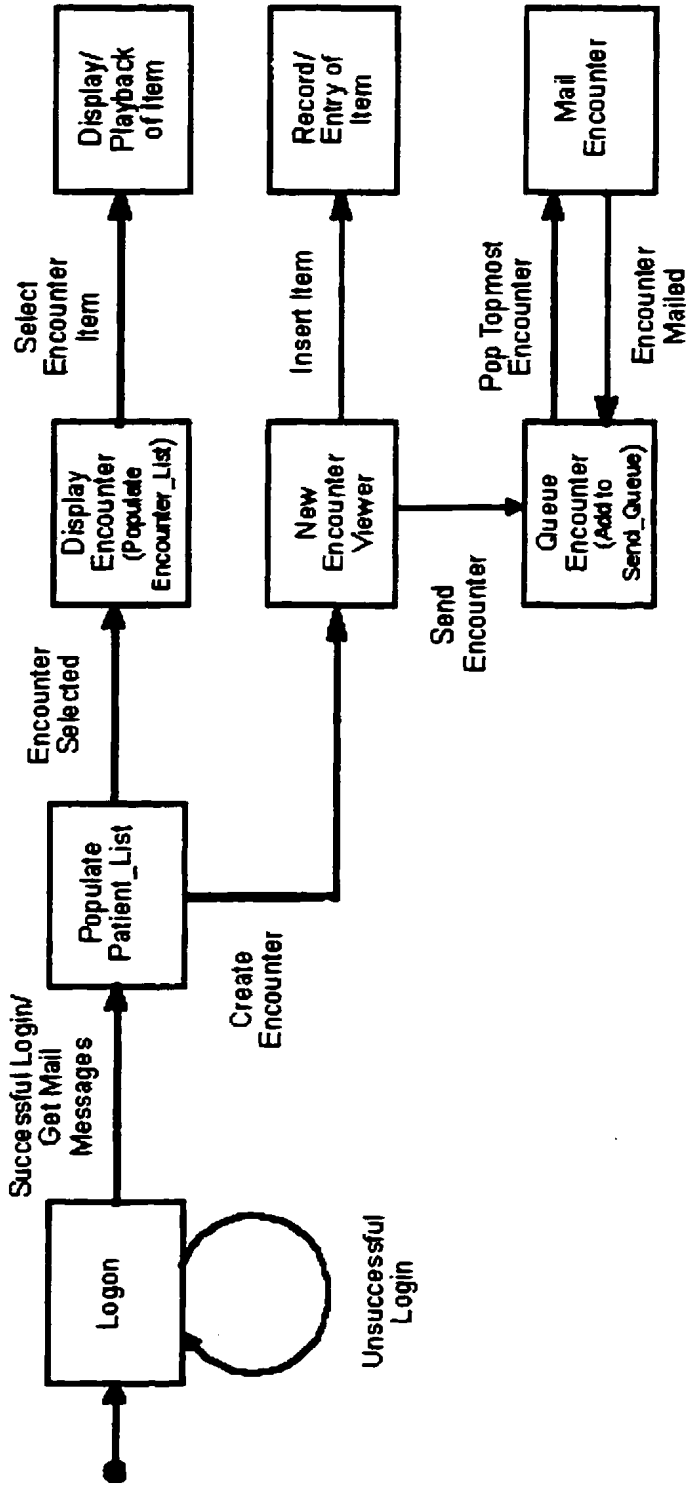

Figure 4–5. State Diagram of MailTeleMed System

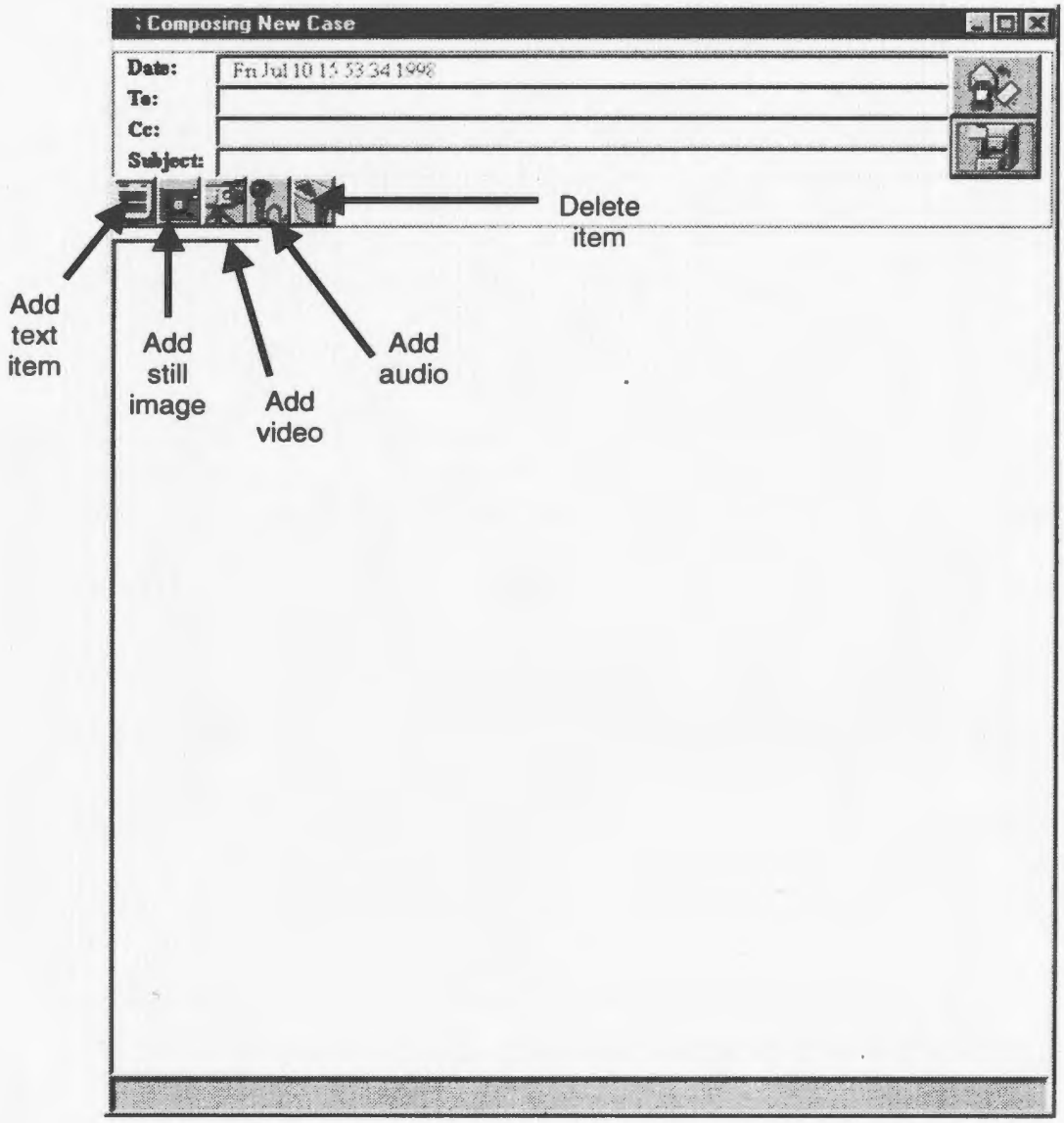

Figure 4–6. Composition of New Encounter

4.3 *WebTeleMed* – Client-Server Model

4.3.1 Overview

The second system developed was based on the client-server model of which the World Wide Web is an example. The user model of the consultation process in *WebTeleMed* is similar to that of *MailTeleMed* in that consultation can be asynchronous, yet is different in that synchronous communication is also possible. In this system, all patient data are stored at one or a few central sites and are accessible using a Web browser. Such a system was developed by implementing the patient database as a mailing list-server where patient encounters are mailed to the server and converted into web pages. While accomplishing many of the same tasks as the *MailTeleMed* system, the *WebTeleMed* system has the advantage of providing richer collaboration between a number of collaborators. This implementation allowed for viewing of the data only and did not provide for the creation or editing of patient encounters. Patient encounters would need to be created using some other means and put onto the central server, e.g. using the mail application provided with many Web browsers.

4.3.2 Functional Specification

The functionality provided by the *WebTeleMed* system is as follows:

- Conversion of patient encounters in mailbox to Web pages i.e. parsing of encounter in mail format for encounter items to be saved as separate items – hyperlinks in patient encounter lead to encounter item.
- Display/playback of encounter items.

As with the *MailTeleMed* system, each patient encounter is a separate entity with data items recorded during the encounter. No provision for the creation of new encounters has been included in *WebTeleMed* itself but can still be achieved using the e-mail composition component of the Web browser, for instance. There are both advantages and disadvantages this solution. The advantage is that standard tools are used in this process – the mail composition component. The disadvantages are that user is forced to switch both programs and modes to accomplish this. For example, the user must capture pictures of an examination and record his/her notes, save both of these to disk, insert them into the e-mail, mail the new encounter to the central site, and then wait for the encounter to be received and processed (converted into a web page). This is not the end as the user must reload the page in order for the new encounter to appear on the index.

4.3.3 Input and Output Interfaces

Since the *WebTeleMed* system is a Web-based system, the user interface is primarily that employed by the Internet browser used to view the patient encounters. As with the *MailTeleMed* system, the input interface is largely point-and-click using the mouse and alternative methods of input, for the creation of patient encounters, involve keyboard entry, video capture and audio recording. Two forms of output are produced by the system – system messages and the display of encounter items. The latter is achieved utilizing the Web browser's "plug-ins" whereas the former is primarily output from the Web browser and is handled as such by the browser. The listing of patient encounters and their display uses an interface similar to that of *MailTeleMed*. Upon visiting the central site, an index of all patient encounters is provided, using hyperlinks for the encounters (**see Figure 4–7**). Selecting a patient encounter from the index displays the encounter with a format similar to that of *MailTeleMed*. The frame is split into two frames – the left frame displays icons for the encounter items and the right frame displays an item when it is selected (see **Figure 4–8**).

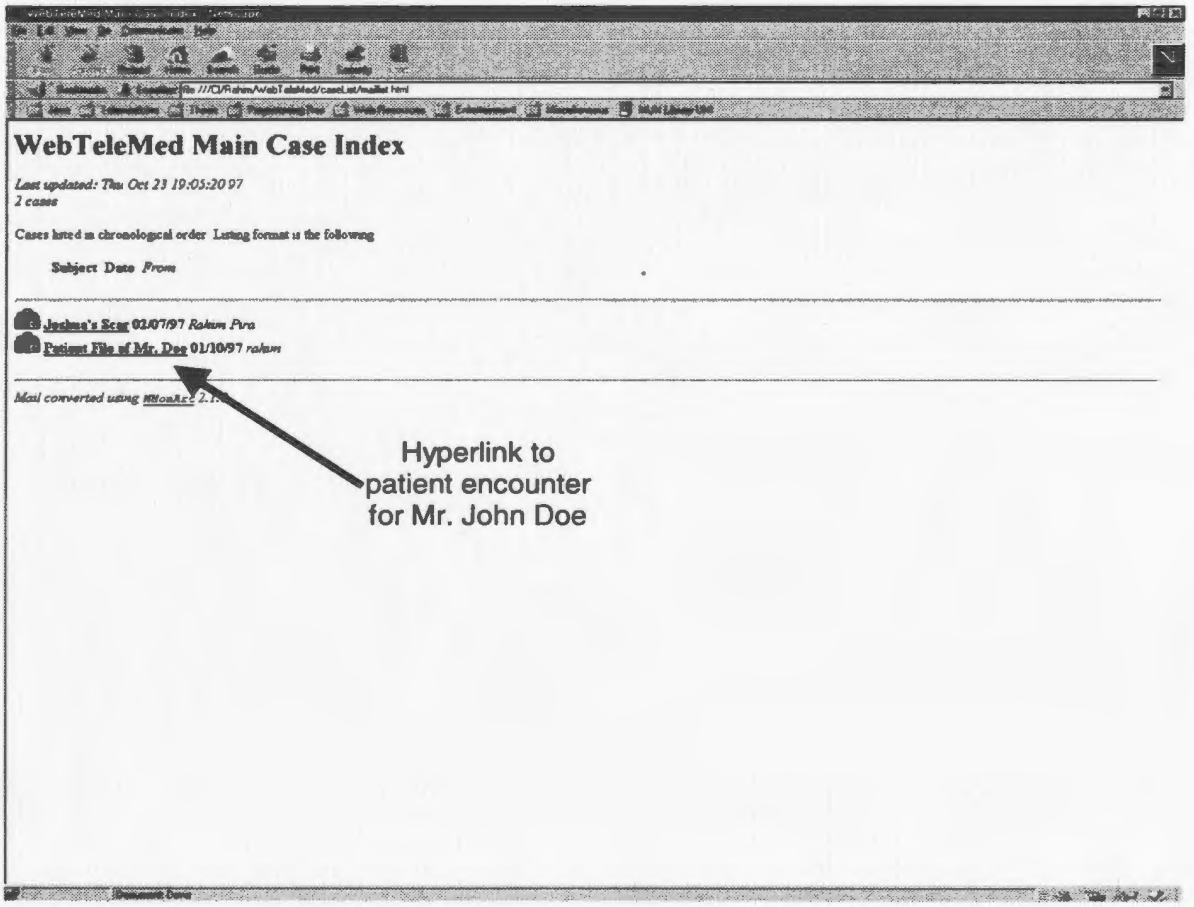

Figure 4–7. Patient Encounter Index

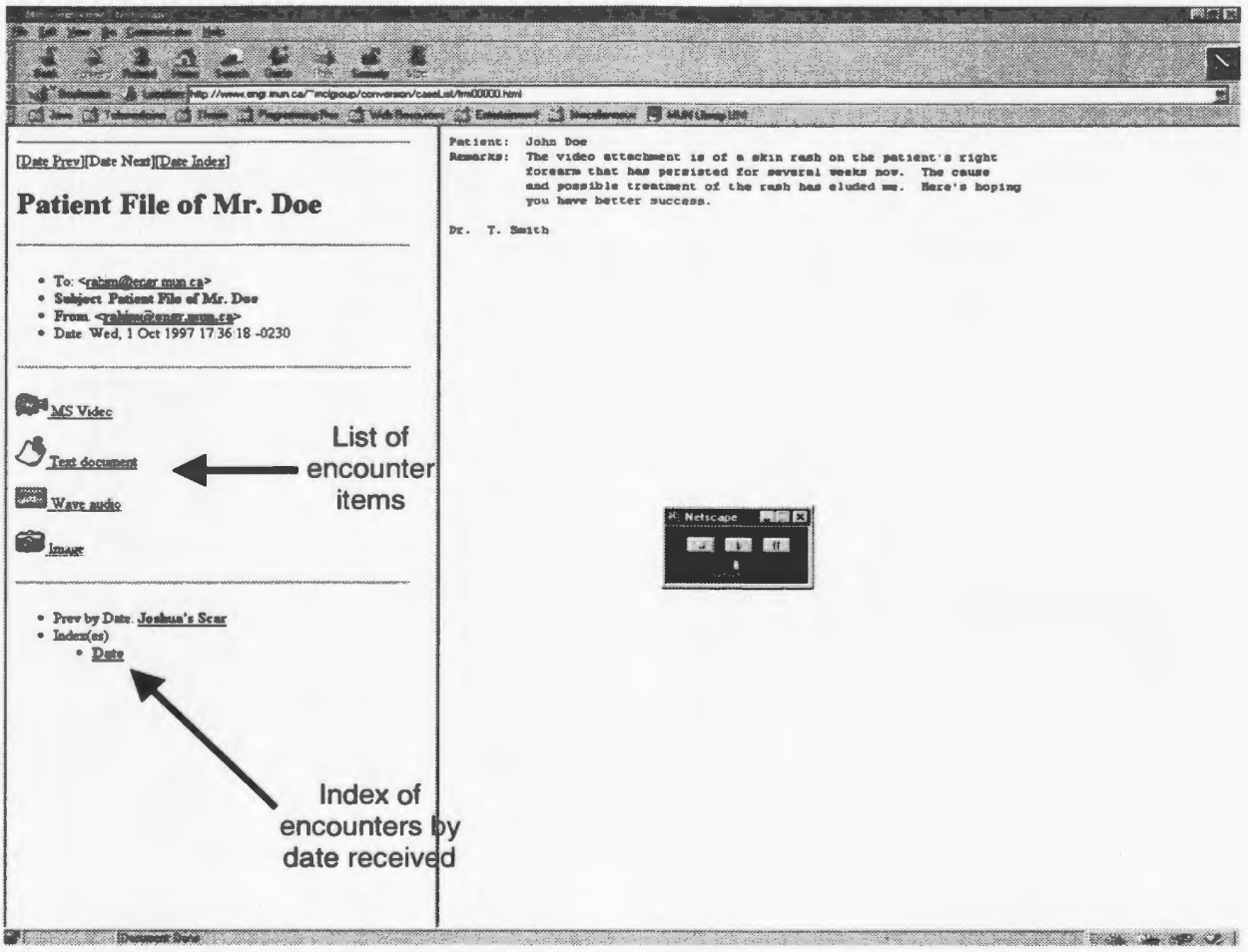

Figure 4–8. Display of Patient Encounter

4.3.4 Procedural Diagram

The *WebTeleMed* system operates by using a Perl script to convert the mail messages in the mailbox to Web pages. As new messages are received in the mailbox, the conversion script is invoked, reading the mailbox at the central site and converting the mail messages into Web pages suitable for the system (see **Figure 4–9**), parsing the attachments into individual items. Once this is completed, the patient encounter indices are updated. Refer to **Appendix C** for the script used in the conversion.

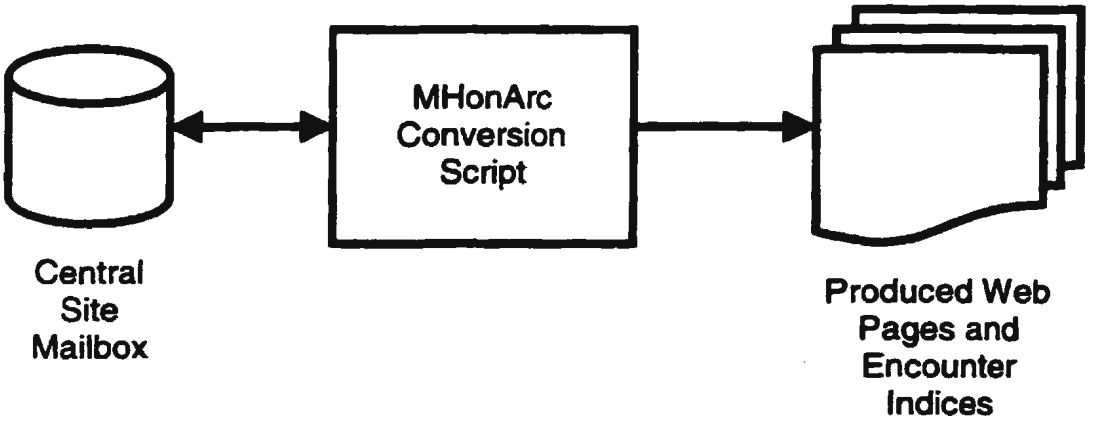

Figure 4–9. *WebTeleMed* – Conversion Process

4.4 *MCLTeleMed* – Replicated Database Model

4.4.1 Overview

The third system developed was based on the replicated database model. Using this model, a patient's medical data is replicated only between those sites that have that patient in common. At the central site, which houses the physicians responsible for all the remote clinics, medical data for all patients is replicated. The user model for this system is entirely different from the models for the *MailTeleMed* and *WebTeleMed* systems. In the previous two systems the user was dealing with separate items and had to send them to the consulting physician. In this model the user is working with a patient record much like the traditional paper-based patient record or chart. When a patient encounter occurs, data is added to the patient's record and placed at the top of the record. Previous encounters are also visible so that the patient's progress can be monitored. Updates to the patient's record are reconciled when the remote clinic connects to the central clinic (see **Section 4.4.2** for a detailed explanation of this process). This automatic patient record reconciliation is transparent to the users who are now able to work as if they are using traditional paper-based medical records. One area of improvement in *MCLTeleMed* is that since the synchronization is transparent to users, it may not always be easy to determine how synchronized the patient records are; i.e. the date when updates were last made to the records. Future work would include a process whereby a check is conducted to determine if the records need to be synchronized.

4.4.2 Patient Data Reconciliation

Since a patient database is used for this system, a method of reconciliation is required to determine the differences in the patient data at two sites and to perform the necessary updates. As low-data rate channels are being used in the application, the amount of data exchanged between two sites during this reconciliation process needs to be minimized as does the time required by this process. This is accomplished by keeping track of the creation time of each item in the patient record. For each patient, the most recent creation time of the items **not** created at that site (i.e. only record items received from other sites are considered) is exchanged. This information for all the patients at a site is packaged into one data packet to minimize bandwidth. This allows a site to determine what items of a patient record are required by the other site, eliminating the need to exchange entire patient records, reducing bandwidth and the time for the reconciliation. This patient record reconciliation is performed every time a remote station connects to the central clinic, which can be made manually by a user or can be set to occur automatically at some pre-set time of day every day.

