TeleMedMail: A Low-Cost Store-and-Forward Telemedicine System for Use in Developing Countries

by

Darius G. Jazayeri

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

March 19, 2002

Author_____

Department of Electrical Engineering and Computer Science
March 19, 2002

Certified by_____

Hamish Fraser
Thesis Supervisor

Accepted by_____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

TeleMedMail: A Low-Cost Store-and-Forward Telemedicine System for Use in Developing Countries

by

Darius G. Jazayeri

Submitted to the
Department of Electrical Engineering and Computer Science

March 19, 2002

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

My thesis proposes an email-based store-and-forward telemedicine system for use in resource-poor, poorly networked developing countries. The system is designed for use in structured network where many remote locations refer cases to few central hospitals with specialist expertise, but it is flexible enough to be used in many other scenarios as well. The system consists of four components: a Java client applications that sends digital camera images and structured XML text as email, a server component which processes referral emails, and enters then into an Electronic Medical Record database, a JavaServer Pages based web interface to that database, and a Java applet for viewing images.

Thesis Supervisor: Hamish Fraser, MBChB, MSc, MRCP
Title: Research Affiliate, MIT Laboratory for Computer Science,
Director of Informatics and Telemedicine,
Program in Infectious Disease and Social Change, Harvard Medical School

Acknowledgements

I would like to acknowledge Lech Banach and Donnie McGrath for their pioneering work in this field, without which I would never have been inspired to pursue the TeleMedMail concept. I would like to thank Professor Peter Szolovitz, Sean Doyle, Agnieszka Szot, and Eduardo Marquez for their advice on the security, computer imaging, radiological, and user-interface aspects of the system.

Part of the funding for this project was provided by the Bill and Melinda Gates Foundation, through a grant to the Program in Infectious Disease and Social Change, Harvard Medical School, and the Fogarty International Center of the NIH through an ITMI grant to the Medical School of the University of Natal, Durban, South Africa, and Tufts University.

Above all, I would like to thank my thesis advisor Hamish Fraser for introducing me to this subject matter, and for the many hours of guidance and support he has given me throughout the last two years.

The idea for this project came from Hamish Fraser and a number of South Africa-based doctors. I did most of the system design and all of the implementation.

Table of Contents

1 Introduction and Previous Work 9

2 Design Considerations 12

2.1 Security 12

2.2 Low Cost and Connectivity 13

2.3 Ease-of-use & Automation 13

2.4 Not Limited to Existing Users 13

2.5 Extensible, Open, and Platform-Independent 14

3 Design 15

3.1 Dual Design 15

3.2 Software Components 15

3.3 Security 16

3.4 Low Cost and Connectivity Requirements 18

3.4.1 Java 19

3.5 Ease-of-use 20

3.6 Automation 20

3.6.1 Referring End 21

3.6.2 Receiving End 21

3.7 Extensible and Based on Open Standards 22

3.8 Platform-Independent 23

4 Client-Server Model 24

4.1 Workflow 24

4.2 Discussion 27

5 Peer-to-Peer Model 29

5.1 Workflow 29

5.2 Discussion 31

6 TeleMedMail 32

6.1 Workflow 32
6.1.1 Sender and Receiver Identification 32
6.1.2 Clinical Patient Data 34
6.1.3 Images 34
6.2 Attached Files 34
6.3 Editing an Image 35
6.3.1 "Crop and Rotate Image" Dialog 35
6.3.2 "Annotate" Dialog 35
6.3.3 "Compression" Dialog 37
6.3.4 "Edit Comment" Dialog 39
6.4 Save Case, Open Case, New Case 39
6.5 Configuration 42
6.6 Advanced Configuration 42
6.6.1 Standard Fields 43
6.6.2 Text Labels and Internationalization 46
6.7 File Types 47
6.8 Address Book 47
6.9 Sent Case Browser 49
6.10 Decrypt Replies 51
6.11 Case UIDs 52
6.12 Java 2 and Installation 53
7 TMMReceiver 55
7.1 External Applications 59
7.2 Further Work 60
8 TMMServer 61
8.1 Workflow 61
8.1.1 New Account Application 61
8.1.2 Login and Case Index 64

8.1.3 Viewing Cases 64
8.1.4 Replies 66
8.2 Administration 69
8.2.1 User Management 69
8.3 Further Work 70
9 Viewer 72
9.1 Window Levels 74
9.2 Multiple Languages 75
10 CaseViewer 78
10.1 Further Work 80
11 Security Risks 81
11.1 Forged Replies 81
11.2 Spurious Referrals 82
12 Conclusions 84
12.1 Other Usage Scenarios 84
12.1.1 Remote Specialists 84
12.1.2 Trusted Third-Party Record-Keeping 86
12.2 Further Work 86
12.2.1 Minor Improvements for Broader Use 87
12.2.2 Message Authentication for Large-Scale Use 88
13 References 90
Appendix A - TeleMedMail Sent Case Archive 92
Appendix B - TeleMedMail Saved Cases 93
Appendix C - Standard Fields Descriptor Files 94
Appendix D - english.lab 95
Appendix E - TeleMedMail AddressBook Interface 101
Appendix F - TeleMedMail CaseBrowser Interface 103
Appendix G - TMMServer SQL Database Setup Code 104

List of Figures

Figure 3.1 – Client-Server Model Overview ____ 17

Figure 3.2 – Peer-to-Peer Model Overview ____ 17

Figure 4.1 – Client-Server Model ____ 25

Figure 4.2 – Client-Server Model Case Packaging ____ 25

Figure 5.1 – Peer-to-Peer Model ____ 30

Figure 5.2 – Peer-to-Peer Model Case Packaging ____ 30

Figure 6.1 –TeleMedMail Main Window ____ 33

Figure 6.2 – TeleMedMail Attached File Panel ____ 33

Figure 6.3 – TeleMedMail Crop and Rotate Image Dialog ____ 36

Figure 6.4 – TeleMedMail Annotate Image Dialog ____ 36

Figure 6.5 – TeleMedMail Image Compression Preview Dialog ____ 38

Figure 6.6 – TeleMedMail Edit Comment Dialog ____ 38

Figure 6.7 – TeleMedMail Save Case Dialog ____ 40

Figure 6.8 – TeleMedMail Open Case Dialog ____ 40

Figure 6.9 – TeleMedMail Configuration Dialog ____ 41

Figure 6.10 – TeleMedMail Compression Settings Dialog ____ 41

Figure 6.11 – TeleMedMail Edit Address Book Dialog ____ 48

Figure 6.12 – TeleMedMail Case Browser ____ 48

Figure 6.13 – TeleMedMail Secure Reply ____ 50

Figure 6.14 – TeleMedMail Decrypt Reply Email Dialog ____ 50

Figure 6.15 - TeleMedMail Installation ____ 54

Figure 7.1 – TMMReceiver Graphical Mode ____ 56

Figure 7.2 – TMMReceiver Text Mode ____ 56

Figure 7.3 – TMMReceiver Received Case Notification Email ____ 57

Figure 7.4 – TMMReceiver Received Case Reply Email ____ 57

Figure 8.1 – TMMServer Account Application Page ____ 62

Figure 8.2 – TMMServer Account Activation Email 62

Figure 8.3 – TMMServer Login Page 63

Figure 8.4 – TMMServer Case Index Page 63

Figure 8.5 – TMMServer View Case Page 65

Figure 8.6 – TMMServer Low-Bandwidth View Case Page 65

Figure 8.7 – TMMServer Send Web Reply 67

Figure 8.8 – TMMServer Send Secure Email Reply 67

Figure 8.9 – TMMServer Send Unsecure Email Reply 68

Figure 8.10 – TMMServer User Management 68

Figure 9.1 – Viewer Window 73

Figure 9.2 – Viewer Window Levels Intensity Transfer Curve 73

Figure 9.3 – Effect of Brightening an Image (After/Before) 76

Figure 9.4 – Effect of Inverting an Image (After/Before) 76

Figure 10.1 – Peer-to-Peer HTML Index Page 79

Figure 10.2 – CaseViewer Window 79

1 Introduction and Previous Work

Developing countries face a severe shortage of doctors. In Africa, the concentration of doctors ranges from 6.2 doctors per 100,000 population in Ghana to 56.3 per 100,000 in South Africa. (The USA, in contrast, has 279 doctors per 100,000, while other developed countries have even higher figures[1].) Other poor areas of the world face similarly grim shortages of medical expertise. The shortage of specialists is even worse: 14 countries in Africa do not have a single radiologist[2], for example.

Further, those doctors that do exist in developing countries are not evenly spread out: many villages and rural areas do not have even the most basic medical and health facilities, and their populations must travel hours or days for medical care. Specialists are also poorly distributed—they generally cannot be found outside of big cities.

The problem of lack of doctors in developing countries is caused by a fundamental lack of resources that causes a lack of training opportunities and a heavy brain drain to rich countries, and as such the solution is very expensive and complicated. The problem of the uneven distribution of the scarce existing medical expertise, however, is more tractable, and can be partly addressed by using telemedicine to allow remote doctors and healthcare workers to refer cases to doctors and specialists at well-staffed hospitals.

Telemedicine, as it is typically practiced in developed countries, focuses on real-time videoconferencing via a high-bandwidth network connection. A typical system requires at least two 64-kbit/sec ISDN lines, and uses equipment costing between $5,000 and $15,000, and additional expensive attachments such as digital ophthalmoscopes. Although such systems have been successfully used in developing countries, the costs make this type of system very difficult to sustain in the developing world outside of small well-funded studies[3]. Even when the equipment is donated, or provided at heavily discounted prices, the connection costs are prohibitive: for example a dual-ISDN (128 kbit/sec) call from Boston to Lima, Peru costs over $150 per hour (2002 price).

However, physicians in developing countries have demonstrated ways to do very low-cost telemedicine referrals, using store-and-forward electronic mail[4] instead of real-time interaction, digital still cameras instead of video[5], and standard computer applications instead of expensive telemedicine systems[5]. Wolfgang Seckler and Don O'Mahony, two doctors in South Africa, began sending referrals as ZIP files

attached to standard email messages[6,7]. The Armed Forces Institute of Pathology, Washington D.C. began using a similar technique, accepting email referrals and replying asynchronously by fax or telephone. These systems clearly show that sufficiently determined and computer-savvy doctors can improve the access to specialist opinions in remote rural areas with the use of cheap hardware and standard software. Indeed, as these methods have become more widely known, considerable evidence has arisen to prove the utility of store-and-forward and still-image telemedicine[8,9]. Some African doctors now refer to the concept of a "standard store-and-forward telemedicine system," which consists of assembling a case either as a ZIP archive with an HTML index page, or as a Microsoft PowerPoint presentation, and then sending it as an email attachment.

While this telemedicine technique is demonstrably useful to computer-savvy doctors, it is not broadly applicable in its current form. Its use is complicated at the referring end, requiring the concurrent use of multiple software packages (digital camera software, possibly an image-editing program, an email client, a web page editor and a zip utility, or, if bandwidth is not a concern, MS PowerPoint); further, it is inefficient at the receiving end, requiring the specialist to download, locate, open, and archive email attachments manually. Finally, this method does not provide any sort of security to ensure patient confidentiality.

Once these ease-of-use, efficiency, and security obstacles are removed, however, the large-scale use of low-cost store-and-forward telemedicine will become possible. This should make it a valuable tool for developing countries that want to address the problem of medically underserved regions—it can become an integral part of a country's health care system, spreading the reach of its centrally located specialists. The same technique is also starting to be used in underserved rural areas of developed countries as well, such as northern Canada[10] and Norway[8].

My thesis describes a store-and-forward telemedicine system, called TeleMedMail, which simplifies the referral process described by Seckler and O'Mahony, and automates the job of viewing and replying to these referrals, while enhancing it to make the entire referral and reply process secure.

The existence of a system facilitating large-scale telemedicine is not in itself a reason to actually practice telemedicine of that sort. Although small studies and anecdotal evidence have shown that telereferrals are medically useful in many situations, the question of whether large-scale, low-cost telemedicine is an appropriate use of the limited healthcare funding available to developing countries is still

open. This question can only be resolved with further projects, studying specific applications of telemedicine to specific situations, and measuring its true costs and associated benefits.

One argument raised against telemedicine is that diagnosing a remote patient with a complicated illness does not necessarily mean that remote health care workers can do anything about that diagnosis. A key focus of pilot telemedicine projects should be to study the degree to which distant specialist diagnoses can be acted upon, whether specialists can tailor their suggested treatments to those available remotely, and to what extent telediagnosis can prevent the expenses and risks of transporting sick patients.

A second issue which should be considered by pilot studies, but is difficult to quantify, is the degree to which having the capability to request a specialist opinion in difficult cases can reassure remote doctors, and make them feel more supported, and perhaps more likely to remain in a remote town, even if that capability is used only sparingly. Another issue to be considered is the extent to which remote healthcare workers can learn from interacting with a specialist on telediagnoses, and provide better care in the future due to a past telereferral[11].

I make no pretense at being able to answer these questions myself. The TeleMedMail system described in this thesis is intended both for pilot studies assessing the costs and benefits of small- and large-scale application of the store-and-forward telemedicine approach, and as a tool that can be widely deployed if that paradigm is shown to be useful.

This thesis builds upon work I did for my Advanced Undergraduate Project[12], and includes work previously described in "Appropriate Technology for Telemedicine in Developing Countries"[13] and "Telemedmail: free software to facilitate telemedicine in developing countries"[14]

The TeleMedMail system I describe is already in actual and test use: the system is being used in Peru to make peer-to-peer referrals, and is being tested in pilot projects in South Africa and Ecuador. In addition, the case-sending portion of the system is being tested in Basel, Switzerland as a tool to send cases to an existing web-accessible pathology database[15].

2 Design Considerations

When beginning the TeleMedMail project, my intention was to build a low-cost store-and-forward telemedicine application, automating the process used by Wolfgang Seckler and Don O'Mahony in South Africa. The project later expanded into designing and building a full telemedicine system, suitable for large-scale use to show that telemedicine could be used as a significant part of a developing country's health care system.

Building such a telemedicine system, appropriate for large-scale use, involves making three improvements that go beyond Seckler and O'Mahony's basic systems:

1. Simplifying the process of referring a case
    - Reducing the amount of human work required to assemble a referral, to save doctor time
    - Reducing the number of different software components needed, to simplify training.
2. Introducing security features into the system
    - Only sending encrypted cases
    - Only sending encrypted replies
3. Allowing both the doctor and specialist to organize cases on their computers
    - Preventing time wasted searching for computer files

These issues led me to set out a set of design guidelines relating to security, low cost, automation, and transparency.

2.1 Security

Above all else, the system would need to be secure. Although an insecure system could still be an improvement over those pioneered by Seckler and O'Mahony, it would be unacceptable for large-scale use: no large institutional user would (or should) invest significant time and effort installing a system with such a glaring deficiency, as it would clearly need to be replaced before too long.

One might think that, given their lack of medical expertise, developing countries are not concerned with patient confidentiality: this is not true. The social stigmas attached to some diseases can make security

literally a life-or-death issue: a woman diagnosed with HIV in southern Africa, for example, may be turned out of her village to starve.

2.2 Low Cost and Connectivity

Secondly, the system should have very low requirements in software, hardware, and network connectivity. For large-scale use in poor countries, the system needs to be as cheap as possible on a per-installation basis. This mainly means that the system should use email for communications, and stick to still images and text, as Seckler's and O'Mahony's works suggest, and that the system should require no software that isn't free. (Although being low-cost is the main point of the system, it is still secondary to security as a design consideration. An expensive system may find large-scale use somewhere; an insecure system is not worth setting up in any large-scale environment.)

2.3 Ease-of-use & Automation

Thirdly, the system would need to be easy to use. Actually putting the system into large-scale use will require significant investment in human training—an easy-to-use system would save significant amounts of time and money by simplifying that training.

In addition to being easy-to-use, the system should automate as much of the process of sending and receiving a case as possible, and ensure that the workflow of the referral process is straightforward to the doctors involved.

In particular, the system should make the job of receiving and replying to a referral as easy and fast as possible: causing scarce specialists to spend unnecessary time clicking and typing would not help improve the state of developing-country health-care.

2.4 Not Limited to Existing Users

One less obvious design consideration is that the system should provide a way to refer cases to specialists who are not affiliated with one of the system's servers.

The basic assumption is not only that specialists are scarce in a particular developing country, but that there may not be any at all, in which case a doctor might want to make a one-time referral to a specialist not normally participating in the system.

Further, a specialist who does participate in the system might not be located at a well-equipped hospital, and thus be unable to access a hospital server over a local network: it should still be possible to refer a case to such a specialist.

2.5 Extensible, Open, and Platform-Independent

Finally, the TeleMedMail system, like all others, will have some flaws and weaknesses. The system should be as transparent as possible to facilitate bugfixes; it should also be extensible and flexible to allow future improvements and enhancements, both by myself and by other users.

First, it should be possible to replace individual components of the system without having to modify others. Second, the system should use open standards wherever possible, and should avoid tying itself to any particular platform.

3 Design

3.1 Dual Design

Two of the design guidelines are in direct conflict. On the one hand, as much of the referral process as possible should be automated, which means that the receiving end of the system needs to have installed software for that purpose. On the other hand, it should be possible to refer a case to a specialist outside of the system, who by definition won't have the system software installed on his computer.

The conflict between these two requirements led me to build the system with a dual design, allowing cases to be referred in two separate modes.

The TeleMedMail system's preferred mode of operation follows a client-server model: it assumes that specialists are able to access a server on a high-speed LAN, that the server has non-negligible computing power, and that the server is set up by knowledgeable IT staff. This preferred mode is designed for the case where there are a few central, well-staffed hospitals, each with several specialists, and many remote clinics from which doctors refer cases to the specialists at the central hospitals.

The more general mode of operation follows more of a peer-to-peer model (in the sense that it doesn't count on any central server), and requires no special software on the receiving end, but is less automated. It is intended for one-time referrals to doctors outside the system, but it has the side effect of allowing computer-savvy doctors to form a flexible, ad-hoc referral network.

From one perspective, this dual design is the most interesting part of the system: no existing telemedicine system that I am aware of has the concept of a one-time secure referral, to any computer user in the world, using only email and requiring no non-standard software or server account.

3.2 Software Components

Case data follows the same format across the two models, but the way it is packaged and encrypted differs. I wanted to reuse as much code as possible between the two models, all the while focusing on automation in the client-server model, and lack of installation in the peer-to-peer model. Towards that end, I divided the system into the following software components:

TeleMedMail An application that allows the user to assemble images and text into a structured case. It packages and encrypts this data, and sends it via email. (See Section 6.)

Viewer A very small Java applet, used to view an image file, with controls for zoom, and brightness. (See Section 9.)

CaseViewer A small Java applet, which decrypts a case, and displays its text and images. (It shares its image-viewing code with Viewer.) (See Section 10.)

TMMReceiver An application that receives cases, decrypts and unzips them, and inserts them into the database used by TMMServer. (See Section 7.)

TMMServer A JavaServer Pages based web application that manages a SQL database of users and cases. It allows users to view and reply to cases that they have permissions on. (See Section 8.)

Under the client-server model (see Figure 3.1), the sender runs TeleMedMail, the receiving server runs TMMReceiver and TMMServer, and the receiving doctor views cases in a web browser, using Viewer for digital images. Under the peer-to-peer model (see Figure 3.2), the sender runs TeleMedMail, and the receiving doctor views cases with CaseViewer, using the JVM included in his web browser. The key difference is that in the client-server model, TMMReceiver automates the process of unzipping and decrypting the case, and managing keys, while in the peer-to-peer model the user must manually enter a password and use an external unzip utility. These two models are described in detail in sections 4 and 5.

This modular design, with open interfaces between its various components, should also allow other programmers to reuse parts of the TeleMedMail system for other telemedicine applications.

Having two different referral models also adds a degree of redundancy to the system. If a specialist's network connectivity is uncommonly bad at any particularly time, then she can ask a doctor to send (or re-send) her a peer-to-peer referral instead of a client-server one, so that she gets a local, offline copy of the case data.

3.3 Security

When deciding what security features to design into the TeleMedMail system, I considered several possibilities:

Figure 3.1 – Client-Server Model Overview

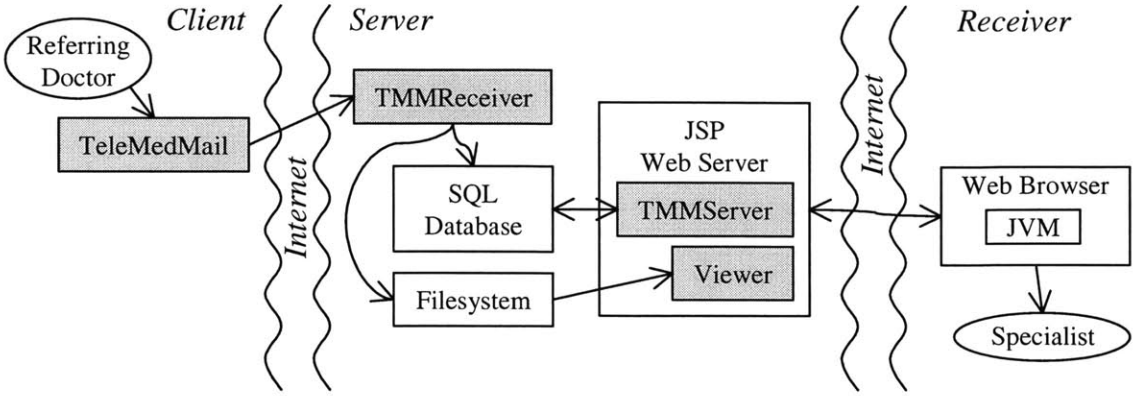
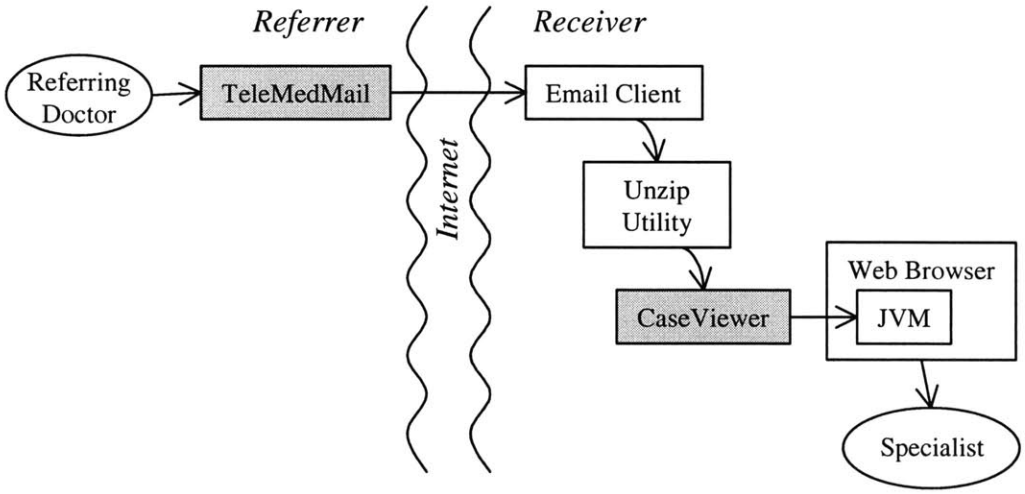


Figure 3.2 – Peer-to-Peer Model Overview



1. Referrals can't be read by anyone but the intended recipient.
2. Referrals can't be tampered with in transit.
3. Replies to referred cases should be also be secured as in (1) and (2).
4. Referral and reply messages should be authenticated.

The first two features are absolutely necessary. A system that does not protect confidential patient data is by definition not secure.

The third feature is slightly less important than the first two, but still highly desirable. A system where replies can be sent securely is significantly more useful than a system where replies are not secure. The latter can still be useful in certain circumstances, however, if it is possible for doctors to reply to referrals via telephone, or if there is a high enough volume of referrals between doctors that using a Case ID instead of a patient's name in replies provides an acceptable degree of patient confidentiality.

The fourth feature is less important than the previous three—exploiting it to do harm to patients requires a break-in at the referring or receiving end. But such a break-in would invalidate most security models, and cause much bigger problems.

After discussing the likely threats with physicians, I decided to implement the first three features, but not the fourth. Authenticating all messages requires a more complicated and less flexible design, while increasing the amount of setup that needs to be done; not authenticating all messages didn't seem to be a major security hole. This is an acceptable decision if TeleMedMail is to be used in a single institution, but it may become a more important problem in a multi-institution enterprise. This is discussed further in Section 12.2.2.

3.4 Low Cost and Connectivity Requirements

For the most part, the system's low requirements stem not from particularly clever optimizations in the software design, but from the system's conscious limitations: using only text and still pictures as case data, and store-and-forward email rather than real-time communication.

One place I did put special effort into keeping the connectivity requirement small was in designing and building the Viewer and CaseViewer applets. After building the applets, I combined event-handling

code for different GUI elements (eliminating anonymous inner classes), which cut the size of both applets by about 5 KB. (This is actually a 20% decrease in the size of "Viewer.jar," and a non-trivial improvement when one considers that the applet may be sent many times over slow dialup connections.)

I also chose to use the IDEA encryption algorithm specifically to decrease the size of the CaseViewer applet. The "IdeaCipher.class" file in the encryption package I used was only 4 KB in size. This compares very favorably to similar block ciphers: "BlowfishCipher.class" takes up 26 KB, and Rc6Cipher.class takes up 8 KB.

I did not put significant effort into keeping the system requirements of the TeleMedMail application small. In order to run a Java program, one must first run a Java Virtual Machine, which is an application with a significant enough memory and processor footprint that any substantial Java application will run slowly on machines with very little memory. On the other hand the TeleMedMail application is straightforward enough that it can run acceptably on modern computers even without being highly optimized.

