Design and implementation of an Internet-based medical image viewing system

Shawn P. Laird a, Johnny S.K. Wong a,*, William J. Schaller b, Bradley J. Erickson b, Piet C. de Groen b

a Department of Computer Science, Iowa State University, 225 Computer Science Building, Ames, IA 50011-1041, USA
b Mayo Clinic and Foundation, 200 First St. S.W., Rochester, MN 55905, USA

Received 13 August 2001; received in revised form 8 December 2001; accepted 13 December 2001

Abstract

Electronic medical image viewers combined with picture archival and communication systems have become a common part of how hospitals manage their images. Typically, these electronic viewers are platform-dependent and are designed to make use of images stored on a workstation or local area network. However, the rise of the Internet and the ubiquity of the Web browser open up a wide range of new possibilities for distributing and accessing these images. The Internet allows us to pull information distributed across many geographically separated data sources, while the Web browser provides a common environment from where programs can be launched. This document details the design of an Internet-based, platform-independent medical image viewing system that harnesses the potential of these technologies.

© 2002 Elsevier Science Inc. All rights reserved.

Keywords: Medical imaging; Telemedicine; Teleradiology; Picture archival and communication systems; CORBA; DICOM

1. Introduction

Over the past several decades, medical imaging has been dominated by the use of expensive film media for the review and archival of medical scans. However, due to developments in networking technology and the widespread acceptance of a standard medical image communication protocol known as digital imaging and communication in medicine (DICOM) Horii et al., 1993, it has become possible to electronically store and review these images in a standard way. Picture archival and communication systems (PACS) Creighton, 1999 have become a mainstay in most hospitals for handling the image data generated by imaging modalities and determining how to store it. As this information has become available electronically, electronic viewers capable of deciphering the image data stored by the PACS have become more and more mainstream. These developments have enabled many improvements in the way hospitals view, distribute, and archive their images. However, the rise of the Internet and the ubiquity of the Web browser provide even greater forums for medical imaging.

Currently, if a hospital does have a medical viewing system, it is typically geared for a particular platform and is used strictly with data stores available on the local area network. This prevents users without the targeted platform from using the system and also limits the available image pool to those image archives connected on the local network. This in turn limits the use of the viewer software to within the hospital itself. Tremendous benefits could be reaped by replacing or at least supplementing these proprietary image viewers with a platform-independent, Internet-based viewing system. Platform-independence creates a much larger user base by enabling access to the system through many types of operating systems and hardware configurations. By being Internet-based and delivering the viewer program through a Web server, the system’s accessibility is further increased since any computer connected to the

---

*Corresponding author. Tel.: +1-515-294-2586; fax: +1-515-294-0258.
E-mail addresses: slaird@cs.iastate.edu (S.P. Laird), wong@cs.iastate.edu (J.S.K. Wong), schalleb@ mayo.edu (W.J. Schaller), bje@ mayo.edu (B.J. Erickson), degroen.piet@mayo.edu (P.C. de Groen).
Internet with a Java™-enabled browser can make use of it. Through the Internet, information for a particular patient can be gathered not just from a local hospital but also from several remote hospitals. However, increasing a system’s accessibility and opening it to the Internet creates several potential security problems that need to be adequately addressed especially when such a system deals with medical information.

This paper details the work we have done developing a complete and secure Internet-based medical image viewing system called JReads that meets these requirements. The development of this system has been funded by the Mayo Clinic in Rochester, MN and has been geared to mesh well with their existing radiology infrastructure. The JReads system consists of a Java applet at the front-end, Java servlets and common object request broker architecture (CORBA) objects in the middle-tier, and relational databases and image file servers at the back-end. By using the platform-independent programming language Java Gosling and McGilton, 1999 and the programming language independent object brokering facilities provided by CORBA Yang and Duddy, 1996, we were able to develop a system that is easily extendible and adaptable to a wide variety of environments. Essentially, the system on both the client and server sides is completely portable. The use of Java applets enables clients to use a variety of different platforms, while the use of Java servlets enables the use of many different types of Web server hardware and software. CORBA provides a standard mechanism for creating and handling the communication between distributed software objects. It provides a clean separation between an object’s interface and implementation so that inter-object communication is simplified. CORBA also enables legacy code written in a variety of languages to be network-enabled. This is accomplished by creating an object implementation that handles the mapping between the CORBA interface and this legacy code. As long as the interface is properly supported, the underlying implementation details are of no concern to clients of this object. CORBA plays an especially important role in this project for sending and retrieving information distributed across multiple hospitals. Overall, the goal was to provide a platform-independent system that is easier to maintain, use, and distribute than the proprietary medical viewers most hospitals use today. In addition, we wanted to make this system capable of teleradiology so that a physician at the Mayo Clinic could access patient information and images stored locally at the Mayo Clinic along with images stored remotely at other hospitals all through one consistent interface. Adding this teleradiology component to JReads expands its usefulness beyond just the Mayo Clinic, since it enables other hospitals to share their data with specialists at the Mayo Clinic and have nearly instant, remote consultations Frey and Spicer, 1999.