4.4.3 Functional Specification

The functionality of the *MCLTeleMed* system differs little from the *MailTeleMed* and *WebTeleMed* systems. The functionality of this system includes:

- Viewing of patient record and encounters.
- Display/playback of patient encounter items.
- Creation of new patient records, encounters and encounter items.
- Reconciliation of patient records, both automatic and manual.

4.4.4 Input and Output Interfaces

The input and output interfaces for the *MCLTeleMed* system are identical to that of the *MailTeleMed* system, predominantly mouse-based input with keyboard, video and audio input for the insertion of encounter items, message dialogs for system output and text, audio and video output for the display of the encounter items. As with the *MailTeleMed* system, a patient record listing displays the records in the system, distinguishing between new or updated records and unchanged records (see **Figure 4–10**). Selecting a patient record displays the browser, separated into three sections (see **Figure 4–11**). At the top of the browser, patient information is displayed including some background information. On the left is the scrolling list displaying the date, time and location of the encounters with the selected patient and on. Also displayed are icons representing the data captured during the encounter. Selection of a

particular item will display it and commence playback, if appropriate, in the right "frame" of the browser (see **Figure 4–12**).

For the addition of new patients to the database, a button displays a blank form consisting of empty data fields (see **Figure 4–13**). The patient's relevant information is entered into these fields as is any additional background information. This results in an empty patient record browser being displayed (see **Figure 4–14**) in which the operator can gather data on the patient, using the buttons on the browser, inserting the items into the browser.

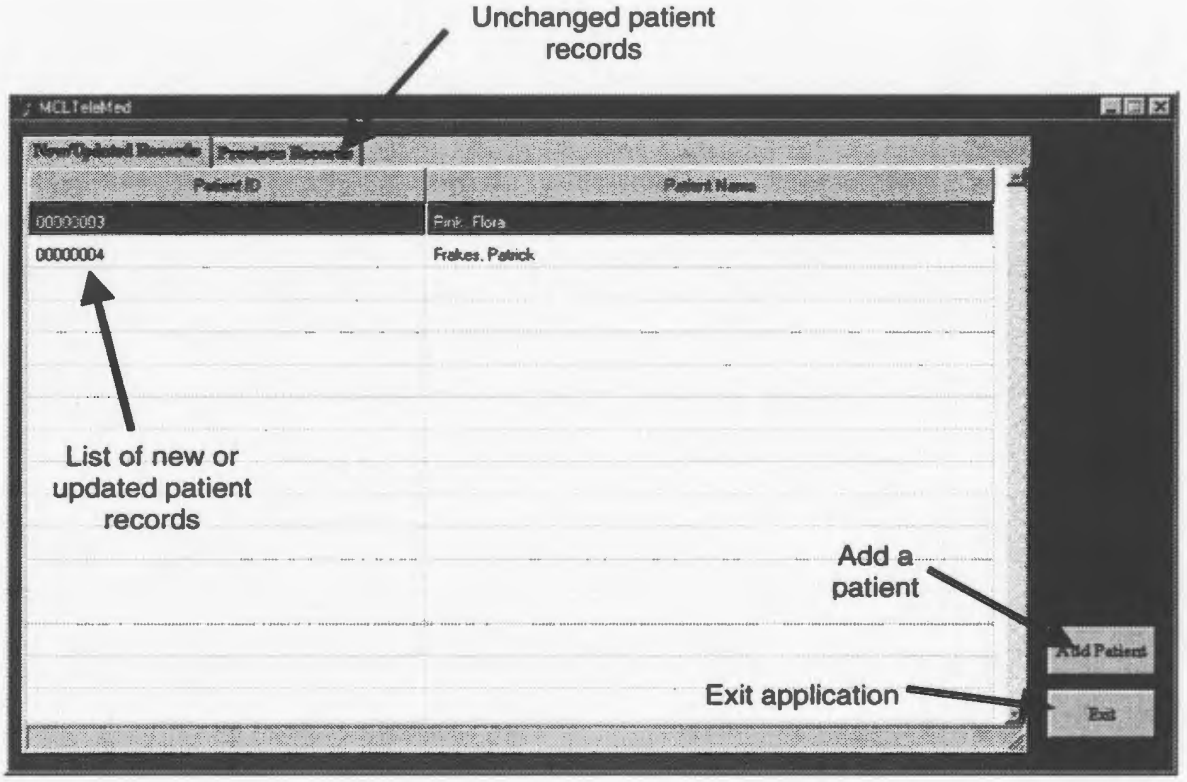

Figure 4–10. *MCLTeleMed* – Patient Record Listing

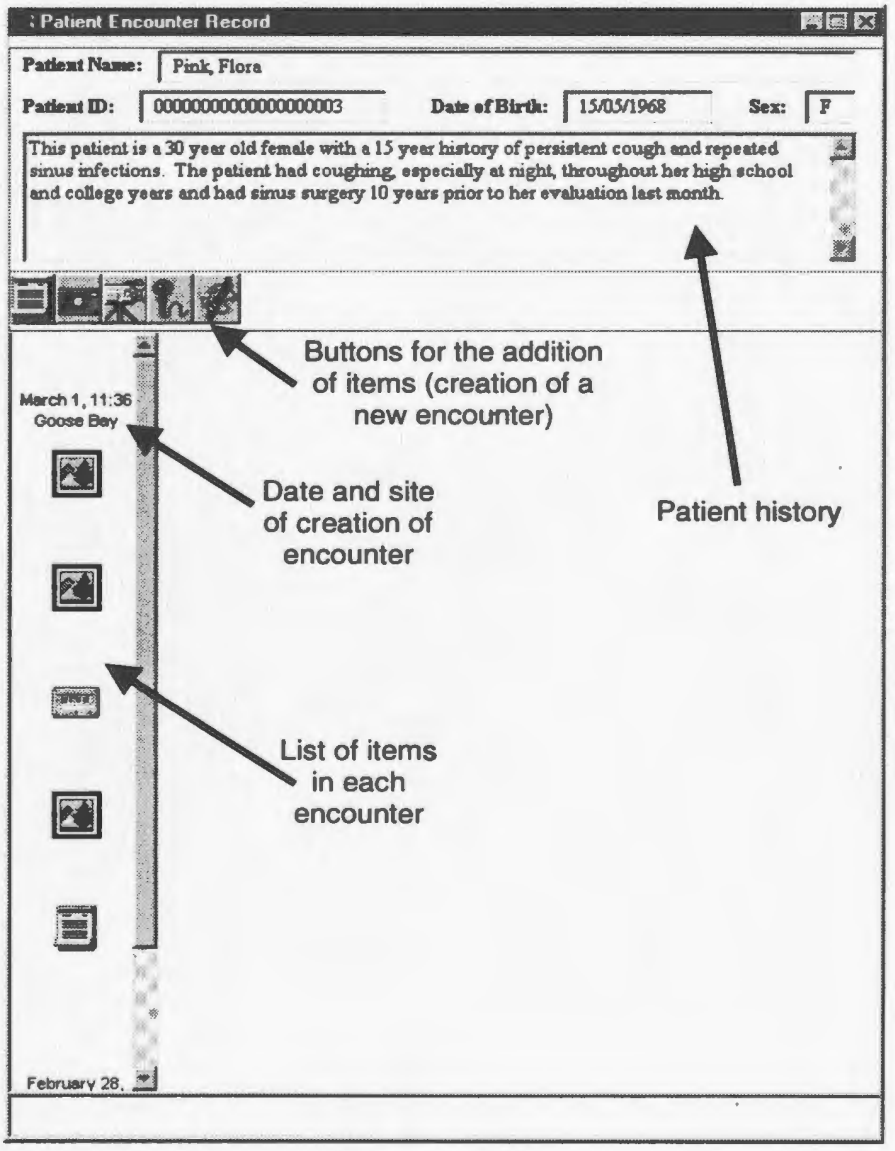

Figure 4–11. Patient Encounter Display

Figure 4–12. Browsing of Patient Encounter

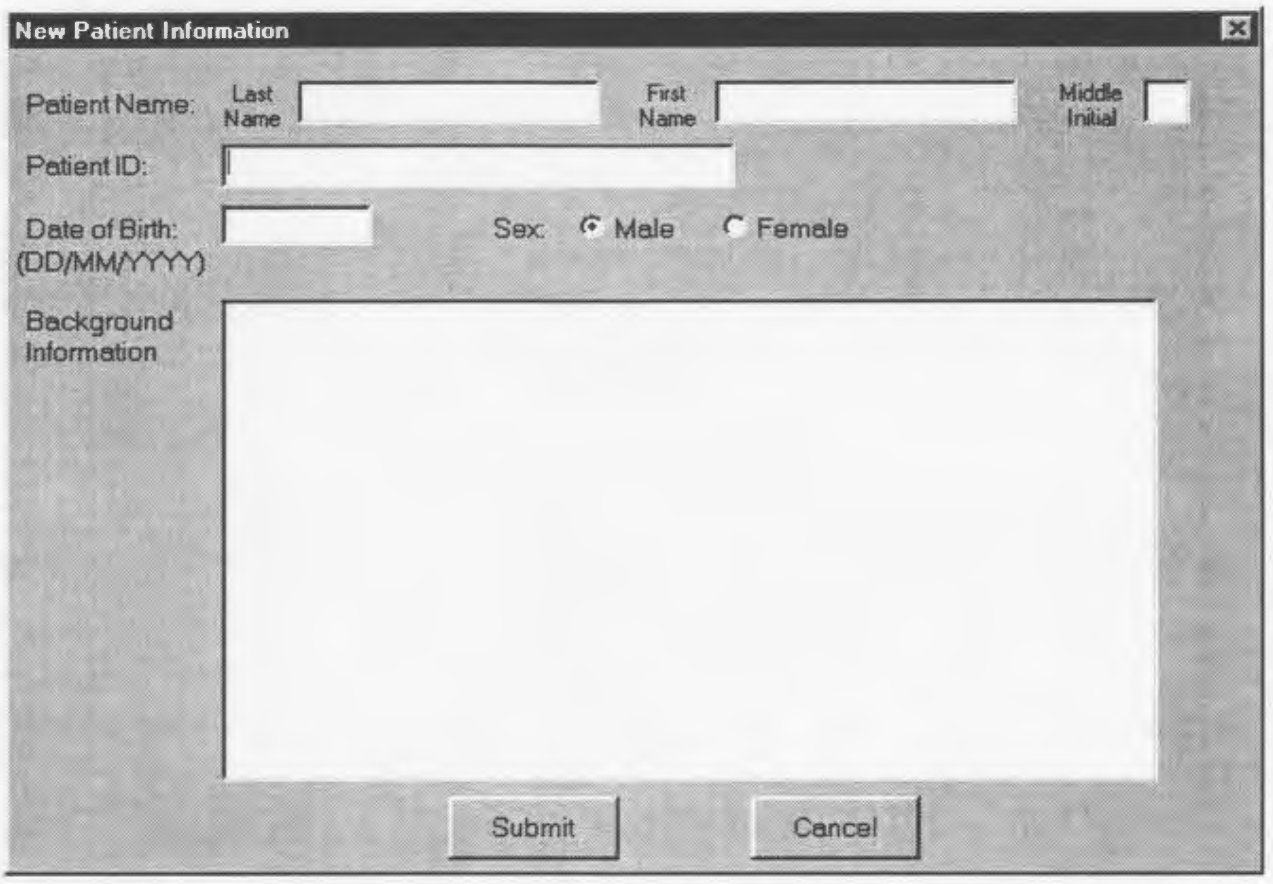

Figure 4–13. Addition of New Patient

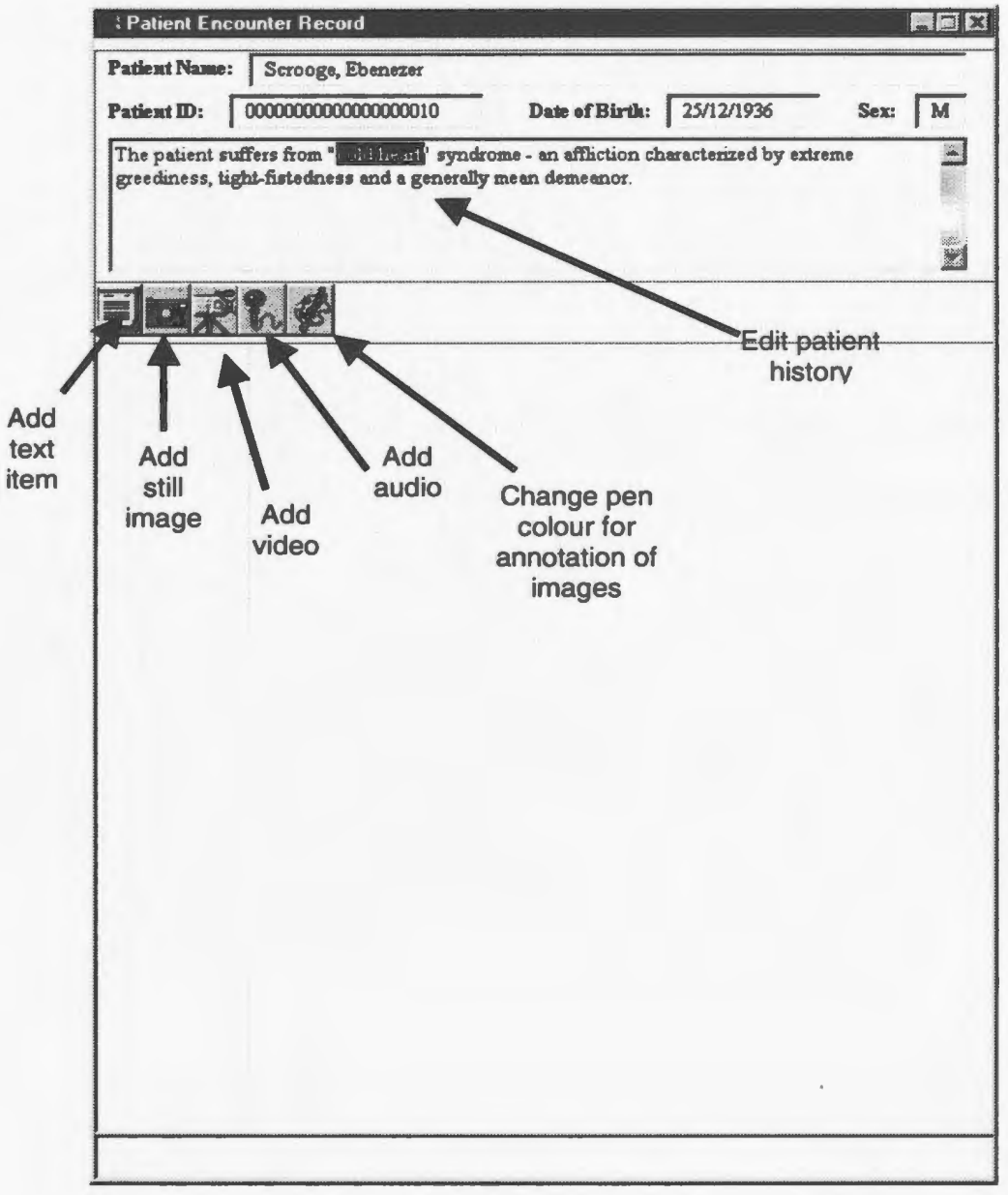

Figure 4–14. New Patient Record Browser

4.4.5 High-Level Data Objects

The primary data object in the *MCLTeleMed* system is the *Station_List* structure. This data object is filled upon start-up and contains information on all the patients on the station's database. The *Station_List* object is comprised of *Patient_Item* objects, which is in turn comprised of *Encounter_Item* objects, containing *Component_Item* objects (see **Figure 4–15**). The *Patient_Item* object contains some information on the patient, the list of encounters for the patient and the creation time of the most recently received encounter item for the patient – used in the patient record reconciliation process. This list of encounters is a collection of *Encounter_Item* objects, each one containing information about the encounter (place, date and time) and the data items created during the encounter. It is important to note that since the location of the encounter is used as information, items created at the same date and time but at two separate sites for the same patient will result in two encounters – one for each site. The collection of data items is simply a collection of *Component_Item* objects, each one corresponding to an individual encounter data item and contains information about the item such as its creation time, its filename and the data type (text, image, audio, video).

Refer to **Appendix D** for a listing of the classes in the *MCLTeleMed* system. As with the *MailTeleMed* system, the classes in *MCLTeleMed* are a result of the development environment used. Of the eleven classes that exist, nine are for the various windows that exist. The remaining two classes are used to hold information on the user and on

the patients in the site's medical database. The relationship between the classes follows the state diagrams depicted in **Figures 4-16** and **4-17.**

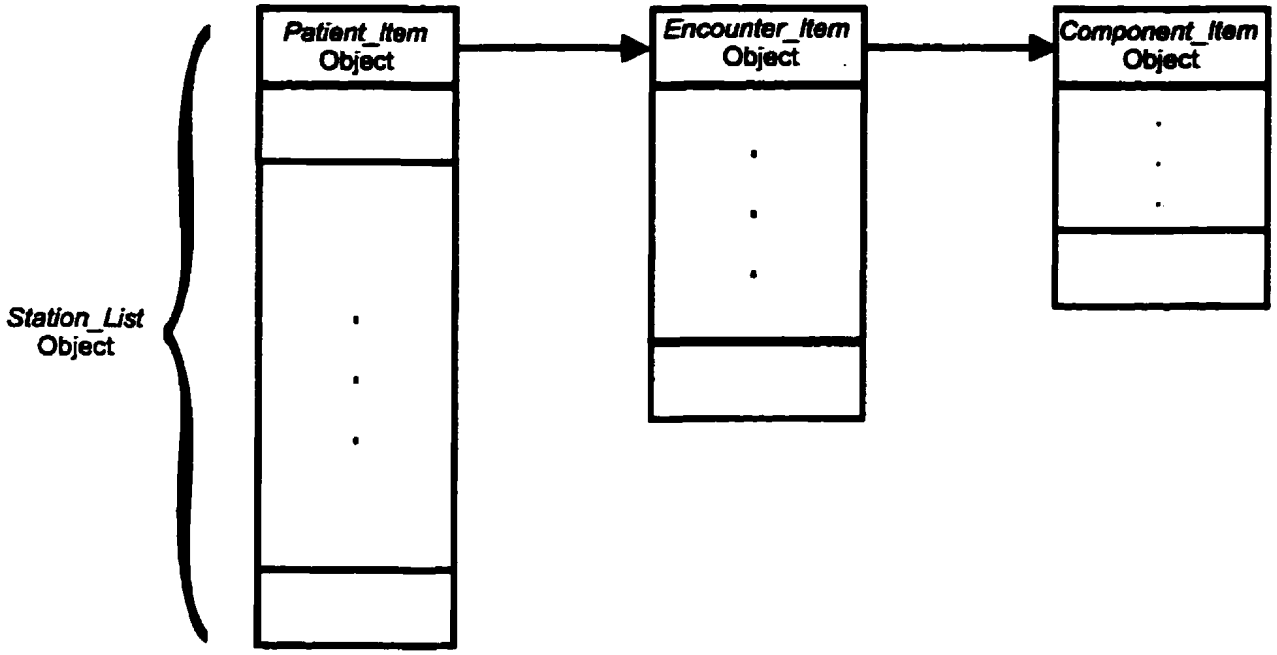

Figure 4–15. *MCLTeleMed* Data Objects

4.4.6 Data Object Interactions

The interactions that occur between the data objects can occur at numerous points throughout the application but have in common the addition/modification of a patient's medical record (see **Figure 4–16** and **4–17** for state diagrams). The instances when this can occur are at start-up (when the site's database is loaded), the addition of a new patient to the database, the addition of encounter items to a patient record, and during the reconciliation process. All operations involving these data objects involve the addition of a new item to the object concerned, never a deletion, except at the termination of the application for the deallocation of the memory used to store the objects.

From the figures, we see that the central and remote sites differ only in the initialization of the synchronization process. The central site acts as the host and thus sets up the session; the remote site is the client and attempts to connect to the host (the central site). One difference between the two that is not noticeable from the figures concerns the addition of new patients during the synchronization process. As the patient database at the central site is deemed to be the "master" list, any new patients at a remote site are added to the central site during synchronization. The converse, however, is not true. Since the central site holds the records for all the remote sites, many of the records may be considered "new" to the current remote site. As a result, information is transmitted at the beginning of the synchronization to notify the other site of the patients held by this site enabling the central site to send

data only for those patients held by the current remote site (see **Figure 4–18**). As indicated in **Figure 4–16**, the user is unable to quit the application during the synchronization process in order to preserve the integrity of the medical database. If the user were allowed to quit during the record reconciliation process, then not all the new record items would be inserted into the patient records during the session. When the user begins a new session by connecting to the central site, the timestamp of the most recent item in the record would already have been updated from the previous session and the record items that were not received during the last session will never be inserted into the patient records (without manual intervention on the user or system administrator's part).