3.4.1 Java

My decision to develop TeleMedMail in Java rather than C++ involved a tradeoff. On the one hand the system could be developed significantly faster, due to Java's built-in networking, security, and image-manipulation code. On the other hand, it means that the computer used at the referring end of the system should be at least a Pentium with 64MB of memory. (It runs correctly, but painfully, on a Pentium-200 with 32 MB of memory.) This is not an issue if a country or institution decides to set up a large-scale network from scratch, since any computer available today will run TeleMedMail well. It *is* an issue, however, when attempting to build a network out of very old existing computers. A future strategy I will take to combat this problem is compiling TeleMedMail into native Win32 code, with a Java bytecode-to-Win32 compiler such as Excelsior JET[16]. This should result in an application with a smaller memory footprint, which will improve its performance on old machines with less than 64MB of memory.

A few optimizations *were* necessary to increase TeleMedMail's perceived speed on slower computers, such as doing key generation and image loading in background threads, and doing some simple image caching.

3.5 Ease-of-use

One key point from the ease-of-use perspective is that the system has to be multi-lingual: it should find significant use in South America and Africa.

I made the TeleMedMail application fully multi-lingual by putting all text that is ever displayed to the user in a user-selectable language file. At present, English, Spanish, and Portuguese translations exist.

The Viewer and CaseViewer applets both look for alternate button labels in the HTML that contains them, before defaulting to English, thus TeleMedMail can specify non-English text for an applet when it sends a case, with a tiny overhead in message size. So far the Viewer applet has been translated into Russian, to view chest X-Rays of drug-resistant tuberculosis patients[17].

The TMMReceiver server component is currently English-only, although it can easily be translated if users require it. This seemed like an acceptable tradeoff between better design and faster implementation: an IT worker at a well-staffed hospital can almost certainly understand computer-related English, and the text of the program contains only standard computer terms.

The second key ease-of-use point is that the workflow of referring a case, and replying to a referral, has to be straightforward and natural. I spent the most significant amount of time working on the User Interface of the TeleMedMail referral application, which over time has gone through several iterations, and received feedback from users in Brazil, South Africa, and the US. At present, the UI is streamlined and straightforward, and seems well accepted. (See Section 6.)

3.6 Automation

A well-designed telemedicine system needs to automate all "computer tasks" involved in sending a case, receiving a case, and replying to a case, such that the doctors involved can focus on "medical tasks." In particular, the doctors should not have to:

1. Handle email attachments
2. Manage email addresses
3. Manage image files and directories
4. Manage case files and directories
5. Remember more than one password

When used in the preferred client-server model, the system should handle all these points; under the peer-to-peer model, it should automate as much of the process as is possible without requiring the receiving specialist to install any software besides a ZIP program.

3.6.1 Referring End

At the referring end of the system, the TeleMedMail application takes care of the first four points. In client-server mode, it also handles the last one.

TeleMedMail sends emails itself, so the user does not need to deal with attachments, or another email client. It maintains an Address Book, so the user can select the receiving doctor and server by name, and not via email address.

TeleMedMail supports drag-and-drop of image files, so the user can drag files directly from the digital camera software without worrying about directories and real file-management. It also includes a "CaseBrowser" component for looking back at sent cases.

In client-server mode, security is handled via server public keys (configured at install-time), so the user doesn't need to worry about passwords. In peer-to-peer mode, however, this isn't possible. The referring doctor and receiving doctor need to agree on a password, and the referring doctor should use a different password for each specialist. TeleMedMail keeps a copy of the password with the sent case, for use in decrypting replies to that case, so that remembering passwords is not an issue for one-time referrals. But it takes no steps to automate the process of sending multiple peer-to-peer referrals to one or more specialists.

It would be possible to build a password-manager, which maps each peer-to-peer specialist to a password shared with that specialist, but in general doctors should not be using peer-to-peer mode in this way.

3.6.2 Receiving End

In the client-server model, the TMMReceiver automates the process of receiving a case: it downloads the referral from the mail server, decrypts and unzips it, and places it in the server's filesystem and database. TMMReceiver then notifies the specialist (via email) of the received case. The specialist merely has to

click on a link in the email and enter his hospital system password to view the case. While viewing the case, a single click takes him to a "send-reply" page.

The TeleMedMail system doesn't automate the process of the specialist looking through past cases, beyond displaying a chronological list of cases received. Better case-management might or might not be an important feature—that will be clearer once the system can be observed in large-scale use. Implementing a more advanced form of received-case-management is a large task, whose design and implementation is better left until the way that TeleMedMail fits in with already-existing systems can be further studied.

Since the cases are put in a database, and available over an SSL web connection, it's straightforward to build a future system to browse and manage them, once it is more clear how exactly that should be done. This could be implemented as a stand-alone application, or else an existing hospital system can be modified for this purpose.

Under the peer-to-peer model, very little of the process of receiving a referral is automated. The receiving specialist must manually unzip an attachment, open the contained html file, and enter the case password. This requires a certain familiarity with email and ZIP programs, and is tedious to do regularly—an acceptable shortcoming, since the peer-to-peer model is intended for one-time referrals. (In fact, the work a specialist has to do to read a peer-to-peer referral is exactly that required in current state-of-the-art manual systems.)

In client-server mode, the process of replying to a case is also automated: clicking a link on the View Case, or Case Index page on the server allows the user to send a text reply without worrying about email addresses or encryption. Peer-to-peer mode does not provide a process for sending replies.

3.7 Extensible and Based on Open Standards

The key design choice I made to keep the TeleMedMail system extensible and open is to make sure that individual components of the system should communicate with open standards, in ways that are transparent enough to understand without looking at any source code. This choice shows up in several instances.

TeleMedMail communicates with TMMReceiver and CaseViewer via an XML case descriptor file included in the case referral, while the referral itself is a standard ZIP file. This allows cases sent by

TeleMedMail to be easily translated for input into other applications. One advanced user built (overnight) a bridge that imports TeleMedMail case referrals into iPath, an open-source telepathology database[15].

TeleMedMail also archives its sent cases as ZIP files which contain an XML descriptor file as a text file, the case session key as a file of bytes, and replies to that case as standard text files. TeleMedMail's address book is saved as straightforward XML in a text file. This should allow the Sent Case Browser, Sent Case Viewer, Decrypt Reply Email, and Address Book components of TeleMedMail to be improved on and replaced by any Java programmer, without him having to understand any obscure data formats. (See Appendices A and E.)

TMMReceiver places the cases it receives in a SQL database, with a straightforward table structure. This allows other front-ends to be built or modified to access TeleMedMail case data.

A second decision, which makes the system more, open and flexible is that to make it open-source, under the GNU General Public License[18]. This allows users to improve upon the system, or alter it to better suit their own situation, by modifying the source code. An institution could easily modify TMMReceiver's `SqlDBHelper.insertCase()` method to insert cases into its existing database system, for example.

3.8 Platform-Independent

A further design choice was to make TeleMedMail as platform-independent as possible. The TeleMedMail and TMMReceiver applications and the Viewer and CaseViewer applets are all written in Java, which means they can run on any major platform. TMMServer, written in JSP, is slightly less portable, but the major web servers do support it. Particularly important is that the entire TeleMedMail system can run on free software: Linux on a client, and Linux, PostgreSQL[19], and Tomcat[20] on a server. (Most users will run Windows on client machines, but Linux's free licensing can cut the TeleMedMail's system's per-machine cost to the bare minimum, if that is important enough to an institutional user.)

4 Client-Server Model

4.1 Workflow

The preferred way to use the system follows a client-server model. This usage model assumes that there are a few central, well-staffed hospitals, each with several specialists, and many remote clinics from which doctors refer cases to the specialists at the central hospitals. Although designed for this scenario, the TeleMedMail system's client-server model can be used in others as well. (See Section 12.1.)

This model is optimized to allow a specialist to handle many incoming referrals, from many different sources. It requires installing software at the central hospitals to receive and decrypt cases automatically, thus saving the specialist work. This is probably an acceptable tradeoff, because the efficiency improvement of making the specialist more productive dwarfs the incurred cost of installing and configuring a software application at a well-staffed hospital. In addition, specialists at locations without sufficient IT expertise to run a server can still use a non-local server, such as the prototype one running at MIT.

Figure 4.1 shows an overview of the client-server security model. A client-server referral begins with the referring doctor assembling and sending a case with the TeleMedMail application (see Section 6), specifying which server the case should be sent to, and which doctor it is intended for. The case is IDEA-encrypted with a random session key; the session key is then encrypted with the server's RSA public key, and sent along with the case so that the server can use it to decrypt the case, and send back secure replies. The TeleMedMail application also archives a copy of the session key for use decrypting these replies.

The case email is then handled by TMMReceiver (see Section 7), the first of two software components at the heart of the server end of the TeleMedMail system. TMMReceiver is a Java application that acts as a POP client, downloading case referrals from a mail server, then decrypting and unzipping them, and inserting them into a SQL database. (The "server email account" may be on a local mail server, or at an outside ISP—cases are stored there in encrypted form.) TMMReceiver sends emails to the referring doctor and receiving specialist, notifying the former that his case was received, and the latter that she has a case waiting, with a link to that case.

Figure 4.1 – Client-Server Model

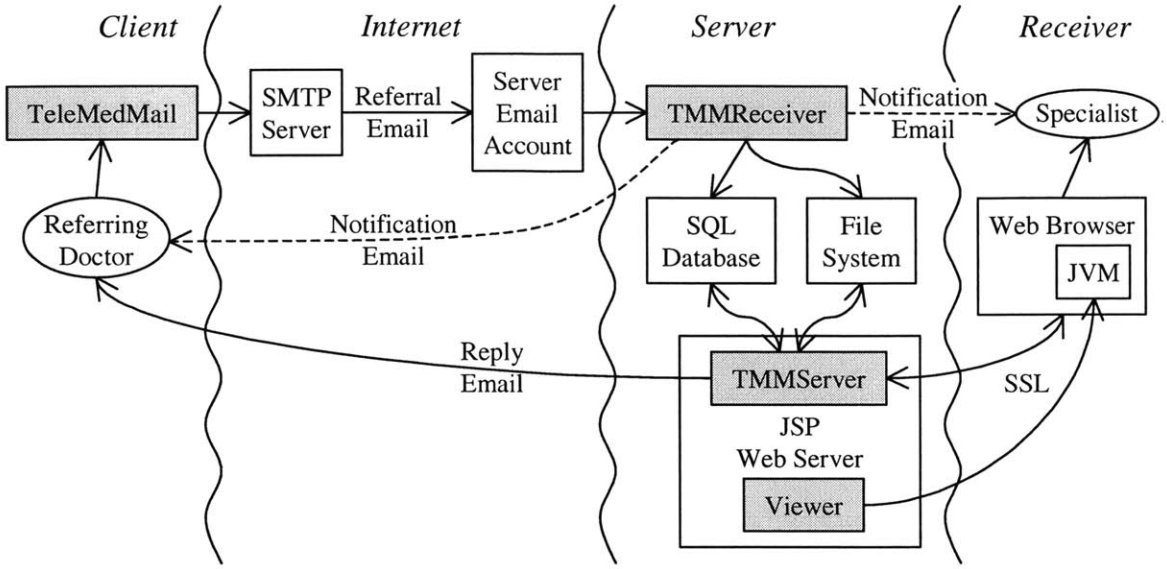
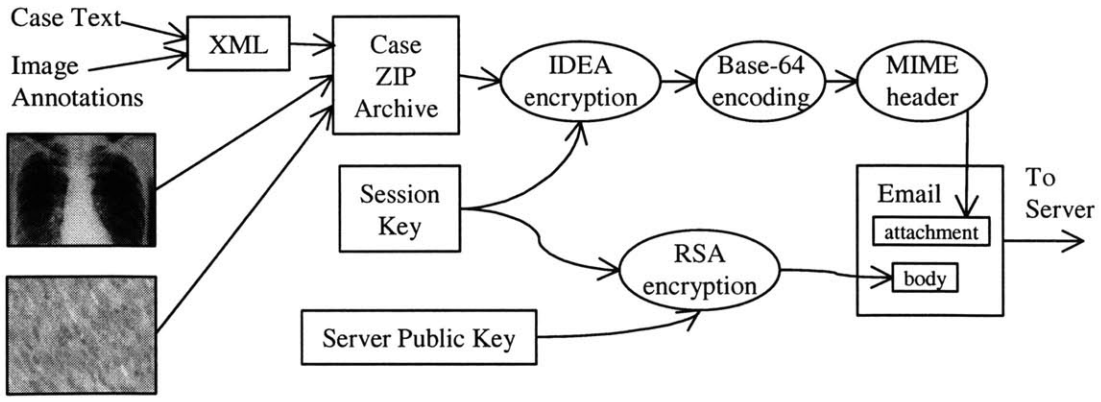


Figure 4.2 – Client-Server Model Case Packaging



The second main server component is TMMServer (see Section 8), a JSP web application that serves cases via HTTPS with SSL, allowing registered users to log in securely to view and reply to their cases from any web browser. When the receiving specialist gets a notification that a case is waiting for her, she browses to the website served by TMMServer, reads the case, and then replies to it.

In order to make the TeleMedMail system flexible, the specialist can reply to a case referral in three ways: on the web, via secure email, or via unsecure email. All three types of replies are sent from the TMMServer reply page and require the same work to be done by the specialist—she just has to specify which delivery method the system should use.

A web reply (the preferred reply method) consists of posting a message on the TMMServer website. TMMServer notifies the referring doctor that his case has a reply, and he logs into the website to view it.

If the referring doctor is so remote or poorly connected that he cannot view a reply on the web (he might have only email access and not web access), then the specialist must reply to him via email. If she specifies that the email should be secure, then TMMServer IDEA-encrypts the text of her reply with the random session key originally used to send the case, and sends that ciphertext to the referring doctor via email, along with English instructions on how to decrypt it. This means using the Decrypt Reply Email feature of the TeleMedMail program he originally used to send the case. The TeleMedMail program determines from the reply text which case it refers to, looks up the archived session key, decrypts the ciphertext, displays the clear-text to the user, and archives it in the record of the relevant case. See Section 6.10.

An insecure email reply is simple: TMMServer just sends the plaintext of the specialists reply to the referring doctor via email, along with the UID of the case it refers to. In order to protect patient confidentiality, this reply method recommends that the specialist avoid mention of any identifying characteristics of the patient. The other two reply methods are strongly preferred, but insecure email replies are still allowed, to make the system more flexible. Doctors who are currently doing TeleMedMail's sort of email-based telemedicine generally use insecure email replies, or alternatives such as fax or phone.

Figure 4.2 shows a more detailed view of the way the client-server-mode referral is packaged. TeleMedMail packages the case text and images into a ZIP file, encrypts the zip with a randomly generated session key, and encodes the result as a MIME email attachment. TeleMedMail then generates an email

message with the encrypted case as an attachment. The email message also contains the session key, encrypted with the receiving server's public key, so that the server (and only the server) can later decrypt the case. TeleMedMail sends this email message to the receiving server.

4.2 Discussion

The client-server model requires non-trivial computer power: the server needs to be running a JSP- and SSL-enabled web server. That is, however, not an unreasonable requirement for a well-equipped hospital, and, even in developing countries, the cost of computer hardware is cheap compared to the cost of human expertise. (The TMMServer database front-end could easily be rewritten with ASP or ColdFusion; the particular choice of JavaServer Pages technology should not prevent any institution from running a TeleMedMail server.)

The client-server implementation also requires non-trivial configuration at the client end in order to function. Remote referring locations need to be configured with the public key and email address of the server they will refer cases to, along with the email addresses of specialists who are accepting referrals at that server. This requirement is acceptable, because a hospital will not be accepting unexpected referrals from unknown remote locations. When a server agrees to accept referrals from a client it can also provide some expertise to help the client configure its software. It is likely that the server site will physically send an IT worker to the client to set up the system, or else prepare a CD or download pre-configured with the correct settings and address book: this makes configuration a less significant issue.

Once configured, the client-server implementation makes it very easy for the referring doctor to send cases, and for the receiving specialist to view them. Neither user has to worry about managing multiple passwords that they share with different doctors—TeleMedMail and TMMReceiver provide security transparently. In fact, this implementation reduces the work the receiving specialist must do to view a case to the minimum. To view a referral he only has to click on a link in the notification email he receives, and enter his hospital system password; to reply he only has to click a link, type the reply text, and press a button.

In exchange for the initial effort of setting up a server, the client-server model makes referring and receiving cases easier-to-use and more flexible than non-server methods would allow. A specialist can view

and reply to his cases in a secure fashion, from any web browser anywhere, and not just locally at his hospital. (Although this can depend on the hospital's firewall rules.) It also allows secure message passing between the referrer and specialist, by two different methods. These capabilities make the client-server model useful in situations beyond the specific one it was designed for. For example, an institution could use it to act as a trusted intermediary that stores legal copies of data, to prevent disagreements and data loss, as a service to unaffiliated specialists throughout its geographical region. (See Section 12.1.1.)

5 Peer-to-Peer Model

5.1 Workflow

The more general way to use the system follows a peer-to-peer model, in the sense that it doesn't count on any central server. It requires only that the receiver have a ZIP program, and a Java-enabled web-browser. These are easy-to-meet requirements—the vast majority of computers capable of checking email provide these capabilities.

The strength of this model is that a doctor can refer a case to any specialist anywhere, with no special software requirements. It is intended for use sending one-time referrals to doctors outside the system. (Although it does allow any informal network of doctors to make regular referrals, this is better done in other ways, as described in Section 12.1.1.) The main drawback of this referral method is that it requires the receiving specialist to manually locate an email attachment in his filesystem, unzip it, and open the contained HTML file. None of this is particularly difficult to do, but it is tedious and inefficient for a specialist to spend time doing it frequently.

A further caveat of this model is that password exchange has to happen outside of the system. (The doctor and specialist will probably agree on a secret key over the telephone or in person, at the same time as the specialist agrees to take referrals from the doctor.)

Figure 5.1 shows a broad overview of the peer-to-peer security model. The referral begins in the same way as a client-server referral, with the referring doctor assembling a case out of text and images. In the peer-to-peer model, he does not specify a receiving server, but he must type in a case password manually.

Unlike the client-server model, there is no special software on the receiving end. The receiving specialist uses his usual email client to retrieve the case, unpacks it with his unzip software, and opens the case "index.html" file in his web browser, which loads the CaseViewer applet (see Section 10) in the web browser's JVM. CaseViewer prompts the specialist for the case password (which the referrer shared with him outside the system) and shows him the case.

Figure 5.2 shows a more detailed view of the way the case is packaged. TeleMedMail encrypts the text of the case with the shared secret key and packages that along with the images and the CaseViewer

Figure 5.1 – Peer-to-Peer Model

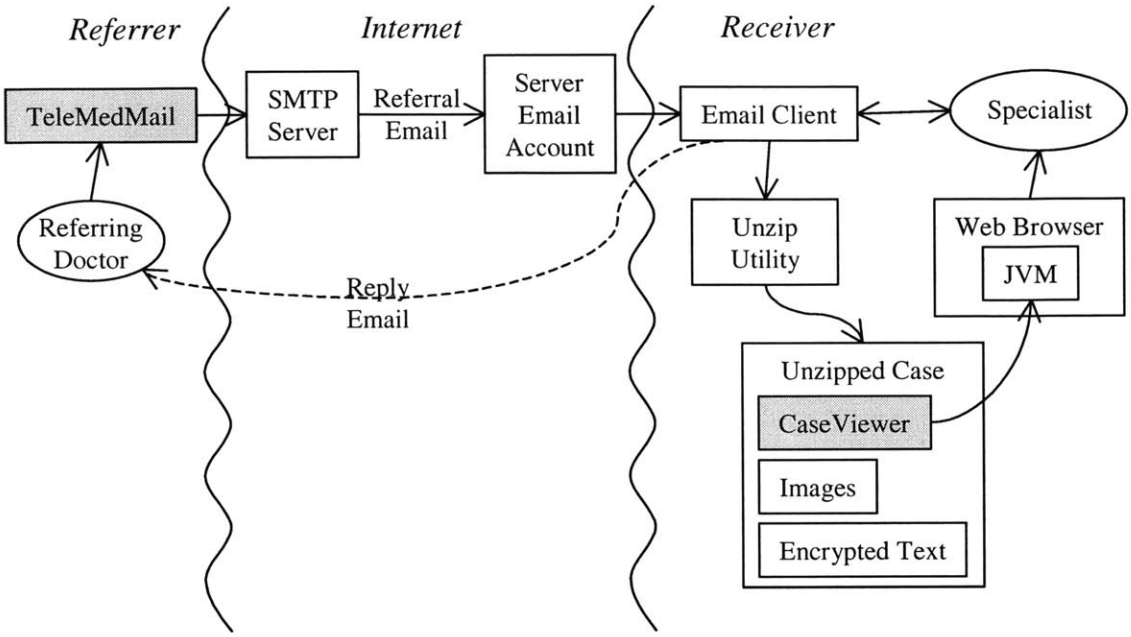
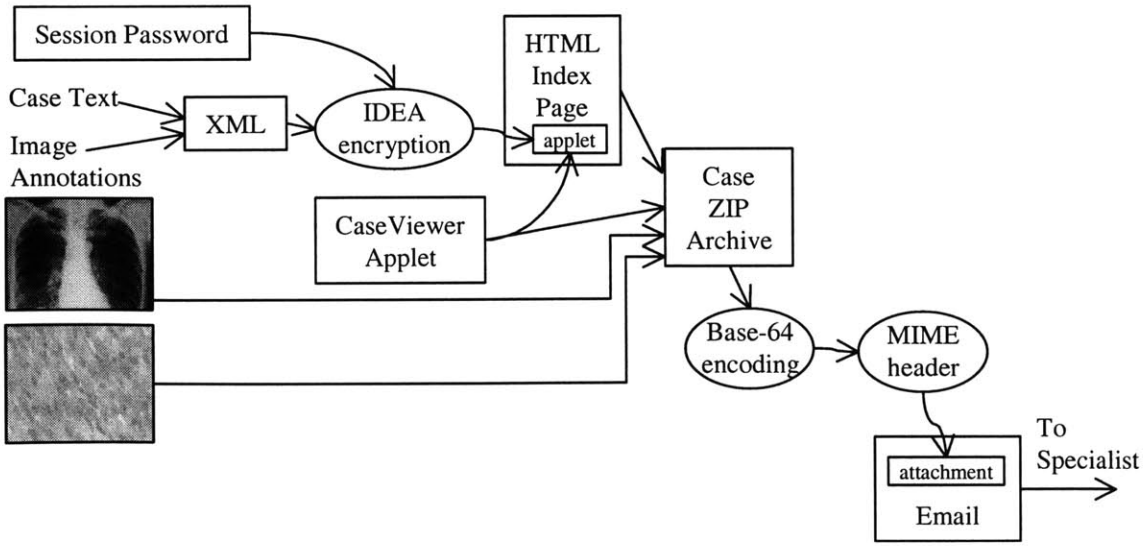


Figure 5.2 – Peer-to-Peer Model Case Packaging



applet into a ZIP file. The ZIP archive is then encoded as a MIME attachment, and sent via email.

5.2 Discussion

This implementation has two significant drawbacks beyond the inconvenience of having to manually unzip the case and manage multiple passwords: it doesn't provide any way to send secure replies, and it doesn't encrypt the images that are sent with the case. The first problem is easily solved: the CaseViewer applet can be enhanced to be able to generate IDEA-encrypted reply text, which the specialist could then paste into his usual email client and email back to the referring doctor. (The referring doctor would then use his TeleMedMail application to decrypt the reply in the same way he would decrypt a secure client-server reply.) The second problem stems from security restrictions on Java applets, although it can be fixed with minor code modifications.

The justification for these shortcomings is that the peer-to-peer model is not meant for large-scale use. The workarounds—making sure that images and replies don't contain patient-identifying information, or perhaps replying by telephone—are not particularly onerous when done infrequently.

Taking a different approach to peer-to-peer referrals could alleviate these problems: the TeleMedMail application could be enhanced to receive cases in addition to sending them. A symmetric peer-to-peer system, where both the sender and receiver use the same software, would be more powerful and automated than TeleMedMail's peer-to-peer referral model.

TeleMedMail purposely avoids this alternate approach. I elected to make the peer-to-peer model flexible, not requiring installed software on the receiving end, rather than automated, since the client-server model can provide that automation, along with other benefits as well. In particular, specialists need almost no computer knowledge to accept case referrals—this would not be the case in a symmetric peer-to-peer system.

6 TeleMedMail

The heart of the TeleMedMail system is a Java application called TeleMedMail, which allows the user to assemble images and text into a structured case, and automates the process of packaging, encrypting, and sending the case.

6.1 Workflow

The workflow of assembling a case is straightforward, and falls into three broad areas:

1. Specifying who is sending and who is receiving the case
2. Entering clinical text data about the patient
3. Attaching, cropping, and annotating images

The user can interleave tasks from each area however he chooses. (For example, the user can still type in patient data while waiting for a particularly large image to load, even if he usually does all image management first, and enters text last.) The conceptual distinction into three areas makes the workflow and user-interface more coherent, and improves ease-of-use: each area falls in a distinct area of the GUI. (See Figure 6.1.)

The Sender and Receiver Identification and Clinical Patient Data fields are implemented as user-configurable standard fields, as described in Section 6.6.1. This allows the user to alter their placement and default values, add other interesting fields, and remove unnecessary ones.

6.1.1 Sender and Receiver Identification

Specifying the sending hospital, sending doctor, receiving hospital, and receiving doctor happens in the top-left of the GUI. These fields generally have one default value, or a few possibilities in a drop-down list, and can be filled out quickly and routinely.

These text fields are placed together so that once the user has set the default values, and entered addresses in his address book, he can enter this "book-keeping" information as efficiently as possible.