The development of a Java-based medical image viewing system provides the following advantages over the typical proprietary image system:

- No installation. An applet can be loaded dynamically through the Internet from a Web server. This enables the code to be controlled and maintained at a central site, while having the capability to be executed at distributed sites. This simplifies the task of updates to the software and also enables every user to run the newest version of the software immediately.
- Low cost and easy maintenance. Java is a modern, object-oriented language with a comprehensive standard API that enables greater programmer productivity and more readable code. Using applets eliminates many of the costs associated with upgrading individual workstations, and wrapping legacy applications with CORBA object interfaces enables existing software to be integrated into the new system rather than being completely rewritten.
- Applets can be loaded through any Java-enabled Web browser enabling physicians to access images and patient information from anywhere in the world rather than at just specific terminals within the local hospital through a proprietary system.
- Platform-independence on both the client (applets) and the server (servlets/CORBA) enables complete portability to other platforms.
- Standardized ways of accessing data and communicating through the network (JDBC, CORBA, Java sockets) allows distributed heterogeneous data sources to work together in a simplified manner.
- The intricacies of legacy systems employed at contributing remote hospitals can be hidden from the rest of the system by the use of CORBA.
- Java’s built-in security manager limits the functionality of applets to protect the client machine making them safer to use and easier to trust than native code.
- Security through the use of the secure sockets layer (SSL) can be provided transparently between the Web browser and server to secure communications between the front-end applet and middle-tier servlets. SSL can also be employed between the servlets and the back-end CORBA objects for authentication and encryption of communications.

There are also many potential benefits of teleradiology. A few of these include Frey and Spicer, 1999:

- Providing easy access to images stored across multiple institutions.
- Improving the timeliness of care.
- Improving the quality of the diagnosis since multiple specialists can examine the results simultaneously despite being geographically separated.
- Allowing access to sub-speciality consultations.
• Covering the needs of many hospitals with just a few specialists through remote consultations.

In summary, the use of Internet technology for medical imaging systems can enable more timely and accurate patient care while significantly cutting the costs associated with distributing and examining medical images.

2. Related work

In the following sections we discuss the architecture, functionality, and focus of three systems that are related to JReads. A features summary for these systems is given at the end of the section.

2.1. QReads

The initial focus of this project was to replicate the base functionality present in the proprietary medical image viewer QReads Erickson et al., 1997 used at the Mayo Clinic in a platform-independent, Internet-ready Java viewer. QReads is a Windows application that was developed using Microsoft’s Visual C++ Version 5.0. It provides an easy-to-use interface for displaying patient information, listings of exams, exam reports, and images taken for an exam. QReads makes use of Mayo’s radiology information management system (RIMS) which is implemented using a Sybase database. It connects to this database using an ODBC driver and calls specialized stored procedures to retrieve information pertaining to the patient and his/her exams. It also receives information regarding the location of any available images pertaining to these exams. Mayo’s PACS along with the clinical images gateway (CIG) manage the storage of medical image scans across multiple clinical image file servers. Using the references provided from the stored procedures in the RIMS database, QReads can access the various image file servers to retrieve the necessary images using the network file system. The images are then displayed in the QReads viewer and can be manipulated by the user.

JReads maintains the basic interface of QReads and makes use of the same data sources as this proprietary program (RIMS and the image file servers). The JReads system was designed to replicate QReads’ functionality while also enabling platform-independent intranet and Internet computing. However, we have added additional functionality and capabilities to the JReads viewer to increase its utility. The development of JReads into a platform-independent version of QReads forms one major part of this research effort. The other research goal was to extend the JReads viewer so that it could make use of medical image sets distributed throughout different hospitals and not just those stored on Mayo’s image servers. QReads is capable of displaying images stored across multiple image servers, but is restricted in scope to just the medical scans taken within the Mayo Clinic. For instance, a physician can view all of the exams and associated images taken for a patient at the Mayo Clinic through the QReads viewer. However, the patient’s exam reports and medical scans taken at another hospital are unavailable to him or her through this same interface. The JReads system was designed so that it would have the flexibility to interface not only with Mayo’s internal image servers but also image servers distributed across any other hospital. Essentially, our ultimate goal was to build JReads into a basic teleradiology system while maintaining the look, feel, and functionality the users at the Mayo Clinic were accustomed to.

There have been several other medical viewing systems developed for a variety of specific applications. Currently, many hospitals and research organizations use proprietary systems either developed in-house or provided from one of several medical imaging vendors. Most of these viewers provide some base functionality that is consistent to what is provided with QReads. Listings of free and commercial medical viewers are available at Clunie, 1999. Nearly all of these viewers have been developed for a particular platform and were not designed to be used through the Internet. However, some new systems have recently been proposed that harness the power of the Internet and the platform-independence of Java. Two of the most significant new Internet-based systems being researched are the digital library based biomedical imaging system being developed at the University of California School of Medicine at San Francisco, CA and the TeleMed project being pursued at Los Alamos National Laboratory.