4.4.7 Call Setup Procedure

Having explained the record synchronization process and the exchange of messages between the two sites, we now turn our attention to the procedure whereby two sites connect, initiating the synchronization procedure. Since the relationship between the central and remote sites is very much like that of host and client, the host (central) site must first begin a session before the client (remote) can connect. Beginning a session at the host site is a simple task of clicking the "*Host Session*" button on the toolbar and then selecting the communication method (**Figure 4–19**). Depending upon the configuration of the host computer, the options may differ. If an internet card is not installed on the computer, then only two options will be available – modem and serial communication – and TCP/IP and IPX will be unavailable. After selecting the

communication method, the "*Host*" button is pressed to begin the session. It is assumed that the host will not terminate a session, so this needs to be done only on rare occasions.

The procedure for the client site to connect to a session is very similar (**Figure 4–20**). The "*Connect Session*" button is pressed, followed by the selection of the communication method. After this is completed, the "*Connect*" button will connect the client site to the host. It is important to mention that a successful connection will only occur if:

- the host site has begun a session
- no client site is currently connected to the host site (only one client site is allowed per session)
- both host and client site are communicating using the same protocol

If any of the above conditions is not met, the connection attempt will fail. If the attempt is successful, the synchronization process will then commence.

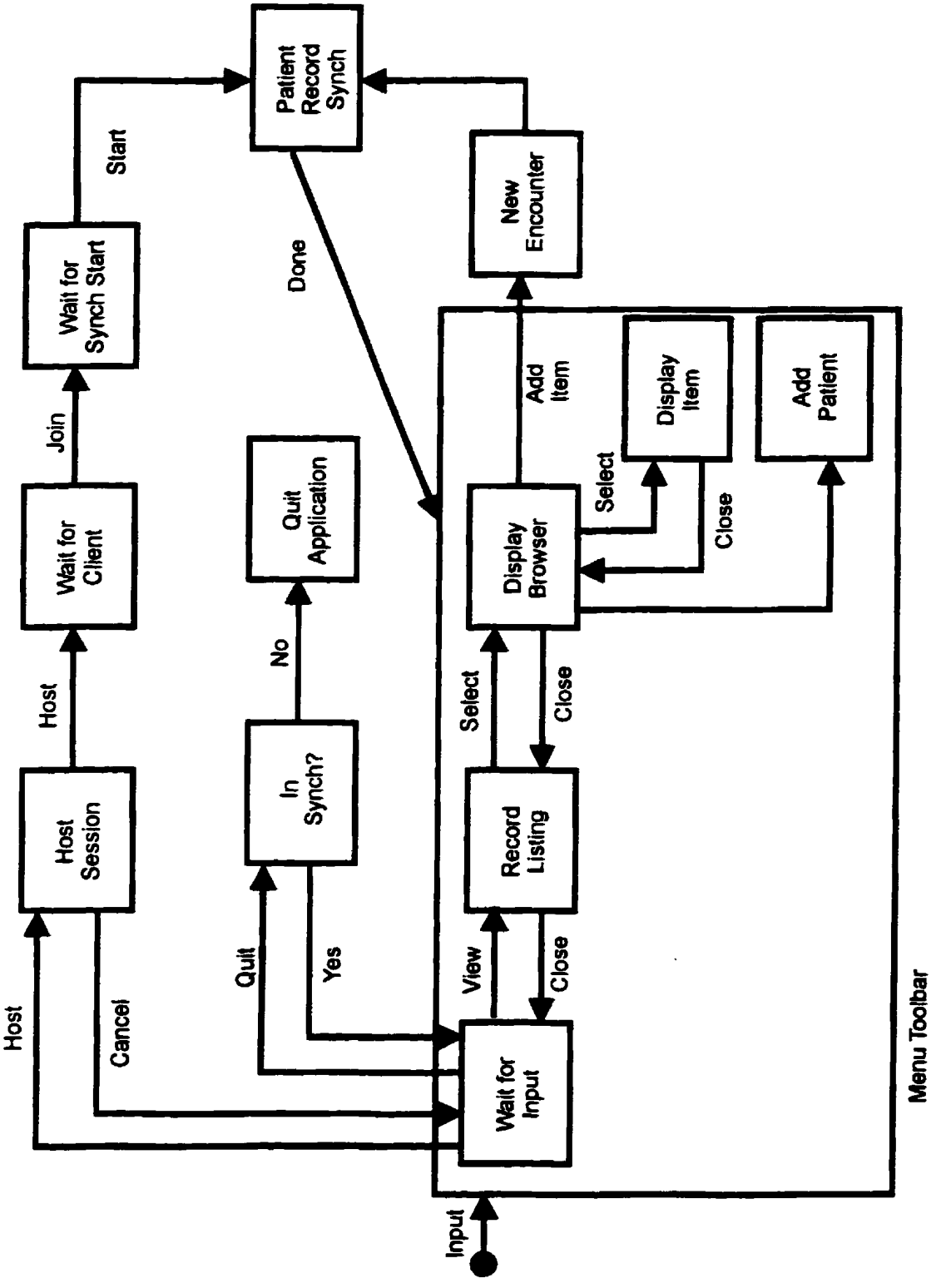

Figure 4–16. State Diagram for Central Site

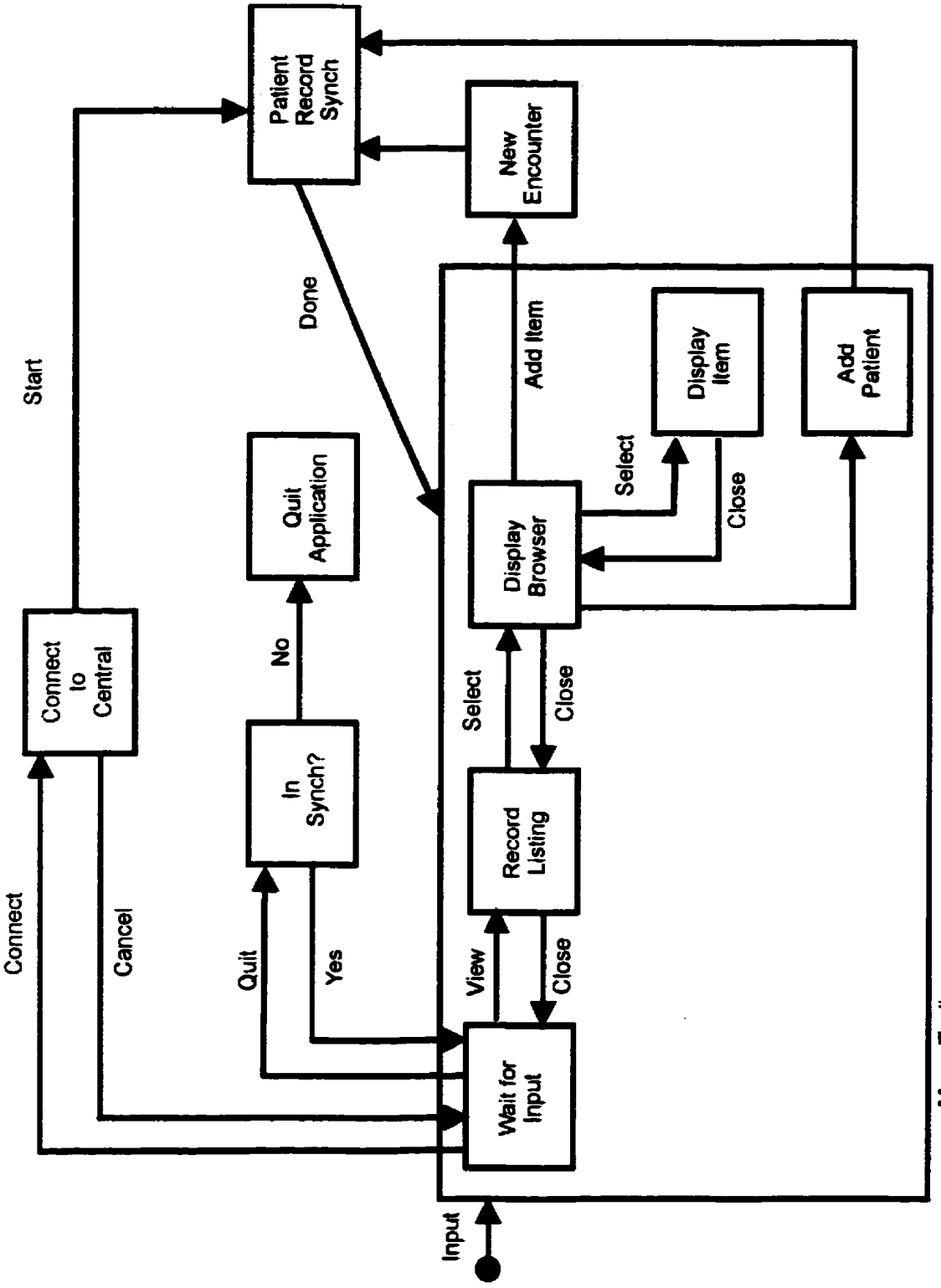

Figure 4–17. State Diagram for Remote Site

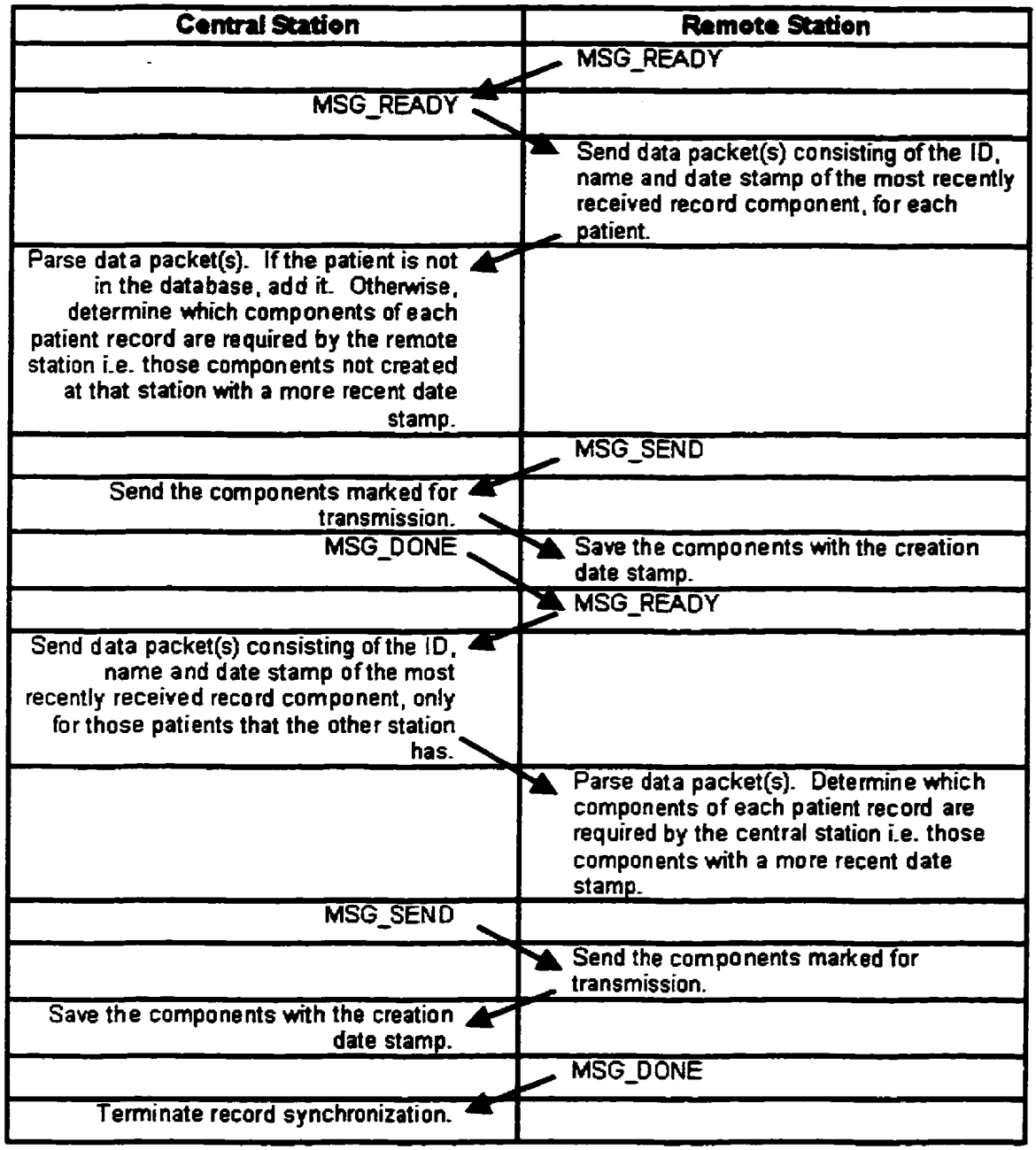

Figure 4–18. Message Exchange During Synchronization

4.4.8 Patient Record Structure

As previously mentioned, the structure of the patient records was designed to be simple and easily convertible to any standards that would be developed in the future. The patient records are organized in a directory structure with (see **Figure 4–21**) the root directory being the "*MCLTeleMed Records*" directory. From here, a directory is created for the station wherein all the patient records reside. In the station directory, is a file in text format, listing all the patients held at that station. Each patient record is a directory created in the station directory using the patient's ID number as the name of the directory. This patient directory is then divided into directories, one directory for each encounter in the patient's record. These encounter directories contain the actual data items collected during the encounter. In the patient directory, there is one additional file used to store the background data for the patient.

4.4.9 Design Evolution

As previously mentioned, input as to the design of the systems was obtained from potential users during the course of development, primarily from Dr. Rod Elford and Andrea Battcock. It was this user input that led to the design of the replicated database system, *MCLTeleMed*. Using *MailTeleMed* as a base, we were informed that users would like a system that resembled the paper medical record, with the patient's medical history broken down by date and place. This lead to the amalgamation of encounters into a single patient record.

Input from the user group was also obtained a number of times during the course of the development of each of the three applications for feedback on the interface, functionality provided and general comments to usage. Input was also obtained at the *Remote Medicine Conference* held in St. John's, Newfoundland in May 1998. The *MCLTeleMed* application was demonstrated at this conference and we obtained valuable input as to the interface and operation, as well as suggestions for desired features. For instance, we learnt that users would like the ability to view multiple images simultaneously, allowing clinicians the ability to track progress and make comparisons easier.

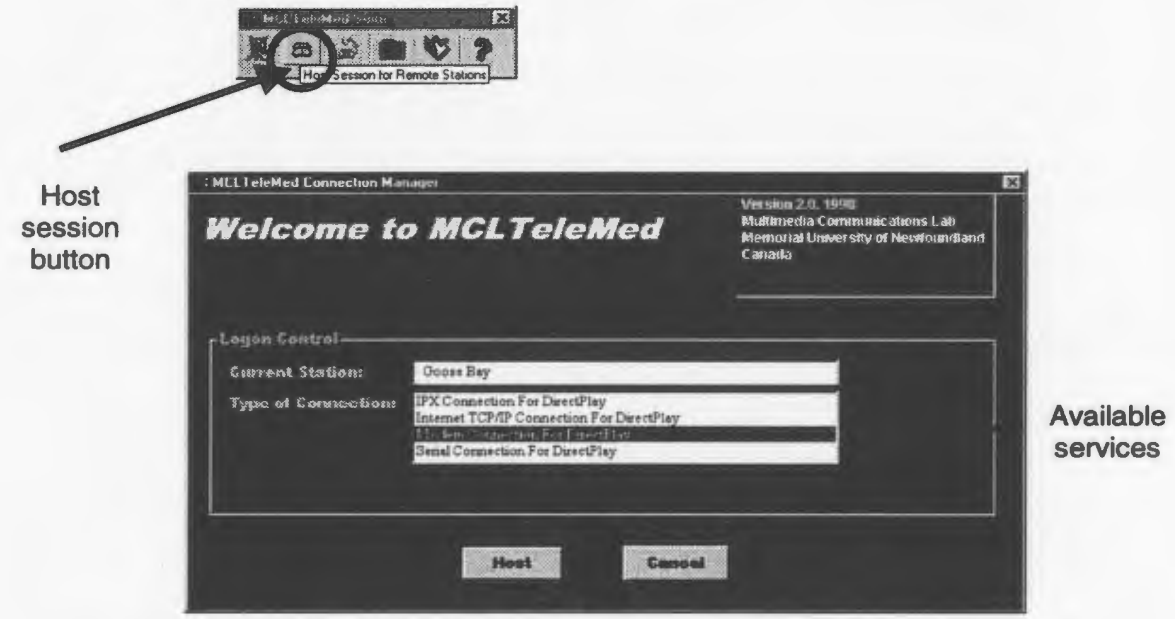

Figure 4–19. Call Setup at Host Site

Figure 4–20. Call Setup at Remote Site

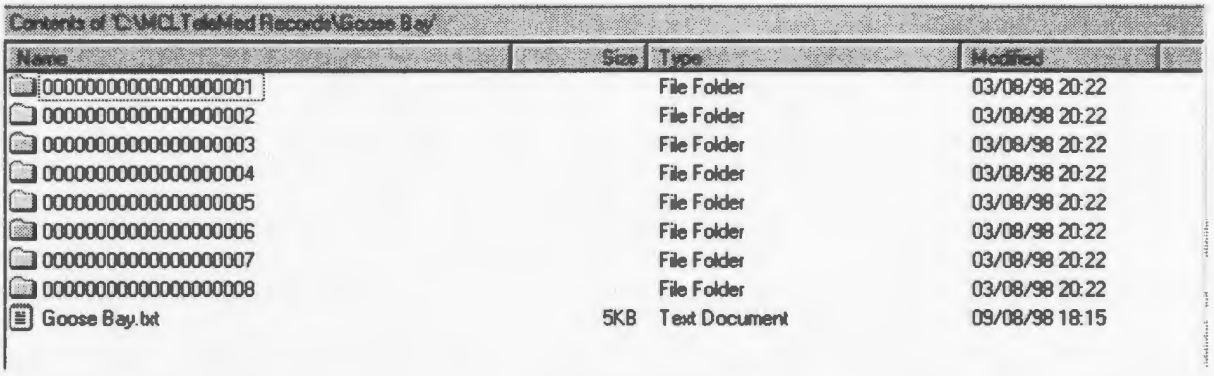

Figure 4–21 (a). Directory Structure at Station Level

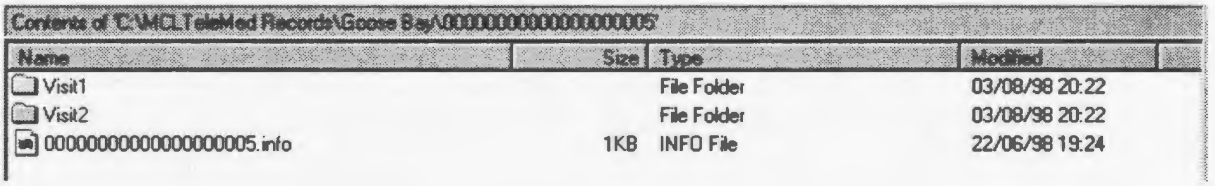

Figure 4–21 (b). Directory Structure at Patient Level

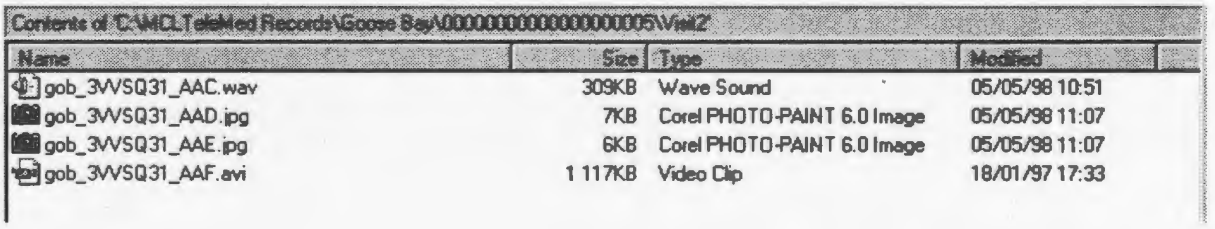

Figure 4–21 (c). Directory Structure at Encounter Level

4.5 Media Acquisition

As the media acquisition process is the same for the *MailTeleMed* and *MCLTeleMed* applications, it will be discussed here in one section and will be divided into five sections: text, audio, still images, moving images and annotated images. The process of acquisition for the *WebTeleMed* application is not definite as it is not incorporated into the application and the user makes use of third-party applications for the acquisition of the media (e.g. Windows Sound Recorder for audio, Corel Photo-Paint™ for still image capture, MediaStudio™ for video capture). Consequently, the exact process for the acquisition will differ depending upon the specific application used. In general, the process is the capture of the item using an application, saving it to disk and then inserting it into the patient encounter.

The discussion below is for the *MailTeleMed* and *MCLTeleMed* applications and the term "patient browser" is used to refer to the patient encounter browser window in *MailTeleMed* and the patient record browser window in *MCLTeleMed*.

4.5.1 Text Media

In the browser window for a patient browser, a toolbar is present below the fields for patient data (see **Figure 4–22**). The left-most button on this toolbar is for the addition of textual data into the encounter (see **Figure 4–23**). Pressing this button displays a text window where text can be entered (see **Figure 4–24**). Pressing the "Close"

button on this window will close the text window, saving the data and inserting it into the encounter (see **Figure 4–25**).