Figure 6.1 –TeleMedMail Main Window

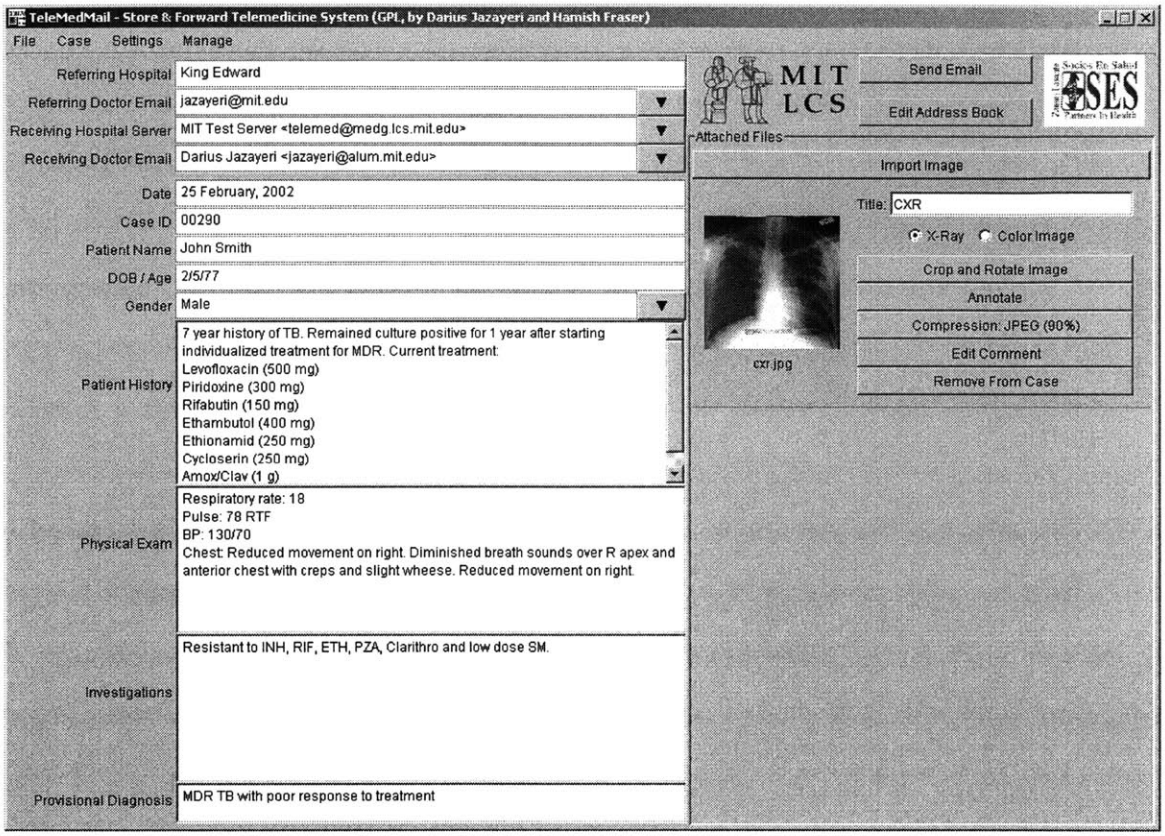
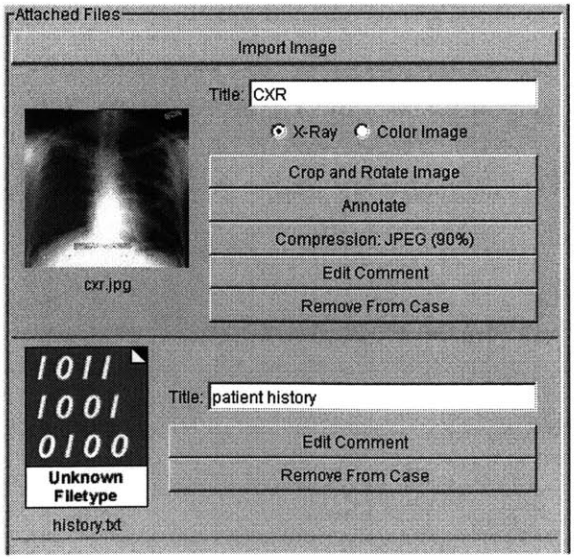


Figure 6.2 – TeleMedMail Attached File Panel



6.1.2 Clinical Patient Data

A TeleMedMail case contains certain standard fields giving clinical information about the patient, such as date of birth, gender, patient history, and physical exam results. These fields are located in the bottom-left of the GUI. Most of this information will be unstructured free text, without preset possible values, although the user can still set default values and possibilities if the cases he refers follow a particular pattern.

Filling out these text fields will usually be the most time-consuming part of assembling a case—it is unavoidably tied to the user's typing speed.

6.1.3 Images

The right side of the GUI is dedicated to attaching images (or other types of files) to the case. The user can attach files to the case either by pressing the "Import Image" button, which brings up a File Chooser dialog, or by dragging the file and dropping it onto the "Attached Files" portion of TeleMedMail's GUI. The drag-and-drop method is more efficient, as it allows the user to drag file icons from the digital camera software, or the Windows file manager, without having to browse through directories. Once the user has attached an image to the case, it appears in the GUI on a panel.

6.2 Attached Files

Once a file has been attached to the TeleMedMail case, it shows up in the GUI as a panel (see Figure 6.2) that allows the user to give the file a short title and a longer comment. If the file is an image of a recognized file type, the panel also has several buttons that bring up dialogs allowing the user to edit the image. TeleMedMail provides only a few simple image-editing features, which are nonetheless sufficient for almost all clinical referrals: the user can crop the image, rotate it in multiples of 90 degrees, and place textual annotations on it. The user can also change the way the image is compressed.

TeleMedMail deliberately does not let the user change the brightness or contrast of an image, although that would have been an easy feature to implement. If the quality of an image does not seem good enough to the referring doctor, then he should re-photograph or re-scan the original data source, taking special care to optimize the camera brightness to deal with the limited dynamic range of 8-bit JPEG images. The referring doctor, and his TeleMedMail software, should provide as clean and faithful a

digitized image as possible to the specialist. The specialist can then use the Viewer applet's window level controls (see Section 9.1) to enhance the image if he feels it is necessary—he knows better than the referring doctor whether adjusting the image is clinically useful.

6.3 Editing an Image

The buttons on the attached image panel bring up dialogs allowing the user to edit the image in several ways.

6.3.1 "Crop and Rotate Image" Dialog

The topmost button in the attached image panel allows the user to crop out unwanted portions of the image, and to rotate the image in increments of 90 degrees (see Figure 6.3). In order to crop the image, the user clicks and drags a rectangle to select the interesting portion: only that interesting portion of the image will be sent in the case referral. The user can clear the crop rectangle by pressing the "No Crop" button.

This dialog also has buttons allowing the user to rotate the image left, right, or 180 degrees, and a checkbox specifying whether the image should be shrunken to fit in the window, or shown at full-size. (Many digital medical images are large; viewing the image at full-size often requires the user to scroll around to see the whole image. Since having to scroll would be an inconvenience while cropping, this checkbox defaults to fit-in-window mode.)

The button for the "Crop and Rotate" dialog is the topmost one for two reasons. First, it is probably the most common way the user will edit the image. Second, cropping and rotating should be done before annotating (as it may influence where the user places annotation text) and before changing the compression level (since cropping will definitely affect the compressed filesize). Placing the button above the others makes it easier to find, and more natural to press first.

6.3.2 "Annotate" Dialog

The second button in the attached image panel allows the user to place textual annotations on the image. An annotation consists of a numeric label, a visual cue to the point or region on the image that the annotation refers to, and a string of text. There are three types of annotations: point annotations, arrow annotations,

Figure 6.3 – TeleMedMail Crop and Rotate Image Dialog

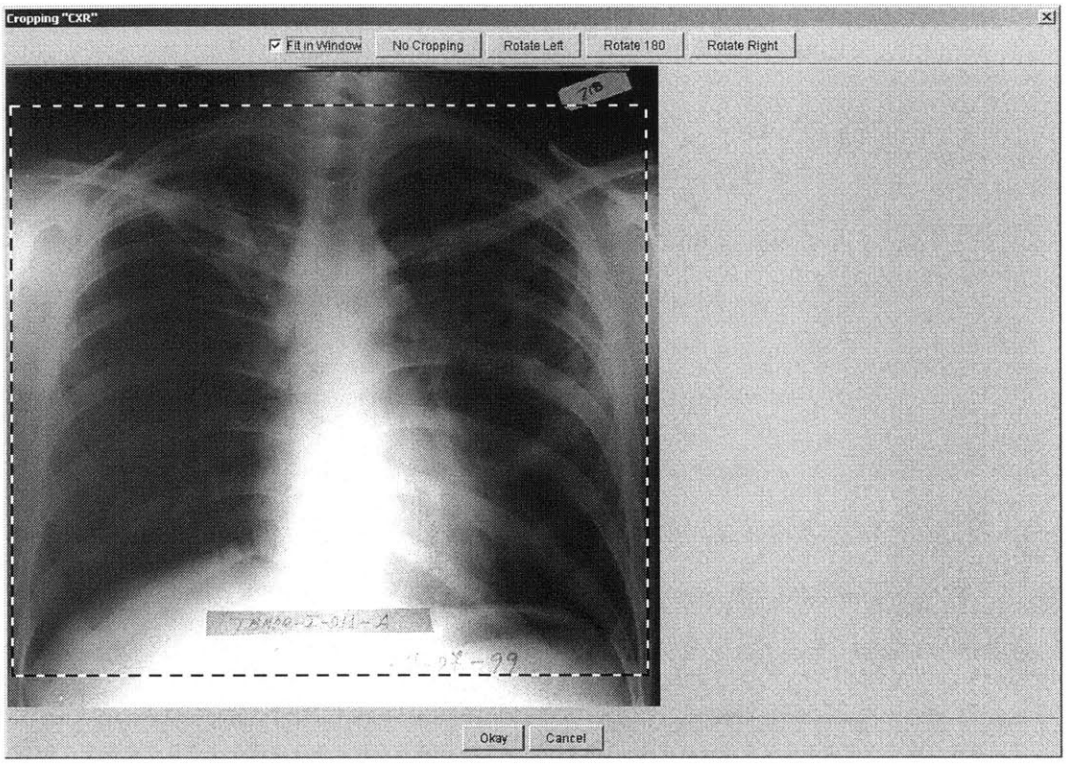
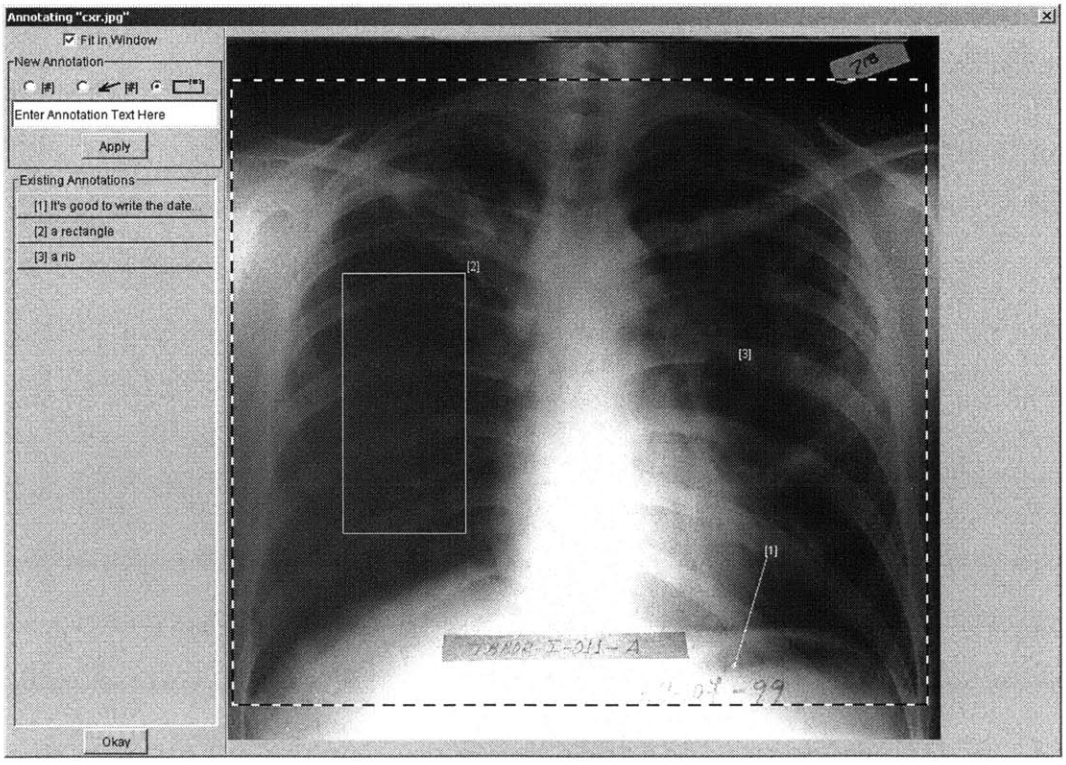


Figure 6.4 – TeleMedMail Annotate Image Dialog



and rectangle annotations. (See Figure 6.4.) In an arrow annotation, an arrow points at the area of interest of the image, with the label drawn at its tail. In a rectangle annotation, a rectangle is drawn around the area of interest, with the label in the top-right corner. In a point annotation, the label is drawn without any extra visual cue.

This dialog shows the image on the right side of the screen, and shows the text of existing annotations on the bottom-left. The top-left of the screen either shows a new annotation being created (allowing the user to enter its text), or the currently selected existing annotation (allowing the user to modify its text, or delete it.)

Like the "Crop and Rotate" dialog, the "Annotate" dialog has a checkbox allowing the user to choose whether to shrink the image to fit in the window or to view it at full size.

The Annotate dialog is not very intuitive at first use, although its operation is easily learned. (This is partly because not all users will be familiar with the concept of annotating an image, although it is easily explained.) Determining how to make this dialog's workflow more intuitive will require studying actual first-time user interaction.

6.3.3 "Compression" Dialog

The Compression dialog (see Figure 6.5) shows the user exactly what the sent image will look like, and exactly what its filesize will be, once it has been cropped and compressed. The top of the dialog's GUI shows the image at full resolution, and allows the user to scroll around it. The bottom of the GUI has a linked slider and text box that allow the user to set the image quality level, between 0 and 100%. The default quality level is 90%.

The specified quality level defines the JPEG quantization coefficients, and corresponds to the idea of JPEG compression level found in typical image-processing applications. (Most other imaging applications have a default JPEG image quality of 70%, and call 80% "high quality.") The choice of a higher-than-usual default quality level means that the image files TeleMedMail sends are larger than they perhaps have to be: a file compressed at 90% quality is about 1/3 bigger than one compressed at 80% quality. Because the effect of JPEG compression on specific types of medical images has not been thoroughly studied, I chose to err on the side of (probably) compressing images too little rather than too much.

Figure 6.5 – TeleMedMail Image Compression Preview Dialog

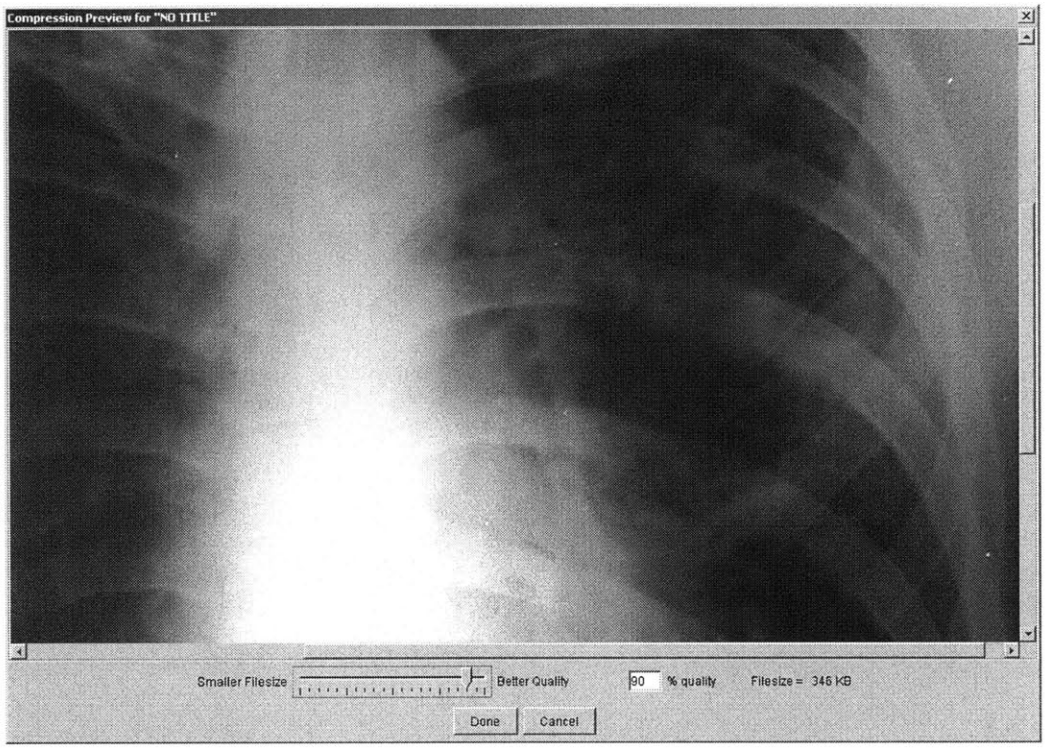
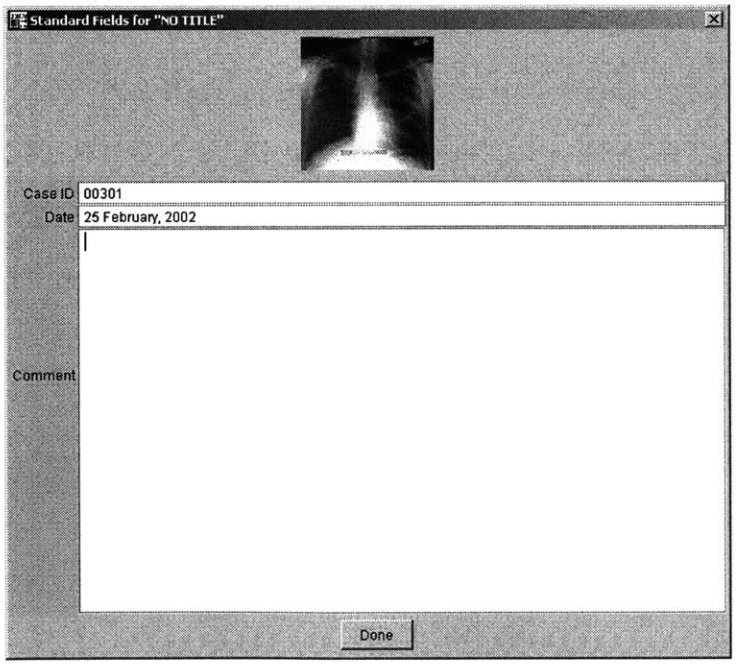


Figure 6.6 – TeleMedMail Edit Comment Dialog



Unlike the Crop and Rotate and Annotate dialogs, the Compression dialog does not have a fit-in-window option. Viewing the image at anything other than full resolution introduces zooming artifacts: this should not be allowed to influence the user's judgment as to whether the image is compressed too much.

6.3.4 "Edit Comment" Dialog

The Edit Comment dialog (see Figure 6.6) allows the user to enter text that applies to a specific image, rather than to the entire case. Since the type of information the user might want to enter depends so heavily on the type and contents of the file, I chose to have only a "Date" field, and a long generic "Comment" field.

Unlike the other three dialogs, the Edit Comment dialog also applies to non-image files. In the case of an image file, the specialist will see information the user enters in these fields as a comment along the top of the image when he views it in the Viewer applet. (See Section 9.) In the case of a non-image file, the user will see the comment next to the link to the file in the html case file.

6.4 Save Case, Open Case, New Case

TeleMedMail's File menu allows the user to save the current case for future editing, to load a previously saved case, or to clear all data from the current case and start a new one.

Choosing "New Case" resets all text fields to their default values, clears those without default values, removes all images from the case, and invalidates the Case ID (see Section 6.11).

Choosing "Save Case" brings up the Save Case dialog (see Figure 6.7). This dialog allows the user to specify a filename (and possibly location), and choose whether the case should be saved for future editing, or exported as a (view-only) sent case. If the user is saving the case for future editing, he can choose to save copies of the attached files, or just save links to them. The second option takes less space, but requires that attached files not be moved or deleted between when the case is saved and when it is reloaded. The first option is safer, and is the default.

Choosing "Open Case" brings up a file chooser asking the user to specify the saved TeleMedMail case to load (see Figure 6.8). Opening a case restores all textual data, attached files, and extra image data (like crop and rotation information) that were in the case when it was saved.

TeleMedMail's saved cases have a ".tmm" file extension, but are actually just zip archives. The saved

Figure 6.7 – TeleMedMail Save Case Dialog

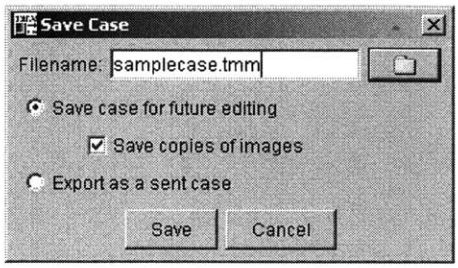


Figure 6.8 – TeleMedMail Open Case Dialog

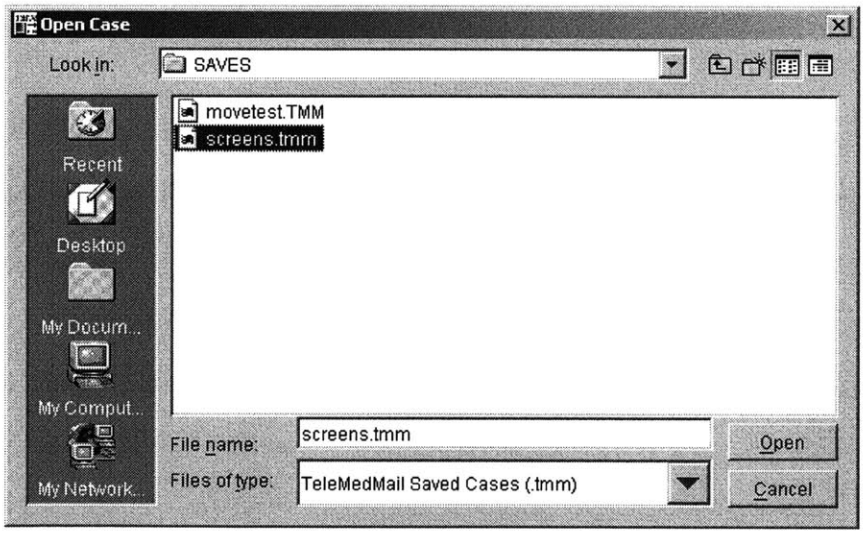


Figure 6.9 – TeleMedMail Configuration Dialog

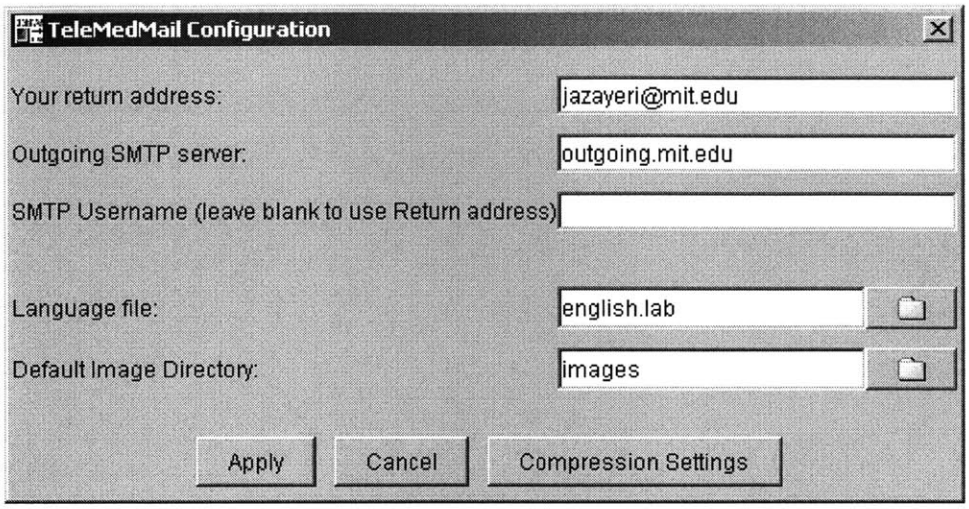
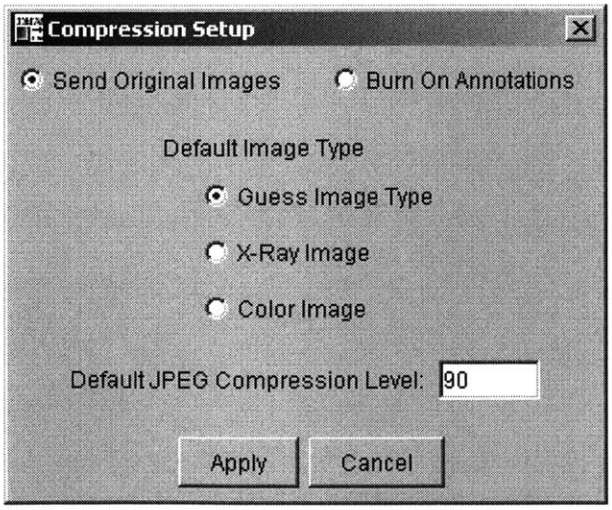


Figure 6.10 – TeleMedMail Compression Settings Dialog



case archive contains an XML file describing the case (see Appendix B) and possibly copies of attached files.

The open case dialog should ideally show the user a preview of the contents of the cases he is choosing from; this information could easily be extracted from the saved case archive and displayed.

6.5 Configuration

The user can configure TeleMedMail in via two configuration dialogs. The main configuration dialog (see Figure 6.9) is accessible from the Settings menu. Its most important purpose is to allow the user to set parameters related to sending email—SMTP server, username, and return address. This dialog also lets the user set the default image directory and language file (see Section 6.6.2), or call up the Compression Settings dialog (see Figure 6.10).