2.2. Neuro-imaging digital library

The radiology research department at the University of California School of Medicine at San Francisco, CA has delved into some very important issues concerning the organization of medical imaging systems. One of their research areas has involved the integration of digital library techniques into medical image archives. The goal of this work is to develop better techniques for organizing and searching image archives. They have developed an imaging architecture to support digital library functions that are discussed in Wong et al., 1997. This is a three-tier architecture with a back-end consisting of an image archive and clinical text database, a comprehensive middleware consisting of specialized servers for image processing, visualization, and retrieval, and a front-end consisting of a special purpose image viewer. Their use of middleware servers for the handling of advanced image processing is interesting and enables the use of very powerful machines to handle computationally...
complex image analysis functions. This also enables them to keep their front-end relatively simple. The downside of this approach is that scalability may become a problem due to the centralizing of image processing. Overall, their basic architecture is similar to that used at Mayo in that they use databases for storing exam information along with references to the images associated with these exams. Their focus is a bit different than the QReads and JReads systems, though, in that they are concerned with more advanced image analysis techniques (handled through specialized middleware servers) rather than general exam and image viewing capabilities. QReads (and JReads) provide a general-purpose viewer with a few easy-to-use analysis tools that are not specialized for any particular imaging modality.

One of the special purpose viewers they have developed for this system is a Java applet for the display of neurological images. It is Web-based and directly communicates to a neuro-imaging database using a type four JDBC driver. This viewer has the capability to display images for exams and their corresponding reports. It also provides some of the basic image manipulation capabilities we provide in JReads such as contrast and brightness adjustment and cine display. The primary difference between our system and theirs is that we are looking to have more of a general teleradiology capability by being able to retrieve images from a variety of hospitals, not just from a local hospital system. The neuro-image viewer they have developed was designed to take advantage of more advanced image processing capabilities available in the middleware servers they are using. So the difference is that we are trying to develop a general-purpose system, while they have been focusing on more specialized viewers.

2.3. TeleMed

TeleMed is a comprehensive telemedicine system being worked on at the Los Alamos National Laboratory (LANL) Forsslund et al., 1999. It is based on Java and CORBA and integrates a set of interoperable CORBA components based on the CORBAmid Jagannathan et al., 1999 interface standards. CORBAmid is the official OMG domain task force for healthcare and is responsible for defining standard object interfaces for common healthcare services. TeleMed’s goal is to bring together all of the pertinent information for a particular patient and present it to the user in a single, unified manner. TeleMed is based upon the concept of a virtual patient record that is transparently constructed from information distributed across different data sources. This virtual patient record construction is aided by an implementation of the person identification service (PIDS) specified by CORBAmid. This identification service can be used to uniquely identify a patient and his or her information across distributed data sources even though these data sources may locally use different identification schemes. This virtual patient record captures all of the medical information for an individual such as immunization history, exam reports, laboratory results, and image studies.

The current implementation of the TeleMed system makes use of Inprise’s Visibroker CORBA implementation. TeleMed is also implemented entirely in Java at both the client and server (much like JReads). The TeleMed system hides the fact that information is gathered from different data source from the user and it provides a standard interface for accessing a variety of features. In addition to providing patient information, it also provides image viewing capabilities TeleMed User Manual, 1999. It can display DICOM format images, and it provides a few basic image manipulation tools such as windowing and leveling (necessary for adjusting the brightness and contrast of an image). For security, they have developed their own authentication scheme using a custom authentication server that the client has to connect with prior to accessing any other service. TeleMed also makes use of public and private key cryptography for securing the communications between the servers and the front-end in a way similar to SSL. For JReads, we have made use of Iona’s CORBA implementation, Orbix™, which can have security features such as authentication and encryption added by using Iona’s OrbixSSL™ package.

TeleMed’s system architecture consists of at least three servers and a front-end Java application. The first server that is contacted by the front-end is the Authenticator. The Authenticator validates users and manages the access policy for the other servers. Once the client is authenticated and the needed secure communication channels have been set up, the front-end can attempt to identify a particular patient using the PIDS server. The PIDS server uses a variety of demographic information to uniquely identify the requested individual. Once the patient has been identified, the client can then connect to one of potentially many Medical Data Servers to request this patient’s virtual medical record. This request may require data to be drawn from distributed locations. When the information is returned to the front-end, the data is displayed in a graphical time-oriented format that can be easily navigated by the user.

The focus of the TeleMed system is to provide a complete telemedicine system that brings together all information for a given patient despite the data being distributed across multiple heterogeneous data sources. Rather than focusing on all of the problems facing the organization of hospital information, the JReads system focuses on one very important aspect of telemedicine known as teleradiology. As such, our system provides a more complete viewing system than the TeleMed project, but at the same time, is more limited in scope. TeleMed provides a good basic model for making use of
patient information that may be distributed across different institutions. The developments of the TeleMed project are significant and show the viability of Java and CORBA in a very large-scale environment.