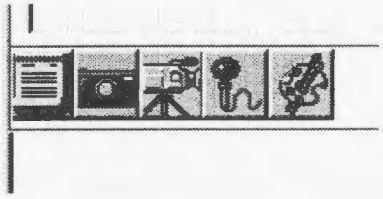

Figure 4–22. Media Acquisition Toolbar

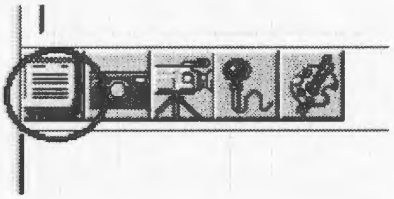

Figure 4-23. Insert Text Media

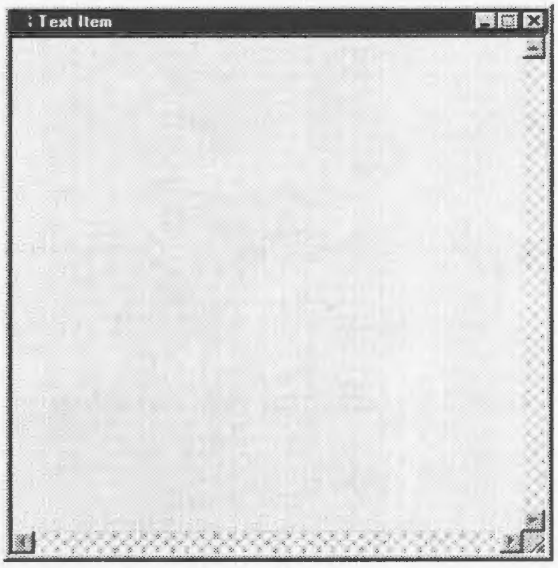

Figure 4–24. Text Notepad

Figure 4–25. Insertion of Text Item

4.5.2 Still Image Media

The second button is for inserting still images into the encounter (see **Figure 4–26**). Pressing this button displays two control buttons for inserting still images (see **Figure 4–27**). For inserting still images, two options are available: the image can either be loaded from disk or captured from a camera device. To load an image from the disk, the user simply presses the "Load" button and is then prompted for the image file to insert into the encounter (see **Figure 4–28**). Capturing an image from a video device is just as simple. The "Preview" button causes a video window to be displayed showing the input from the video device. This allows the user to view the input and when the desired image appears, capture it. The button text changes to "Capture" so that when pressed a second time, an image will be captured and also displays two additional buttons: "Source" and "Format" (see **Figure 4–29**). The "Source" button displays options available on the video device, if supported, such as brightness and contrast control (see **Figure 4–30**). The "Format" button, if supported by the video device, allows the user to change the size of the image captured, increasing the resolution (see **Figure 4–31**). In both cases, the image is inserted into the encounter (see **Figure 4–32**) and displayed in an image viewer.

4.5.3 Moving Image Media

The third button is for the insertion of video items into the encounter (see **Figure 4–33**). Pressing this button displays a preview window with controls for the video capture (see **Figure 4–34**). The "Source" and "Format" buttons function the same as they do with inserting still images. The "Preview" button, like with the image insertion, displays the input of the video device on the preview window. To begin recording the video, the user simply presses the "Record" button to commence recording, and the same button to stop (the button text changes to "Stop" once capturing commences). The captured video is then inserted into the encounter (see **Figure 4–35**). It is important to note that the user need not have to press the "Preview" button before capturing a video sequence; pressing "Record" at any time will start the capture, it's just that the user will not be able to preview the scene before capturing it.

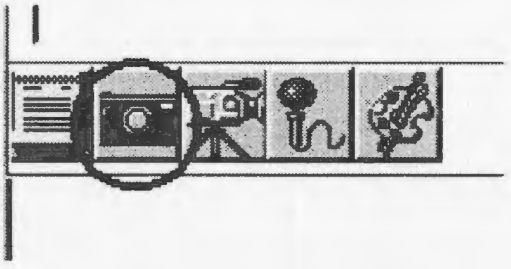

Figure 4–26. Insert Still Image Media

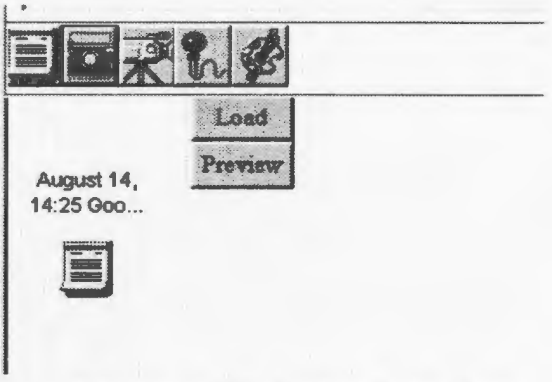

Figure 4–27. Image Insertion Controls

4.5.4 Audio Media

The fourth button is to insert audio data into the encounter (see **Figure 4–36**). Pressing this button displays the "Record" button and a microphone icon (see **Figure 4–37**). Pressing the "Record" button starts the audio capture, changing the button text to "Stop". During recording, the microphone icon changes to provide the user with feedback indicating that recording is taking place. Once the "Stop" button is pressed, recording stops and the item is inserted into the encounter (see **Figure 4–38**).

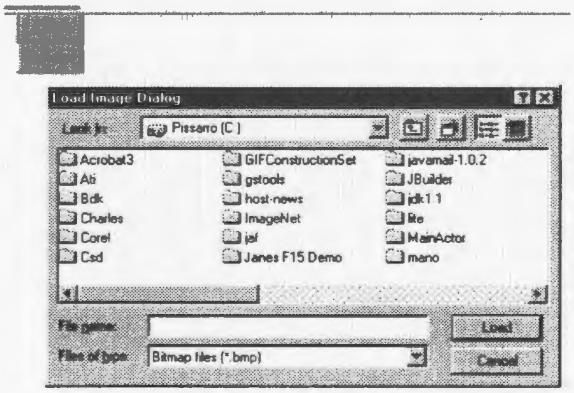

Figure 4–28. Image Load Dialog

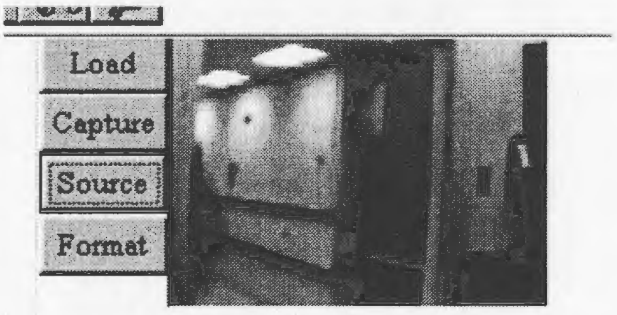

Figure 4–29. Still Image Preview Window and Controls

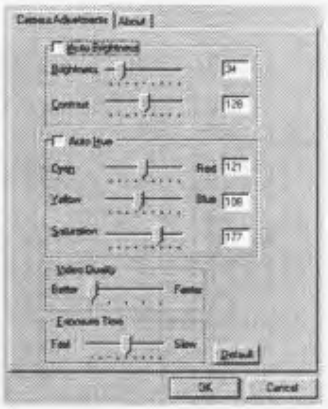

Figure 4–30. Video Source Options

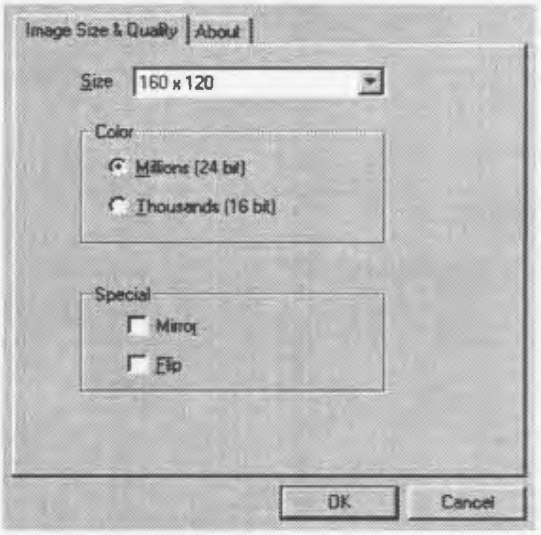

Figure 4–31. Video Format Options

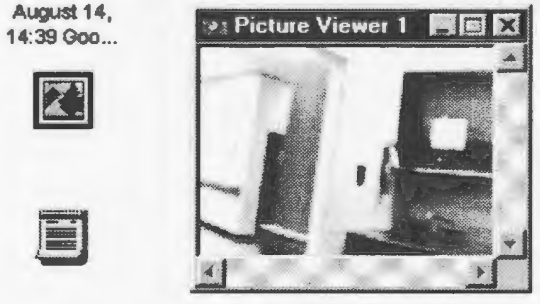

Figure 4–32. Insertion of Still Image and Image Viewer

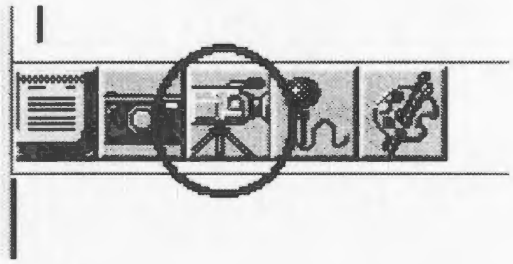

Figure 4–33. Insert Video Media

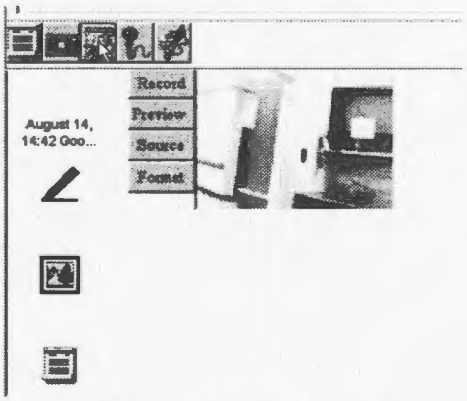

Figure 4–34. Video Preview Window with Video Controls

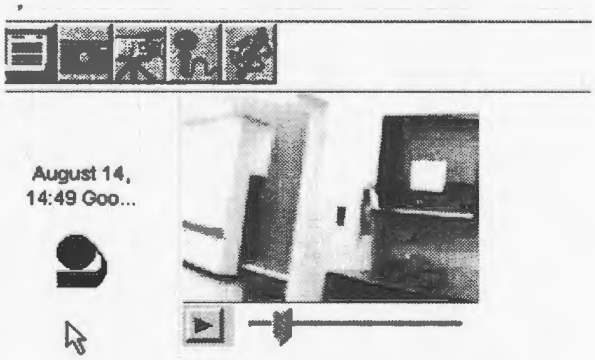

Figure 4–35. Insertion of Video Item

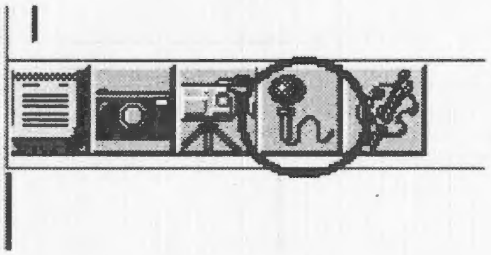

Figure 4–36. Insert Audio Media

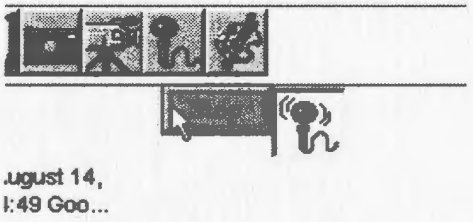

Figure 4–37. Recording of Audio Item

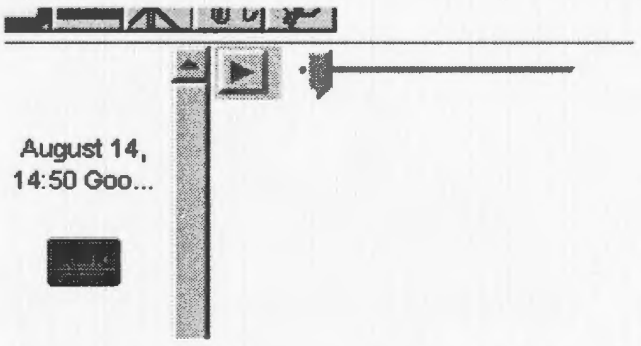

Figure 4–38. Insertion of Audio Item

4.5.5 Annotated Image Media

The last media item that can be inserted into patient encounters is the annotated image. When viewing a still image, the cursor for the window changes to a pen indicating the ability to annotate the image (see **Figure 4–39**). The last button on the patient browser toolbar allows the user to change the pen's colour (see **Figure 4–40**), by default the pen's colour is cyan. Pressing the left mouse button and moving the mouse over the image overlays graphics onto the image. Releasing the left button stops the annotation. When the image viewer window is closed, the annotated image is inserted into the encounter (see **Figure 4–41**). As the original image may be required later to view, it is left unchanged. Instead a copy of the image is made with the graphics overlaid on it.

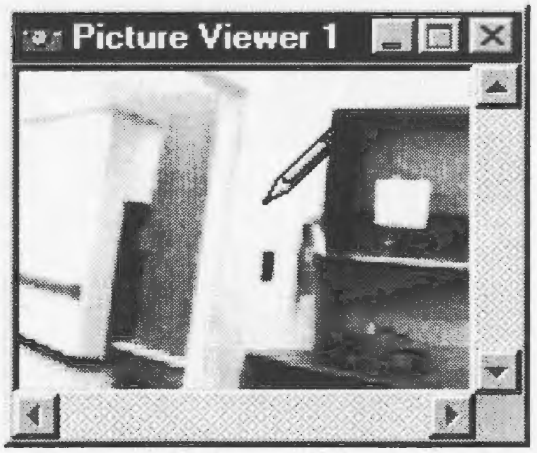

Figure 4–39. Annotation of Still Image

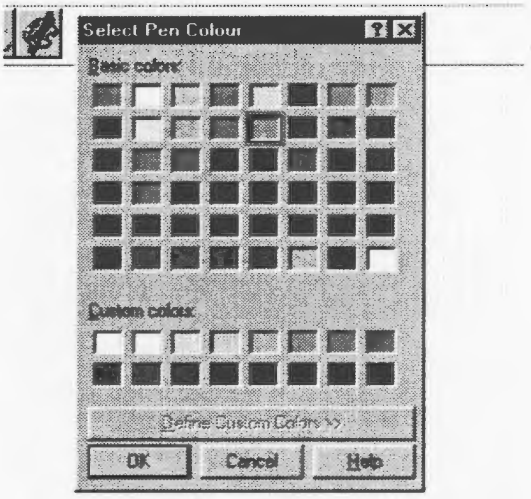

Figure 4–40. Selection of Pen Colour for Annotation

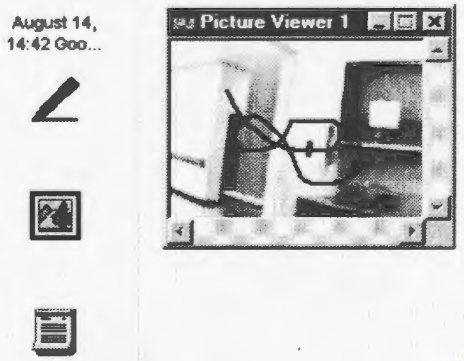

Figure 4–41. Insertion of Annotated Media

CHAPTER FIVE

IMPLEMENTATION ISSUES

5.1 Development Environment

Development and testing of the telemedicine applications was done on Pentium PCs, running Windows© 95. The environment used to design and develop the applications was *Powersoft's Power++*[5], a commercially available C++ rapid application development (RAD) tool. Application design also used *MCLGallery*, a C++ multimedia library developed in-house for *Power++* [Chen97]. Information regarding the number of lines of code for the three systems is as follows:

[5] Available from *Sybase Inc., http://www.sybase.com/products/powerpp/index.html*

- *MailTeleMed*[6] – 5270 lines
- *WebTeleMed*[7] – 170 lines
- *MCLTeleMed*[8] – 8600 lines

5.2 Third-Party Tools and Components

In the development of the three systems, third-party software tools and components were used to facilitate their operation. In *MailTeleMed*, the use of the *NetManage* POP (Post Office Protocol) and SMTP (Simple Mail Transfer Protocol) software components[9] were used in order to read the user's mailbox and to transmit electronic mail messages, respectively. In *WebTeleMed*, the *MHonArc*[10] mail-to-web converter was used to convert the patient encounters within the user's (the central site) mailbox to a series of web pages, displaying the patient encounters for the site.

5.3 Issues and Problems in Implementation

Outstanding issues remain in the three telemedicine applications dealing with security issues and user testing. The outstanding security issues are discussed in **Section 7.3** and user testing in **Chapter 6**. The only problem of serious importance faced during

[6] The number of lines of code refers to code written, not generated, and includes comments and white space.
[7] The number of lines listed is for the *caselist.rc* resource file used for conversion.
[8] The number of lines of code refers to code written, not generated, and includes comments and white space.
[9] Software components are now distributed by *NetMasters LLC – http://www.startups.com/default.htm*
[10] *MHonArc Email to HTML Converter – http://www.oac.uci.edu/indiv/ehood/mhonarc.html*

the implementation of the telemedicine systems, concerns the time elapsed from the conception of the project to the approval for clinical trials of the proposed telemedicine applications. It is strongly felt that had acceptance been granted earlier, the clinical trials would have offered invaluable data.

5.4 Testing and Verification

Testing and verification was conducted for two of the three telemedicine systems – testing was performed on the *MailTeleMed* and *MCLTeleMed* systems. Since the *WebTeleMed* system simply uses a script for conversion to HTML (Hypertext Mark-up Language), testing simply involved editing the script to produce the desired look of the produced Web pages.

For the *MailTeleMed* and *MCLTeleMed* systems, white-box testing of the systems was conducted followed by black-box testing of the systems. The purpose of white-box testing is to test the program based on knowledge of the internal workings of the application [Robi94]. Black-box testing is functional testing for the purposes of system tests and to determine if the applications meet the functional specification [Myna90]. White-box testing was conducted during the development phase of these systems to ascertain that the system functioned properly and for debugging purposes.

Following the debugging and white-box testing phase, a stable version of the system was produced to undergo black-box testing by a member of the *Multimedia Communications Lab* (MCL) group, who was unfamiliar with the inner workings of the

system. The purpose of this testing was not only for stress-testing, i.e. to determine the limits of the application such as the maximum number of patients allowed, but to confirm that the system behaves in accordance with the specification. The tester's responsibilities included providing actions and behaviours unexpected or unanticipated by the application to ensure that it fails "gracefully" – without causing the computer system to "lock up" or terminate without warning. Black-box user testing also provided an alternative perspective to the application affording suggestions as to improvements in the interface and removing ambiguities in the user interface. After the black-box user testing was completed, modifications and bug-fixes were made to the application, followed by another round of user testing, both white-box and black-box.

Examples of feedback from the black-box testing include:

- Notifying the user that mouse movement with the left mouse button pressed annotates images. Previously, no indication that this could be done was provided. Additionally, the ability to change pen colour was wished.

- If the program is terminated while a patient is added or items are added to an encounter, then these additions are neglected. It was originally conceived that such an operation amounts to the user making a mistake and does not wish the addition to be made. After the testing it was decided that additions of patients would be accepted as well as data items, as long as the operation has been committed by the

user. If the user records a video and terminates the program without stopping the recording, then the recording will not be saved.

5.5 Design Decisions

During the course of development, some decisions were made that lead to a non-optimal solution. Some of these decisions will be listed as well as their reasons.

5.5.1 *MailTeleMed*

To speed up the reading of the user's mailbox and to allow the user to continue working while an encounter is mailed, the decision to spawn two processes, one for reading and one for sending, was made. The program would start up, access the user's mailbox, retrieve each item, and display it in the folder. While the mail items are being retrieved, the user is able to access already retrieved encounters or create new encounters. On the sending side, the process for mailing the messages can occur while the user continues to work. In this way the user need not pause and wait for either process to finish before continuing with his/her work. Unfortunately, this decision resulted in an increase of required working disk space to store the encounters, both incoming and outgoing, for temporary files which are removed after the application terminates. As a result, if a large number of encounters are being retrieved and/or sent, the amount of working disk space may take up all the available free disk space on the machine causing adverse effects to the application and the operating system. Unfortunately, the only solution to this is to ensure that a large

amount of free disk space is available and a general guideline as to how much space is required cannot be specified as the disk space required for a patient encounter is dependent upon the quantity and type of items in each encounter.

5.5.2 *WebTeleMed*

As has been mentioned numerous times, the design of the *WebTeleMed* system did not provide for the addition of new encounters – this would be performed using the mail component of the browser or other standard mail applications. The reasoning for this is that since *WebTeleMed* relied on the user interface of the Web browser favoured by the user, providing a new interface for adding patient encounters may affect usability. Users already familiar with a specific browser and its components would surely prefer to use that than learn a new application or component.