The Compression Settings dialog lets the user specify several settings generally related to image compression. The user can specify whether to "burn" annotations directly onto the images it sends (so that the images may be viewed with annotations in any standard image viewer) or whether it should send the original images and include the annotations in the case's XML descriptor. The user can specify whether TeleMedMail should by default convert images to black-and-white (e.g. if the user only sends X-Ray), treat them as color images, or try to guess their image type based on the variance in hue of the image's pixels. Finally, the user can set the default JPEG quality level. The values in the Compression Settings dialog are not likely to be changed by most users. When a future version of TeleMedMail can do other types of file compression besides JPEG, this dialog will become more useful.

The main configuration dialog pops up by default the first time the user runs TeleMedMail, prompting him to enter his email information. Few users will see the configuration dialog more than that once: TeleMedMail's default settings are appropriate for the vast majority of users.

6.6 Advanced Configuration

TeleMedMail can also be configured in two more advanced ways that are not accessible through the program itself. First, an advanced user can modify the standard text fields, either to change which fields appear in the GUI (for example to add a field for a common test in a particular specialty) or to change the

default values of the fields already there. Second, the user can change any of the text that ever appears in TeleMedMail (for example to make the GUI use Spanish or UK English instead of US English).

6.6.1 Standard Fields

The left side of the TeleMedMail GUI is dedicated to "Standard Fields," or named text fields that contain information about the case. Analogously, the Edit Comment dialog for an attached file (see Section 6.3.4) stores textual information about an attached file. These fields are weakly-structured: they can have default values, or drop-down lists of standard possibilities, but the user can ignore the defaults and enter arbitrary free text if he chooses.

These standard fields are not hardwired—at startup TeleMedMail reads two files that specify the types and default values of standard fields for the overall case, and on a per-attached-file basis. The file that describes the overall case standard fields is called "overall.stf", while the one describing per-attached-file standard fields is called "perfile.stf". They function identically except that per-file fields file can inherit fields from the overall fields file, as described below.

A standard fields descriptor file is a text file that describes the fields in the order they will be displayed. Each line of the file describes one field, and takes the general form

```
NameOfField = TypeOfField
```

with some extra modifiers. TypeOfField can be any of the following: STRING, LONGSTRING, AUTO-DATE, CASE-UID, CONFIG, ADDRESSBOOK, or SERVERS. Only the first two are useful to a user changing TeleMedMail's standard fields; the last five serve specific purposes, and are already used in the default standard fields descriptor file in ways that shouldn't be changed.

A STRING field is a one-line text field, which can have default values. These would be specified as follows:

```
PatientName = STRING("Darius", *"Alice", "Bob")
```

A field with several default values will be displayed along with a button that allows the user to choose from a drop-down menu of those values. If a field has just a single default value, or if it has multiple default values and one is marked with an asterisk, then the field defaults to that one value when TeleMedMail is started, and whenever the user starts a new case.

A LONGSTRING field is a multi-line text field, specified as:

```
PatientHistory = LONGSTRING<9>
```

where the parameter tells how many lines long the field should be. A LONGSTRING field can have default values, but it generally won't—a multi-line field exists for free-text entry, not for situations with just a few possibilities.

An AUTO-DATE field is a one-line text field that defaults to today's date. The user cannot specify another default value, as that would be meaningless in the context of "today's date."

A CASE-UID field is a special field that cannot be edited by the user. It shows the unique identifier for the current case, as described in Section 6.11.

A CONFIG field is a one-line text field whose drop-down list values are given in TeleMedMail's config.ini file. For example a field specified as

```
ReferringDoctor = CONFIG(ReturnAddress)
```

where the config.ini file contains the line

```
ReturnAddress = alice@mit.edu, bob@mit.edu, carl@mit.edu
```

will have a button that allows the user to choose from a drop-down menu of the three addresses given in the ReturnAddress config.ini entry. This field type exists for two reasons. First, it allows configuration-related values to function as standard field defaults (something most users want, as in the case of Return Address) without the user having to know anything about the standard fields descriptor file (which only very advanced users will even be aware of); second, it prevents information from having to be duplicated between the standard fields descriptor file and the TeleMedMail configuration file, where the two instances will potentially become out of sync.

An ADDRESSBOOK field and a SERVERS field are similar, and related. They are both one-line text fields that allow the user to choose from a drop-down menu of values from the user's address book. A SERVERS field's drop-down shows all the Server entries in the address book, while an ADDRESSBOOK field's drop-down list shows People entries. If TeleMedMail contains both a SERVERS and an ADDRESSBOOK standard field (which should always be the case), then the two fields interact.

In the default setup (see Appendix C), TeleMedMail has a SERVERS field and an ADDRESSBOOK field as follows:

```
ReceivingServer = SERVERS
ReceivingDoctor = ADDRESSBOOK(ReceivingServer)
```

If the ReceivingServer field has a value entered in it, and the user presses the drop-down-menu button for the ReceivingDoctor field, then the list will only show the Person entries in the address book whose affiliation matches the value in the ReceivingServer field, or who have no affiliation. On the other hand, if the user chooses a value from the ReceivingDoctor drop-down while the ReceivingServer field is still blank, then the ReceivingServer field will be automatically filled in with the server that the person chosen from the ReceivingDoctor drop-down is affiliated with. This interaction between ADDRESSBOOK and SERVERS fields does generalize to other fields of these two types, but there's no real need of it outside the ReceivingDoctor and ReceivingServer fields. (See Section 6.8 for a description of Servers and People.)

The user can also put a vertical spacer in the standard fields list, specified as

```
<SEPARATOR(6)>
```

in the standard fields descriptor file, where the parameter tells how many pixels high to make the spacer. This allows standard fields to be split into physical groupings, as in the case of "Sender and Receiver Information" versus "Clinical Patient Data." Physically separating groups of fields from each other can give the user hints as to how to work through the case most efficiently.

Per-file standard fields can be of one additional type not available in overall standard fields: they can be "inherited fields." If the per-file field descriptor file specifies

```
CaseAssemblyDate = @Date
```

then the set of per-attached-file standard fields will contain a field called "CaseAssemblyDate" which takes the value of the overall-case field called "Date." Any changes to one of the two will show up in the other. Inherited fields allow data regarding the overall case to be displayed when the specialist views an attached file, without requiring the referring doctor to enter that data multiple times.

TeleMedMail parses through the "overall.stf" and "perfile.stf" text files when it is run, and exits with an error message if either file is missing or misformatted. The default overall and per-file standard fields descriptor files are given in Appendix C.

Although the standard fields concept was designed to make TeleMedMail very generic, and allows a great deal of user customization, it is missing a generalization that would make TeleMedMail more powerful and useable: the user should be able to, within the TeleMedMail program, select one of several

sets of standard fields, each customized for a particular specialty. For example a dermatology referral should include a field like

```
LocationOfRash = STRING("Head", "Chest", "Arm", etc)
```

which would be less relevant in other specialties.

Rather than using a single overall-case standard fields descriptor file, TeleMedMail should, on startup, read a several such descriptor files, perhaps all those in a specified directory. The top-left corner of TeleMedMail's GUI, just above all the standard fields, could then contain a drop-down menu or a group of radiobuttons allowing the user to choose one particular set of standard field groups.

6.6.2 Text Labels and Internationalization

When the TeleMedMail program is run, it reads all the text that it will possibly display, from a text file specified as "LabelFile" in the "config.ini" file, and referred to in the program itself as a "Language File." This allows TeleMedMail's GUI to be easily translated into different languages merely by editing a text file, and without having to change any of the Java code. The default installation of TeleMedMail contains an English language file, and a Spanish language file: "english.lab" and "spanish.lab".

A language file is a standard text file, with a ".lab" (for "label") extension. Since Java is able to correctly read text files with non-ASCII encodings (e.g. Unicode), this text file can contain fully international text, with accent marks and non-English characters.

Each line in the language file specifies a "label", which maps an abstract name to a language-specific value. A label can be a static string, or it can contain parameters to be filled in by TeleMedMail during runtime. An example of a line in the English label file is:

```
ADDFILE_FileAlreadyThere = File (%1%) is already in case.
```

This label can be referenced from anywhere in the program by calling

```
TeleMedMail.getLabel("ADDFILE_FileAlreadyThere", filename);
```

The language file also specifies the keyboard shortcuts associated with menu items (e.g. "`KEYBOARD_Save = control S`".) to allow further internationalization. Finally, the language file specifies translations for TeleMedMail's standard fields, so that the standard fields descriptor file can be independent of language. (For example "`STF_PatientName = Nombre del Paciente`".)

The default English language files is given in Appendix D.

The user can select which language file to use from the configuration dialog (see Section 6.5). Choosing a different language file requires TeleMedMail to be restarted.

I have not tested TeleMedMail's ability to handle languages that are written right-to-left, or top-to-bottom. I expect that the code would need modifications to do so, such as specifying the Java Locale setting. Non-Latin left-to-right character sets, such as Cyrillic, do work properly.

6.7 File Types

TeleMedMail can import JPEG, GIF, and TIFF images. It converts all of these to JPEG images before sending them. TeleMedMail reads JPEG and GIF images via Java's built-in image loading capabilities; it reads TIFF images using an open-source Java image-processing library called ImageJ[21].

The user can also include any other type of file in a TeleMedMail case. Files of unrecognized types are included in the referral zip archive as-is, and are accessible via a link on the case's index.html page. Thus a case could conceivably include sound files or short video files (such as ultrasound), which the receiving specialist's web browser would handle with whichever plug-in it has registered for that file type. A more likely use of this feature might be to include a detailed patient history as an attached file, rather than in the provided standard field.

6.8 Address Book

TeleMedMail provides a simple address book for the user to store the email addresses of the specialists he will be referring cases to. This serves two main purposes: it saves the user the inconvenience of repeated typing in the same few email addresses, and it allows server public keys (for client-server mode referrals) to be handled transparently to the user.

All address book entries have a Name, and an Email address. (GUI elements that interface with the address book will usually let the user select from a list of names, and then map that to "Name <Email address>" in some way.) There are two different kinds of entries in the TeleMedMail address book: Server entries, and Person entries. A Server entry represents a hospital server that will accept case referrals, archive them, and forward them on to a specialist (i.e. a Server is a location running TMMReceiver. See Section 7.) Beyond the standard Name and Email fields, a Server entry specifies the name of the file

Figure 6.11 – TeleMedMail Edit Address Book Dialog

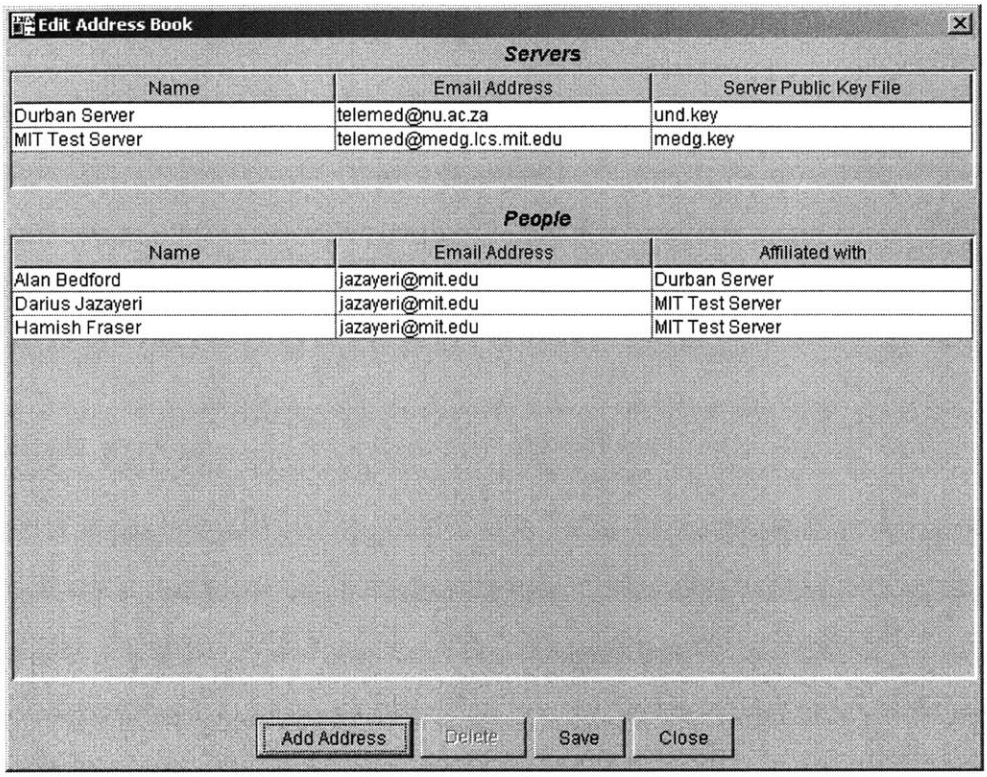
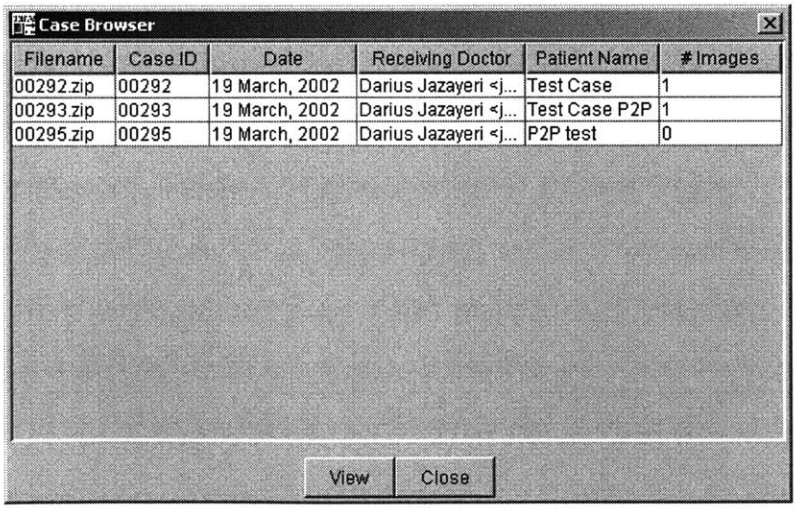


Figure 6.12 – TeleMedMail Case Browser



containing the server's public key, which TeleMedMail needs to send a referral to that server. A Person entry specifies which server, if any, the person is affiliated with.

Affiliating a doctor with a Server means that by default any cases referred to that doctor will actually be sent to him via his affiliated server. A doctor should generally be affiliated with a server that he has a fast Internet connection to, and where he has an account, although other usage models are possible. (See Section 12.1.)

TeleMedMail's address book component is actually a Java interface, rather than a class. (See Appendix E.) The config.ini file specifies which class (that implements the AddressBook interface) should actually be instantiated when the program is first run. This allows the address book component to be easily modified or improved without having to change (or understand) any of the rest of the TeleMedMail code. While the default address book class provided with TeleMedMail has sufficient functionality, its GUI is not production-quality—it is a likely candidate for open-source improvement. (See Figure 6.11.)

In the TeleMedMail system's most likely large-scale usage scenario, a member of the IT staff at a server hospital will go around and install the TeleMedMail software at the nearby client clinics who will be referring cases to that hospital. While doing so, this IT worker will configure the user's address book with a Server entry for the server hospital, and Person entries for all relevant specialists. (Most likely he will copy over a pre-compiled address book file, and the local server's public key.)

6.9 Sent Case Browser

TeleMedMail also provides a basic GUI for browsing through previously sent cases. (See Figure 6.12.) The browser shows some basic information about each case the user has sent (including date sent, receiving doctor, and patient name) and allows the user to view the full text

Like the address book, TeleMedMail's sent case browser is a Java interface (see Appendix F) rather than a class, and the actual instantiating class is specified in config.ini. The default case browser included with TeleMedMail implements that interface in a basic way, and the current version is meant just as a placeholder for the improved browser that will replace it in a future release of TeleMedMail. The current one has sufficient—though very basic—functionality, but its GUI is far from production-quality.

Figure 6.13 – TeleMedMail Secure Reply

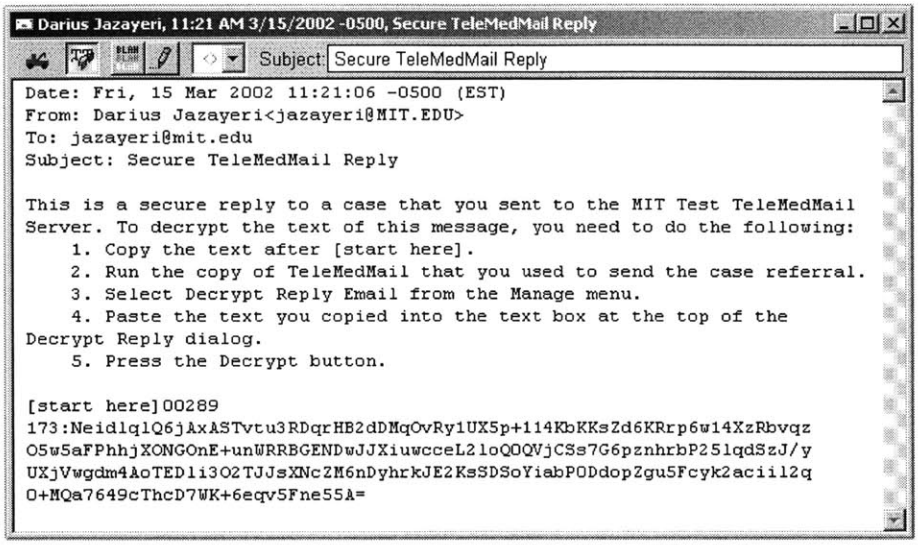
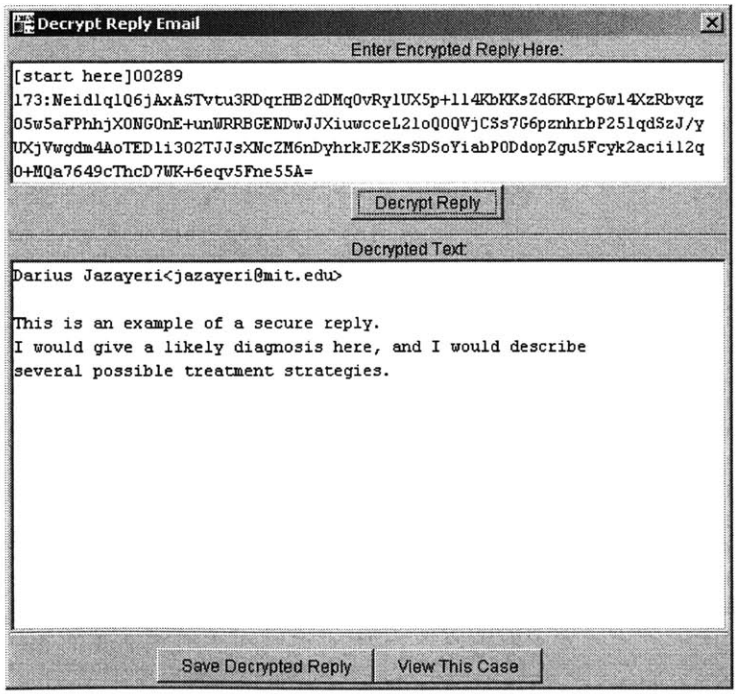


Figure 6.14 – TeleMedMail Decrypt Reply Email Dialog



6.10 Decrypt Replies

One of the ways the TeleMedMail system allows the specialist to reply to a doctor's referral is by encrypting the reply text with the secret session key that was used to send the case in the first place, and emailing it to the referring doctor. (See Section 8.1.4.) The referring doctor uses TeleMedMail to decrypt such a secure reply, through the "Decrypt Reply Email" dialog in the Manage menu. This dialog (see Figure 6.14) is very straightforward—it has a text area in which the user pastes the encrypted text, and a button he presses to indicate that TeleMedMail should perform the decryption, and a text area that shows the resulting decrypted text.

Once TeleMedMail has decrypted the reply text, it gives the user the option of saving the reply in the archive of the case it refers to, or of viewing the relevant case beforehand. (The user should always save a reply unless it was somehow corrupted in transit.)

A sample encrypted reply email is shown in Figure 6.13. The email contains instructions on how to decrypt its contents along with the encrypted contents themselves. The instructions and contents are separated by a line saying "`Copy From Here-->`" indicating what specific part of the text needs to be copied and pasted into the Decrypt Reply Email dialog.

The Decrypt Reply Email dialog allows the TeleMedMail system to send secure replies using standard email; as-is it is just a proof-of-concept of that possibility, and not meant to be used outside of a pilot study. Even more so than in the Address Book and Case Browser dialogs, the GUI needs to be redone to improve workflow and ease-of-training.

Beyond obviously needed cosmetic improvements, there are several easy changes to this dialog that would improve the workflow of decrypting a reply email. First, the dialog should support dragging-and-dropping of the decrypted text, as an alternative to copy-and-paste. The user could use whichever method he felt more comfortable with. Second, the dialog should automatically decrypt the contents of the encrypted text area whenever they change, rather than waiting for the user to press a button. Since the reply text will be placed there all-at-once by a paste or a drag-and-drop, not typed in a character at a time, it is easier for the user (and not computationally wasteful) to decrypt the text area whenever it changes. Third,

the dialog should recognize when the decrypted text is in the correct format (i.e. the email address of the doctor who sent the reply, followed by a line break, followed by the text of the reply) and automatically save the decrypted reply to the archive of the case it refers to. Replies that are decrypted correctly should always be saved—there's no reason to make it possible for the user to forget to press the Save button. Further, a single character data-transmission error will lead to unmistakably wrong decrypted text, so the likelihood of accidentally saving a garbled reply is effectively zero.

6.11 Case UIDs

A unique case number, called a "Case UID" identifies each TeleMedMail case. This number serves two purposes: first, it allows the sending and receiving doctors to discuss the case in an anonymized way over an insecure medium, and second, it allows encrypted replies to be automatically stamped with an indication of which case they refer to. This saves the replying specialist from having to include any case-identifying information in his reply (a waste of time, since it is generally irrelevant to actually diagnosing the case), and saves the referring doctor considerable work managing archived case files.

Each new TeleMedMail case starts without a UID assigned to it. Only when the case is sent does it actually get assigned a UID. Modifying a case (e.g. changing a standard field, attaching an image, modifying an attached image) after it has been assigned a UID causes the case to revert back to the unassigned-UID state. Further, if the same case is sent out multiple times, even without being modified, each sent instance gets its own UID. (The same case, sent twice, will have two different session keys—TeleMedMail needs to be able to distinguish which is which.)

If a doctor uses the TeleMedMail system in the preferred client-server mode, and replies are made on the web, or via secure email (see Section 8.1.4), then the user never has to pay attention to case UIDs. However, making the Case UID visible to the user allows TeleMedMail to be used in other ways as well, planned, (such as peer-to-peer usage mode, and client-server mode with unsecure replies) or ad-hoc. I decided to make case UIDs visible to the user in order to make the system more flexible, even though that risks confusing some users: placing a piece of important-sounding information prominently in the GUI, but intending for most users to ignore it, is not usually good user-interface design.

6.12 Java 2 and Installation

TeleMedMail is a Java 2 application. This made it easy and fast to develop, due to standard Java libraries for Internet connectivity, security, and graphics, but means that TeleMedMail needs a JVM with Java 1.2 or later standard classes to run. This creates some issues in distribution and installation.

Since most PCs (Windows or Linux) do not have a Java 1.2 runtime installed, TeleMedMail needs to be distributed with a Java Runtime Environment (JRE). The solution to this problem that I first experimented with was distributing Sun's latest JRE along with TeleMedMail, and having the user install the JRE before running TeleMedMail's installation. This turned out to be insufficient: although its installation process is straightforward, users sometimes found it difficult to install the JRE properly. The reason for this is not entirely clear, since users who had this problem were all remote and inaccessible until weeks after it occurred, but it may be due to the JRE installation not correctly setting the classpath environment variable on all versions of Windows.

To prevent users from having to install the JRE themselves, TeleMedMail's current installation CD contains the JRE files in unpacked form, rather than as a single installation executable. The installation copies the JRE files directly onto the users computer, placing them in the "c:\TeleMedMail\JRE" directory. Knowing the precise location of the JVM and classes allows the TeleMedMail to be run from the program directory with a command like "..\JRE\bin\java TeleMedMail".

Distributing the JRE like this has some incidental advantages. It allows the installation script to be written in Java, since it can use the (unpacked) JRE on the distribution CD. Thus installation can be arbitrarily powerful, without the monetary cost of an installer like InstallShield. In addition, since the absolute pathname of the JRE\bin directory is known at install-time, the installation program can copy a pre-made Windows shortcut to TeleMedMail onto the user's Desktop, which would otherwise be very difficult using Java.

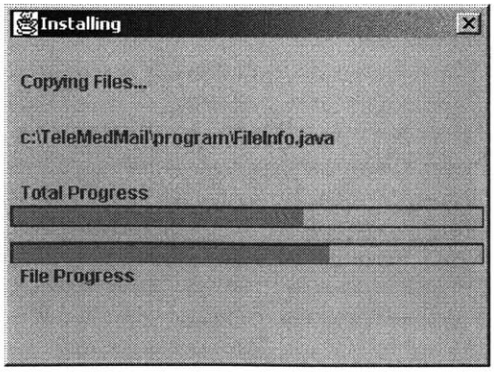
The downside of doing the installation in this way is that it installs a private copy of a large (~6.5 MB) program that was designed to be shared. If the user already did have Java installed, or if the user later installs another Java application, this would waste several megabytes of space. A further downside is that if the user installs later, improved versions of the JRE, TeleMedMail will see no improvement.

These disadvantages are all outweighed, however, by the main advantage of this installation method: it is as simple as can be. The user merely has to insert the TeleMedMail CD-ROM, double-click on the "Install TeleMedMail.bat" file it contains, and press an "Okay" once to accept the default installation directory. The JRE and all TeleMedMail files will be copied over to the users computer (see Figure 6.15), and a desktop icon will be created.