3. System architecture

We have created two main architectures for the telediagnosis-enabled JReads system. One of these architectures makes exclusive use of CORBA objects that serve as wrappers for the RIMS database and the local and remote image servers. We will call this the pure CORBA architecture. This architecture makes the location of the images completely transparent to the user and provides a consistent mechanism for loading images. However, it is expected that the majority of accesses that the JReads system will make will be to images generated locally at the Mayo Clinic. Because of this, we have developed a second architecture that makes more limited use of CORBA objects for efficiency. We call this the partial CORBA architecture. Queries to the RIMS database will be made directly through the query servlet using JDBC instead of through the RIMS CORBA object. Also, all accesses to image servers on the local area network will be made directly by the image retrieval servlet through the network file system (NFS) rather than through a ResourceServer object. These optimizations eliminate the overhead of going through a CORBA object when dealing with sources readily available within Mayo’s intranet. This approach also meshes well with the existing architecture at Mayo, since it does not require any CORBA servers to be implemented for their local RIMS and image servers. However, this approach causes some loss of transparency and makes the design a bit less consistent. We will cover the workings of these architectures in the coming sections.

3.1. Pure CORBA architecture

Fig. 1 shows the layout of the components for this architecture. For a physician to begin a JReads session, he or she first needs to download the JReads applet. This applet is served by a Web server configured with servlet capability within the network at the Mayo Clinic. The applet communicates with a query servlet installed on the Web server to access information stored in the RIMS. RIMS provides patient and exam information, as well as references to the images associated with each exam. This servlet obtains this information by getting a reference to the appropriate RIMS CORBA object that handles access to the actual RIMS and calling appropriate methods on it. To retrieve the images from the image servers, the applet then contacts an image retrieval servlet and passes to it the references for the desired images. These references point to files located on the image servers and are used by the servlet to retrieve the images. This servlet will then get a reference to a ResourceServer CORBA object representing the desired server and call the appropriate methods for retrieving the desired images. The results are then passed back to the applet.

We will now take a closer look at how all of the individual components interact during a typical JReads session.

1. The user first needs to download the JReads applet front-end from a servlet-enabled Web browser. The system also allows a JReads front-end application to be used so that download time can be reduced. However, either front-end will need to communicate with servlets running on the Web server to get the information they need.

2. The user accesses the login function of the front-end and enters their username and password. This login information is then sent through HTTP to the Web server.
server that the applet was originally loaded from. SSL is used to protect all information traveling between the applet and the Web server. This information is directed to a query servlet that is running on the Web server’s servlet engine. This query servlet then calls a method on the RIMS CORBA object to test if a connection is possible using the provided username and password. The query servlet then sends back a reply to the applet letting it know whether or not the login to RIMS was successful.

3. If the login was successful, the user can then enter in a patient id to bring up information and a listing of current exams for this individual. When one of the listed exams is clicked upon, the corresponding textual report and notes for this exam will be displayed in the report text area. The applet retrieves the necessary information by making requests to the query servlet running with the Web server. This servlet calls the appropriate methods of the RIMS CORBA object to acquire the desired information from the RIMS database.

4. If images are available for a selected exam, the user can click on the open button in the image toolbar to retrieve all of the associated image series for this exam. When the applet retrieves information pertaining to all of the exams for the patient, it also receives the references to the associated image files for each exam and the server they are located on. The applet then passes this information to the image retrieval servlet also running with the Web server. The retrieval servlet obtains a reference to a ResourceServer CORBA object through the Naming Service using the server information passed to it. It then calls a method on this object to retrieve the desired images. The ResourceServer object will then handle retrieving the images from the server it is attached to and send them back to the servlet. The image retrieval servlet then pipes these image files directly back to the applet.

5. When the applet receives image information back from the retrieval servlet, it uses specialized classes to interpret and render them on-screen. Initially, all of the images are displayed as thumbnails in the upper portion of the viewer. When a user left or right clicks on any of the thumbnails, a larger version of the image is shown in the lower left or right displays. From here, the user can manipulate and analyze the images in a variety of ways. The specific tools provided by the JReads front-end are covered in a later section.

3.2. Partial CORBA architecture

Unlike the pure CORBA architecture discussed previously, this middleware architecture has two distinct sections. One section deals with images that are generated locally at the Mayo Clinic. These images may be generated by the use of imaging modalities distributed across the institution. However, they are considered local since they all interface with the Mayo PACS through a local area network. The other main section serves a teleradiology role and deals with exams and related images that were generated from a remote hospital. This remote hospital may have a very different organization for its imaging stations and may or may not have a PACS or similar system. However, these details are not important to the JReads system due to the development of CORBA objects that hide the inner workings of Mayo’s and the remote hospital’s systems from each other.

Fig. 2 shows the layout for the partial CORBA architecture. For a physician to be able to access any of

![Fig. 2. Partial CORBA architecture.](image-url)
these stored images, he or she needs to download the JReads applet. This applet is served by a Web server configured with servlet capability within the network at the Mayo Clinic. This applet communicates with a query servlet installed on the Web server to access information stored in the RIMS. RIMS provides patient and exam information, as well as, references to the images associated with each exam. To retrieve the images from the image servers, the applet then contacts an image retrieval servlet and passes to it the references for the desire images. These references point to files located on the image servers and are used by the servlet to retrieve the images. This servlet will then contact the proper image server, acquire the requested image files, and send them back to the applet for display.