5.5.3 *MCLTeleMed*

In designing the *MCLTeleMed* system, originally no bounds were placed on the size of the audio and video media items inserted into patient records. It was assumed that users would be knowledgeable enough to be able to estimate the size of a video sequence, for example, using the length of the recording. This proved to be a poor design choice when we learnt that the user group had little or no experience with computers. A decision was made to place a limit on the size of audio and video items inserted. Two choices were available: set a maximum time for the recording or set a maximum size in terms of disk usage. The latter option was the preferred choice

using a limit of half the available disk space available – the reason for this is that the Windows© operating system saves recordings to a temporary file before saving it to the specified file. However, implementing this requires Windows© functions available in the Windows 95 OSR2 (OEM Service Release) release of the operating system and later. Functions are available for pre-OSR2 releases but are unreliable for hard disks greater than 2 GB in size. It was decided that until the specifications of the target operating systems could be determined, we would implement a restriction based on the recording time.

A second design decision involves the editing and updating of the patient history files. As these files are editable and are treated as patient media items during the site reconciliation process, a decision must be made on whether the creation (or edit) time of this file is included in determining the most recent patient media item. Either choice produces undesirable results but one must be chosen.

The following scenario was developed to test the outcomes of each design decision: A new patient is created at the remote site and is sent to the central site when it connects at 11:00 a.m.. Three situations then arise (it is assumed that each site disconnects while these test cases are conducted and then reconnect):

1. A new item is added to the patient's record at the central site.

2. A new item is added at the central site and the patient history is edited at the remote site some time after the creation of the new item (i.e. the new item is added at 1:00 p.m. and the history file is edited at 2:00 p.m.).

3. The patient history is edited at both sites in between two record updates.

The results for these cases are as follows:

1. If the patient history is not included in the calculation of the most recent file, then in addition to sending the new item, the patient history file sent initially is re-sent to the remote site. If the file is included in the calculation, then only the new item created is sent to the remote site.

2. In this case, not including the history file in the calculation results in a correct transmission – the new item is sent from the central site to the remote site and the edited history file is sent from the remote site to the central site. If the history file is included then the remote site fails to obtain the new item created at the central site but sends the edited history file to the central site.

3. For this case, both choices result in the history file at the remote site being overwritten by the file from the central site. The history file is also retransmitted to the central site by the remote site if the file is not included in the calculation, as the remote site sends what it thinks is its version of the file.

Now, this whole problem can be solved but not without adding complexity to the operation and affecting the speed of the updates. As a result, the lesser of the two evils was chosen and the history file is not included in the calculation of the recent file for each patient.

CHAPTER SIX

USER TESTING

At the time of this writing, testing of the three telemedicine systems under consideration by the target user group had not been performed. This was because of delays in the approval process for clinical testing. User testing of the *MCLTeleMed* system has been scheduled to commence in September 1998. However, informal testing was performed by administrators over-seeing telemedicine operations in Labrador and informal input was obtained during the development process. The persons involved in both the informal testing and discussions were Dr. Rod Elford, Andrea Battcock, Dr. Carl Robbins, and Dr. Michael Jong. During the course of development, the testers were shown the systems being developed in order to obtain their input. Input was sought regarding the user interface, the operation of the system and additional functionality useful to clinicians.

For the informal testing, the "testers" were shown the three telemedicine systems, *MailTeleMed*, *WebTeleMed* and *MCLTeleMed*, and went through the steps of performing a consultation – both initiating a consultation and responding to a query. The perceived user model for each system was explained as well. Following the demonstration of the three systems, the participants were asked to answer a number of questions (see **Figure A–1** in **Appendix A** for the questionnaire). During the course of the demonstration and in the questionnaire, all attempts were made to prevent any bias from being introduced regarding our opinion of which system is most suitable for teleconsultations, namely *MCLTeleMed*. This was accomplished by explaining all three systems to the same level of detail, providing examples for each system of the same level of quality and by not mentioning, in our opinion, the advantages and disadvantages of each system.

The results of the informal testing, based upon the questionnaire given and discussion meetings held during the course of development, provided valuable insight as to the best application for teleconsultation operations in Labrador, and the associated user model (see **Figure A–2** in **Appendix A** for a sample response).

6.1 *MailTeleMed* – Mail-Based System

Concerning the mail-based system, respondents liked the electronic mail-based solution as it presented a low cost solution. However, the need for organization of the patient encounters on the user's part and the inability to view the encounter history of the patient, limit the advantage of such a solution.

The Telemedicine Department has been involved in trials with the VisiTran® mail-based teleconsultation system (see **Section 2.3**) between the Hibernia oil platform and a hospital in St. John's. Users involved with the trials have expressed the following comments regarding the system:

- Over extended use, a large number of cases are created – numerous patient encounters for the personnel on the platform. These cases are all listed in a single folder ordered by date, making searching for a patient encounter difficult. The operator, to make using the system manageable, is forced to invest time into organizing all the cases into folders (e.g. A-F, G-M, etc.). While this does solve the problem, it means that a great amount of time is spent simply "tidying" up the cases.

- The VisiTran system used compression for the still images inserted but did not compress video sequences captured! Considering the size of typical video items compared to still images, it seems logical to compress video sequences with compression optional for still images. In fact, users have mentioned that they would prefer if images were *not* compressed. This is reasonable as depending upon the compression algorithm used and the amount of detail required by clinicians, information may be lost during compression, proving unsuitable for clinicians.

- Users noted that VisiTran did not allow the operator to open multiple windows for images, hence being unable to compare images. Multiple windows for other modes of media were not viewed as important. Text items would not number greatly as clinicians preferred to record audio than type. Multiple instances of video and audio items would not be possible, at least under Windows.

6.2 *WebTeleMed* – Web-Based System

User responses for the Web-based system were in general accord with those comments for *MailTeleMed*. This is not surprising as the two systems are very similar aside from the method used to access the medical data. As expected, respondents liked the ability for simultaneous, multiple user access to the data.

6.3 *MCLTeleMed* – Replicated Database System

It was a general consensus among respondents that the *MCLTeleMed* system was the best application for their use. The ability to view a patient's entire encounter history as well as the automatic organization of all encounters into one record, were features most liked by all respondents. Respondents commented on how the system most resembled the traditional paper-based view of a medical record. An additional benefit identified was that the user need not have to be concerned about with the transfer of patient encounters or keeping patient records up-to-date since all sending and retrieval of patient records is done automatically and is transparent to the user.

6.4 General Remarks

In response to the question of which system respondents would use, both *MCLTeleMed* and *MailTeleMed* were named, with *MCLTeleMed* being the most preferred. The inclusion of *MailTeleMed* in the response likely stems from trials recently conducted in Labrador clinics of the *VisiTran* system, an electronic mail-based telemedicine system (as discussed in **Section 2.3**).

Discussions with respondents revealed that the asynchronous operation of the teleconsultation systems is sufficient for their needs and best matches their perception of the consultation process. While synchronous communication would be valuable on occasion, usage of this mode is negligible due to scheduling difficulties of the consultation.

Owing to the small sampling, it is recognized that the results of these informal tests and discussions cannot be reliably used to arrive at conclusions of the effectiveness of each model presented. It is our firm conviction, however, that they do provide a general guide as to the effectiveness of each.

From the informal discussions held during development and during the testing phases, it was determined that the security and confidentiality of the patient data and limiting access to the information were of concern to clinicians and administrators.

From both the informal tests and discussions, the following user interface issues arose:

- It was determined, as previously mentioned, that multiple image windows would prove beneficial to clinicians. This feature would allow users to compare images and be able to track changes between images.

- The sorting of the list of patient encounters/records within the folder should be performed in the following manner; in *MailTeleMed* and *WebTeleMed*, the encounters should be sorted by in most-recently received to least recently received order. For *MCLTeleMed*, the records should be sorted in the folder alphabetically by last name. This would cause records of persons in the same family to be grouped together and would ease searching for a record in the folder.

- When browsing a patient record in *MCLTeleMed*, the encounters should be sorted by date with most recent encounter at the top of the scrolling list. The reason for this simple; users are most likely to view data from a recent encounter than one from the patient's past.

- The ability to print items, text and still image items only, from within the application would avoid the necessity to search for the component item in the directory structure in order to print.

- The provision of a comments box allows clinicians to view a patient encounter and compose a response at the same time, without having to create a new encounter to

do so. This was implemented in *MCLTeleMed* and provided for both textual and audio comments.

Arising from the user testing, respondents suggested additional functionality they would like to see in the application – searching capabilities for ailments, sorting by last names, automatic backups of all records to disk, printing of text and still image items, etc. Due to time restrictions, we were unable to include these features into the applications but have been slated for future work.

6.5 User Training

User training of the systems is expected to be easy, requiring a few sessions to become proficient with each system assuming users have some basic computer training. It is expected that the *WebTeleMed* system is the easiest to learn, provided the user is familiar with Web browsers. Training will likely require more time for the *MailTeleMed* and *MCLTeleMed* systems to get used to the interface and the operations possible. The most difficult aspect of the applications is likely going to be for the user to become proficient with the capturing of the media for the patient encounters. Aside from this, the setting up of the calls may prove tricky if the user is unfamiliar with the communications protocols available.

CHAPTER SEVEN

CONCLUSIONS

7.1 Major Results

Three applications were developed addressing the issue of low-data rate medical teleconsultation in Labrador. The applications provided easy-to-use interfaces and functionality customized to the target user group. The applications developed are robust, efficient and well documented.

From the study conducted using these applications, two major conclusions can be drawn:

- asynchronous communication is a natural mode for medical teleconsultations
- an electronic analogy to the traditional paper-based medical record is the most appropriate application model for medical teleconsultations

The first conclusion agrees with those reached by *Allely* [Alle95]. It is, however, puzzling in the light of the present trend of a proliferation of videoconferencing-based telemedicine systems, which operate on a synchronous basis, for the purposes of teleconsultations. This does, however, explain the status of electronic mail-based systems as the current state-of-the-art.

The second conclusion is a natural and intuitive one – presenting users with a new system that functions similarly to whatever method was used before will naturally be preferred to a system which is unfamiliar. This conclusion is also in agreement with the conclusion made by *Forslund and Kilman* [Fors96] *MCLTeleMed* takes advantage of this by making use of patient records that are similar to the traditional paper-based medical records and partly explains the preference of testers of this system over the other systems presented. Additionally, the ease of use of the system and the automation of patient record updating performed by the system add to the advantages of the *MCLTeleMed* system over the others.

7.2 Discussion

The conclusions reached in this study have implications on systems and applications developed for medical consultations between physicians, now and in the future. The first conclusion reached – that asynchronous communication is the natural operational mode – points away from the current trend of developing synchronous teleconsultation systems (e.g. videoconferencing systems). The difficulties in scheduling consultations coupled with typical bandwidth requirements, severely limits the application of synchronous systems in remote areas. This is, essentially, a common-sense notion, yet there is an abundance of videoconferencing teleconsultation systems on the market directed for use in remote areas. However, the poor video quality and communication requirements frequently result in the deployment of such systems in areas where the communications infrastructure required is in place – large city centres – and not rural and remote areas.

The second conclusion reached provides a clear direction as to the development of electronic patient medical records – that such records should take the traditional paper-based medical record as a model. In so doing, advantage is taken of the familiarity health care professionals have with the paper-based medical while providing an electronic medium for the information which allows for better record-keeping and handling and for efficient data mining.

7.3 Future Research

The first objective of any future research should be to validate the conclusions reached in this study through formal user testing and clinical testing. Other areas of future research would include:

- quantifying the additional information provided during a consultation, if any, using the telemedicine systems developed during this research when compared to using telephones for the consultation

- determine the benefits of utilizing multimedia items for patient medical data in physician teleconsultations

Future areas of work would involve adding more functionality desired by users (searching capabilities for ailments, sorting by last names, automatic backups of all records to disk, printing of text and still image items). As mentioned in **Section 6.3**, these desired features arose from the user testing and the informal discussion but could not be included in the release versions. Another area of future work is the patient record synchronization. Presently, it may not always be clear to the user how current the records are. Informing the user of the last synchronization would solve this problem. As well, the capability to perform a quick check to see if the records do need to be synchronized should be provided. The user could then decide if they wish to synchronize the records or delay it. Related to this is the automatic updating of the records. To take advantage of time-of-day savings for long-distance calls, the ability to

set the system to automatically connect to the central site to update the records. In this manner cost savings can be achieved without the requirement of an operator to operate the system.

REFERENCES

[Alle95] Allely, E.B., "Synchronous and Asynchronous Telemedicine", *Journal of Medical Systems*, vol. 19 no. 3, pp. 207-212, 1995.

[Bash77] Bashshur, R., and Lovett, J., "Assessment of telemedicine: results of the initial experience", *Aviation, Space and Environmental Medicine*, vol. 48 no. 1, pp. 65-70, 1977.

[Bash97] Bashshur, R.L., Sanders, Jay H. and Shannon, Gary W., eds., *Telemedicine: Theory and Practice*, Charles C Thomas Publisher Ltd., Springfield, Illinois, pp. 10-12, 1997.

[Bens65] Benschoter, R.A., Wittson, C.L., Ingham, C.G., "Teaching and consulting by television", *Hospital and Community Psychiatry*, vol. 16, pp. 99-100, 1965.

[Chen97] Cheng, L., "MCLGallery Version 1.1 Documentation", *Internal Technical Report 1997:1*, Multimedia Communications Lab, Memorial University of Newfoundland, 1997.

[Date90] Date, C.J., *An Introduction to Database Systems, Volume I (Fifth Edition)*, Addison-Wesley Publishing Company, Reading, Massachusetts, pp. 6-7, 1990.

[Dohr91] Dohrman, P.J., "Low-cost teleradiology for Australia", *The Australian and New Zealand Journal of Surgery*, vol. 61, pp. 115-117, 1991.

[Eber94] Eberts, Ray E., User Interface Design, Prentice-Hall, Englewood Cliffs, New Jersey, 649 p., 1994.

[Fiel96] Field, Marilyn J. (ed.), *Telemedicine: A Guide to Assessing Telecommunications in Health Care*, National Academy Press, Washington, D.C., p. 81, 1996.

[Fors96] Forslund, D. and Kilman, D., "The Virtual Patient Record: A Key to Distributed Healthcare and Telemedicine", 1996, *http://www.acl.lanl.gov/TeleMed/Papers/virtual.html*

[Fuch79] Fuchs, M., "Provider Attitudes Toward STARPAHC: A Telemedicine Project on the Papago Reservation", *Medical Care*, vol. 17, pp. 59-68, 1979.

[Gali97] Galitz, Wilbert O., *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*, John Wiley & Sons, New York, New York, pp. 220-224, 271-294, 1997.

[Grig95] Grigsby, R.K., "Telemedicine", *Journal of the American Medical Association*", vol. 274, no. 6, pp. 461-462, 1995.

[Gome94] Gomez, E.J., et al., "A Telemedicine Distributed System for Co-operative Medical Diagnosis", *American Medical Informatics Association*, pp. 433-437, 1994.

[Grah96] Graham, A.M., "Telepsychiatry in Appalachia", *American behavioral scientist*, vol. 39, no. 5, pp. 602-615, 1996.

[Hous77] House, A.M., and Roberts, J.M., "Telemedicine in Canada", *Canadian Medical Association Journal*, vol. 117, no. 4, pp. 386-388, 1977.

[Hous91] House, M., "Canadian experience: using telemedicine for the support of medical care at remote sites", *International Telemedicine/Disaster Conference*, NASA, Washington, 1991.

[Hous94] House, A.M., "Distance Health Systems of International Significance", *National Conference on Hospital Management and Services*, Guangzhou, China, April 1994.

[Moor96] Moore, D., "Mobile Code in Networked Medical Systems", *Masters Thesis, Dept. of Biomedical Engineering*, University of Virginia, August, 1996.

[Myna90] Mynatt, Barbee Teasley, *Software Engineering with Student Project Guidance*, Prentice-Hall, Englewood Cliffs, New Jersey, pp. 280-297, 1990.

[Oroz92] Orozco-Barbosa, L., et al., "A Multimedia Interhospital Communications System for Medical Diagnosis", *IEEE Journal on Selected Areas in Communication*, vol. 10 no. 7, pp. 1145-1157, 1992.

[Paiv71] Paivio, A., *Imagery and verbal processes*, Holt, Rinehart and Winston, New York, New York, 596 p., 1971.

[Pres92] Preston, J., Brown, F.W., Hartley, B., "Using telemedicine to improve health care in distant areas", *Hospital and Community Psychiatry*, vol. 43, pp. 25-32, 1992.

[Robi94] Robinson, J. , *Essential Software Design (Second Edition)*, University of Waterloo, Waterloo, Ontario, pp. 294-297, 1994.

[Stan70] Standing, L., Conezio, J. and Haber, R.N., "Perception and memory for pictures: Single-trial learning of 2500 visual stimuli", *Psychonomic Science*, vol. 19, pp. 73-74, 1970.

BIBLIOGRAPHY

1. Bellon, E., van Cleynenbreugel, J., Suetens, P., Marchal, G., van Steenbergen, W., Plets, C., Oosterlinck, A., and Baert, A.L., "Multimedia E-mail systems for computer-assisted radiological communication", *Yearbook of Medical Informatics*, pp. 356-365, 1995.

2. Cameron, John R. (1989). *JSP & JSD: The Jackson Approach to Software Development (Second Edition)*, IEEE Computer Society Press, Washington, D.C., 526 p.

3. Coiera, Enrico (1997). *Guide to Medical Informatics, the Internet and Telemedicine*, Chapman & Hall, London, U.K., 376 p.

4. DeMarco, Tom (1978). *Structured Analysis and System Specification*, YOURDON Inc., New York, New York, 352 p.

5. Field, Marilyn J. (ed.) (1996). *Telemedicine: A Guide to Assessing Telecommunications in Health Care*, National Academy Press, Washington, D.C., 271 p.

6. Forslund, D.W., Phillips, R.L., Kilman, D.G., Cook, J.L., "Experiences with a Distributed Virtual Patient Record System", *Proceedings of the 1996 American Medical Informatics Association Annual Fall Symposium*, pp. 483-487, 1996

7. Jacobson, Ivar (1994). *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Publishing Co., New York, New York, 524 p.

8. Kilman, David G. and Forslund, David W., "An International Collaboratory Based on Virtual Patient Records", *Communications of the ACM*, vol. 40 no. 8, pp. 111-117, 1997.

9. Reis, H., Brenner, D. and Robinson, J., "Multimedia Communications in Health Care", *Annals of the New York Academy of Sciences*, vol. 670, pp. 257-268, 1992.

10. Roe, P.H., Soulis, G.N., and Handa, V.K. (1992). *The Discipline of Design*, University of Waterloo, Waterloo, Ontario, 283 p.

11. Taylor, P., "A survey of research in telemedicine. Telemedicine systems", *Journal of Telemedicine and Telecare*, vol. 4 no. 1, pp. 1-17, 1998.

12. Treves, S.T., Hashem, E.S., Majmudar, B.A., Mitchell, K., and Michaud, D.J., "Multimedia Communications in Medical Imaging", *IEEE Journal on Selected Areas in Communications*, vol. 10 no. 7, pp. 1121-1131, 1992.

13. Zajtchuk, R., Goeringer, F. and Mun., K. (1995). *Proceedings of the National Forum, Military Telemedicine On-Line Today: Research, Practice and Opportunities, March 27 – 29, 1995, McLean, Virginia*, IEEE Computer Society Press, Los Alamitos, California, 179 p.

APPENDIX A

USER QUESTIONNAIRE AND RESPONSE

Questionnaire

1. What did you like about the *MailTeleMed* system? What did you not like about the system? How can the *MailTeleMed* system be improved?

2. Provide any comments on the usability of the *MailTeleMed* system.

3. What did you like about the *WebTeleMed* system? What did you not like about the system? How can the *WebTeleMed* system be improved?

4. Provide any comments on the usability of the *WebTeleMed* system.

5. What did you like about the *MCLTeleMed* system? What did you not like about the system? How can the *MCLTeleMed* system be improved?