The standard TeleMedMail installation directory is "c:\TeleMedMail". Using this standard directory makes it relatively easy to send updates and bugfixes to users who aren't very computer-savvy. At present, these updates consist of a batch file which copies files into the appropriate TeleMedMail program directory; in the future, however, the TeleMedMail application will probably need some automated way of receiving updates and bugfixes.

One problem caused by TeleMedMail being written in Java, which has no workaround, is that the standard distribution must necessarily be large: it has to include several megabytes of zipped JRE files. (The non-JRE portion of the installation is under 1 MB.) This is a significant problem because remote users with good dialup connections may be able to download one megabyte, but will not even consider downloading 7.5 megabytes. As a result, TeleMedMail will have to be distributed on CD-ROM. This is not such an onerous requirement—the most common large-scale installation scenario will probably involve an IT worker from a server site physically traveling to a client site, and performing installation, configuration, and training. In this scenario, CD-ROM based installation is actually preferred.

Figure 6.15 - TeleMedMail Installation



7 TMMReceiver

The first of two software components at the server end of the Client-Server model is TMMReceiver, a Java application that runs on a hospital server, and handles incoming cases. TMMReceiver is implemented as a POP client: it downloads referred cases from an email account, decrypts them, places any attached files in the server's filesystem, and inserts the case data into a SQL database, where it can be served by TMMServer (see Section 8). Finally, TMMReceiver sends notification emails to the sending and receiving doctors.

TMMReceiver can be run either graphically, or in text mode. Graphical mode (see Figure 7.1) allows the user to configure the program more easily, while text mode (see Figure 7.2) is more flexible—it can be started remotely and in the background on a Linux server, for example. In text mode the user must edit TMMReceiver's configuration file manually.

TMMReceiver has two modes of operation: an advanced mode where it uses a SQL database to record received cases and manages permissions on those cases, and a basic mode where it merely places unzipped cases in a directory to be served by an external web-server application.

In advanced mode, the TMMServer database front-end takes care of user-specific security, but it still requires the JSP server to use SSL to encrypt communications. In basic mode, the external web-server must provide security, by using SSL and requiring a user password.

The actual operation of the program is very straightforward. TMMReceiver runs in the background on a hospital's server and, at regular intervals, connects to a POP mail server, downloads any messages and deletes them from the server. It then handles all downloaded messages.

A downloaded message consists of two parts: the body of the message contains the IDEA session key, encrypted with the receiving server's RSA public key, while the message's one MIME attachment contains the case data in a ZIP file, encrypted with the session key. Handling a downloaded message entails the following steps:

1. Decrypting the body of the message, with the server's private key.
2. Extracting the IDEA session key from the decrypted body.
3. Extracting the attached file from the message.

Figure 7.1 – TMMReceiver Graphical Mode

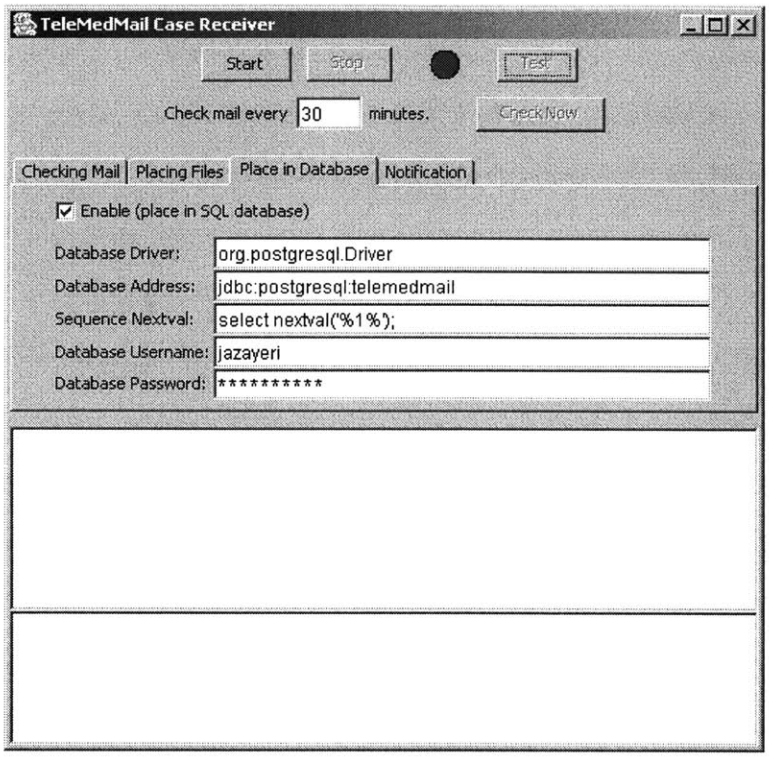


Figure 7.2 – TMMReceiver Text Mode

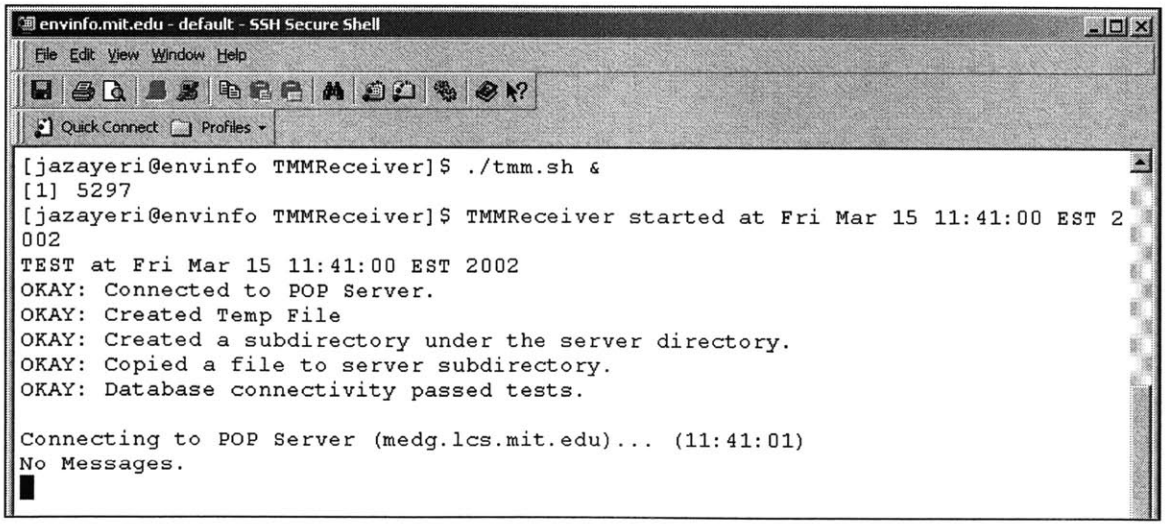


Figure 7.3 – TMMReceiver Received Case Notification Email

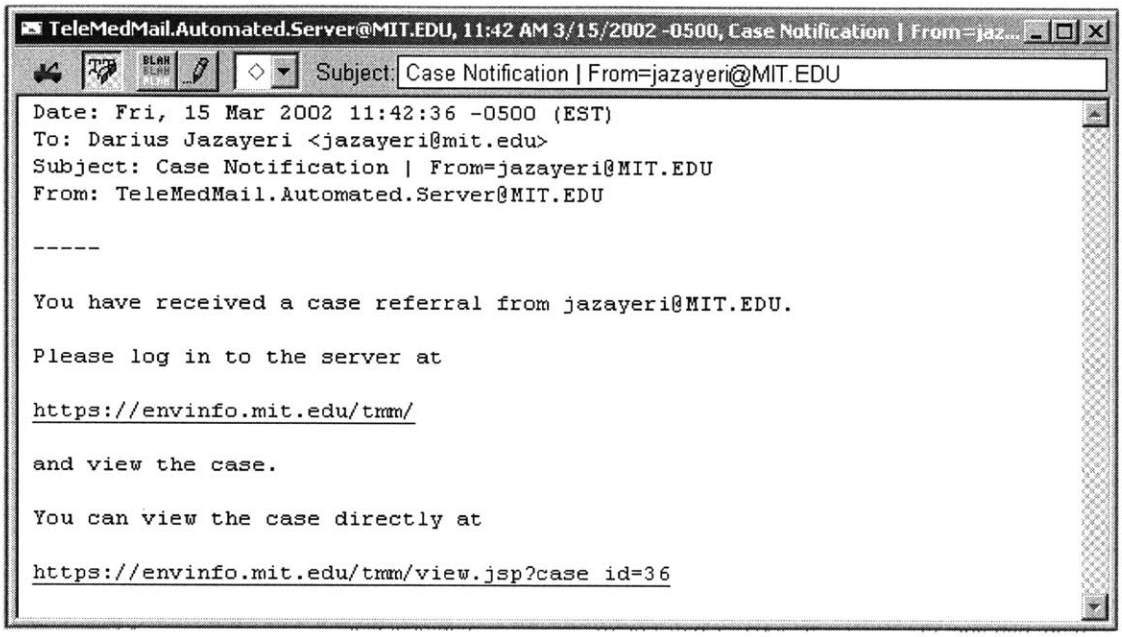
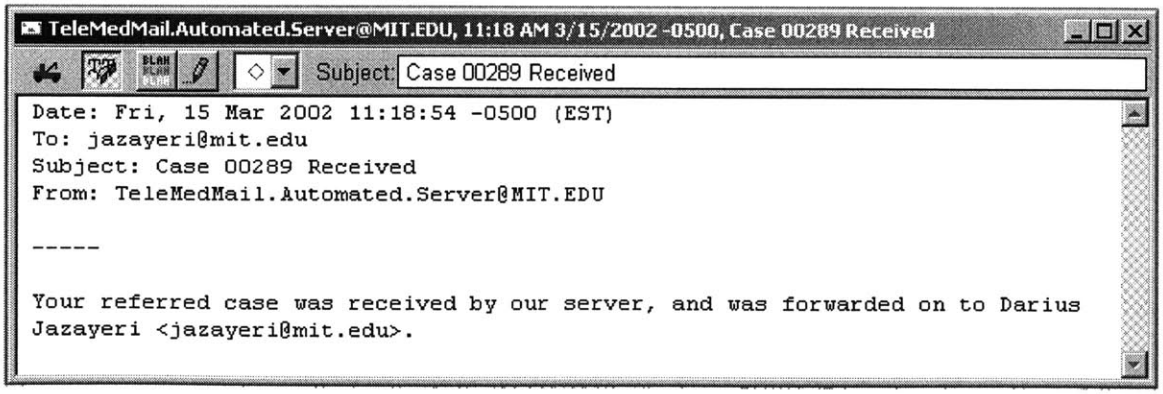


Figure 7.4 – TMMReceiver Received Case Reply Email



4. Decrypting the message attachment with the session key.
5. Unzipping the resulting ZIP file and placing its contents in the server's filesystem.

In advanced (database connectivity) mode, TMMReceiver then:

6. Places the case details in a SQL database.
7. Grants permission on that case to the receiving doctor.
8. If the receiving doctor has an account on the server:
    - Notifies the sender the case was received. (See Figure 7.4.)
    - Notifies the receiving doctor of the received case, along with a link to TMMServer's login page and a direct link to the case. (See Figure 7.3.)
9. If the receiving doctor does not have an existing user account on the system:
    - Notifies the receiving doctor that he needs to request an account on the system, with a link to the TMMServer account-request page.
    - Notifies the sender that the receiving doctor will only get the case when he requests an account on the system.

If TMMReceiver is configured to run in basic (web-server only) mode, it then:

6. Notifies the sender the case was received.
7. Notifies the receiving doctor of the received case, with a link to its web-viewable index file.

When running in advanced mode, TMMReceiver makes a receiving specialist's workflow very simple and straightforward. He receives an email notifying him that the server has received a case referral for him, and clicks on the link in that message. (See Figure 7.3.) The specialist then enters his username and password to log into the system, and views the case as described in Section 8. All in all, the process only requires a handful of mouse clicks, and a password dialog. (Of course manually typing in the reply is unavoidable.)

The workflow is even less for a specialist who receives many TeleMedMail referrals: he can handle several referrals while only having to log in to the system (and enter a password) once.

TMMReceiver's basic mode automates decryption and unzipping of cases, improving the workflow of the TeleMedMail system over point-to-point referral mode, but it has two major shortcomings. First, it doesn't allow the specialist to send encrypted replies back to the referrer, thus requiring replies to be made

outside the system (by phone, for example) or else requiring that replies don't include any patient-identifying data. Second, it doesn't keep any structured record of past cases. TMMReceiver's advanced mode is much preferred, as it solves these two problems.

The database structure used by TMMReceiver is simple, and really only serves as a proof of concept—it doesn't allow any EMR functionality beyond permission-management. Getting sophisticated functionality in the real world would require integrating TMMReceiver with the hospital's existing EMR system. This is not difficult, because of TMMReceiver's open-source design, but it is a highly EMR-specific task.

The SQL code used to produce this database structure is given in Appendix G. Case information is stored in *cases*, *patient_details*, and *case_details*, and *replies* tables. Information about files attached to those cases is stored in *attached_files*, *attached_images*, *image_annotations*, and *image_fields* tables. User information is stored in a *users* table. User permissions on cases are given in the *permissions* table.

7.1 External Applications

The fundamental design choice in TMMReceiver is to use external applications to do all the real server-related work. This decision happens three times: I chose to use an external SSL web server, SQL database server, and SMTP mail server, rather than specialized components to serve, store, and receive cases. None of the three decisions are difficult: they each allow the system to be more flexible and powerful, while making implementation easier. For the first two, this is obvious, as SSL and SQL are the industry-standard ways of storing and serving this sort of data, and provide all the necessary features with no downsides. Implementing TMMReceiver as a pop client (using an existing mail server) rather than as a specialized SMTP server (capable of receiving mail by itself) is a similarly easy decision. Although the latter method would have offered the advantage of allowing the server to know as soon as a referral email was received, rather than several minutes later when polling a mail account for messages, this advantage is very minor, and the former method is far easier to implement.

7.2 Further Work

TMMReceiver's day-to-day operation is straightforward, and acceptable for large-scale use in its current form. However one enhancement needs to be made to TMMReceiver's configurability before it achieves wider use, so that administering it requires less specialized knowledge.

The text of the emails that TMMReceiver sends to the referring doctor and receiving specialist under various circumstances is specified directly in TMMReceiver's code. As a result, changing those messages (e.g. translating them for use with Spanish-speaking doctors) requires modifying Java code, and recompiling TMMReceiver. Although the fragments of Java code that need to be modified are trivial, requiring system administrators to know a particular programming language is not acceptable. TMMReceiver needs to be modified to load a text file specifying the contents of the various emails it sends, thus allowing a TeleMedMail server to be administered by an IT worker who knows nothing about Java.

8 TMMServer

TMMServer is the front-end to the database that TMMReceiver places cases in while running in advanced mode. TMMServer allows specialists to log in and view referrals that have been sent to them, and reply to those referrals. It also allows referring doctors to log in and see replies to their cases. TMMServer is implemented as a JavaServer Pages based web application that can run on any JSP-enabled web-server. (It could just as easily be implemented in any other server-side scripting language that can connect to a SQL database and generate HTML output.)

TMMServer also provides very basic administrative user-management capabilities. These will need to be enhanced before the system achieves widespread use.

8.1 Workflow

8.1.1 New Account Application

When TMMReceiver receives a referral for a specialist who does not have an account on the system, then it sends that specialist a notification that he has received a case, along with a link to TMMServer's New Account Application page. It is of course preferable for the TeleMedMail system administrator to introduce a specialist to the system and create an account for him before this happens. But since TeleMedMail's flexibility allows ad-hoc doctor networks to spring up unplanned, this may be the first interaction that some specialists have with the system.

The New Account Application page is shown in Figure 8.1. The user is required to enter his name, email address, requested username, and requested password. He may also specify whether he wants to get the low- or high-bandwidth version of pages when he logs in.

When the user submits his account application, TMMServer enters it into the *users* table in the database, marks it as inactive, and notifies the administrator via email that there is an account request waiting to be approved.

Once the administrator approves the account, TMMServer sends the user an email with instructions on how to log in to the system (see Figure 8.2).

Figure 8.1 – TMMServer Account Application Page

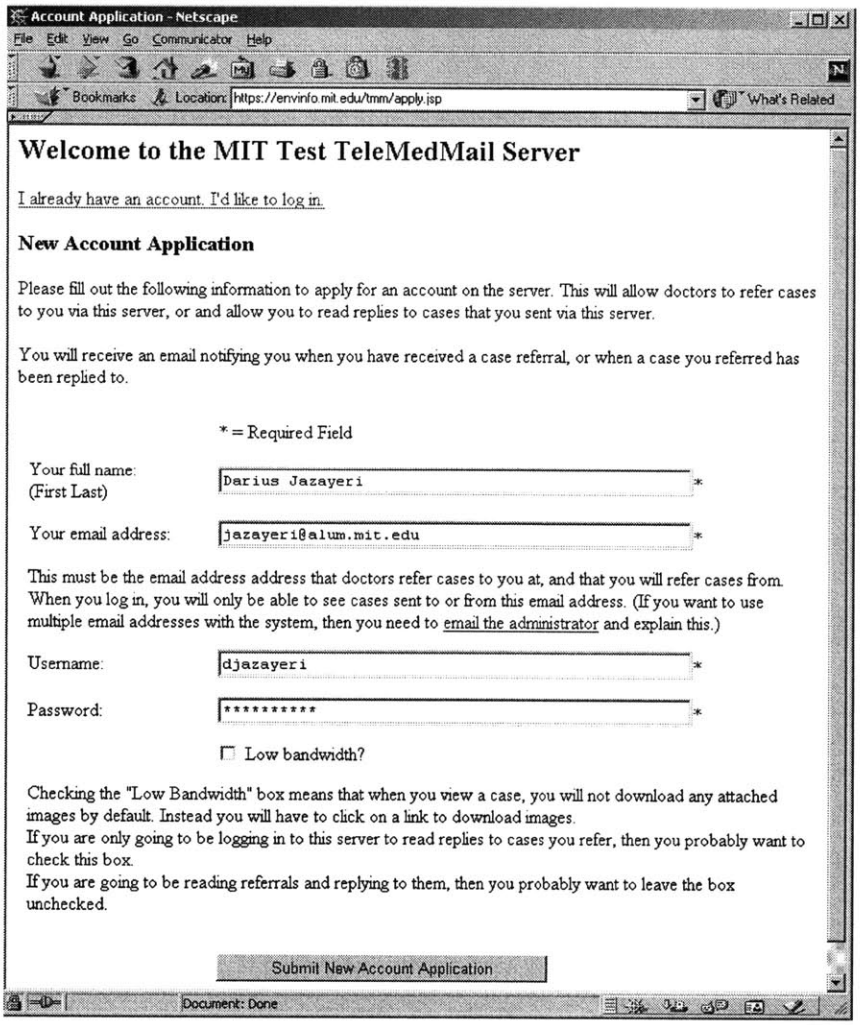


Figure 8.2 – TMMServer Account Activation Email

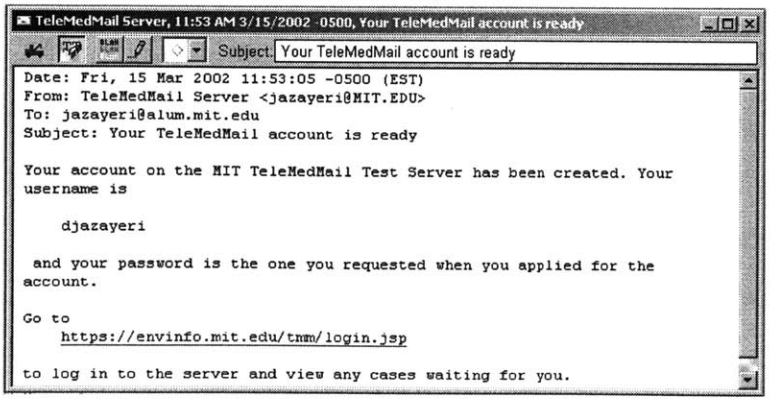
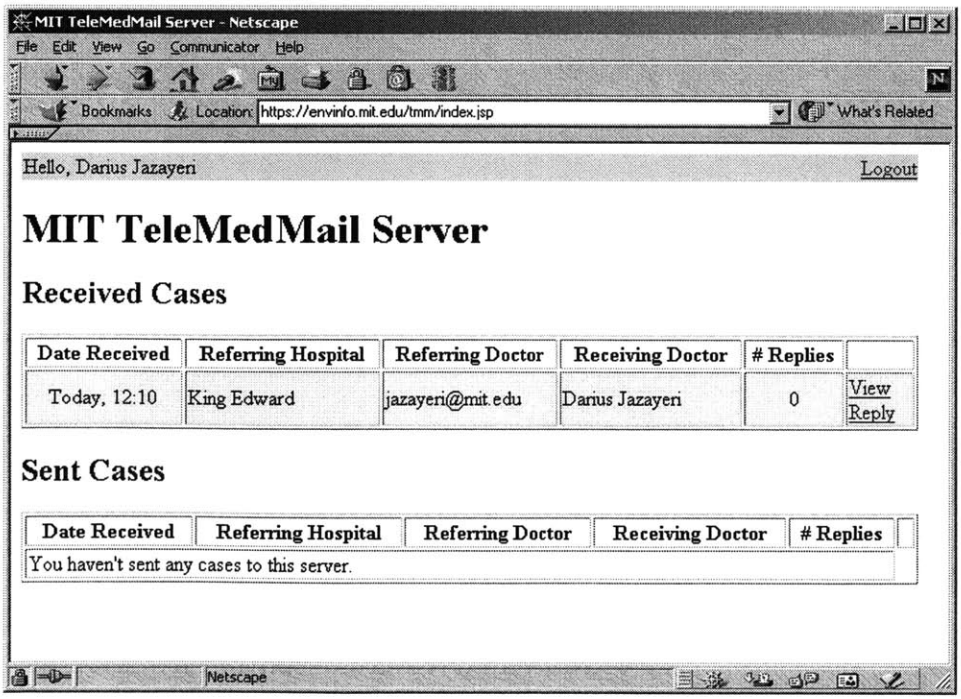


Figure 8.3 – TMMServer Login Page

Figure 8.4 – TMMServer Case Index Page



8.1.2 Login and Case Index

TMMServer's login page (see Figure 8.3) is very straightforward, merely prompting the user for his username and password. Once he has logged in, he is taken to the case index page. (Email notifications of received cases and replies include a link that will take the user directly to the relevant case or reply after he logs in. If the user reaches TMMServer by clicking on one of these, then he will skip the index page entirely.)

The Case Index page (see Figure 8.4) shows the user a chronological listing of all the cases he has received, followed by a similar listing of all cases he has sent. The received-case list has links next to each case that allow the user to view or reply to that case, or to view previous replies to that case, if any exist. Cases that are still awaiting replies are highlighted.

Each entry in the sent-case list has links allowing the user to view the case, or read replies that have been made to that case. Cases that have already received replies are highlighted.

The Case Index page, like all pages the users sees after he has logged in, has a navigation bar at the top, showing him who he is logged in as, allowing him to log out of the system, and allowing him to step back to more general pages, if appropriate. (In the Case Index page, there is nowhere to step back to; all other pages can at least step back to the index page.) In the future, this navigation bar may allow more advanced features, such as user-to-user messaging.

8.1.3 Viewing Cases

When the user clicks on the "View" link for a case on the Case Index page, or when he clicks on a direct-to-case link in a TMMReceiver case-notification email, he is taken to the View Case page (see Figure 8.5). This page shows textual case details (i.e. Standard Fields, see Section 6.6.1), followed by attached files. Attached images are shown first, and then attached non-image files.

The View Case page displays attached images differently depending on whether the user has requested low-bandwidth pages or not. For each image, a high-bandwidth user will see an image thumbnail, a button to view that image in the Viewer applet, and a link to view the image directly in his browser, without the applet. (The button is in fact the applet itself. See Section 9.) In low-bandwidth mode, a user will merely see the title of the image, and a link that will load the attached images. (See Figure 8.6.)