For images that are saved at different institutions, the physician makes use of the same JReads applet. However, when logging in to the system, he or she will select to connect to the TeleRIMS instead of the default RIMS database. The TeleRIMS system contains patient, exam, and image information for exams taken from outside of the Mayo Clinic. Essentially, the TeleRIMS database shares the same architecture as the RIMS database, but it only stores information for patients examined at remote hospitals. An approved remote hospital updates the TeleRIMS system with exam information and references to images through a CORBA object running at Mayo. The JReads servlets then communicate with the TeleRIMS in the same fashion as the RIMS database to provide exam and patient information to the JReads applet. However, when the applet requests images that are not located on the local Mayo image servers, the image retrieval servlet makes a request to a CORBA object running at the remote hospital that handles serving images from there. The servlets transparently retrieve the images locally or remotely; the functionality of the applet is exactly the same despite where the images are loaded.

The following two subsections describe how these components interact for local images and for remote images, respectively. We will take a closer look at how the individual components interact during a typical image review session.

3.2.1. Local images

We will first describe the architecture needed to support the handling and exams of local images:

1. The user first needs to download the JReads applet front-end from a servlet-enabled Web browser just like the pure CORBA architecture previously discussed.
2. The user accesses the login function of the front-end and enters their username and password along with their choice of RIMSs to use. In this case we are using the standard RIMS, which contains information about all patients examined at the Mayo Clinic. The login information is then sent through HTTPS (HTTP over SSL) to the Web server that the applet was originally loaded from. This information is directed to a query servlet that is running on the Web server’s servlet engine. This query servlet connects to the RIMS database using JDBC to test that a connection is possible using the provided username and password. The query servlet then sends back a reply to the applet letting it know whether or not the login to RIMS was successful.

3. If the login was successful, the user can then enter in a patient id to bring up information and a listing of current exams for this individual. When one of the listed exams is clicked upon, the corresponding textual report and notes for this exam will be displayed in the report text area. The applet retrieves the necessary information by making requests to the query servlet running with the Web server. This servlet then performs queries on the RIMS database for the requested information and returns it to the applet.

4. If images are available for a selected exam, the user can click on the open button in the image toolbar to retrieve all of the associated image series for this exam. When the applet retrieves information pertaining to all of the exams for the patient, it also receives references to the associated image files for each exam. The applet then passes the appropriate image references to the retrieval servlet also running with the Web server. The retrieval servlet then uses these references to download the images from a collection of image servers available on the local area network. The retrieval servlet then pipes these image files directly back to the applet. The applet then makes use of specialized classes to interpret and display the medical images as thumbnails.

5. When the applet receives image information back from the retrieval servlet, it displays them as clickable thumbnails and provides the same functionality as in the pure CORBA architecture.

3.2.2. Remote images

We will now describe the architecture needed to support the handling and exams of remote images:

1. When the user wants to browse exams and images for a patient examined at a remote hospital, he or she will make use of the exact same front-end used for viewing patients treated at the Mayo clinic. This front-end can be downloaded as an applet from a servlet-enabled Web browser or used as an application. However, either front-end will need to communicate with the Web server to get the information they need to display.
2. Once the front-end has been loaded, the user then needs to log in providing their username, password,
and radiology management system of choice. In this case, we want to display remote exams and images, so we need to make use of the teleradiology information management system (TeleRIMS). The login information is then sent through HTTPS to a query servlet running on a Web server. This query servlet connects to the TeleRIMS database using JDBC to test that a connection is possible using the provided username and password. The query servlet then sends back a reply to the applet letting it know whether or not the login to TeleRIMS was successful. Note that the TeleRIMS database is set up much like that of the RIMS database, so the process of connecting to it and performing queries on it is nearly identical to that of Mayo’s standard RIMS.

3. If the login was successful, the user can then enter in a patient id to bring up information and a listing of current exams for this individual. When one of the listed exams is clicked upon, the corresponding textual report and notes for this exam will be displayed in the report text area. The applet retrieves the necessary information by making requests to the query servlet running with the Web server. This servlet then performs queries on the TeleRIMS database for the requested information and returns it to the applet.

4. If images are available for a selected exam, the user can click on the open button in the image toolbar to retrieve all of the associated image series for this exam. When the applet retrieves information pertaining to all of the exams for the patient, it also receives references to the associated image files for each exam. The applet then passes the appropriate image references to the retrieval servlet running on the Web server. Since the user has chosen to make use of TeleRIMS, the retrieval servlet knows that the images will not be available on the local image file servers. When the applet makes the request to the retrieval servlet, it provides information about which server to use along with all of the image references. The specified server refers to a specific CORBA ResourceServer object that the servlet needs to access. The servlet gets a reference to the specified CORBA object through the Naming Service and calls a retrieval method on it providing the image references given to it by the applet. The CORBA ResourceServer object runs at the remote hospital and is configured to interface with the particular image storage mechanisms they have in place. The remote hospital is responsible for the actual implementation of this ResourceServer object, and the details of this implementation are hidden from the outside by the ResourceServer’s interface. The important thing is that the ResourceServer object properly honors its interface and can correctly decipher the references given to it. It needs to use these references to access the images stored locally at the remote hospital and pass them back to the client (the image retrieval servlet). The retrieval servlet then pipes these image files directly back to the applet.