6. Provide any comments on the usability of the *MCLTeleMed* system.

7. Do you see yourself using any of these systems? If so, which one would you use and why? If not, why not?

8. Which of the three systems shown is best suited for medical teleconsultations? Why?

Figure A–1. Questionnaire Following Demonstration

```
From mjong@cancom.net Fri Jun 26 13:00:56 1998
Date: Fri, 26 Jun 1998 11:23:04 -0400 (EDT)
Message-Id: <199806261523.LAA24599@ns.cancom.net>
X-Sender: mjong@mail.cancom.net
X-Mailer: Windows Eudora Light Version 1.5.2
Mime-Version: 1.0
Content-Type: text/plain; charset="us-ascii"
To: Rahim S Pira <rahim@engr.mun.ca>
From: Michael Jong <mjong@cancom.net>
Subject: TeleMed

Rahim,

Thanks for the demo. You did an excellent job.  I will be happy to pilot
and
with continuing improvement it will be have great potential for
marketing.

The following is my response to your questionnaire:
1. Mail system offers a low cost operation.  The inclusion of patient
data and the ability to pull the file out for storage in another system
for querying/statistical analysis would be useful.  As it is, the main
concern is with security.

2. Mail system is usable in all our communities if the security issue
can be solved.

3. I am not certain how the Web system is different from the mail system
other than the potential for multiple users to access the info.

4. Am concern with security. Usable. More patient data will be useful -
including name, date of birth, age, sex, ID no., address, tel. no.,
active medical problems (itemised), past medical history, social history
(smoking, alcohol, occupation, marital status), current medications,
family history, allergies and side-effects to drugs.

5. MCL system seems to be the best mainly because of the top window for
patient data and more security.  Ability to synchronize file is good.
Having past interactions in the same window is great. Like the most
current interaction on top.

6. Certainly usable

7. Would use MCL and perhaps mail. Would like to try mail to determine
difference in operational cost.

8. MCL - security, additional window for patient data, potential for
querying with another software package.

Michael
```

Figure A–2. User Response to Questionnaire

APPENDIX B

MAILTELEMED CLASS LISTING

Figure B–1. *TeleMed* Class Listing

```
//
// TeleMed - This is the main class (the main window containing the
//           inbox and outbox).
//
// created by Li-Te Cheng, April 1997
// modifications:
//      Rahim Pira, July, October 1997
//          - addition of Record_Viewer
//          - create and load case buttons to inbox
//          - added outbox
//          - added logoff button (batch mailing)
//

class TeleMed
{
    protected:
        Connection_Info    info;           // logon information
        Outgoing_Queue     outgoingQueue;  // queue of outgoing cases
        Record_Viewer      *caseViewer;    // the case viewer
    public:
        void InitMailbox();
        // perform some initialization of the mailbox
};
```

Figure B–2. *Logon_Form* Class Listing

```
//
// out_mailbox - This class handles the queuing of outgoing cases and
//               their sending via the SMTP control.  It also manages
//               the queue of cases scheduled to be mailed.
//
// created by Rahim Pira, October 1997
//

class Logon_Form
{
    protected:
        Connection_Info *info;  // pointer to object
        // add your protected instance data here
    public:
        void SetInfo( Connection_Info *info_ptr);
        // sets pointer to Connection_Info structure
};
```

Figure B-3. *in_mailbox* Class Listing

```
//
// in_mailbox -  This class handles incoming cases via the POP
//               control and is based on Alan Armstrong's Optima++
//               mail reader example from the Absolute Power++ WWW
//               site.
//
// created by Li-Te Cheng, April 1997
// modifications:

//      Rahim Pira, May 1997
//          - modifications made to mail receiving to allow obtaining
//            of sender's real name, as well as to message reading to
//            correctly parse multimedia content
//

class in_mailbox
{
    protected:
        static Record_Viewer *currentCase;
        static Connection_Info *info;
        static int status;
        int n_messages,         // total number of messages in mailbox
            current_message,    // the current message
            displayed_case;     // the case currently being displayed

        WBool connectedToPOP,
              resolvedHost,
              first_time;
        WCursor oldcursor, hourglass;
    public:
        void SetInfo( Connection_Info *info_ptr );
        // set pointer to Connection_Info structure
    public:
        void ScanMail();
        // begins process of scanning incoming mail
    public:
        void AddDate( WString & date );
        // puts "date" in current position in incoming mail grid and
        // current case viewer
    public:
        void AddFrom( WString & from );
        // adds "from" data to current position in incoming mail grid
        // and current case viewer
    public:
        void AddSubject( WString & subject );
        // adds "subject" to current position in incoming mail grid
        // and current case viewer
    public:
        void TrimTabs( WString & str );
        // removes leading and trailing tabs in string
    public:
        void AddCc( WString & cc );
        // add "cc" data to curent case viewer
```

```
    public:
        WString formatHdrString( WString & str );
        // parse str, from the mail's header, to obtain the
        // recipient's or sender's real name
    public:
        WBool FindCaseviewer();
        // if the Record_Viewer is unavailable, it is destroyed
};
```

Figure B-4. *out_mailbox* Class Listing

```
//
// out_mailbox - This class handles the queuing of outgoing cases and
//               their sending via the SMTP control.  It also manages
//               the queue of cases scheduled to be mailed.
//
// created by Rahim Pira, October 1997
//

class out_mailbox
{
    protected:
        static Connection_Info  *info;
        static Outgoing_Queue   *mailQueue;
        WStatusBar              *statusBar;
        int   mailingCase;     // index of case currently being mailed
        _DualDocHeaders         *dispHeaders;
        _DocHeaders             *HDRS;
        char                    *data;
        ifstream                ifile;
        WString                 filename;
        unsigned long           fileLength;
    public:
        void SetInfo( Connection_Info *info_ptr );
        // set pointer to Connection_Info structure
    public:
        WBool AddCase( WString & filename, WString & date,
                       WString & to, WString & cc,
                       WString & subject, WString & from );
        // adds the composed case to the queue - filename, and mail
        // headers of case
        // returns true if case was queued, false if too many cases
        // in the queue
    public:
        void RemoveCase( int caseIdx);
        // removes the specified case from the queue
    public:
        int GetQueueSize();
        // return the size of the queue
    public:
        void MailQueue();
        // mail off the topmost case in the queue
    public:
        void SetQueue( Outgoing_Queue *queue_ptr );
        // set the pointer to the queue
    public:
        void MailCase( int index );
        // send only one case
};
```

Figure B-5. *connection_info* Class Listing

```
//
// Connection_Info - A class storing the logon information which is
//                   passed between forms.
//
// created by Rahim Pira, April 1997
//

class Connection_Info
{
public:
        WBool   use_email,      // use e-mail address
                use_phone,      // connect by phone
                logon;          // logon or cancel
        WString name,
                address,
                userid,
                password,
                pop,
                smtp,
                mailfile,
                phone,
                save_path;   // must end with "\"
};
```

Figure B-6. *noteviewer* Class Listing

```
//
// noteviewer - This class handles the notepad and text box used for
//              entering and displaying text items in cases.
//
// created by Rahim Pira, May 1997
//

class noteviewer
{
    public:
        void AddText( const WString & text );
        // puts text in text window.
    public:
        void NewText( const WString & text );
        // replaces text box contents with "text"
    public:
        void SetReadOnly(  WBool readOnly );
        // set the read-only property of the text box
    public:
        void AtTop();
        // set the cursor at the beginning of the text box & clear
        // the undo buffer
    public:
        WString GetMessage();
        // get the text entered into the text box
    public:
        WBool IsEmpty();
};
```

Figure B-7. *Record_Viewer* Class Listing

```
//
// Record_Viewer - This class is the viewer in which an incoming case
//                 is displayed and used to create a case for
//                 sending; as such, this class is responsible for
//                 all the recording of the case multimedia items
//                 and for sending the case off (actually, it's
//                 passed on to out_mailbox which queues the case and
//                 then sends it off).
//
// created by Rahim Pira, April 1997
//

class Record_Viewer
{
    protected:
        static Connection_Info  *info;
        WBool  available,     // flag for availability of viewer
               noteVisible,   // visibility flag for noteviewer
               addingText,    // flag for adding textual component
               composing,     // flag for composing or reading mail
               noteSaved;     // flag for status of saving note

        // these are flags used during the composition of a new
        // case for the various multimedia components; in case the
        // user switches from button to another button without
        // stopping these actions, some way to track them is needed
        WBool  videoPreview,
               picturePreview,
               videoRecording,
               audioRecording;

        WImageList iconlist;       // icons for list view
        WLong      iconindex[MAX_ICONS];
        WString    item_filename[MAX_ITEMS]; // attachment file names
        int        item_type[MAX_ITEMS],     // MIME types
                   n_items,
                   current_item,
                   item_protect[MAX_ITEMS];  // delete item?
        Filter_Base64MIME   base64filter;
        MCLBitmap  *bitmap;
        int        hscrollPosition,
                   vscrollPosition;
        WString    buffer;
        WULong     buffered_pos;
        int        reading_file;
        WString    boundary;     // MIME boundary string to be used
        MCLVideo   video_camera;
        WCursor    normal, hourglass;
        UINT       wDeviceID;       // waveform-audio device ID
        WBool      recordTimerOn;   // still recording?
    public:
        void SetDate( WString & date );
        // set the Date field for mail message
```

```
public:
    void SetFrom( WString & from );
    // set the From field for mail message
public:
    void SetSubject( WString & subject );
    // set the Subject field for mail message
public:
    int MakeNote();
    // creates a new note window
    // returns 1 on success, 0 on failure.
public:
    int AddAttachment( WString & text, WString & mime_boundary );
    // adds multimedia data to current case window
    // returns 1 on success, 0 on failure.
public:
    void SetTitle( WString & title );
    // sets title on form and note pad
public:
    WBool IsAvailable();
    // whether or not it's being used to display a mail message
public:
    int SaveItem( const WBuffer & data, int type,
                  int encoding=MIME_BASENONE,
                  int n=-1, const char *filename=NULL );
    // saves item n with data to filename; if filename=NULL,
    // creates a unique temporary filename to save the data into;
    // will automatically append type extension to
    // filename if necessary.  If n=-1, adds filename to
    // item_filename protected variable and increments n_items.
    // returns 0 on failure, 1 on success.
public:
    void SetInfo( Connection_Info *info_ptr );
    // passes along connection information parameters to this
   //  class
public:
    int ShowItem( WLong n );
    // displays the contents of item n in the appropriate window
public:
    void SetBitmapScrolling();
    // sets up bitmap window for scrolling
public:
    void ShowBitmap();
    // enables bitmap window; disables all other overlapping
    // windows (e.g. media player)
public:
    void HideBitmap();
    // hides bitmap window and associated controls
public:
    void HideMediaBox();
    // hides the media box and closes/saves associated data
public:
    void ShowMediaBox();
    // shows the media box
public:
    void BeginReading();
    // indicates that we're reading one e-mail message component.
```

```
public:
    void EndReading();
    // indicates we're finished reading one e-mail component.
public:
    void ComposeCase();
    // begin composition of a new case
public:
    void SetTo();
    // set the to/from label of the mail message to "To:"
public:
    void AddDate();
    // add the current date & time to the date field of the mail
    // message
public:
    void SaveComponents( ostream & outfile, WString & boundary );
    // write the mail attachments to outfile which must be opened
    // prior to calling this function
public:
    void SetCc( WString & cc );
    // set the Cc field of the mail message
public:
    WString FormatSmtpHeader( WString str );
    // format the string to comply with SMTP standard for headers
public:
    void SaveNoteItem();
    // save the note item to text file
public:
    void ShowVideoBox();
    // show the MCLVideo box and its controls
public:
    void HideVideoBox();
    // hide the MCLVideo box and its controls
public:
    void ShowBitmapControls();
    // show the MCLBitmap controls
public:
    void HideBitmapControls();
    // hide the MCLBitmap controls
public:
    void SetStatusBar( WString text );
    // set the text of the status bar
public:
    WBool LoadCase( WFilePath & casePath );
    // load the specified case into the Record_Viewer
public:
    void SetTo( WString & text );
    // set the to/from label of the mail message to "To:" and set
    // the field to the given text
public:
    void ShowAudioBox();
    // show the controls for recording audio
public:
    void HideAudioBox();
    // hide the controls for recording audio
public:
    void StopOtherRecordings();
    // stop the recording of other components
```

```
    public:
        void ShowNote();
        // show the notepad
    public:
        void HideNote();
        // hide the notepad
    private:
        void DeleteItem( int item );
        // delete the selected item
    public:
        WBool IsComposing();
        // is the user composing a case?
    public:
        WBool CompressVideo( int itemNumber );
        // compress the video using Microsoft's Video 1 codec
    public:
        int GetNoteItem();
        // get the item number of the note or -1 if none
};
```

Figure B-8. *Filter_Base64MIME* Class Listing

```
//
// Filter_Base64MIME - This class can encode and decode a byte buffer
//                     as per the Base 64 MIME specification.
//
// created by Li-Te Cheng, April 1997
// modifications:
//      Rahim Pira, June 1997
//          - addition of encoder for Base 64 MIME specification.
//

class Filter_Base64MIME
{

    public:
        int DecodetoFile( const unsigned char * in_buffer,
                          ostream & outfile, unsigned long size );
        // decodes "size" bytes in in_buffer and writes them to
        // "outfile"
        // returns 1 on success, 0 on failure; "outfile" must be
        // opened prior to calling this.
    public:
        int EncodeToFile( WString & infilename, ostream & out );
        // encode the data in "infilename" writing it to "out", which
        //  must be opened prior to calling this function
    public:
        void output64chunk( int c1, int c2, int c3, int pads,
                            ostream & out );
        // encode a block of three characters (c1, c2, c3) in base64
        // to "out"
};
```

Figure B-9. *Outgoing_Queue* Class Listing

```
//
// Outgoing_Queue - This class is the queue for outgoing cases.
//
// created by Rahim Pira, October 1997
//

class Outgoing_Queue
{
    public:
        // individual item in mailing queue
        struct QueueItem {
            WString filename;       // temp file for case
            WString date;           // date & time case was composed
            WString subject;        // case subject
            WString to;             // recipients
            WString from;           // sender
            WString cc;             // CC
        };
        // queue of items to be mailed
        QueueItem   queue[MAX_QUEUE_SIZE];
    public:
        WString GetFilename( int index );
        // returns the specified patient case's filename
    public:
        void AddFilename( int index, WString & filename );
        // set the filename for the specified case
    public:
        WString GetDate( int index );
        // returns the specified patient case's date
    public:
        void AddDate( int index, WString & date );
        // set the date for the specified case
    public:
        WString GetCc( int index );
        // returns the specified patient case's CC tag
    public:
        void AddCc( int index, WString & cc );
        // set the CC tag for the specified case
    public:
        WString GetSubject( int index );
        // returns the specified patient case's subject
    public:
        void AddSubject( int index, WString & subject );
        // set the subject for the specified case
    public:
        WString GetTo( int index );
        // returns the specified patient case's TO tag
    public:
        void AddTo( int index, WString & to );
        // set the TO tag for the specified case
    public:
        WString GetFrom( int index );
        // returns the specified patient case's FROM tag
```

```
    public:
        void AddFrom( int index, WString & from );
        // set the FROM tag for the specified case
    public:
        void InitQueue();
        // initialize the queue
    public:
        void RemoveCase( int start, int end );
        // update the queue, removing the item indexed by start
};
```

APPENDIX C

WEBTELEMED RESOURCE LISTING

The following is the code listing of the resource file used by *MHonArc* to generate the patient encounter Web pages.

Figure C–1. *caselist.rc* File Listing

```
<!-- ========================================================================== --
><!--  MHonArc Resource File

       caselist.rc 1.1 06/10/97

       This resource file takes the patient encounters and "webifies" them.  An
       index is generated listing the patient encounters in alphabetical order.
       Each case is listed in the main index and the message components are
       extracted, represented as hyperlinks to the component.  Additionally, an
       icon is placed next to the link to indicate the type of component. The
       message page is formatted in a frame with the component listing in one
       frame and the displaying of the item in the other frame.
  -->

<!-- ========================================================================== -->
<!--    Variable Definitions                                                    -->
<!-- ========================================================================== -->

<!--    Variables for frame names
  -->
<definevar>
FRAME-MAIN
target="MAIN"
</definevar>

<definevar>
FRAME-IDX
target="INDEX"
</definevar>

<!-- ========================================================================== -->
<!--    Derived Files                                                           -->
<!-- ========================================================================== -->

<!--    The following derived file is the main file the message pages will
       link to.  It defines the subframe definitions for the multimedia
       components and the component data itself.  We also derive a blank
       HTML file to be used for the right frame (used to display the multimedia
       components) since a source page is required for a frame.
  -->
<definederived>
frm$MSGNUM$.html
<html>
<head>
<title>Message View</title>
</head>
<frameset cols="35%, 65%">
<frame src="msg$MSGNUM$.html" name="MESSAGE">
<frame src="blank.html" name="MSG_ITEM">
</frameset>
</html>
</definederived>
```

```
<definederived>
blank.html
<html>
</html>
</definederived>

<!-- ========================================================================== -->
<!--    Options                                                                 -->
<!-- ========================================================================== -->

<main>
<sort>
<nothread>
<noreverse>

<!-- ========================================================================== -->
<!--    Main Index Resources                                                    -->
<!-- ========================================================================== -->

<title>
MCLTeleMed Main Case Index
</title>

<!--    LISTBEGIN defines the markup for the start of the message listing.
        Here we simply state when the update was performed.
  -->
<listbegin>
<address>
Last updated: $LOCALDATE$<br>
$NUMOFMSG$ cases<br>
</address>
<p>
Cases listed in chronological order.  Listing format is the following:
<blockquote>
<strong>Subject</strong><code>  </code>
<strong>Date</strong><code>  </code>
<em>From</em>.
</blockquote>
<p>
<hr>
</listbegin>

<!--    LITEMPLATE defines the markup for a message list entry for the main
        index.  The index has a simple listing with the exception that the
        listing is of the "frameized" version of the message.  A medical case
        icon is also added to each list item.

        The $SUBJECTNA$ variable requests the subject of the message without
        the hypertext wrapping.
  -->
<litemplate>
$ICON$ <a $A_NAME$ href="frm$MSGNUM$.html"><strong>$SUBJECTNA$</strong></a>
<strong>$DDMMYY$</strong> <em>$FROMNAME$</em><br>
</litemplate>

<listend>
</listend>

<!-- ========================================================================== -->
<!--    Message Page Resources                                                  -->
<!-- ========================================================================== -->
```

```
<excs>
xref
</excs>

<labelstyles>
-default-
subject:strong
from:strong
</labelstyles>

<fieldstyles>
-default-
subject:strong
from:strong
</fieldstyles>

<!-- ========================================================================== -->
<!--    MIME Resources                                                          -->
<!-- ========================================================================== -->

<!--    Override the filters for the components.  The original MIME resource
        (mhexternal.pl) is used for WAVE files as we cannot redirect the WAVE
        file to a frame.
  -->
<mimefilters override>
image/*:m2telemed'filter:telemed.pl
audio/*:m2h_external'filter:mhexternal.pl
text/*:m2telemed'filter:telemed.pl
video/*:m2telemed'filter:telemed.pl
</mimefilters>

<!--    Specify the filter to use the given name, if any, of the component,
        to place each message in its own subdirectory, and to use the
        defined icons for the various components
  -->
<mimeargs>
m2telemed'filter: usename subdir useicon
m2h_external'filter: usename subdir useicon
</mimeargs>

<!-- ========================================================================== -->
<!--    Icon Resources                                                          -->
<!-- ========================================================================== -->

<!--    Specify icons for message types
  -->
<icons>
image/ms-bmp:icons/camera.gif
image/x-MS-bmp:icons/camera.gif
text/plain:icons/note.gif
video/msvideo:icons/video.gif
audio/wav:icons/tape.gif
multipart/mixed:icons/case.gif
</icons>
```

APPENDIX D

MCLTELEMED CLASS LISTING

Figure D–1. *Connection_Info* Class Listing

```
//
// Connection_Info - A class storing the logon information which is //
                passed between forms.
//
// created by Rahim Pira, April 1997
//

class Connection_Info
{
    public:
        class Station_Info_Item : public WObject
        {
            public:
                WString prefix;
                WString stationName;
        };

        WString station,
                recordsDir,
                recordsPath,
                filePrefix,
                centralPhone,
                centralIPAddress;
        WBool   hostOrClient;       // TRUE if host, FALSE if client
        WBool   connected;          // TRUE if connected to another
                                    // station, FALSE otherwise
        WBool   waitAfterSend;      // TRUE if should sleep after
                                    // each send, FALSE otherwise
        int     sendFlag;           // setting to be used for
                                    // MCLNet::Send()

        // list of information items for each station in the system
        WVector<Station_Info_Item> stationsInfo;

    public:
        Connection_Info();
    public:
        ~Connection_Info();
    public:
        WBool AddStationInfo( WString & prefix,
                              WString & stationName );
        // add a station to the list
        // returns TRUE on success or FALSE on fail
    public:
        WString GetStationName( WString & prefix );
        // returns the station name corresponding to the prefix
};
```