Figure 8.5 – TMMServer View Case Page

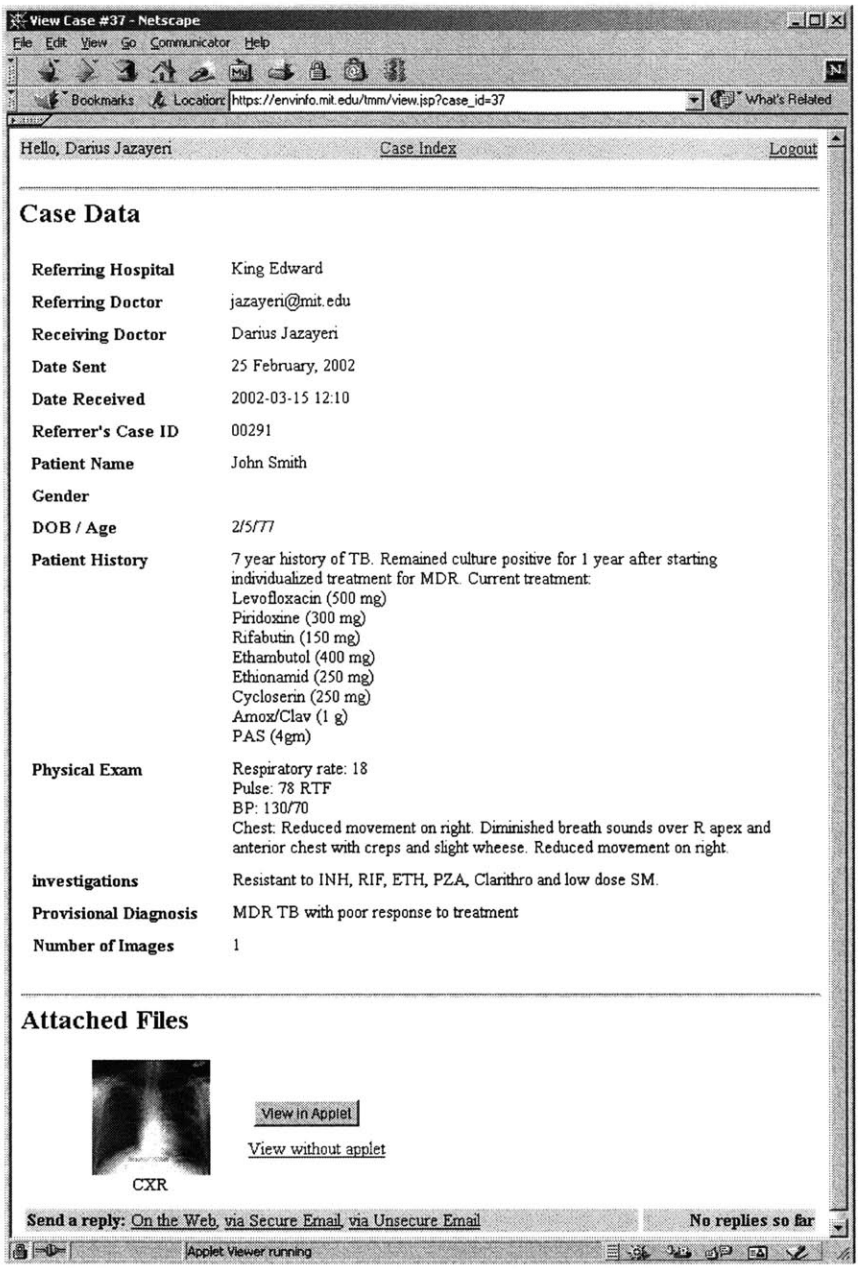
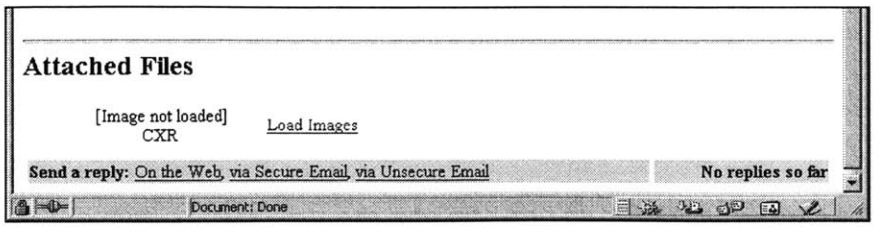


Figure 8.6 – TMMServer Low-Bandwidth View Case Page



Attached non-image files are displayed the same way in both low- and high- bandwidth versions of the page: the title of the attached file, followed by a link to the file. The user's web browser is responsible for handling the file—it will generally guess its content type based on its file extension, and open it, save it, or offer to download a plug-in to view it. Handing off unknown file types to the web browser makes the TeleMedMail system very flexible: a TMMServer installation should be enhanced by adding Java viewers for specific medical file types it will receive that are not handled by standards browser plugins, but even if this is not done, unrecognized file types are handled gracefully.

The navigation bar at the top of the View Case page has a link back to the Case Index page. The bottom of the View Case page has a Reply bar, which allows the user to reply to a case, or view any replies that have already been made to this case. The Reply bar has links for each of the TeleMedMail system's three types of replies.

8.1.4 Replies

The TeleMedMail system allows specialists to send replies with three different transport mechanisms: the text of the reply can be posted on the TMMServer website, it can be IDEA-encrypted and sent via email, or it can be sent as a plain-text email. The reply-on-web mechanism is the preferred one in all situations where it can be used; in situations where it cannot, the secure email mechanism should be used. The unsecure email mechanism should not really be used at all, but the TeleMedMail system provides in the interest of flexibility, and for emergency situations where a speedy, instantly-readable reply is more important than a secure one that may take extra minutes to read. (The TMMServer administrator can disable any of the three transport mechanisms, and in some scenarios may disable the unsecure one.)

The appearance of the pages for the three different types of replies (see Figures 8.7, 8.8, and 8.9) is similar overall, to emphasize the fact that it is only the transport mechanism that differs, and not the content of the message. Still, they have subtle differences. Each page looks generally like an email message, with To, From, and Subject fields filled out to the correct values by TMMServer, a large text area for the user to enter the text of his reply, a description of the message transport mechanism that will be used, and a button to send the reply.

Figure 8.7 – TMMServer Send Web Reply

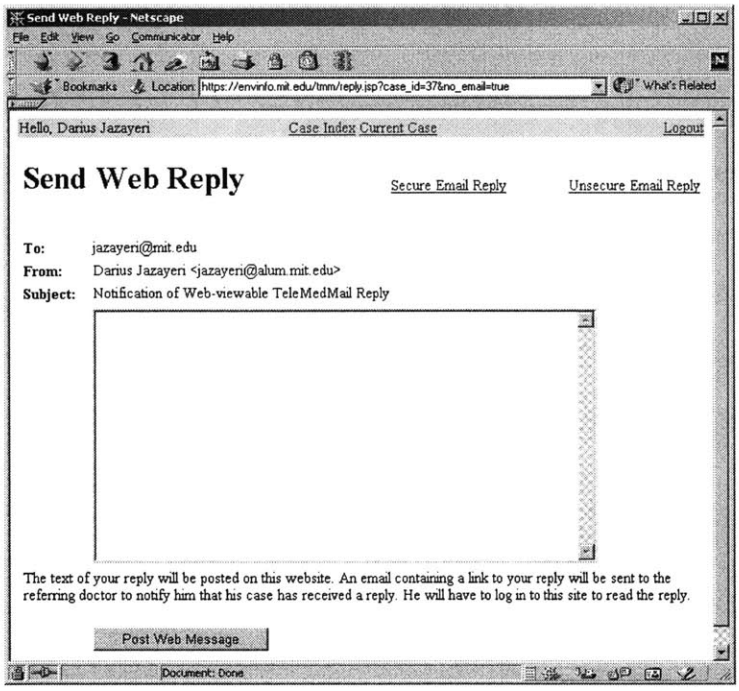


Figure 8.8 – TMMServer Send Secure Email Reply

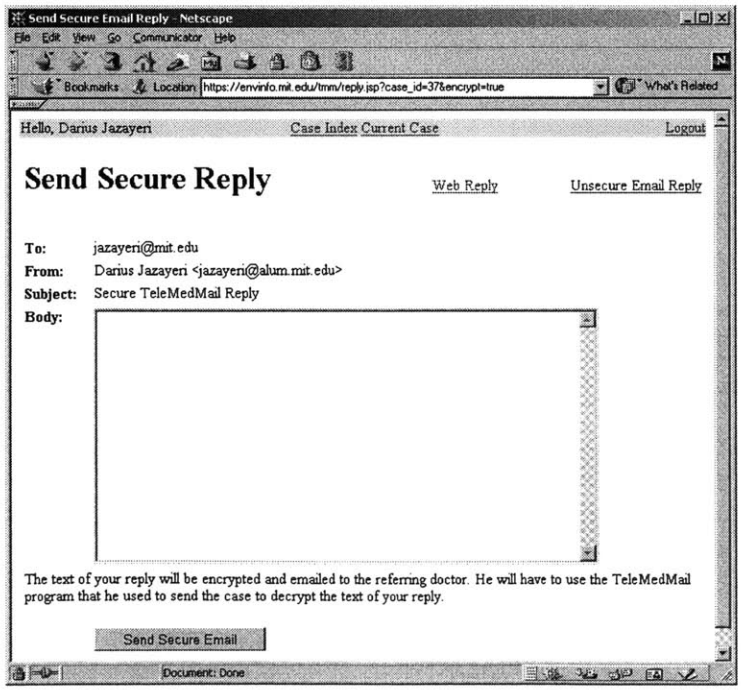


Figure 8.9 – TMMServer Send Unsecure Email Reply

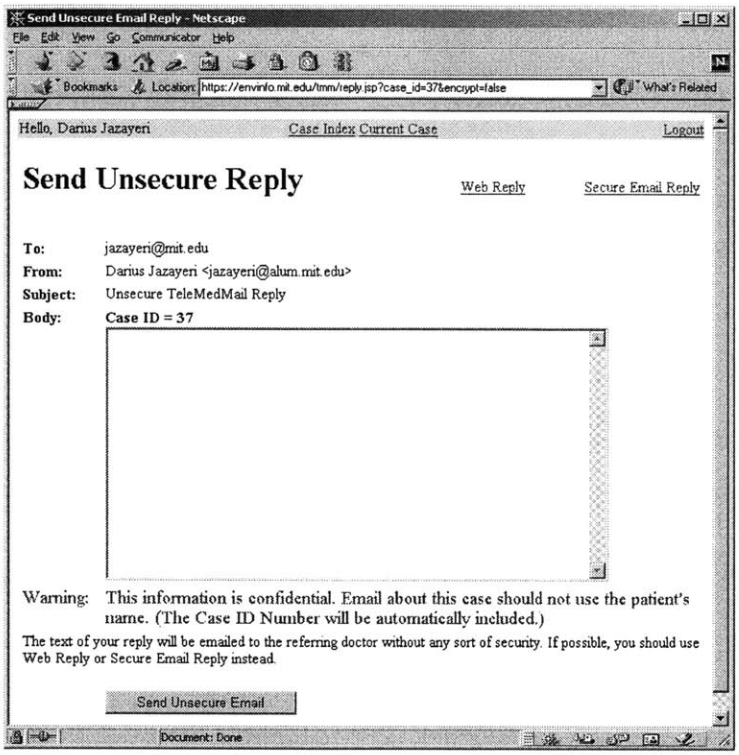


Figure 8.10 – TMMServer User Management

In addition to this, the Unsecure Email Reply page shows an obvious warning, in large red text, that the text of the reply should not contain the patient's name, and a reminder that TMMServer will automatically include the appropriate Case ID number.

The navigation bar at the top of the Reply pages has links back to the Case Index page and the View Case page.

8.2 Administration

TMMServer's administrative toolset is currently very limited, providing only the bare minimum of functionality necessary to run the TeleMedMail system. When an administrator logs is, he is taken to the standard Case Index page, with three changes:

1. His received-case list shows all cases sent to any doctor.
2. Each displayed case has an extra link, to delete that case.
3. His navigation bar has an extra link, which takes him to the User Management page.

8.2.1 User Management

TMMServer's User Management page (see Figure 8.10) allows an administrator to approve and create accounts, view user account information, or assume the identity of any active user. The top of the page shows a list of user accounts awaiting approval, while the bottom of the page shows a list of active user accounts.

The most important function of the User Management page is to allow the TMMServer administrator to approve accounts for users that have requested them. To approve an account, the administrator just has to select the checkbox of an account in the awaiting-approval list at the top of the page, and press the "Activate" button. Activating an account will send a notification to that user that his account is ready, and move that account from the awaiting-approval list to the active list at the bottom of the page.

An administrator can also create a user account without waiting for that person to request an account, via a link at the top of the page.

As an aid to troubleshooting, the administrator can also log in to the TMMServer system as any of the active users by clicking on that user's username in the active list. This allows the administrator to see pages exactly as a particular user would, which can help him better answer questions about the system.

8.3 Further Work

TMMServer is currently little more than a simple front-end to the SQL database storing referred cases. It has sufficient functionality to demonstrate the possibilities of a store-and-forward telemedicine system, but it lacks features that might make it more powerful for real-world use. The lack is not fundamental to the TeleMedMail system—these features can easily be added at a later date.

One shortcoming of the current implementation of TMMServer is its simplistic view of case permissions. At present, the specialist to whom a case is referred, and the doctor who referred the case, are the only two users with permission to view a case; only the receiving specialist can reply to that case. This makes sense, since some users who have accounts on a particular TMMServer instance (such as a remote referring doctor whose account exists only to view replies) should not have permission to view and reply to all cases on the server.

This shortcoming could be fixed by implementing the idea of different user groups and roles, but that solution would complicate the system considerably. A much simpler improvement would be to allow any user with permission on a case to grant permission on that case to other users as well. This would allow fairly sophisticated case permission management, while keeping TMMServer just as simple as it currently is. Specialists could ask a colleague for help on a difficult case, or "refer" it along to someone with more relevant knowledge; the referring doctor could ask a second specialist for an opinion on a case without having to send an entire new TeleMedMail referral. Allowing a case's receiver to grant permissions to other users would also make it possible for TeleMedMail to be run as a sort of clearinghouse system, where all referrals to a hospital would go to a single user, who would then assign them to the appropriate specialists, thus relieving the referring doctors from the burden of keeping track of where various specialists work.

A trivial but necessary improvement would be to allow the administrator to delete or deactivate users, and to modify their email addresses and passwords—currently there is no web front-end for this, and the administrator has to do it directly in the SQL database.

Another trivial adjustment involves having TMMServer store some sort of information about referring doctors, specifying whether they can read replies on the website, or whether they need to have replies delivered to them by one of the email methods. (This information could be used to pick the correct default

reply mechanism for a particular recipient.) A third would be to establish two distinct types of user accounts, one geared towards receiving cases, and one towards sending them, and have visually different index pages for each user type.

A more significant weakness of the current TMMServer implementation is that it doesn't allow any sort of sophisticated user-to-user messaging. One particular way this can be fixed is by allowing referring doctors to post messages about cases they sent to the server. (TMMServer would then have to notify the receiving specialist, and anyone else who previously replied to that case, that there was a further development or question.) Currently only the receiving specialist can post replies, so if the referring doctor has any questions about the specialist's reply, the system doesn't provide any way to address them. Allowing him to post replies on the TMMServer website would allow such follow-up questions to be asked in a secure manner, and make the entire referral process more of a dialog.

9 Viewer

Viewer is a very small Java Applet (~21 kb) that the receiving specialist uses to view images attached to a TeleMedMail case. Web browsers already have the built-in capability of displaying images, but the simple image display they provide does not allow the sort of viewing that a specialist needs when looking at clinical images. In particular, while looking at an image in a web browser, the user must view the image at full-size. Since digital camera images will often be larger than the specialist's screen (most clinical image types should be *at least* 1600x1200, while a typical screen resolutions is 1024x768), he would often not be able to see the whole image at once. This is not acceptable—at the very least it will be an inconvenience to the specialist, and more likely it will hurt his diagnostic accuracy. The problem of zooming an image can be easily solved by a Java applet, which can perform arbitrarily complicated image manipulations.

Viewer pops up an external window (see Figure 9.1) with a small toolbar on the left, and a large image display on the right. The toolbar has four sections: zoom controls, window level controls, the navigation thumbnail, and annotation view controls.

The zoom controls allow the user to scale the image to fit in the window, view it at full resolution, or zoom in or out. Zooming in or out enlarges or reduces the size of the image, to the nearest power-of-two zoom. (Thus if the image is currently at 35% zoom due to fit-in-window, and the user presses "Zoom In," the image will scale to 50% zoom.) I chose to use only power-of-two zooms when possible because that probably minimizes the zooming artifacts that are inevitable when viewing the image at anything other than full resolution.

The window level controls allow the user to brighten and darken the image, either by pressing the "Darken" or "Brighten" buttons or by directly entering high and low window levels. ("Window levels" is a radiological term, described below in Section 9.1.) The user can also choose to invert black and white in the image display, possibly making some image features more visible.

The navigation thumbnail is a scaled down version of the image, with a box highlighting the portion of the image that is currently visible on the right side of the window. The user can drag the mouse around on the navigation thumbnail to move the rectangle around, thus scrolling the view on the right side of the window. (The user can also scroll the image by dragging the view on the right side around directly. The

Figure 9.1 – Viewer Window

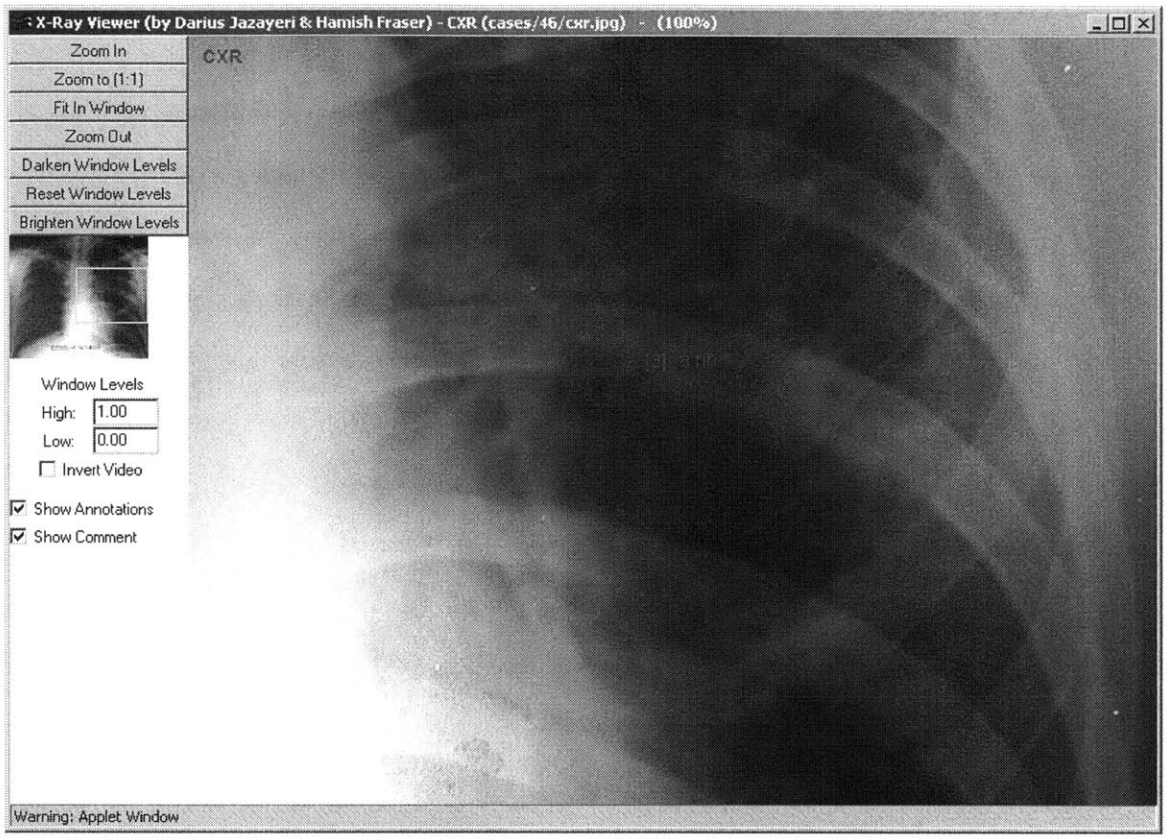
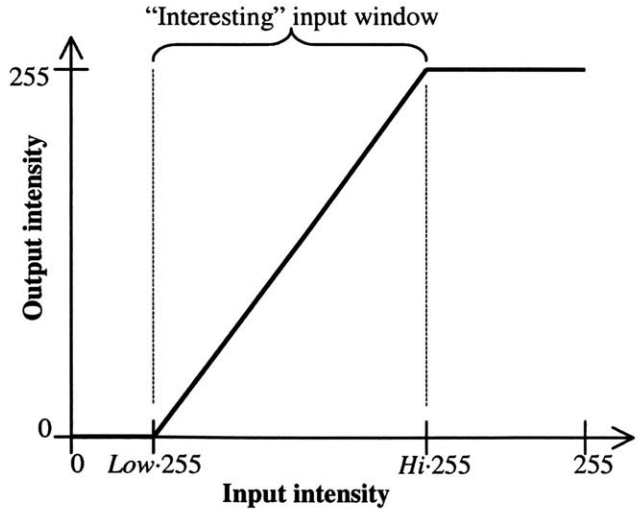


Figure 9.2 – Viewer Window Levels Intensity Transfer Curve



navigation thumbnail merely provides another mechanism, which can provide a better perspective when Viewer is zoomed in.)

The annotation view controls allow the user to specify whether or not Viewer should display annotations, and whether or not it should show the image comment. When visible, annotations are drawn directly on the image; the user might want to hide them if one is obscuring a feature of the image. The comment is displayed in a fixed position towards the top of the image display on the right side of Viewer's UI. Since it doesn't scroll with the image, the user will rarely need to hide it.

Both the comment and annotation are drawn in red. This makes them easily visible on many images (such as X-rays), but very difficult to see on red-biased images (such as some pathology slides). Viewer should be enhanced to automatically display text in a color that doesn't show up in much in the rest of the image, or at least allow the user to toggle between different text colors.

The text of the image comment that Viewer displays is specified in the HTML file containing the APPLET tag for Viewer—as such it is specified by TeleMedMail when it sends the case in peer-to-peer mode, or by TMMServer when it generates the "view.jsp" page. This comment text will generally consist of the per-file standard fields (see Section 6.3.4) concatenated together.

Once it has been downloaded and initialized, the Viewer applet shows up in its containing web page as a button labeled "View In Applet." (The button is not visible before Viewer is ready to display its image.) When the user presses the button, Viewer brings up the image and associated toolbar in a new window. (Since the new window is a Java Frame object rather than a new browser window, the user's web browser will generally label it with some sort "Untrusted Applet Window" warning.)

9.1 Window Levels

Viewer provides some simple image filtering capabilities, allowing the user to change the image's brightness and contrast, or to invert light and dark. This filtering is done on a per-pixel basis—it is not affected by other nearby pixels, or global image characteristics.

Regardless of whether it is displaying a color image or a grayscale image, Viewer uses the standard Java Image class, at the default 32-bit RGB color depth, with the alpha channel set to be fully opaque. The

red, green, and blue channel values can vary from 0 to 255; for a grayscale image, all three channels have the same value at each pixel.

Each color component of each pixel in the image is modified by a filtering function before being drawn to the screen. This function, which maps actual pixel channel intensity to displayed pixel channel intensity, is piecewise linear, and depends on three user-specified parameters: *High*, *Low*, and *Invert*. (See Figure 9.2.)

High and *Low* are real numbers between 0 and 1. Color components with values below (*Low* × 255) are mapped to 0, while color components above (*High* × 255) are mapped to 255. The intensity transfer function between those two values is a straight line connecting the points (*Low*×255, 0) and (*High*×255, 255).

These controls are referred to as Window Level controls because *High* and *Low* together specify a window of interesting color component values which is mapped to the entire 8-bit displayable range.

Invert is a boolean value which, when true, specifies that the color component value resulting from the transfer function just described should be inverted (i.e. subtracted from 255) before being drawn to the display.

If the user chooses, he can enter values of *High* and *Low* directly as real numbers. More likely, however, the user will change window levels by pressing the Darken and Brighten buttons. The former increases *High* if possible and if not increases *Low*, while the latter decreases *Low* if possible and if not decreases *High*. For a single press of either button, the total change in *High* and *Low* is 0.1, unless the image is already so bright or so dark that this is not possible.

Enhancing the image by changing the window level does not happen instantaneously, and depends on the speed of the user's computer. Future work should go into optimizing the window-level algorithms for speed. The results of Brightening and Inverting an image are shown in Figures 9.3 and 9.4.

9.2 Multiple Languages

Viewer's is built such that its user interface can be easily translated into different languages. The APPLET tag in the HTML file containing Viewer can specify all strings that show up in the UI with parameters.