5. Just like the local case, when the applet receives image information back from the retrieval servlet, it uses specialized classes to interpret and render them on-screen.

4. Implementation details

This project has been implemented entirely using Java technologies. At the front-end, a Java applet or application serves as the user interface to the system. Java servlets running on a servlet engine installed with the Web server act as middleware and handle the details of interfacing with databases and image servers. These servlets make use of CORBA when necessary for interfacing with these data sources. Iona’s Java ORB product, OrbixWeb™, was used as our CORBA implementation so that security through SSL could potentially be added through integration with their OrbixSSL package. Our code will also work with the ORB provided freely with the Java 2 distribution, since we only make use of the standardized APIs defined by the OMG. Therefore, both our CORBA servers and clients can be compiled and run using either ORB implementation. However, the ORB provided with the Java 2 runtime is not as efficient as the one provided with OrbixWeb, and more importantly to this project, the Java 2 ORB does not have any security capabilities such as SSL support.

4.1. Tools used

For the development of the Java front-end and servlets, we used the Java development kit (JDK) version 1.1.7. The Web server that we used to deliver the applet front-end and to host the servlets was Microsoft® Internet Information Server® (IIS) version 4.0. Allaire’s™ JRun™ version 2.2 was used as the servlet engine for this Web server. The test RIMS and TeleRIMS databases were created using Microsoft’s SQL Server® version 6.5. Iona’s OrbixWeb version 3.2 was used as our CORBA environment. Certificates for the various components that use SSL were created using OpenSSL version 0.9.4.

4.2. Code organization

The code for the JReads system is broken across eight main packages: jreads, jreads.data, jreads.dicom, jreads.gui, jreads.image, jreads.middleware, jreads.middleware.corba, and jreads.util. Fig. 3 shows the hierarchy for these packages.
4.2.1. The JReads package

This is the main package for the JReads program and contains classes for launching the front-end either as an applet or an application as well as a class for keeping track of global settings for all of the other classes.

**JReads.java**

This class launches the JReads front-end as an application.

**JReadsApplet.java**

This class launches the JReads front-end as an applet.

**Settings.java**

This class contains static (class) variables and methods for managing settings used by a variety of JReads classes. It keeps track of user preferences for the various analysis tools, whether or not the front-end was started as an application or applet, the URLs for the query and image retrieval servlets, and other settings. It also provides methods for showing messages in the status bar or in a dialog window that can be called from any other class.

4.2.2. The JReads.data package

This package contains all of the classes that hold data retrieved from a radiology management system. These classes are instantiated by the query servlet, initialized with data from the RIMS database, and then serialized and sent to the applet. The applet then can reconstruct (deserialize) the data objects and access the data they contain. Essentially, these classes are like C structs, since they just hold data and have no methods. They simplify the packaging, transmission, and interpretation of complex data that is sent between the front-end and the middleware.

**PatientInfo.java**

This class contains information pertaining to a particular patient such as his or her name, gender, and age.

**ExamInfo.java**

This class contains information pertaining to a particular exam such as the status of the exam, the time the exam was performed, and the imaging modality used in the exam.

**SeriesInfo.java**

This class contains information pertaining to a particular series of images for a specific exam. It gives details on the server the images reside on, the filenames for the images on that server, the number of images making up the series, and other details.

4.2.3. The JReads.dicom package

This package contains all of the classes needed for parsing, interpreting, and rendering a DICOM file. Fig. 4 shows how these classes work together to construct an image.

**DICOMDataElement.java**

A DICOM data stream consists of a series of data elements that each have a tag and a value (among other things). This class represents a single data element and provides convenience methods for retrieving the value it holds.

**DICOMInputStream.java**

This class provides a buffered stream for reading in DICOM files and methods for reading the data structures (data elements) that compose these files. It can handle the three primary DICOM transfer syntaxes: implicit VR, little endian; explicit VR, little endian; and explicit VR, big endian. It is also capable of properly extracting encapsulated (compressed) pixel data in addition to uncompressed pixel data (the default). This class forms the heart of the DICOM parsing process.

**DICOMObject.java**

This class represents a single DICOM image object. It makes use of a DICOMInputStream to parse out all of the DICOM data elements present, and then stores all of them in a hash table for quick random access.

**DICOMImage.java**

This class takes a DICOMObject and uses the information it holds to construct instances of Java Images. It is capable of handling 8, 12, and 16-bit grayscale images, as well as 24-bit RGB-encoded color images. It provides methods for constructing thumbnail and full-size image objects that can then be displayed by the viewer.
4.2.4. The JReads.gui package

This package contains all of the classes needed to construct the graphical user interface for the front-end. Fig. 5 shows a screen capture of the JReads front-end interface.

**JReadsFrame.java**

This class serves as the main frame for the front-end. All of the other gui components are either contained by this frame or are launched from it. It displays patient and exam textual information as well as providing the interface for viewing medical images.