Figure D–2. *Station_List* Class Listing

```
//
//  Station_List - Listing of all patients and their component
//                 filenames.
//
//  Created by Rahim Pira
//

class Station_List
{
    public:
       // structure to store information on patient record components
        class Component_Item : public Wobject
        {
            public:
                WString componentFile;  // name of component
                WBool updated;          // new component
                WBool markForSend;      // mark component for sending
                WBool checked;      // already checked this component
                WString createTime; // the creation date & time of
                                    // the component
        };

       // structure to store information on patient visit (encounter)
        class Visit_Item : public Wobject
        {
            public:
                // list of record items
                WVector<Component_Item> components;
        };

        // structure to store patient information
        class Item : public WObject
        {
            public:
                WString patientId;   // patient's ID
                WString patientName;  // patient's name
                WBool updated;       // patient is new or has new
                                     // component(s)
                WBool markForSend;   // mark patient for sending
                                     // (implies at least one
                                     // component)
                WBool checkPatient;  // should check this patient
                WBool hasInfoFile;   // whether info file exists for
                                     // patient
                WString infoFileTime;   // timestamp for patient
                                        // information file
                WBool sendInfo;     // whether should send
                                    // patient's background info
                int folder;   // folder in which patient is listed
                              // (0 for new, 1 for old)
                int rowIndex;  // row number of folder
                WString newestTime;   // the date & time of most
                                      // recently received component
                WVector<Visit_Item> visits;     // list of encounters
        };
```

```
        WVector<Item> list;  // list for station of all patients
public:
        Station_List();
    public:
        ~Station_List();
    public:
        int AddPatient( WString & patient );
        // add a patient to the list; returns patient index
    public:
        int AddComponent( int index, int visitIdx,
                          WString & filename, WString & createTime );
        // add a component to the patient; returns component index
    public:
        int GetPatientCount();
        // get the number of patients
    public:
        int GetComponentCount( int index, int visitIdx );
        // get the number of components for the patient index
    public:
        WString GetComponentName( int patientIdx, int visitIdx,
                                  int componentIdx );
        // get the name of the component (as per componentIdx) for
        // the patient index
    public:
        void DeleteAll();
        // clear the list (delete all patients and components)
    public:
        WString GetPatient( int index );
        // get the patient ID as specified by the index
    public:
        int GetPatientIndex( WString & patient );
        // get the index of the patient as per the patient ID; if not
        // found, -1 is returned
    public:
        int GetComponentIndex( int patientIdx, int visitIdx,
                               WString & component );
        // get the index of the component as per the component name for
the patient index
    public:
        int AddNewPatient( WString & patient );
        // add a patient to the list as new; returns patient index
    public:
        int AddNewComponent( int index, int visitIdx,
                             WString & filename );
        // add a component to the patient as new and mark the patient
        // as updated returns the index of the newly added component
    public:
        WBool NewPatient( int index );
        // is the patient new?
    public:
        WBool NewComponent( int patientIdx, int visitIdx,
                            int componentIdx );
        // is the component new?
```

```
public:
    void MarkPatient( int index, WBool value = true );
    // mark the patient for sending (some or all components need
    // to be sent)

public:
    WBool SendPatient( int index );
    // do we need to send the patient?
public:
    void MarkComponent( int patientIdx, int visitIdx,
                        int componentIdx, WBool value = true );
    // mark the component for sending
public:
    WBool SendComponent( int patientIdx, int visitIdx,
                         int componentIdx );
    // do we need to send the component?
public:
    void MarkComponentChecked( int patientIdx, int visitIdx,
                               int componentIdx );
    // mark the component as checked
public:
    WBool CheckedComponent( int patientIdx, int visitIdx,
                            int componentIdx );
    // return whether the component has been checked
public:
    void AddPatientName( int index, WString & patientName );
    // set the patient's name
public:
    WString GetPatientName( int index );
    // retrieve the patient's name
public:
    void SetCheckPatient( int index, WBool value = true );
    // set the checkPatient flag for the specifed patient
public:
    WBool GetCheckPatient( int index );
    // should we check for this patient (and its components)
public:
    void SetFolder( int patientIndex, int folder, int row );
    // set the folder and row number where the patient is listed
public:
    int GetFolder( int patientIndex );
    // return the folder in which the patient is listed - 0 for
    // new, 1 for old or -1 if not assigned
public:
    int GetRowNumber( int patientIndex );
    // return the row number listing the patient
public:
    WString GetComponentCreateTime( int patientIndex,
                                    int visitIdx,
                                    int componentIdx );
    // get the creation time of the specified component
public:
    WString GetRecentTime( int index );
    // get the timestamp of the most recently received component
```

```
    public:
        void SetRecentTime( int index, WString & recent );
        // set the time of the most recently received component
        // for the patient
    public:
        void SetComponentTime( int patientIdx, int visitIdx,
                               int componentIdx,
                               WString & createTime );
        // set the creation timestamp for the component
    public:
        int AddVisit( int patientIndex );
        // add a new encounter to the patient
        // returns the visit number
    public:
        int GetVisitCount( int patientIndex );
        // returns the number of encounters for the patient
    public:
        void RemovePatient( int patientIndex );
        // remove the specified patient entry from the list
    public:
        int GetPatientIndexFromName( WString & patientName );
        // returns the patient index given the name
    public:
        void DetermineRecentFile( WString & stationPrefix );
        // determines the most recent patient item not created at
        // this site
    public:
        WBool SendPatientInfo( int patientIndex );
        // returns TRUE if patient's info needs to be sent,
        // FALSE otherwise
    public:
        void SetSendInfo( int patientIndex, WBool value = true );
        // set the sendInfo flag for the patient
    public:
        void SetHasInfoFile( int patientIndex, WBool value );
        // set the hasInfoFile flag for the patient
    public:
        WBool GetHasInfoFile( int patientIndex );
        // get the hasInfoFile flag for the patient
    public:
        void SetInfoFileTime( int patientIndex,
                              WString & timeStamp );
        // set the timestamp for the patient info file
    public:
        WString GetInfoFileTime( int patientIndex );
        // returns the timestamp for the patient information file
};
```

Figure D–3. *Main_Form* Class Listing

```
//
// Main_Form - As the name suggests, this is the main from which the
//             user can start the MCLTeleMed application.  The form
//             simply displays a toolbar with buttons that starts up
//             the application as a thread.  The user is also able to
//             bring up help, quit and setup a call.
//
// Created by Rahim Pira, February 1998
//

class Main_Form
{
      protected:
              TeleMed              *telemedForm;
              MCLNet               *mclnet;
              Connection_Info      info;
              Station_List         stationList;
              WToolbarItemHandle   updateHndl,
                                   connectHndl,
                                   disconnectHndl,
                                   telemedHndl,
                                   quitAllHndl,
                                   helpHndl;
              WBool                telemedRunning,
                                   rolesReversed,
                                   immediate,
                                   synchInProgress;
              int                  commStatus;
              int                  numConnections;

      // structure to store info on items queued for immediate send
      class Immediate_Item
      {
            public:
                int patientIndex;
                int visitIndex;
                int componentIndex;
      };

      // queue for immediate send
      WVector<Immediate_Item> immediateQueue;
    public:
        void Logon();
        // "Logon" button has been pressed on logon form
    public:
        void CloseLogon( WBool reset = false );
        // destroy Logon form; if reset is TRUE, recreate the
        // MCLNet object
    public:
        void TerminateTelemed();
        // terminate the MCLTeleMed application (set it invisible)
    public:
        WBool LoadConfig();
        // load the station's configuration file
```

```
    public:
        WBool GetStationList();
        // load all patient data into structure
        // returns FALSE if there was a problem
    public:
        friend int ReceiveCallback( MCLData * data, void * my_data );
        // receive callback for patient synchronization
    public:
        friend int SendCallback( MCLData * data, void * my_data );
        // send callback for patient synchronization
    public:
        void ImmediateSynch();
        // perform immediate synchronization of an updated record
    public:
        friend int SystemCallback( int eventCode, DPUSER userInfo,
                                   void * myData );
        // MCLNet system callback
    public:
        void StartSynch();
        // disable toolbar buttons
    public:
        void EndSynchMessage();
        // display message for completion of record synchronization
    public:
        void EndSynch();
        // re-enable toolbar buttons
    public:
        void UpdatePatientFile();
        // update the patient file for all new/updated patients
    public:
        void UpdateStationList();
        // update the station file
    public:
        WBool SynchInProgress();
        // return whether record synchronization is in progress
    public:
        void AddItemtoQueue( int patient, int visit, int component );
        // add the item to the Immediate Send Queue
    public:
        void CheckImmediateQueue();
        // check the immediate queue and send any items in the queue
    public:
        friend int ConnectCallback( MCLData *  data,
                                    void * my_data );
        // callback for MCLNet to determine if connection is
        // refused or not
    public:
        WBool ResetStationList();
        // resets each patient to "unchecked"
};
```

Figure D–4. *TeleMed* Class Listing

```
//
// TeleMed - The lead form for the MCLTeleMed application, consisting
//           of two folders - "Previous" and "New/Updated" patients.
//
// Created by Rahim Pira, November 1997
//

class TeleMed
{
    protected:
        Encounter_Viewer encounterViewer;  // patient record viewer
        Connection_Info  *info;       // logon information
        Station_List *stationList;    // list of patients & components
                                      // for the station
        New_Patient   newPatientInfo;  // from to add new patients
        MCLNet      *mclnet;   // ptr to MCLNet object
        WWindow     *parentForm; // the parent window
        WBool       addingPatient;  // are we adding a patient?
        WString     patientDisplayed;  // ID patient of patient being
                                       // viewed; "0" of none
    public:
        void InitializeTeleMed();
        // perform some initialization once the user logs on
    private:
        WBool OkayToQuit();
        // determine if it's okay to quit the application
    public:
        void UpdateStationList();
        // update the list of patients and components for the station
    public:
        void UpdatePatientFile();
        // update patient file for all new/updated patients
    public:
        void UpdateFolders( int patientIndex );
        // update the folders moving the specified patient from the
        // "Old" folder to the "New/Updated" folder
    public:
        void ProcessUpdate( int patientIndex );
        // patient record update has been completed; if the record is
        // currently being viewed, update the browser; otherwise,
        // just update the folders
    public:
        void NewPatient( int patientIndex );
        // new patient has been added to the database
    public:
        void SetInfo( Connection_Info *infoPtr );
        // set the pointer to the Connection_Info structure
    public:
        void SetNet( MCLNet *netPtr );
        // set the pointer to the MCLNet object
    public:
        void MyCloseEvent();
        // as the name suggests, this is my own Close event handler
```

```
    public:
        void SetList( Station_List * listPtr );
        // set the pointer to the Station_List object
    public:
        void ImmediateSynch();
        // function for immediate synchronization of patient records
    public:
        WBool SynchInProgress();
        // returns whether record synchronization is in progress
    public:
        void SetParentForm( WWindow * window );
        // set the pointer to the parent form
    public:
        void StartSynch();
        // synchronization starting; change cursor to Wait
    public:
        void EndSynch();
        // synchronization ended; revert cursor to Pointer
    public:
        WBool AddingNewPatient();
        // returns TRUE if new patient being added to database, FALSE
        // otherwise
    public:
        void SetPatientDisplayed( WString & patientId );
        // set the ID of patient being viewed
    public:
        WString GetPatientDisplayed();
        // get the ID of the patient being viewed
    public:
        void SetAddingPatient( WBool value );
        // set the value of the "addingPatient" variable
    public:
        void ClosePictureViewers();
        // close all Bitmap_Viewers
};
```

Figure D–5. *In_mailbox* Class Listing

```
//
// in_mailbox -  This tab folder stores the newly received or updated
//               patient records.
//
// Created by Rahim Pira, December 1997
//

class in_mailbox
{
    public:
        int selectedMessage,     // the message selected to be viewed
            displayed_case;      // the case currently being displayed
        WCursor oldcursor, hourglass;
    protected:
        static Encounter_Viewer *currentCase;
        static Connection_Info  *info;
        static Station_List     *patients;
        static TeleMed          *parent;
    public:
        void SetInfo( Connection_Info * info_ptr );
        // set pointer to Connection_Info structure
    public:
        void DisplayRecord( int msgNumber, WString & key );
        // display the selected patient case (where 'key' is the
        // patient ID)
    public:
        void GetRecords();
        // retrieve all records (updated) to be displayed in the
        // folder
    public:
        void SetList( Station_List * listPtr );
        // set the Station_List pointer
    public:
        void SetViewer( Encounter_Viewer * viewerPtr );
        // set the Encounter_Viewer pointer
    public:
        void AddPatient( int patientIndex );
        // add this patient to the "New/Updated" folder listing
    public:
        int GetNumRecords();
        // return the number of records in the folder
    public:
        void MyCloseEvent();
        // to be called on closing of form
    public:
        void StartSynch();
        // synchronization started; change cursor to Wait
    public:
        void EndSynch();
        // synchronization has ended; revert cursor to Pointer
};
```

Figure D–6. *out_mailbox* Class Listing

```
//
// out_mailbox - Tab folder to store "old" (i.e. unupdated) patient
//               records.
//
// Created by Rahim Pira, December 1997
//

class out_mailbox
{
    public:
        int selectedRecord,     // the message selected to be viewed
            displayedRecord;    // the case currently being displayed
            WCursor normal, hourglass;
    protected:
        static Encounter_Viewer *patientRecord;
        static Connection_Info  *info;
        static Station_List     *patients;
        static TeleMed          *parent;
    public:
        void SetInfo( Connection_Info * info_ptr );
        // set pointer to Connection_Info structure
    public:
        void GetRecords();
        // retrieve all cases to be displayed in the folder
    public:
        void SetList( Station_List * list_ptr );
        // set the Station_List pointer
    public:
        void DisplayRecord( int recIndex, WString & key );
        // display the selected patient case where 'key' is the
        // patient ID)
    public:
        void SetViewer( Encounter_Viewer * viewerPtr );
        // set the Encounter_Viewer pointer
    public:
        void RemovePatient( int rowNumber );
        // remove the patient in the selected row of the folder
    public:
        void MyCloseEvent();
        // to be called on closing of form
    public:
        void StartSynch();
        // synchronization has started; change cursor to Wait
    public:
        void EndSynch();
        // synchronization has ended; revert cursor to Pointer
};
```

Figure D–7. *Logon_Form* Class Listing

```
//
// Logon_Form -  This class simply presents the user with the logon
//               window and collects the information, storing them
//               into the Connection_Info structure.
//
// Created by Rahim Pira
//

class Logon_Form
{
    protected:
        Connection_Info *info;
        MCLNet *net;
    public:
        void SetInfo( Connection_Info *info_ptr);
        // set pointer to Connection_Info data structure
    public:
        void SetNet( MCLNet * mclnet_ptr );
        // set pointer to MCLNet object
    public:
        WBool HostSession();
        // host a session using the selected service
    public:
        WBool ConnectSession();
        // connect to the central station using the selected session
    public:
        void CloseLogonForm( WBool reset = false );
        // close the Logon form and returns control to the
        // application; if reset is TRUE, the MCLNet object
        // is recreated
};
```

Figure D–8. *Encounter_Viewer* Class Listing

```
//
// Encounter_Viewer -   Used for viewing/creating objects in the
//                      patient's record.  Newly added objects are
//                      sent for immediate updating.  As well, the
//                      browser is updated to reflect any new objects
//                      received.
//
// Created by Rahim Pira, April 1997
//

class Encounter_Viewer
{
    protected:
        static Connection_Info  *info;
        static Station_List     *stationList;

        WBool available,  // flag for availability of viewer
              noteVisible,      // visibility flag for Note_Viewer
              addingText,       // flag for adding textual component
              newNote;          // new note?

        // these are flags used during the composition of a new
        // case for the various multimedia components; in case the
        // user switches from button to another button without
        // stopping these actions, some way to track them is needed
        WBool videoPreview,
              picturePreview,
              videoRecording,
              audioRecording;
        WImageList iconlist;    // icons for list view
        int hscrollPosition,
            vscrollPosition;
        int n_items,
            current_item,
            patientIndex,
            numEncounters,
            lastVisit;
        int biasHour, biasMinute;
        int listEntries;    // number of patient items & visit labels
        int penRed, penGreen, penBlue;
        WString patientId, patientName;
        MCLBitmap *bitmap;
        MCLMatrix redMatrix, blueMatrix, greenMatrix;
        MCLVideo video_camera;
        WCursor normal, hourglass;
        UINT wDeviceID     // the ID for the waveform-audio device
        WBool recordTimerOn;   // flag indicating whether recording
                               // timer is still going
        WBool itemAdded;
        WWindow *parentForm;
        WToolTip *lvToolTip;    // ToolTip for the ListView
        WPoint lvMousePos;      // mouse position info
        WRange listScrollRange;
        WBool quitting;
```

```
    // structure to store information on picture viewers
    class Bitmap_Viewer_Item
    {
        public:
            // pointer to window displaying picture
            Bitmap_Viewer *viewer;
            // window handle of the picture viewer
            WWindowHandle handle;
            // item number of the picture (to prevent multiple
            // instances of same image)
            long itemNumber;
    };
    // viewer windows for still images
    WVector<Bitmap_Viewer_Item> pictureViewer;

    // structure to store information on media items
    class Record_Component_Item
    {
        public:
            int protect;  // delete item after closing window?
            int newItem;        // whether item is new or not
            int type;           // type of item
            int visit;          // encounter that item belongs to
            int index;          // index of item in scroll list
            WString filename;   // filename of item
    };
    // list of items displayed in Encounter_Viewer
    WVector<Record_Component_Item> itemVector;
public:
    WBool IsAvailable();
    // whether the Encounter_Viewer is available i.e. whether or
    // not it's being used to display a patient record
public:
    void SetInfo( Connection_Info *info_ptr );
    // set the pointer to the Connection_Info structure
public:
    int ShowItem( WLong listIndex );
    // displays the contents of item specified by 'listIndex' in
    // the appropriate window
public:
    void HideBitmap();
    // hides bitmap window and associated controls
public:
    void HideMediaBox();
    // hides the media box and closes/saves associated data
public:
    void ShowMediaBox();
    // shows the media box
public:
    void SaveNoteItem();
    // save the note item to text file
public:
    void ShowVideoBox();
    // show the MCLVideo box and its controls
```

```
public:
    void HideVideoBox();
    // hide the MCLVideo box and its controls
public:
    void ShowBitmapControls();
    // show the MCLBitmap controls
public:
    void HideBitmapControls();
    // hide the MCLBitmap controls
public:
    void SetStatusBar( WString text );
    // set the text of the status bar
public:
    void ShowAudioBox();
    // show the controls for recording audio
public:
    void HideAudioBox();
    // hide the controls for recording audio
public:
    void StopOtherRecordings();
    // stop the recording of other components
public:
    void ShowNote();
    // show the notepad
public:
    void HideNote();
    // hide the notepad
private:
    void DeleteItem( int item );
    // delete the selected item
public:
    WBool CompressVideo( int itemNumber );
    // compress the video using Microsoft's Video 1 codec
public:
    void AddAttachment( WString & attachFile,
                        int attachType=-1 );
    // add an attachment given its filename & component type
public:
    void SetNoteTitle();
    // set the title of the note window
public:
    void SetPatientId( WString & patientNumber );
    // set the patient ID (in the viewer only)
public:
    void UpdateStationList();
    // update the patient data in Station_List object
public:
    void SetPatientName( WString & patient_name );
    // set the patient name (in the viewer only)
public:
    void MyCloseEvent();
    // my method for the Close event handler
public:
    void SetStationList( Station_List *stationListPtr );
    // set the pointer to the Station_List used by the
    // application
```

```
public:
    void ImmediateSynch();
    // update the patient file and send new object
public:
    int GetPatientIndex();
    // returns the index of the patient being displayed or -1 if
    // none is displayed
public:
    void UpdateBrowser();
    // the patient record being viewed has been updated; update
    // the browser to reflect the change (i.e. Additional
    // components)
public:
    DWORD OpenAudioDevice();
    // open the waveform-audio device for audio recording
public:
    DWORD BeginAudioRecording();
    // start audio recording
public:
    void CloseAudioDevice();
    // closes the waveform-audio device
public:
    DWORD StopAudioRecording();
    // stop audio recording
public:
    DWORD SaveAudioRecording( WString & filename );
    // save the recorded audio to filename specified
public:
    void SetParentForm( WWindow * window );
    // set the pointer to the parent form
public:
    int CreatePictureViewer( int itemNumber);
    // create a new picture viewer
    // returns index of Bitmap_Viewer
public:
    void DeletePictureViewer( int index );
    // destroy the picture viewer indicated by the index
public:
    int GetPictureViewer( WWindowHandle hndl );
    // get the index of the picture viewer from its handle
public:
    WString GenerateFileName();
    // return the path for the component to be saved
public:
    void InsertItem( int listIndex, int itemType );
    // insert the item at the top of the List View
public:
    int NewRecordItem( int position=-1 );
    // add a new item to the vector of items at the specified
    // position; returns index of new item
public:
    void SetupToolTips();
    // sets up the tooltips for the ListView
public:
    WBool SaveDrawObject( MCLBitmap * image );
    // saves a bitmap with annotations to file
```

```
    public:
        void ConvertUTCtoLocal( int *year, int *month, int *day,
                                int *hour, int *minute );
        // converts the UTC time to local time
    public:
        WBool ReadPatientInfo();
        // read the patient's info file and transfer the data to the
        // Encounter_Viewer
    public:
        void StartSynch();
        // starting synchronization; change cursor to Wait
    public:
        void EndSynch();
        // synchronization ended; revert cursor to pointer
    public:
        void CheckPatientFields();
        // checks patient fields for changes and saves patient info
        // file if any changes made, updating file creation time
};
```

Figure D–9. *Bitmap_Viewer* Class Listing

```
//
// Bitmap_Viewer - Used for viewing pictures in a patient record (the
//                 Encounter_Viewer).  Each picture is presented in a
//                 separate window.
//
// Created by Rahim Pira, May 1998
//

class Bitmap_Viewer
{
    protected:
        WWindow *parentForm;
        int hscrollPos, vscrollPos;
        WBool drawLines;
        WBool linesDrawn;
        WBool penColourSet;
        MCLDraw *lines;
    public:
        void SetParentForm( WWindow * window );
        // set the pointer to the parent window
    public:
        WBool LoadImage( WString & filename );
        // load image file
    public:
        void CopyImage( MCLBitmap & copy );
        // copy an MCLBitmap object
    public:
        void DisplayImage( long height, long width );
        // display image, resize form and picture box
    public:
        int SetPenColour( int red = 5, int green = 255,
                          int blue = 255 );
        // sets the colour of the pen (by default cyan is set)
    public:
        void MyCloseEvent();
        // overrides Close event handler
};
```

Figure D–10. *Note_Viewer* Class Listing

```
//
// Note_viewer -  The note in which text is displayed during browsing
//                and entered during composition.
//

class Note_Viewer
{
    protected:
        Encounter_Viewer *encounterViewer;
    public:
        void AddText( const WString & text );
        // puts text in text window.
    public:
        void NewText( const WString & text );
        // replaces text box contents with "text"
    public:
        void SetReadOnly(  WBool readOnly );
        // set the read-only property of the text box
    public:
        void AtTop();
        // set the cursor at the beginning of the text box & clear
        // the undo buffer
    public:
        WString GetMessage();
        // get the text entered into the text box
    public:
        WBool IsEmpty();
        // whether the note is empty or not
    public:
        void MyCloseEvent();
        // overrides the Close event handler
    public:
        void SetEncounterViewer( WObject * viewer );
        // set the pointer to the Encounter_Viewer
};
```

Figure D–11. *New_Patient* Class Listing

```
//
// New_Patient - Class to retrieve information on new patient being
//               added to patient database for the station.
//
// Created by Rahim Pira, January 1998
//

class New_Patient
{

    protected:
        Station_List *stationList;
        Connection_Info *info;
        WBool sexFlag;  // flag for patient's sex - TRUE for male,
                        // FALSE for female
    public:
        void SetList( Station_List * listPtr );
        // set the pointer to the Station_List structure
    public:
        void NewPatient();
        // user wishes to add patient to database; display form and
        // clear all fields
    public:
        void MyCloseEvent();
        // override the default Close event handler
    public:
        void SetInfo( Connection_Info * infoPtr );
        // set the pointer to the Connection_Info object
};
```

APPENDIX E

MAILTELEMED SETUP GUIDE

Installation

To install the *MailTeleMed* application, simply place the executable and the *MCLGallery* DLL in a working directory (e.g. C:\MailTeleMed). To activate the application, simply double-click the *MailTeleMed* executable.