Figure 9.3 – Effect of Brightening an Image (After/Before)

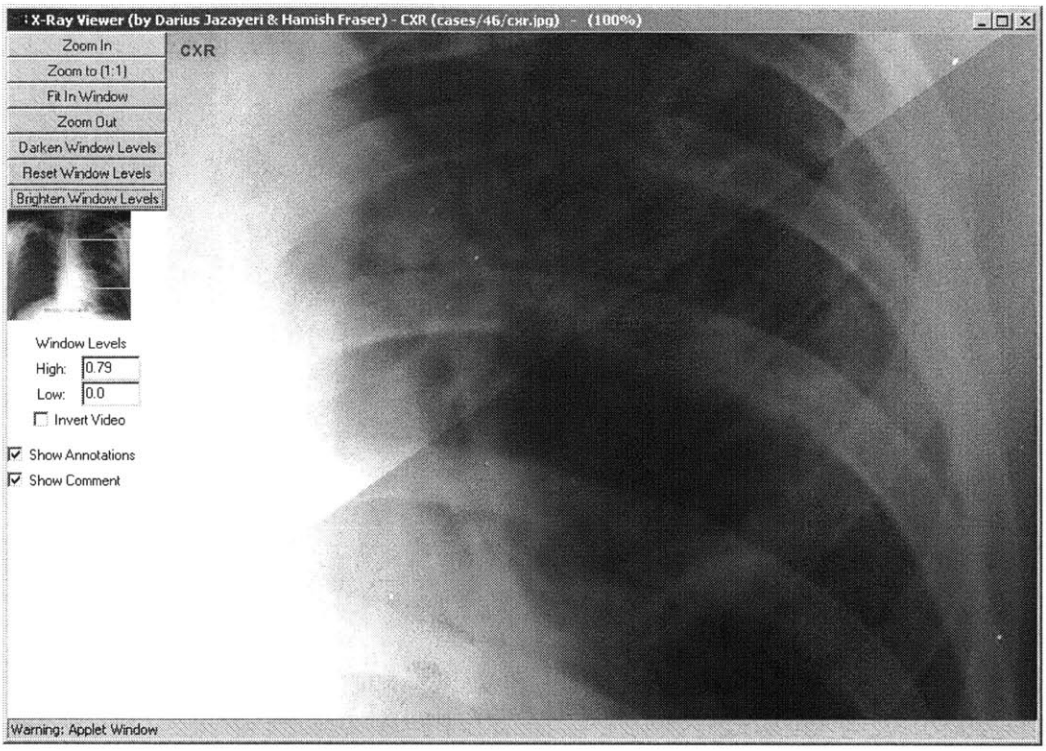
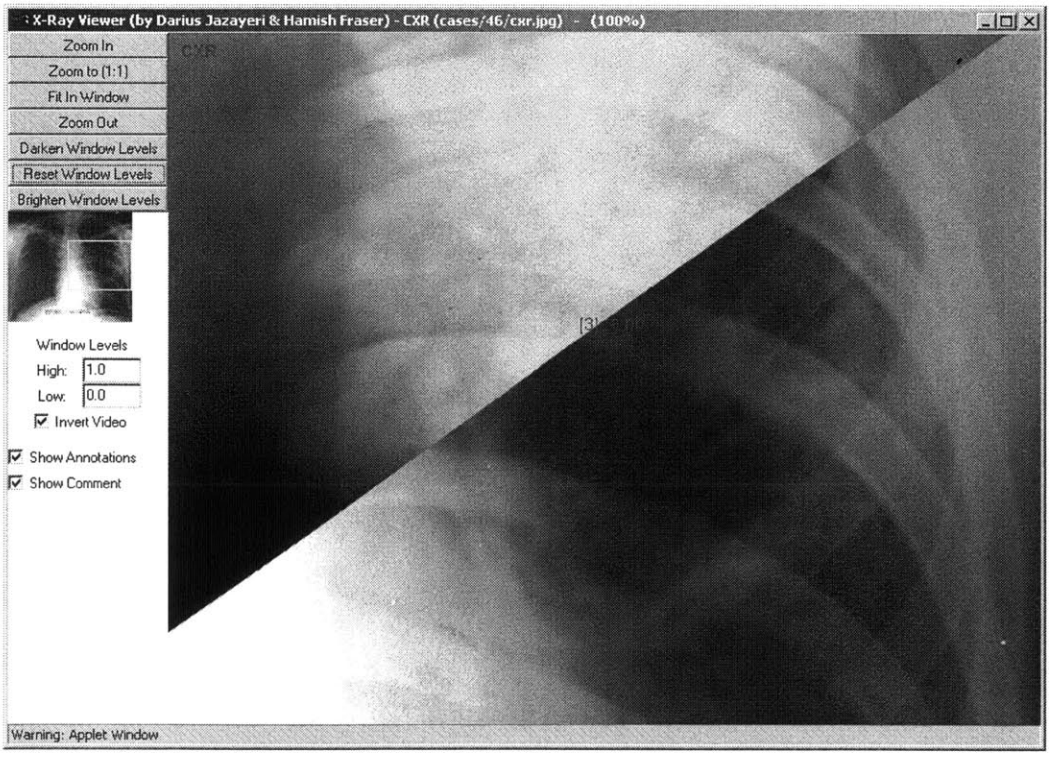


Figure 9.4 – Effect of Inverting an Image (After/Before)



Any labels not specified in the HTML file default to standard phrases in English. The configurable parameters, and their default values, are shown in this HTML fragment:

```
<APPLET CODE="Viewer.class" ...>
    ...
    <PARAM NAME="VIEWBUTTON" VALUE="View in Applet">
    <PARAM NAME="ZOOM" VALUE="Zoom">
    <PARAM NAME="ZOOMIN" VALUE="Zoom In">
    <PARAM NAME="ZOOMOUT" VALUE="Zoom Out">
    <PARAM NAME="RESETZOOM" VALUE="Zoom to (1:1)">
    <PARAM NAME="ZOOMFIT" VALUE="Fit In Window">
    <PARAM NAME="BRIGHTEN" VALUE="Brighten">
    <PARAM NAME="DARKEN" VALUE="Darken">
    <PARAM NAME="RESETBRIGHTNESS" VALUE="Reset">
    <PARAM NAME="WINDOWLEVELS" VALUE="Window Levels">
    <PARAM NAME="WINDOWLEVELHIGH" VALUE="High">
    <PARAM NAME="WINDOWLEVELLOW" VALUE="Low">
    <PARAM NAME="INVERT" VALUE="Invert Video">
    <PARAM NAME="SHOWANNOTATIONS" VALUE="Show Annotations">
    <PARAM NAME="SHOWCOMMENT" VALUE="Show Comment">
</APPLET>
```

10 CaseViewer

CaseViewer is a small Java applet (~39 kb) that the receiving specialist uses to view a peer-to-peer referral. (Viewer only displays a single image, while CaseViewer displays the entire case, text and images.) Once unzipped, a peer-to-peer referral consists of an index.html file (which contains the APPLET tag necessary to load CaseViewer, and the XML information describing the case, IDEA-encrypted, and stored in a PARAMETER tag for the applet) and any files attached to the case. (These attached files are not encrypted.)

The CaseViewer applet shows up in an HTML page as a button labeled "Open Case" (see Figure 10.1). When clicked, the user is prompted for the case password. CaseViewer tries to use the user-entered password to decrypt the XML ciphertext; if the resulting decrypted text is of the correct format (i.e. `<TELEMEDMAIL TYPE="SEND"> . . .`) then the password was correct, otherwise CaseViewer prompts the user for the password again.

Once the user has entered the correct password, and CaseViewer has decrypted the case XML, it opens a new window containing the case data. (See Figure 10.2.) The left side of the window shows the textual case data (i.e. overall-case Standard Fields from TeleMedMail as described in Section 6.1.2), while the right side shows thumbnails of the attached images, along with a button to view each image. These buttons begin grayed out, and only become active when CaseViewer is prepared to display the relevant image. This preparation happens in a background thread, so as not to prevent the user from viewing the rest of the case while it happens.

When the user presses one of the image buttons, CaseViewer displays the image in a new window that looks exactly like the window brought up by Viewer (see Section 9), and has exactly the same functionality and interface. CaseViewer's image-displaying code is the same as Viewer's, with very minor modifications: the variables specifying UI-component labels are taken from the CaseViewer class, and the window popped up is a Dialog window rather than a stand-alone Frame window.

The index.html page containing the CaseViewer applet also has links to any non-image attached files (CaseViewer wouldn't be able to handle them—that is left to the web browser's standard mechanisms), and a mailto link allowing the specialist to send a reply back to the referring doctor.

Figure 10.1 – Peer-to-Peer HTML Index Page

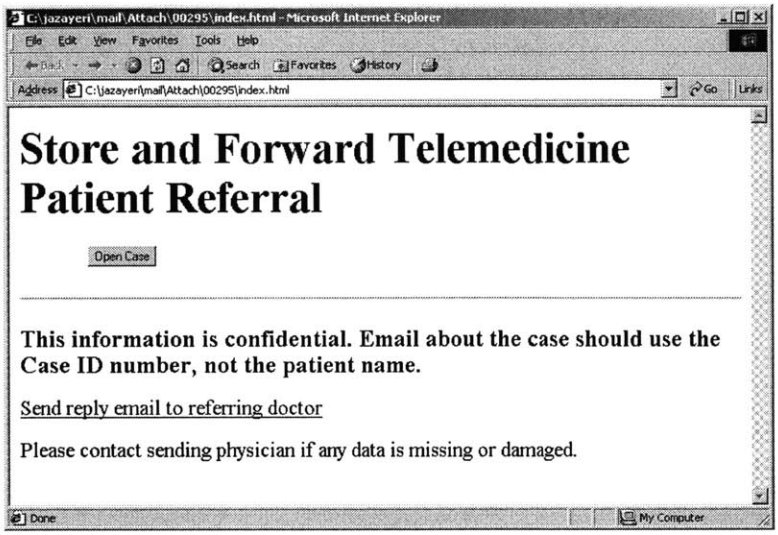
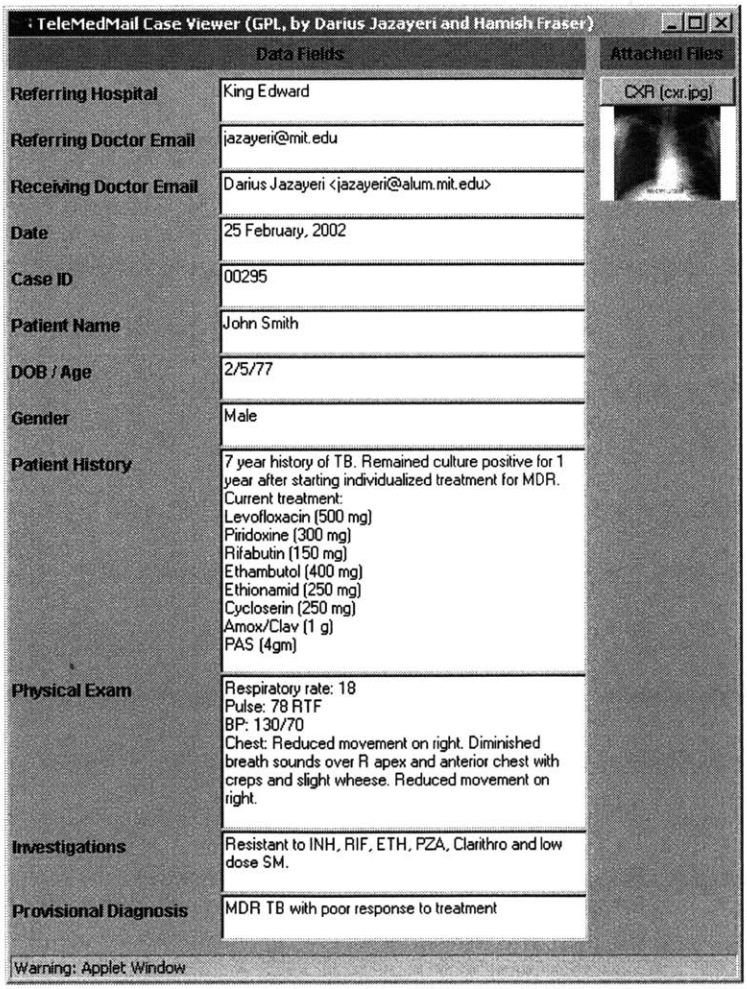


Figure 10.2 – CaseViewer Window



10.1 Further Work

The TeleMedMail system's peer-to-peer referrals do not provide any mechanism for the specialist to send a secure reply back to the referring doctor. But CaseViewer already includes all the Java classes necessary to generate such a reply. CaseViewer should be extended to allow the specialist to type in a reply to the case, and then to IDEA-encrypt the text of that reply with the case password. CaseViewer could then produce instructions and ciphertext for the user to copy and paste into his standard email client, and send to the referring doctor. The referring doctor's TeleMedMail application has an archived copy of the case password, and can decrypt the reply with its existing Decrypt Reply Email function.

One shortcoming of peer-to-peer referrals is that attached files are not encrypted, so to ensure patient confidentiality, they should not include any patient-identifying information. This is not a desirable situation, however, since in general files *should* contain patient-identifying information, in case they ever become separated from the XML information that describes them. In the case of image files, this can be fixed with a small change in the image-loading code, and a change in TeleMedMail to specify in the XML whether an image is encrypted or not.

In the case of non-image files, there is no easy solution to this problem: CaseViewer could decrypt them, but due to security restrictions on Java applets, it would not be able to write the decrypted file to disk. To address this shortcoming, CaseViewer would need to request permission to write to disk, which involves making CaseViewer a signed applet. This would inconvenience the user, as few people are used to granting permission to signed applets. It would also introduce security risks—even an unsophisticated adversary could easily exploit users' acquired propensity to grant write-permission to applets.

11 Security Risks

TeleMedMail has all the security risks of a distributed computer system: passwords can be compromised due to human error or cracked by brute-force attacks, or an adversary can break into a client or server machine and steal data. These risks are unavoidable in any distributed computer system, and the system administrator should be familiar with how to deal with them.

The TeleMedMail system has two security risks particular to its design, besides familiar issues common to all client-server systems. First, secure reply emails from the server to a referring doctor are encrypted but not authenticated, so they are vulnerable to a limited type of forgery attack. Second, referring clients do not have their own RSA keypairs, so a server does not know for certain whether a referral really comes from the client it claims to have been sent by.

Both of these attacks are extremely unlikely: they require far greater effort and expertise than would plausibly be employed by an attack on a remote, poor area of a developing country.

11.1 Forged Replies

An adversary can break in to the client or server computer, steal an archived session key, and then use that key to send an encrypted reply to the client who referred that case. This forged reply might contain an incorrect diagnosis, which could lead the referring doctor—trusting the reply—to perform a harmful procedure on the patient. (This attack requires the adversary to write some Java cryptography code—an unlikely scenario.)

Even if replies were authenticated, a successful break-in at the server end of the system would still allow an adversary to send this kind of forged reply. Since replies are not authenticated, a break-in at the client end of the system allows the same forgery. This is significant, because the client end of the system is likely to be physically less secure than the server.

On the one hand, an adversary who breaks into the system at the client end can do significantly more damage by other means than by forging a reply, so this type of attack is not a big deal. On the other hand, this kind of attack could go unnoticed, while more damaging attacks (deleting files, etc.) have a noticeable impact.

This risk can be avoided entirely (assuming the server is never compromised) by using on-the-web replies instead of secure email replies (see Section 8.1.4). Since on-the-web replies are the preferred reply method, most users will already be immune to this risk.

One change that would make this kind of attack more difficult would be to have the server sign the ciphertext of secure email replies with its private key. (This requiring TeleMedMail to decrypt incoming replies first with the public key of the server it sent the case to, and then with the archived session key for that case—not difficult, since TeleMedMail has a copy of the public key, and a record of which server that case was sent to.)

An intruder who changes the server public key file in TeleMedMail's keystore could circumvent this additional security measure. So a further security enhancement, which would make forged replies nearly impossible, would be to modify the TeleMedMail client application to require a password, and to use that password to encrypt sensitive information (i.e. the sent-case archive, and the keystore). This would prevent an intruder with access to the client computer from being able to copy archived session keys, or modify server public keys.

Under the current TeleMedMail system, the risk of forged replies is acceptably small for pilot studies and small- to medium-scale use. Specifically, it requires an attacker to break in to the client machine, and such a break-in would cause larger security problems. The TeleMedMail client application should be enhanced with a password before large-scale use, however.

11.2 Spurious Referrals

Under the TeleMedMail system, referring clients do not have any way of certifying their identity. As a result, a server cannot know whether a case was really referred by the client that the case claims to be from. This is not really a security risk, as it doesn't allow any data to be stolen or modified, but it does allow an adversary to send case referrals to a server, claiming they are from some other (real or fake) person. This could be a type of Denial of Service attack, causing receiving specialists to waste time reading bogus cases.

Another variant of this attack would be for an attacker to get a receiving specialist to diagnose a real case, but charge a referral fee to some other referring client. (In order to succeed, this type of attack would require that the user network be *very* large, or that the system administrator be particularly inattentive.)

In small- to medium-scale uses of the TeleMedMail system, these types of attacks would be quickly noticed, because the system administrator and users would be relatively familiar with each other. Very-large-scale use of the system, however, would probably require that messages be cryptographically authenticated. Large-scale health-care providers whose concerns go beyond the purely medical (such as a multi-hospital organization or an entire country) should probably not use the TeleMedMail system until these changes have been made.

Solving this problem, by requiring client locations to register their RSA public keys with servers before sending cases actually makes the TeleMedMail system considerably less flexible, especially before it is widely installed. Registering public keys takes time on an administrative scale, which is generally slow in developing countries. This would mean, for example, that a doctor who travels to a remote clinic and installs TeleMedMail there would probably not be able to use it for a considerable amount of time.

For TeleMedMail's initial users, especially those involved in well-supervised pilot studies, the small risk of spurious referrals is far preferable to the considerable inconvenience and lack of flexibility required to prevent that risk. Thus the changes necessary to eliminate this risk are not warranted unless they are particularly required in a specific situation.

12 Conclusions

The TeleMedMail system fulfills its design aims of security, ease-of-use, automation, low cost, flexibility, and openness. As such it makes the large-scale use of low-cost store-and-forward telemedicine a real possibility; this can be a powerful tool, allowing developing countries to spread the reach of their expertise.

Telemedicine has been shown to be cost-effective in some specific situations[22]. TeleMedMail is a simple, efficient, and inexpensive system that can make telemedicine more widely useful, allowing its use by less computer-savvy doctors.

But there is a larger question. Beyond lacking specialists, developing countries often lack the funding and resources necessary for even very basic health services. Is it worthwhile to divert a portion of that scarce funding towards telemedicine infrastructure, when the cost of a single laptop and digital camera could buy hundreds of doses of basic vaccinations? The existence of the TeleMedMail system does not answer these questions. Rather it helps lower the costs of telemedicine (software, training, and labor) and makes it easier to perform studies quantifying the real costs and benefits that telemedicine will bring to specific clinical areas in specific locations and scenarios.

In addition to gathering data for future cost-benefits analyses, these sorts of pilot studies will also provide valuable user-driven insight into the strengths and weaknesses of the TeleMedMail system, and suggest the ways in which it should be modified and improved.

After pilot studies have been carried out, and a few minor improvements are made to the TeleMedMail system (see Section 12.2), low-cost store-and-forward telemedicine will be a useful tool for developing countries' health-care systems, with evidence to support its real utility and costs.

12.1 Other Usage Scenarios

12.1.1 Remote Specialists

I designed the TeleMedMail system under the assumption that all specialists would be located at central hospitals with good IT support. In practice this is usually, but not always, the case. Remotely located specialists do exist, and they are just as valuable as centrally located ones. In fact they are more valuable in some ways: since remote specialists may be physically located in underserved areas that have no easy

access to large central hospitals, they may be able to recommend more practical local treatment, or that the patient travel to their clinic for further consultation. A centrally located doctor might not be able to make such a recommendation.

The options that the TeleMedMail system provides to such a remote specialist are not sufficient in all cases. Under the constraints of the system, the preferred way for a remote specialist to receive case referrals would be for him to have a user account on a TMMServer installation that he can reach via a dialup web connection. This is satisfactory for most remote specialists—the idea of giving server accounts to specialists not actually located at that hospital is very flexible and powerful—but does not work in two cases: if the specialist has only email access and cannot use the web, or if the specialist needs a local copy of case referrals, for offline viewing.

In these two scenarios, where a remote specialist cannot use an account on a server, he instead has to rely on peer-to-peer referrals. But TeleMedMail's peer-to-peer referral method is designed for one-time communications rather than frequent traffic, and as a result does not optimize the workflow of the receiving specialist. In particular, a specialist who receives many TeleMedMail peer-to-peer referrals has to unzip attachments and archive past cases in the same inconvenient way a specialist not using the TeleMedMail system would have to.

The best solution to this problem would be to enhance the TeleMedMail application such that it automates the process of receiving, decrypting, and archiving cases. Instead of opening received cases with his standard unzip client, the specialist would open them with (enhanced) TeleMedMail.

This particular solution has two key strengths. First, it would not require any changes to the case-sending portion of TeleMedMail: as such it would happen completely transparently to the referring doctor. Second, these enhancements could directly use TeleMedMail's existing code: referring doctors could automatically be put in the Address Book, and cases could be displayed using the existing Case Browser infrastructure.

Implementing this TeleMedMail enhancement would make handling peer-to-peer referrals nearly as easy as handling client-server referrals. But the actual implementation of this TeleMedMail enhancement should wait on one important thing: a demonstrated need. It is not worthwhile to expend effort enhancing

the system unless there actually are enough specialists who need the enhancement because they cannot use TeleMedMail's existing client-server referral method.

12.1.2 Trusted Third-Party Record-Keeping

Although the TeleMedMail system was designed under the assumption that an installation of TMMServer would serve several specialists at a single institution, this is not a requirement. As alluded to in the discussion of remote specialists, the fact that specialists with server accounts can be physically far from the server makes the TeleMedMail system very flexible. One particular usage scenario it allows is that of an institution serving as a third-party record-keeper and arbitrator of disputes. A well-staffed and trusted institution such as a University teaching hospital could run a TeleMedMail server, and provide an account to any interested specialist. This server would store (and backup) a permanent record of the interactions between referring doctors and specialists. This would both allow specialists to accept telemedicine referrals without needing any local IT expertise, and allow any disputes about cases to be resolved by a trusted third party.

This record-keeping function works even though referring clients do not have RSA keypairs: any case residing on the server is guaranteed to have been unaltered since it was sent. Even the risk of forged replies (see Section 11.1), does not affect this arbitration function: any reply stored on the server is guaranteed not to have been the result of such a forgery attack following a break-in at the client.

The trusted server does not solve the problem of spurious referrals (see Section 11.2)—it has no way to be sure that a message is actually from the person it claims to be from. But since this cannot lead to misdiagnoses, only to Denial of Service-style wasted specialist time, it is an acceptable risk if the server is well administered and monitored.

12.2 Further Work

In its current state, the TeleMedMail system is perfectly sufficient for use in pilot studies measuring the costs and benefits of store-and-forward telemedicine. However, a few minor improvements need to be made before it is ready for daily use. In addition, it will probably need one fundamental change before being used in a very-large-scale enterprise, such as an entire country's medical system.

12.2.1 Minor Improvements for Broader Use

The TeleMedMail system's software components need the various minor improvements already mentioned in previous sections. Among the key ones:

TeleMedMail needs to have multiple different sets of standard fields, each customized for a particular clinical specialty. (See Section 6.6.1.)

Pilot studies can use a single standard field configuration, since they will probably geared to study referrals of a particular clinical type, but a general-purpose telemedicine application needs to provide its users with the flexibility to choose what type of referral they are sending, and provide data required by that particular specialty.

TeleMedMail's Address Book, Case Browser, and Decrypt Reply Email components need to have their GUIs reworked. (See Sections 6.8, 6.9, and 6.10.)

The present user-interfaces of TeleMedMail's plug-in components provide the necessary functionality, but their appearance is not as professional, and their workflow is not as well streamlined, as the rest of the TeleMedMail application.

TeleMedMail should be password-protected. (See Section 11.1.)

In order to lessen security risks in scenarios that are larger-scale and not as well monitored as a pilot study, TeleMedMail should password-protect the sensitive data it contains.

The text of reply and notification emails sent by TMMReceiver and TMMServer needs to be easily configurable. (See Section 7.2.)

Presently, the text of reply and notification emails TMMReceiver and TMMServer send is defined directly in the Java and JSP code. This is not a problem in a pilot study with a highly skilled system administrator, but if general users should be able to translate and customize email text in a single text file, without having to see any Java code. This same configuration file should also specify other information (e.g. the name of the server) that will need to be changed at each installation of the TeleMedMail server.

TMMServer should have more sophisticated case-permission management. (See Section 8.2.)

At the very least, the TMMServer administrator should be able to see which cases do not belong to any user, and to grant permissions on those cases to the appropriate doctor, without having to access or modify

the SQL database directly. (As an example, if a referring doctor's Address Book contains an incorrect email address, the TMMServer administrator can assign cases sent to that wrong address to an appropriate specialist, without requiring the referring doctor to resend the case.)

In addition, letting users to grant other people permissions on their cases will make the TeleMedMail system more flexible, and allow interesting usage scenarios. (See Section 8.3.)

Since this improvement requires such a small change in TMMServer, yet would make the whole system much more flexible, it should be implemented before putting the system in large-scale use. In fact, as soon as this change is implemented, a pilot project should be designed to study the strengths and weaknesses of the current TeleMedMail system as compared to a clearinghouse-style system where all referrals to a server go to the TMMServer administrator, who then assigns them to the appropriate doctors.

Peer-to-peer referrals should encrypt images. (See Section 5.2.)

I have discovered a simple modification to the CaseViewer applet and to the format of peer-to-peer XML case descriptors that would allow attached images to be encrypted in peer-to-peer referrals. At present TeleMedMail's peer-to-peer referral method is easier to use, but no more powerful, than a process that can be performed with other standard tools—implementing this change would make TeleMedMail's peer-to-peer referrals not just easier than existing current methods, but also functionally better.

12.2.2 Message Authentication for Large-Scale Use

At present, referring locations do not have any way of certifying their identity. This is not really a security risk (see Section 11.2) but it means that messages sent by the system are not authenticated. A very large health-care provider (e.g. an entire country) would require message authentication, so in its current form the TeleMedMail system should not be used by such a large provider.

Before such very-large-scale use, the system would need to be modified by giving referring locations their own RSA keypairs (server locations already have them), and requiring remote locations to register their public keys with the central servers that they intend to refer cases to before they can actually send any cases. The structure of referral and reply messages would also have to be altered to support include authentication. (This could involve including private-key-signed checksums of message content, or private-key-signing the messages themselves.)

This change should only be made if necessary for very-large-scale use, since it makes the system less flexible for general use. (See Section 11.2).

13 References

[1] WHO Estimates of Health Personnel; 1998. http://www3.who.int/whosis/health_personnel/health_personnel.cfm?path=whosis,health_personnel

[2] WHO advisory meeting on radiology education. Geneva: World Health Organization; 1999.

[3] Wootton R. The possible use of telemedicine in developing countries. J Telemed Telecare 1997;3(1):23-6.

[4] Della Mea V, Beltrami CA. Telepathology applications of the Internet multimedia electronic mail. Med Inform (Lond) 1998;23(3):237-44.

[5] Corr P, Couper I, Beningfield S, Mars M. A simple telemedicine system using a digital camera. Journal of Telemedicine and Telecare 2000;6:233-36.

[6] O'Mahony D, Banach L, Mahapa D, Lancaster E, van derLinde G, Williams B, et al. Teledermatology in a Rural Family Practice. S Afr Fam Pract 2001(In Press).

[7] Seckler W, Hoffman N, Banach L. A practical approach in Teleconsultation using standard technologies in rural areas of Eastern Cape. http://www.nemc.org/itmi/presentations/conf/seckler/tsld001.htm

[8] Sørensen T, Rundhovde A, Kozlov VD. Telemedicine in north-west Russia. J. Telemed and Telecare 1999;5(3):153-56.

[9] Malani J, Dever GJ. Telemedicine demonstration projects in the Western Pacific. J Telemed Telecare 1997;3(Suppl 1):43-6.

[10] Johnson MA, Davis P, McEwan AJ, Jhangri GS, Warshawski R, Gargum A, et al. Preliminary findings from a teleultrasound study in Alberta. Telemed J 1998;4(3):267-76.

[11] Vassallo DJ, Hoque F, Farquharson Roberts M, Patterson V, Swinfen P, Swinfen R. An evaluation of the first year's experience with a low-cost telemedicine link in Bangladesh. J Telemed Telecare 2001; 7: 125-138.