**ReportPanel.java**

This class forms the top portion of the main JReads interface and displays patient information, exam listings, and exam reports.

**DICOMViewer.java**

This class forms the bottom portion of the main JReads interface and consists of an image toolbar, a thumbnail panel, and two large image displays.

**ImageCanvas.java**

This class provides a display area for Java images. It also automatically handles scaling of an image so that the proper aspect ratio is always maintained.

**ImageDisplay.java**

This is a subclass of ImageCanvas that provides image manipulation and analysis features. It is the gui component that is used to show a larger version of an image in the DICOMViewer.

**ImagePanel.java**

This class is used for displaying all of the images of an exam as clickable thumbnails. Each thumbnail is an ImageCanvas, and when it is selected, a larger version of the image is shown in one of the two ImageDisplays of the DICOMViewer.

**ImageFrame.java**

This class enables an ImageDisplay to be shown in a window separate from the main viewer. This is useful when a user wants to concentrate on a single image and be able to resize it separate from the rest of the interface.

**CineFrame.java**

This class provides an interface for animating through a series of images. It provides controls for stepping through the images in either the forward or reverse direction, adjusting the speed of the animation, and setting the looping style (reset to the first image or...
reverse direction of animation when the last image is reached).

**ToolbarPanel.java**

This class serves as the main toolbar for the viewer and contains a variety of controls for loading and manipulating images as well as launching other tools.

**LoginDialog.java**

This is a Dialog window used for logging a user into the specified RIMS. It requests a username, password, and RIMS choice.

**OptionFrame.java**

This is a Frame that displays the current configuration information and allows a user to adjust it. For example, it allows a user to adjust the increments used in the brightness and contrast controls, the colors used for the text and analysis tools, and the size and magnification level of the magnification tool.

---

### 4.2.5. The JReads.image package

This package contains classes for image processing. Fig. 6 shows a CT image after being processed by these classes.

**Flipper.java**

This class takes an image and flips it along its vertical axis.

**Rotater.java**

This class takes an image and rotates it 90° counterclockwise.

**EdgeFinder.java**

This class uses a simple edge finding technique that takes a threshold representing the maximum difference between pixels belonging to the same “section”. Areas of the image that exceed this threshold belong to different sections, and an edge is drawn between them. This class reduces the given image to black and white where...
white lines represent the edges in the image. This class can be helpful when used in conjunction with the StructureFiller, since it eliminates any color gradation within the image. It also enables the boundaries of a structure within the image to be clearly delineated so that the fill does not spill over.

**Inverter.java**

This class takes an image and inverts all of its pixels (creates a “negative” of the image).

**StructureFiller.java**

This class fills in a region of an image starting at a given point. The pixels to be included in the fill are determined by the color value of the initial point and the current threshold. The total area of the filled region is also displayed in square millimeters.

### 4.2.6. The JReads.middleware package

This package contains the two servlets that form the heart of the middleware for the JReads system. Our two system architectures are differentiated solely by changes in how the query and image retrieval servlets are implemented. In each architecture, we have a different implementation for each of these servlets, while keeping the front-end and back-end components the same. This package also contains a class for abstracting database communication details that is used by both the query servlet and the RIMS CORBA object.

**QueryServlet.java**

This is a servlet that handles all of the database querying for the JReads front-end. In the pure CORBA architecture, this query servlet communicates with the RIMS through a RIMS CORBA object. In the partial CORBA architecture, it connects directly to either the RIMS or TeleRIMS databases through JDBC to acquire information. This servlet can handle five basic requests from clients. These include requests for database connection testing, patient information retrieval, basic exam information retrieval, exam report retrieval, and exam series information retrieval. It uses the DBQuery class to actually query the database for the requested information and makes use of object serialization to pass initialized data objects back to the JReads front-end.

**ImageRetrievalServlet.java**

This is a servlet that handles the loading of images from various image file servers that are either local or remote. In the pure CORBA architecture, this servlet communicates through ResourceServer CORBA objects to acquire images no matter where they are located. In the partial CORBA architecture, this servlet is more discriminating as it will only use a ResourceServer CORBA object for retrieving remote images and will directly load images available on the local image servers. This is done to avoid the overhead of CORBA where necessary. In either architecture, once the DICOM file for the requested image is acquired from an image server, it is sent in its entirety to the JReads front-end for further processing and display.

**DBQuery.java**

This class takes care of the lower level details of database access for the QueryServlet. It has static methods that handle querying a database, processing the results, and packaging and returning the relevant information to the QueryServlet.

### 4.2.7. The JReads.middleware.corba package

This package contains the IDL definitions and CORBA object implementations that support the telediagnosis portion of the JReads system.

**RIMSServer.java**

This class comprises the server for querying or updating a RIMS database by remote clients. Its main purpose is to create a new RIMSServant object and register it with the ORB and the naming service so that clients can access it.

**RIMSServant.java**

This class implements the RIMS interface specified in the TeleRad IDL file and performs the actual work of querying or updating a RIMS database for a remote client.