Operation

Once the application has been started, you are prompted for information – your name, email address, userid, and password to access your mail account (see **Figure E–1**). Pressing the "**Settings**" button reveals additional options available; primarily the means used to retrieve the mail – via network or via telephone. If the network is used, the name of the POP and SMTP servers used to send and retrieve mail are required. If the telephone is used, the name of the mailbox containing the mail messages and the phone number used to connect to the mailbox are required. The last option is the working directory for *MailTeleMed*. This directory is used to temporarily store all incoming and outgoing mail messages and is initially set to "c:\temp". If you wish to use another directory, this value needs to be altered. Once all the information is entered, press the "**Log On**" button to connect to the mailbox and being retrieval of al the mail messages. The application can also be terminated by pressing the "**Cancel**" button.

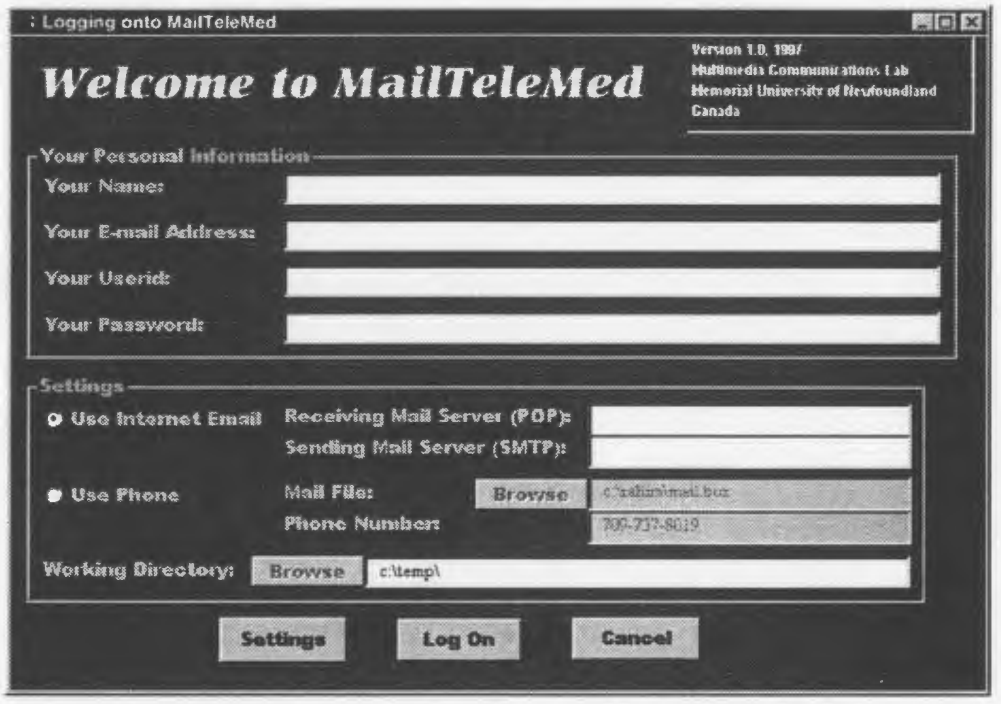

Figure E–1. Logon Window for *MailTeleMed*

Once you have logged on, the items in your mailbox are retrieved and displayed in a folder with the date, sender and subject listed (see **Figure E–2**). Clicking on a case will display it in a window (see **Figure E–3**) where the media items in the patient encounter are listed on the side. Selecting an item will display the media item. From the main folder, a new patient encounter can be created or a previously saved encounter can be loaded. If a new patient encounter is to be created, an empty window is displayed with fields to be filled and a toolbar for the insertion of the media items (see **Figure E–4**).

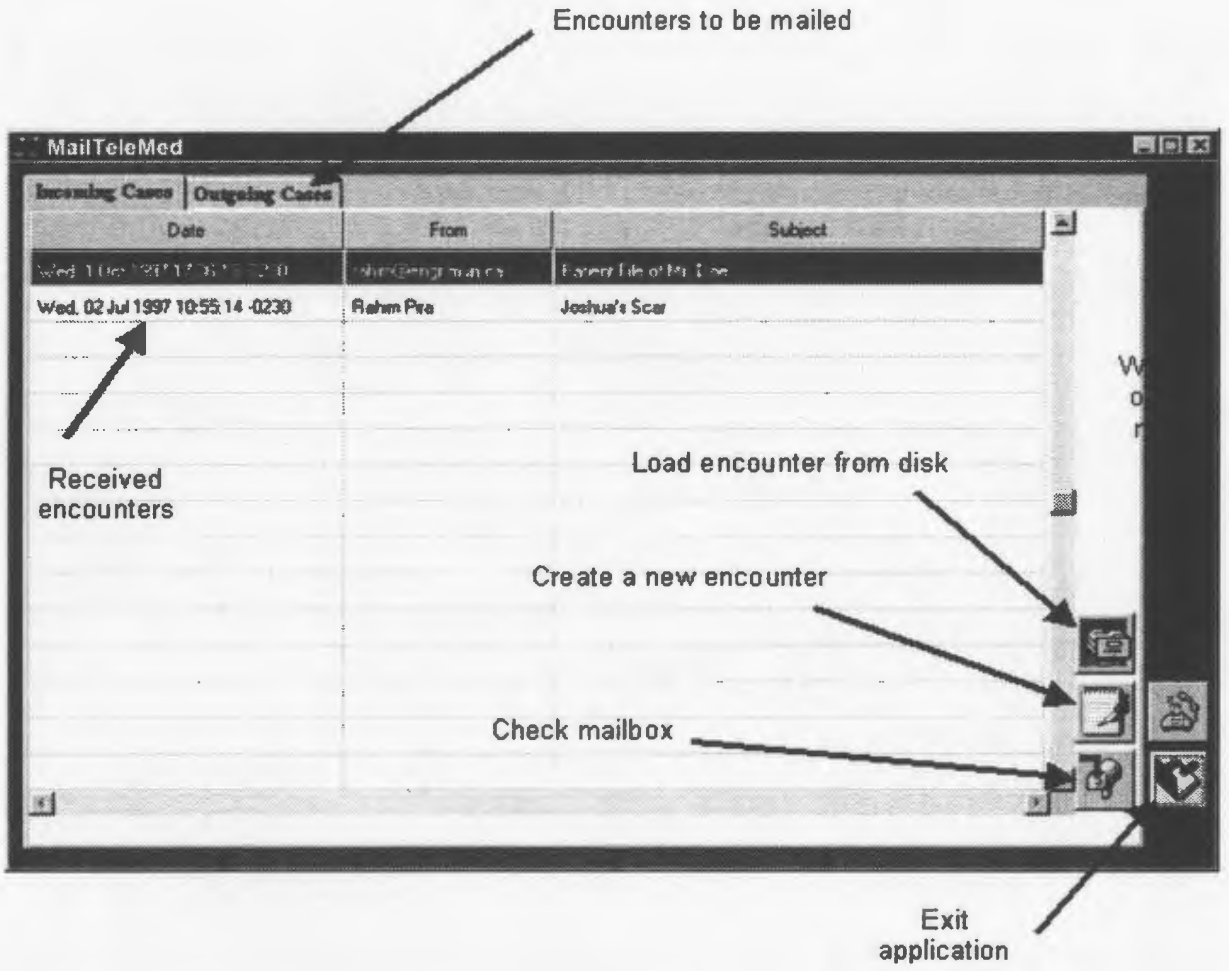

Figure E–2. Retrieval of Patient Encounters

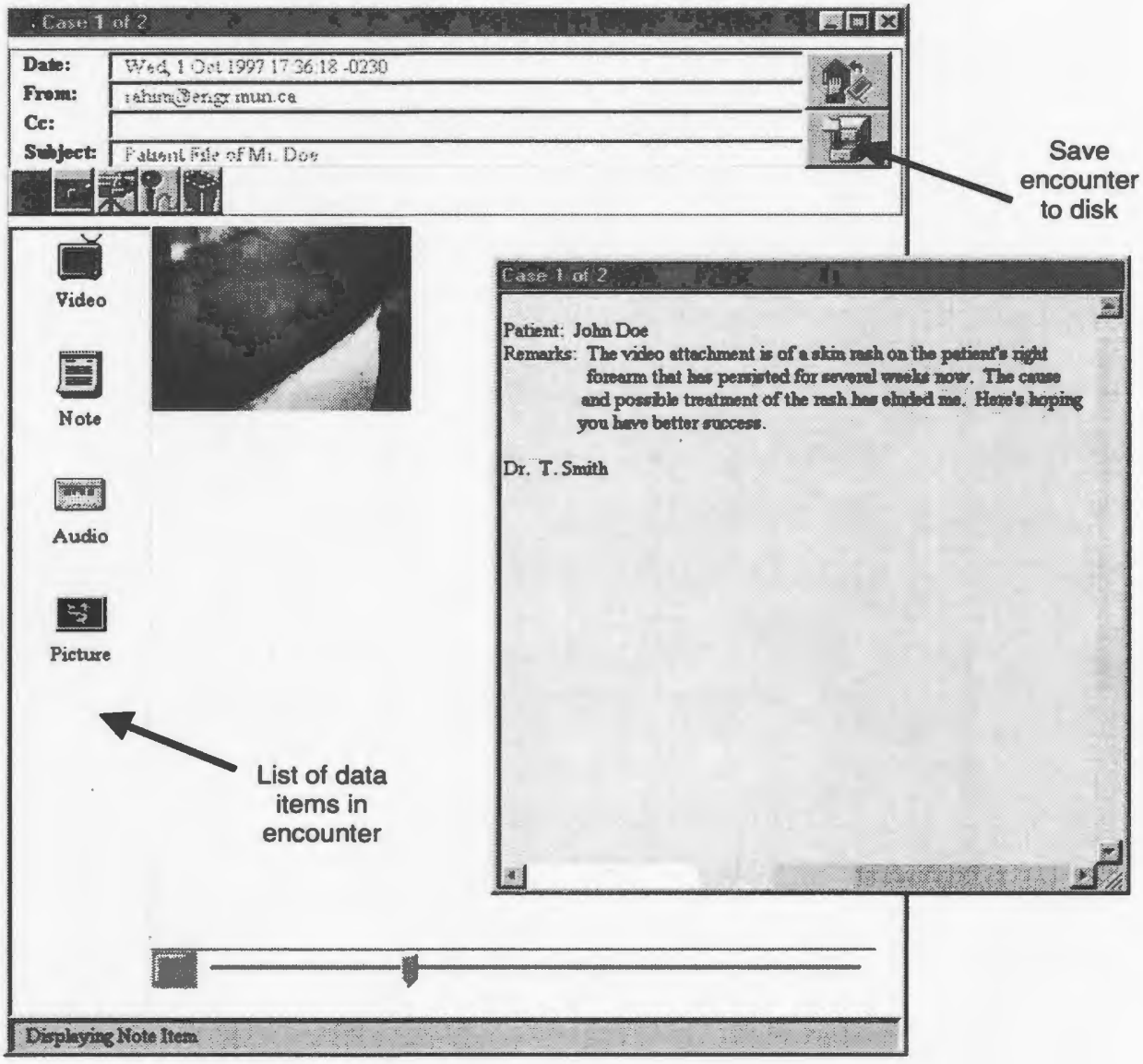

Figure E–3. Viewing of Patient Encounter

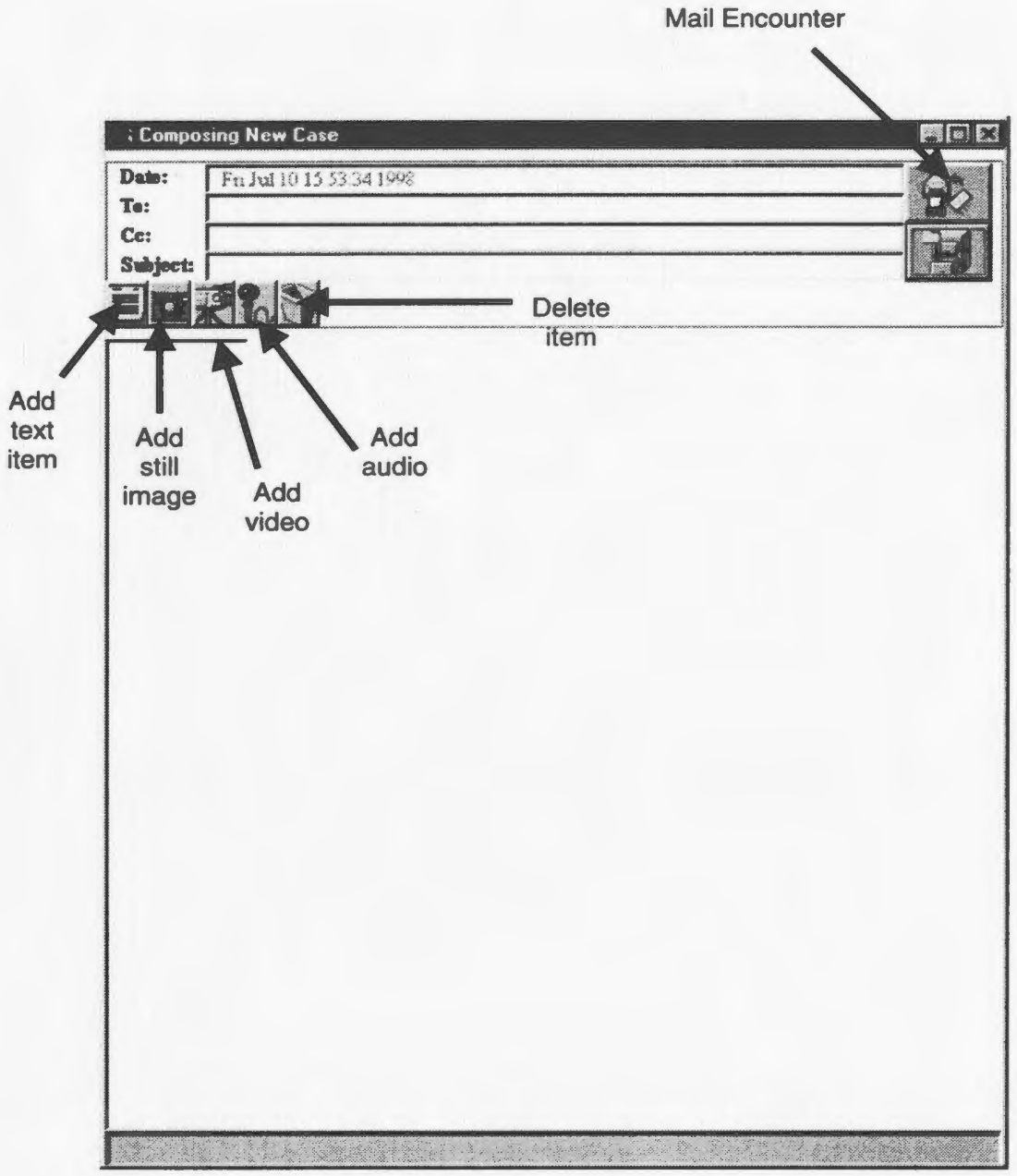

Figure E–4. Creation of New Patient Encounter

APPENDIX F

WEBTELEMED SETUP GUIDE

Installation

Installation of the *WebTeleMed* system involves the creation of a user account for the system, the installation of *MHonArc* application, and the creation of scripts to use the conversion resource file when encounters are received by the mail account for the system. These steps require root level access to the system and some familiarity with system administration is required. The steps are as follows to install *WebTeleMed*:

1. Install the *MHonArc* application and its libraries with access to the systems Perl v5.0 libraries. If Perl v5.0 is not installed on the system, this must be done prior to installing *MHonArc*.

2. Create an account for the *WebTeleMed* system giving it any name. Modify the shell script for this account placing the *MHonArc* path in the account's search path. Create a web directory for this account.

3. Modify the account's mail script to execute *MHonArc* whenever patient encounters are retrieved outputting the created Web pages to the web directory created above.

APPENDIX G

MCLTELEMED SETUP GUIDE

System Requirements

MCLTeleMed Suite will run on any PC running Windows® 95 or 98 but not on Windows NT as it makes use of *DirectX 5.0* which is not available for the Windows NT operating system. A camera is required in order to capture images and videos as well as a microphone for recording voice data. It is advised that the PC have at least 16 MB of memory but 32 MB is recommended.

Installation

To install the program, simply run the installation program named **"Setup.exe"** and follow the instructions for installing the application. This setup program will create the program group for the *MCLTeleMed Suite* application in your Windows **Start** menu. Once this has been completed, some configuration is required prior to running the application. Once installation is complete, the *MCLTeleMed Configuration* program will execute and will require the following information:

1. **Station Name:** This is the name of the station and is used in the identification of the various clinics.

2. **Station Type:** Two types of stations or clinics exist in the system – a remote nursing clinic or a central physician clinic. Simply select the button that best describes your site.

3. **Medical Records Directory**: This is the path of the directory where all the patient records will be stored. The default value for this is "*C:\MCLTeleMed Records*".

4. **Central Phone Number**: If your site is a remote clinic site, then you will need to connect to the central site for the transfer of patient records. This can be performed using either a modem or the Internet. The value of this field is the modem phone number of the central site to which this site will connect.

5. **Central IP Address**: Connections to the central site can also be made using the Internet. In order to do this, the IP (Internet Protocol) address of the computer at the central site to which this system will connect, is required. See your network administrator to determine this value.

Special Setup for Central Clinic

Some special setting up of the application is required at the host site before the application can be used. After the installation of the system, information regarding the other sites (the remote clinics) involved in the system is required. In the directory in which *MCLTeleMed Suite* was installed (by default "**C:\Program Files\MCLTeleMed Suite**") there is a file named "**stations.lst**" (see **Figure G–1**). This file contains a list of all the medical clinics involved in the system consisting of the station name and its 3-letter prefix. Open this file using **Notepad** and enter the prefixes and station names for all the stations involved, in the following format:

3-letter prefix:station name

For example, if the stations involved in the system were Goose Bay (gob), Black Tickle (btk) and Nain (nai), then the "**stations.lst**" file will look like this:

gob:Goose Bay

btk:Black Tickle

nai:Nain

This editing of the file needs to be done when the application is first installed and whenever a new clinic is added to the system.

```
gob:Goose Bay
dhu:Black Tickle
btk:Nain
```

Figure G–1. *stations.lst* File Listing

DirectX Installation

In order to use the *MCLTeleMed Suite* application, *DirectX*® is required. The set-up file for *DirectX 6.0* is included with the installation disk and is titled "**dx6eng.exe**". If *DirectX 5.0* is installed on your machine, it is recommended that you still install *DirectX 6.0* as improvements have been made to the drivers and to ensure that your system is ready for any future enhancements to the *MCLTeleMed Suite* application.

Operation

Once the application has been started a number of choices are available. If the site is a remote site, a connection can be established to the central site (see **Section 4.4.7** for instructions on how to do this). For both sites, old patient records can be viewed and new ones can be made. To view an existing record simple click on the row in the folder. To create a new patient record, press the "Add Patient" button and a form is displayed requiring information on the patient. To add media items to a patient record, either new or existing, refer to **Section 4.5**.