[12] Jazayeri D. XraySender: A System to Digitize and Transport Medical X-ray Images. AUP Writeup, May 2000.

[13] Fraser H. S. F., Jazayeri D., Szolovits P., McGrath S. D . Appropriate technology for telemedicine in developing countries. Proc AMIA Symp 2000;:1010

[14] Fraser HS, Jazayeri D, Bannach L, Szolovits P, McGrath SJ. Telemedmail: free software to facilitate telemedicine in developing countries. Medinfo. 2001;10(Pt 1):815-9.

[15] Brauchli K. iPath: Internet Pathology Suite. http://ipath.sourceforge.net/

[16] Excelsior JET: The Java Performance Solution. http://www.excelsior-usa.com/jet.html

[17] Tuberculosis in Russia. http://www.tuberculosis.ru

[18] GNU General Public License: http://www.fsf.org/copyleft/gpl.html

[19] PostgreSQL. http://www.postgresql.org/

[20] Jakarta Tomcat. http://jakarta.apache.org/tomcat/

[21] ImageJ: Image Processing and Analysis in Java. http://rsb.info.nih.gov/ij/

[22] Wootton R, Bloomer SE, Corbett R, Eedy DJ, Hicks N, Lotery HE, et al. Multicentre randomised control trial comparing real time teledermatology with conventional outpatient dermatological care: societal cost-benefit analysis. Bmj 2000;320(7244):1252-6.

Appendix A – TeleMedMail Sent Case Archive

ZIP Contents

The TeleMedMail application archives copies of cases that it sends for future reference. A sent case archive is a standard ZIP file, with a ".zip" file extension, containing:

info.xml – An XML text file describing the sent case. (See below.)
key.dat – The IDEA session key used to send this case. (See below.)
index.html – The html index file that was sent with the case
Applet JAR – The applet (Viewer.jar or CaseViewer.jar), if any, sent with the case.
Attached Files – Any files that were sent with the case.
Replies – The text of any secure replies regarding this case. (See below.)

XML Descriptor

The XML case descriptor is formatted as follows:

```
<TELEMEDMAIL TYPE="CASEINFO">
    <CASEINFO case_id = "..."
              date = "..."
              referring_hospital = "..."
              ...all overall standard fields...
              />

    <ATTACHEDIMAGE TITLE="..." FILENAME="..." WIDTH="X" HEIGHT="Y"
                   BW="true|false" COMMENT="...">
        <ANNOTATION NAME="..." TEXT="..." TYPE="ANCHOR|ARROW|RECTANGLE"
                    P1="X,Y" [P2="X,Y"]/>
        ...all annotations...
        <FIELD NAME="..." VALUE="..."/>
        ...all per-file standard fields...
    </ATTACHEDIMAGE>

    ...all attached images...

    <ATTACHEDFILE TITLE="..." FILENAME="..."/>
    ...all attached files...

</TELEMEDMAIL>
```

Each XML attribute value is URL-encoded. (E.g. an image whose title is "Chest X-Ray" would have the attribute:

```
TITLE="Chest+X-Ray+%28CXR%29"
```

key.dat

The "key.dat" file contains the raw 128-bit IDEA session key, with no encoding.

Replies

Replies are stored in text files named "reply#.txt" where # generally increases sequentially starting from 1. (This order may not always hold, since replies can be deleted.) The first line of a reply text file gives the name and email address of the person who sent it, followed by a blank line, followed by the text of the reply.

Appendix B – TeleMedMail Saved Cases

A TeleMedMail saved case is a ZIP file with a ".tmm" file extension. The ZIP contains a text file called "save.xml" that contains all the case data, and possibly copies of images that are attached to the case.

save.xml

```
<TELEMEDMAIL TYPE="SAVE" ID="...">
    <FIELD NAME="ReferringHospital" VALUE="..."/>
    <FIELD NAME="ReferringDoctor" VALUE="..."/>
    ...all overall fields...

    <IMAGE FILENAME="..." TITLE="..." TARGET="..." FORCEBW="true|false">
        <INFO CROP="NONE|X,Y" ROTATION="0|90|180|270" COMPRESSIONTYPE="JPEG"
              COMPRESSIONLEVEL="..."/>
        <ANNOTATION TEXT="..." TYPE="ANCHOR|ARROW|RECTANGLE" P1="X,Y"
                    [P2="X,Y"]/>
        ...all annotations...
        <FIELD NAME="CaseID" VALUE="..."/>
        ...all per-file fields...
    </IMAGE>

    <IMAGELINK FILENAME="..." TITLE="..." TARGET="..." FORCEBW="true|false">
        <INFO CROP="NONE|X,Y" ROTATION="0|90|180|270" COMPRESSIONTYPE="JPEG"
              COMPRESSIONLEVEL="..."/>
        ...all annotations...
        ...all per-file fields...
    </IMAGE>

    <FILE FILENAME="..." TARGET="..." TITLE="...">
        ...all per-file fields
    </FILE>

    <FILELINK FILENAME="..." TARGET="..." TITLE="...">
        ...all per-file fields
    </FILE>

</TELEMEDMAIL>
```

All XML attribute values are URL-encoded.

The attached files in a saved case can take two forms: physical copies of the original files, or links to the original files. The former are stored in the save ZIP, and are specified by `IMAGE` or `FILE` tags. The latter are specified by `IMAGELINK` or `FILELINK` tags.

Appendix C – Standard Fields Descriptor Files

TeleMedMail's default Standard Field descriptor files are called "overall.stf" and "perfile.stf". Their contents are as follows:

overall.stf

```
ReferringHospital = STRING
ReferringDoctor = CONFIG(ReturnAddress)
ReceivingServer = SERVERS
ReceivingDoctor = ADDRESSBOOK(ReceivingServer)
<SEPARATOR(6)>
Date = AUTO-DATE
CaseID = CASE-UID
PatientName = STRING
DOBAge = STRING
Sex = STRING("Male","Female")
PatientHistory = LONGSTRING<9>
PhysicalExam = LONGSTRING<8>
Investigations = LONGSTRING<8>
ProvisionalDiagnosis = LONGSTRING<2>
```

perfile.stf

```
@CaseID
Date = AUTO-DATE
Comment = LONGSTRING<20>
```

Appendix D - english.lab

```
#General

EncryptedHTML = encrypted.html
UnencryptedHTML = unencrypted.html
Yes = Yes
No = No
Apply = Apply
Cancel = Cancel
Done = Done
Okay = Okay
Close = Close
Delete = Delete
Quit = Exit
CloseMenu = -Close Menu-
Error = Error
ERROR = ERROR:
NotYetImplemented = This feature is not yet implemented by the programmer.
Title = Title
NoTitle = NO TITLE
FitInWindow = Fit in Window
AllFiles = *All Files*
ShowFieldsInteresting = Show Interesting Fields
ShowFieldsAll = Show All Fields
ShowExtraFields = Show Extra Fields
Message = Message
Select = Select
None = None
Date = Date

#Error Messages

ERROR_ErrorAddingFile = Error adding file (%1%) to case.
ERROR_InvalidImageFile = Error: Image file (%1%) is invalid.
ERROR_UndefinedLabel = Error: "%1%" does not define "%2%".
ERROR_ErrorInOverall = Error: Error in overall standard fields file (%1%).
ERROR_ErrorInPerFile = Error: Error in per-file standard fields file (%1%).
ERROR_CantWriteConfigFile = Error: Can't write to CONFIG.INI.
ERROR_WorkingDirDNE = Error: Default Image Directory specified in CONFIG.INI
(%1%) doesn't exist. Using "images\" instead.
ERROR_WorkingDirCantCreate = Error: Specified Default Image Directory (%1%)
exists, and is not a directory. Please delete it and re-run TeleMedMail.
ERROR_ErrorReadingConfig = Error: Can't read CONFIG.INI.
ERROR_ErrorConnecting = Error: Can't connect to SMTP Server. %1%
ERROR_ErrorSending = Error: Error sending email message. %1%
ERROR_CantSave = Error: Can't save case. %1%
ERROR_CantLoad = Error: Can't load case. %1%
ERROR_ErrorWritingJPG = Error: Error writing jpeg file (%1%). %2%
ERROR_NoWildcardsYet = Wildcards not supported yet.
ERROR_CantWriteAddressBook = Error: Can't write Address Book file (%1%). %2%
ERROR_IllegalCompressionLevel = Error: Compression level must be an integer
between 0 (highest compression) and 100 (lowest compression).
ERROR_InvalidLabelFile = Error: The language file chosen (%1%) is not valid.
ERROR_FileDNE = Error: Chosen file ("%1%") does not exist.
ERROR_UIDFileInvalid = Error: Cannot read the UID data file (%1%).
ERROR_UIDFileCantCreate = Error: The file "next_uid.dat" is missing, and
TeleMedMail can't create a new copy. (%1%).
ERROR_EncryptedHTMLFileMissing = Error: encrypted.html file is missing.
ERROR_EncryptedHTMLFileError = Error: encrypted.html file has errors. %1%
ERROR_UnencryptedHTMLFileMissing = Error: unencrypted.html file is missing.
```

```
ERROR_UnencryptedHTMLFileError = Error: unencrypted.html file has errors. %1%
ERROR_NoContents = Error: Case contains no data to be sent.
ERROR_BadAppletSize = Error: Applet Width and Height must be positive integers.
ERROR_FillInFrom = Error: In order to send a case to a server, both the
Referring Doctor and the Receiving Doctor fields must be filled in.
ERROR_ErrorExporting = Error: Can't export file. %1%
ERROR_MailTimeout = Error: Timeout connecting to SMTP Server. (limit = %1%
seconds)
ERROR_KeyFileMissing = Error: The server's public key file (%1%) is missing, or
can't be opened.
ERROR_FailedToDeleteReply = Error: Failed to delete reply. File is locked.
ERROR_CantDeleteFile = Error: Failed to delete file "%1%". File may be locked.
ERROR_CantReadZip = Error: Can't read ZIP file. %1%
ERROR_ErrorInXml = Error reading XML file. %1%
ERROR_CantFindSentCase = Can't find the sent case this reply refers to.
ERROR_CantFindKey = Can't find secret key used to encrypt this case. (Perhaps
it was sent with a previous version of TeleMedMail.)
ERROR_BadAddressXml = The following entry in the address book file is invalid:
%1%

#Main TeleMedMail Frame

MAIN_Title = TeleMedMail - Store & Forward Telemedicine System (GPL, by Darius
Jazayeri and Hamish Fraser)
MAIN_ImportImage = Import Image
MAIN_RemoveImage = Remove Image(s)
MAIN_SendEmail = Send Email
MAIN_SaveCase = Save Case
MAIN_LoadCase = Open Case
MAIN_NewCase = New Case
MAIN_Setup = Edit Settings
MAIN_CaseBrowser = Browse Sent Cases
MAIN_DecryptReply = Decrypt Reply Email
MAIN_AttachedFiles = Attached Files
MAIN_FileMenu = File
MAIN_CaseMenu = Case
MAIN_SettingsMenu = Settings
MAIN_ManageMenu = Manage

#Keyboard Shortcuts

KEYBOARD_New = control N
KEYBOARD_Save = control S
KEYBOARD_Load = control O
KEYBOARD_Exit = alt X
KEYBOARD_Import = control M
KEYBOARD_Send = control E
KEYBOARD_Address = control A
KEYBOARD_Cases = control B
KEYBOARD_Decrypt = control D

#Add File

ADDFILE_FileAlreadyThere = File (%1%) is already in case.
ADDFILE_AddingImage = Loading Image (%1%)
ADDFILE_AddingFile = Loading File (%1%)

#Setup

SETUP_Title = TeleMedMail Configuration
SETUP_EmailAddress = Your return address
SETUP_LabelFile = Language file
SETUP_LabelFileFilter = Language files
```

```
SETUP_SMTPServer = Outgoing SMTP server
SETUP_SMTPUsername = SMTP Username (leave blank to use Return address)
SETUP_WorkingDirectory = Default Image Directory
SETUP_EditAddressBook = Edit Address Book
SETUP_CompressionSetup = Compression Settings
SETUP_AdvancedSetup = Advanced Settings
SETUP_LabelsChanged = Label file changed. Please restart TeleMedMail for
changes to take effect.
ADVANCEDSETUP_Title = TeleMedMail Advanced Configuration
ADVANCEDSETUP_OverallFile = Overall Case STF file
ADVANCEDSETUP_PerImageFile = Per-Image STF file
ADVANCEDSETUP_AppletWidth = Applet Width
ADVANCEDSETUP_AppletHeight = Applet Height
COMPRESSIONSETUP_Title = Compression Setup
COMPRESSIONSETUP_BurnOnAnnotations = Burn On Annotations
COMPRESSIONSETUP_SendOrigImage = Send Original Images
COMPRESSIONSETUP_DefaultJPEGLevel = Default JPEG Compression Level
COMPRESSIONSETUP_ImageType = Default Image Type
COMPRESSIONSETUP_Guess = Guess Image Type
COMPRESSIONSETUP_XRay = X-Ray Image
COMPRESSIONSETUP_Color = Color Image

#Edit Address Book

EDITADDRESS_Title = Edit Address Book
EDITADDRESS_Add = Add Address
EDITADDRESS_Delete = Delete
EDITADDRESS_Save = Save
EDITADDRESS_NotSavedTitle = Save Address Book?
EDITADDRESS_NotSavedQuestion = Address Book has been changed. Do you want to
save those changes to disk?
EDITADDRESS_Name = Name
EDITADDRESS_Email = Email Address
EDITADDRESS_IsServer = Server
EDITADDRESS_IsPerson = Person
EDITADDRESS_KeyFile = Server Public Key File
EDITADDRESS_Comment = Affiliated with
EDITADDRESS_People = People
EDITADDRESS_Servers = Servers
EDITADDRESS_AddPerson = Add Person
EDITADDRESS_AddServer = Add Server
EDITADDRESS_PublicKey = Public Key

#Attached File Panel

ATTACHED_TypeXRay = X-Ray
ATTACHED_TypeColor = Color Image
ATTACHED_CompressionLevel = Compression: %1% (%2%%)
ATTACHED_EditFields = Edit Comment
ATTACHED_Annotate = Annotate
ATTACHED_Remove = Remove From Case
ATTACHED_CropRotate = Crop and Rotate Image

#Compression Level Dialog

COMPRESSION_Title = Compression Preview for
COMPRESSION_Smaller = Smaller Filesize
COMPRESSION_Better = Better Quality
COMPRESSION_PercentQuality = % quality
COMPRESSION_Filesize = Filesize
COMPRESSION_ShowDegredation = Show Image Degredation

#Edit Fields Dialog
```

```
FIELDS_Title = Standard Fields for

#Annotate Dialog

ANNOTATION_Title = Annotating
ANNOTATION_NewAnnotation = New Annotation
ANNOTATION_ExistingAnnotation = Annotation #
ANNOTATION_ExistingAnnotations = Existing Annotations
ANNOTATION_DefaultAnnotationText = Enter Annotation Text Here
ANNOTATION_Delete = Delete

#Crop Image Dialog

CROP_Title = Cropping
CROP_NoCropping = No Cropping
CROP_Rotate90 = Rotate Right
CROP_Rotate180 = Rotate 180
CROP_Rotate270 = Rotate Left

#Save Dialog

SAVE_Title = Save Case
SAVE_Filename = Filename
SAVE_SaveCase = Save case for future editing
SAVE_ExportCase = Export as a sent case
SAVE_SaveCopies = Save copies of images
SAVE_SaveButton = Save
SAVE_NothingToSave = There is no data in this case to save.

#Load Dialog

LOAD_Title = Open Case
LOAD_Button = Open
LOAD_FileFilterTitle = TeleMedMail Saved Cases

#Send Mail Dialog

SENDMAIL_To = To:
SENDMAIL_From = From:
SENDMAIL_Subject = Subject:
SENDMAIL_Body = Body:
SENDMAIL_Encrypt = Encrypt Case
SENDMAIL_SendApplet = Send Applet
SENDMAIL_Send Applet = Send Applet
SENDMAIL_Send = Send
SENDMAIL_SendServer = Send To Server
SENDMAIL_Export = Export
SENDMAIL_MessageSize = Message Size
SENDMAIL_FillInHeaders = Please fill in all header fields.
SENDMAIL_BuildingCase = Preparing Case...
SENDMAIL_SendingMessage = Sending Message...
SENDMAIL_StatusBuilding = Building Recipient List...
SENDMAIL_StatusEncoding = Encoding Attached Files...
SENDMAIL_StatusEncodingFile = Encoding %1%...
SENDMAIL_StatusConnecting = Opening Connection to SMTP Server...
SENDMAIL_StatusConnected = Connected to SMTP Server...
SENDMAIL_StatusSendingRecipients = Sending Recipient List...
SENDMAIL_StatusSendingHeaders = Sending Message Headers...
SENDMAIL_StatusSendingBody = Sending Message Body...
SENDMAIL_StatusSendingFile = Sending %1%...
SENDMAIL_StatusAssembling = Assembling images and other files.
SENDMAIL_StatusGeneratingHTML = Generating HTML files.
```

```
SENDMAIL_StatusGeneratingKey = Generating session key.
SENDMAIL_StatusEncrypting = Encrypting Case.
SENDMAIL_StatusBuildingZIP = Building ZIP file.
SENDMAIL_StatusSending = Sending Email...
SENDMAIL_MessageNotSent = Error: Message not sent.
SENDMAIL_MessageSent = Message sent.

#Password

PASSWORD_Title = Enter Password
PASSWORD_EnterPassword = Enter Password:

#Standard Fields

STF_CaseID = Case ID
case_id = Case ID
STF_ReferringHospital = Referring Hospital
referring_hospital = Referring Hospital
STF_ReferringDoctor = Referring Doctor Email
referring_doctor = Referring Doctor Email
STF_ReceivingHospital = Receiving Hospital
receiving_hospital = Receiving Hospital
STF_ReceivingServer = Receiving Hospital Server
receiving_server = Receiving Hospital Server
STF_ReceivingDoctor = Receiving Doctor Email
receiving_doctor = Receiving Doctor Email
STF_PatientName = Patient Name
patient_name = Patient Name
STF_DOBAge = DOB / Age
dob_age = DOB / Age
STF_Sex = Gender
sex = Gender
STF_PatientHistory = Patient History
patient_history = Patient History
STF_PhysicalExam = Physical Exam
physical_exam = Physical Exam
STF_Investigations = Investigations
investigations = Investigations
STF_ProvisionalDiagnosis = Provisional Diagnosis
provisional_diagnosis = Provisional Diagnosis
STF_Date = Date
date = Date
STF_Comment = Comment
number_of_images = # Images

#Date

DATE_January = January
DATE_February = February
DATE_March = March
DATE_April = April
DATE_May = May
DATE_June = June
DATE_July = July
DATE_August = August
DATE_September = September
DATE_October = October
DATE_November = November
DATE_December = December


#UID
```

```
UID_Unassigned = [UNASSIGNED]

#APPLET

APPLET_PopupButtonLabel = Open Case
APPLET_UnencryptedPopupButtonLabel = View In Applet
APPLET_AttachedFile = Attached File : %1%

#IMAGE FILTERS

IMAGEFILTER_0 = Image Files ||| jpeg, jpg, tiff, tif, gif
IMAGEFILTER_1 = JPEG and GIF images ||| jpeg, jpg, gif
IMAGEFILTER_2 = TIFF images ||| tif, tiff

#REPLY

REPLY_DialogTitle = Reply #%1%
REPLY_ReplyFrom = Reply From: %1%

#CASE BROWSER

CASEBROWSER_Title = Case Browser
CASEBROWSER_Filename = Filename
CASEBROWSER_UID = Case ID
CASEBROWSER_ReceivingDoctor = Receiving Doctor
CASEBROWSER_PatientName = Patient Name
CASEBROWSER_NumImages = # Images
CASEBROWSER_View = View
CASEBROWSER_AttachedImage = Attached Image:
CASEBROWSER_AttachedFile = Attached File:
CASEBROWSER_Case = Case #%1%

#DECRYPT REPLY

DECRYPT_Title = Decrypt Reply Email
DECRYPT_Password = Password:
DECRYPT_Password2 = (leave blank if this is a reply from a server)
DECRYPT_EnterHere = Enter Encrypted Reply Here:
DECRYPT_Decrypt = Decrypt Reply
DECRYPT_DecryptedText = Decrypted Text:
DECRYPT_Save = Save Decrypted Reply
DECRYPT_View = View This Case
DECRYPT_BadInput = Error in the encrypted text. Failed to decrypt. (%1%)

#Default email fields

EMAIL_Subject = Telemedicine Referral
EMAIL_Body = Telemedicine Referral. Please unzip the ZIP file attachment, and
open the "index.html" file it contains.
```

Appendix E - TeleMedMail AddressBook Interface

```
import java.io.File;
import javax.swing.JDialog;

/*
 * Rather than implementing this Interface, you probably want to
 * subclass DefaultAddressBook, and override the editDialog()
 * method.
 */
interface AddressBook {

    /*
     * Sets the File that load and save will read from and write to.
     * This method must be called before load and save can be called.
     */
    public void setFile(File f);

    /*
     * Returns the File that load and save will read from and write to.
     */
    public File getFile();

    /*
     * Writes the contents of the address book to the file specified
     * by the last call to setFile.
     * Returns null if the saving worked, or an error message if not.
     */
    public String save();

    /*
     * Empties the address book, then reads the contents of the file
     * specified by the last call to setFile into it.
     * Returns null if load worked, or an error message if not.
     */
    public String load();

    /*
     * Adds or overwrites an entry to the address book.
     */
    public void add(AddressEntry e);

    /*
     * Removes an entry from the address book.
     */
    public void remove(String name);

    /*
     * Returns the entry in the address book for a given name,
     * or null if that name isn't in the book.
     */
    public AddressEntry get(String name);

    /*
     * Returns an array of all entries in the address book.
     */
    public AddressEntry[] entries();
```

```
    /*
     * Returns the filename (relative to the KEYS directory)
     * of the public key of the server with a given email address.
     * email can be of the form "name@email.com" or
     * "Real Name <name@email.com>".
     */
    public String getKeyFor(String email);

    /*
     * Brings up a dialog allowing the user to edit the address
     * book.
     */
    public JDialog editDialog();

}
```

adrbook.dat

The TeleMedMail AddressBook is stored as XML in a text file ("adrbook.dat" by default). It has the following form:

```
<ADDRESSBOOK>
    <SERVER NAME="..." EMAIL="..." KEY="..."/>
    ...all server entries...

    <PERSON NAME="..." EMAIL="..." [COMMENT="..."]/>
    ...all person entries...

</ADDRESSBOOK>
```

All XML attribute values are URL-Encoded.

The key attribute for a server entry specifies the filename (relative to the keys directory) of that server's public key file.

The comment attribute for a person entry specifies the server that person is affiliated with. It should match the name of one of the server entries.

Appendix F - TeleMedMail CaseBrowser Interface

```
interface CaseBrowser {

    /*
     * Classes which implement CaseBrowser must support these two constructors:
     *
     *      public CaseBrowser(JFrame parent);
     *      public CaseBrowser(JDialog parent);
     */

    /*
     * This pops up a modal dialog allowing the user to browse through sent
     * cases.
     */
    public void display();

}
```

The contents of the TeleMedMail sent case folder are ZIP files as specified in Appendix #.

The sent case folder can be obtained with the

```
TeleMedMail.getSentCaseFolder()
```

method.

Appendix G - TMMServer SQL Database Setup Code

Before TMMServer can be run for the first time, its database structure must be created with the following SQL statements:

```
create table users (
    id              integer not null primary key,
    name            varchar(100),
    email           varchar(100) not null,
    email2          varchar(100),
    email3          varchar(100),
    username        varchar(100) not null,
    password        varchar(32) not null,
    admin           boolean not null default false,
    active          boolean not null default false,
    added_by        integer references users,
    added_date      timestamp default CURRENT_TIMESTAMP,
    low_bandwidth boolean default false
);

create sequence user_id_seq;

create table cases (
       id                      integer not null primary key,
       referring_hospital      varchar(100),
       referring_doctor        varchar(100),
       receiving_doctor        varchar(100),
       date_sent               varchar(32),
       date_received           timestamp default CURRENT_TIMESTAMP,
       sender_case_id          varchar(32),
       url                     varchar(100),
       session_key             varchar(256),
       old_style               boolean not null
       --an old style case must be viewed by going to the case's index.html
);

create sequence case_id_seq;

create table patient_details (
       case_id              integer not null references cases on delete cascade,
       name                 varchar(100),
       sex                  varchar(16),
       dob_age              varchar(32)
);

create table case_details (
       case_id             integer not null references cases on delete cascade,
       name                varchar(100),
       value               varchar(4000)
);

create table attached_images (
    case_id      integer not null references cases on delete cascade,
    image_id     integer not null primary key,
    old_style    boolean not null,
    filename     varchar(100) not null,
    title        varchar(100),
    width        integer,
    height       integer,
    comment      varchar(4000)
```

```
);

create sequence attached_images_id_seq;

create table image_annotations (
    image_id    integer not null references attached_images on delete cascade,
    name        varchar(20) not null,
    text        varchar(4000) not null,
    type        varchar(20) not null,
    p1          varchar(20) not null,
    p2          varchar(20)
);

create table image_fields (
    image_id    integer not null references attached_images on delete cascade,
    name        varchar(100) not null,
    value       varchar(4000)
);

create table attached_files (
    case_id      integer not null references cases on delete cascade,
    filename     varchar(100),
    title        varchar(100)
);

create table permissions (
       case_id           integer not null references cases on delete cascade,
       user_id           integer references users on delete cascade,
       email             varchar(100)
);

create table replies (
    case_id      integer not null references cases on delete cascade,
    user_id      integer references users,
    reply_text   varchar(4000),
    reply_date   timestamp default CURRENT_TIMESTAMP,
    encrypted    boolean
);
```