**RIMSUpdater.java**

This is just a sample client for a hypothetical remote hospital that makes use of an instance of the RIMS CORBA object running at Mayo. Each hospital would actually need to create their own client class so that they could control what information is sent to Mayo’s RIMS. They may want to send information automatically from their PACS when medical scans are performed for a particular patient or they may want some sort of manual system. It really just depends on the needs of the remote hospital and what information they want to provide to Mayo.

**ResourceServer.java**

This class comprises the server that needs to be running at any remote hospital participating in the JReads system. Its main purpose is to create a new ResourceServant object and register it with the ORB and the naming service so that clients can access it.

**ResourceServant.java**

The class implements the ResourceServer interface specified in the TeleRad IDL file and performs the actual work of retrieving requested images. This class needs to contain code for dealing with the specific image archiving systems at the remote hospital and for interpreting the references passed to it by the client. It returns DICOM image data to clients.

### 4.2.8. The JReads.util package

This package contains utility classes that simplify access to the middleware.
DataRetriever.java
This class handles the details of getting data from the middleware for the JReads front-end. It has several static methods that return a data object that the applet can directly process such as a PatientInfo or DICOM-Image object. It makes use of the ServletInvoker class to communicate with the middleware.

ServletInvoker.java
This class handles the details of connecting to a given servlet and passing the appropriate parameters to it. It returns a DataInputStream to the clients of this class that can be used to read any output from the servlet.

4.3. JReads functionality

The JReads front-end applet was designed to provide all of the basic tools needed by a physician to examine a wide range of medical images. It is capable of interpreting and rendering images stored in the standard DICOM format. Currently, this front-end can handle images that are stored as 8, 12, or 16 bit grayscale or are stored as 24-bit RGB color.

The primary features include:

• Patient, exam, and exam report information given in a concise view.
• Display of all images for a given exam as clickable thumbnails.
• Ability to adjust the brightness and contrast of an image (change its window width and level).
• Rotation of image in 90° increments.
• Flip image along vertical axis.
• Cycle through previous/next image in a series.
• Cycle through previous/next series.
• Cine mode.
• Individual display mode (image shown in separate, resizable window).
• Edge finding.
• Color inversion.
• Analysis tools
  • Ruler (provides linear measurements in mm)
  • Area calculator (provides rectangular area measurements in mm²)
  • Magnifier
  • Structure filler.
• Options menu for selecting defaults for magnification levels, brightness and contrast increments, edge finding and structure filler thresholds, and various other attributes of the viewer’s environment.

4.4. Comparison of systems

Table 1 provides a concise summary of features present in each of the related systems and JReads. It lists features that were deemed important in the creation of the JReads system and is by no means an exhaustive list of the capabilities of any of these systems. Each system really has its own focus, capabilities, and advantages that differ from the others.

QReads was designed to be a general medical image viewer with some basic image manipulation tools for the Windows platform. The Neuro-Imaging Digital Library System focuses on the more specialized task of presenting neurological images and providing advanced imaging features. TeleMed is a general telemedicine system with the goal of creating virtual patient records from distributed data sources and presenting them to the user in a graphical fashion. JReads has a focus similar to QReads in that it was designed for general medical image viewing and providing easy-to-use image manipulation tools. However, we have added some additional image analysis tools and have expanded its data retrieval capabilities so that it can function in a teleradiology capacity over the Internet.

5. Conclusions and future work

The Internet provides a tremendous forum for medical imaging and medicine in general. It frees physicians
from being tied to specific workstations within an institution and allows them to review information from almost any computer around the world. It also allows us to bring together information that is stored at geographically distributed sites and present it to the user in one uniform view. In this paper, we discussed a medical imaging viewing system that harnesses this power of the Internet. This system uses a wide variety of technologies such as Java, CORBA, and DICOM to provide an Internet-enabled, platform-independent, standards-based solution. We presented two architectures for this system. The first one uses CORBA throughout the middleware to provide location transparency for the images and a consistent mechanism for accessing all of the data stores (RIMS and image servers). The second one uses direct connections to local data stores available at the Mayo Clinic and only uses CORBA for accessing images stored at remote hospitals. This approach is more efficient since it allows us to avoid the overhead of CORBA when dealing with local data stores, and it also enables this system to fit more easily into Mayo’s existing architecture without requiring changes to major system components such as RIMS.

Future work for this system includes:

- Elimination of the middleware servlets so that there is less overhead especially when using SSL. It was not possible to do this in the current system due to the connection restrictions placed on applets by browsers (an applet can only make a network connection back to the machine from which it was originally loaded). Using trusted applets would have resolved this problem but would have created others due to inconsistencies in how the current Web browsers support them.

- Integration of the JReads system with other hospital information systems to provide a more complete telemedicine system. In this way, patient records could be provided in addition to exam reports and images.

- Addition of the capability to decode JPEG2000 encoded pixel data once this standard is finalized. JPEG2000 offers a wavelet compression scheme that enables excellent compression ratios. Currently, JReads supports the standard DICOM format which calls for uncompressed pixel data.

References


Shawn P. Laird is a Software Programmer at a financial company in Des Moines, Iowa. He has BS degree in Mathematics and Computer Science from William Penn College in May 1997 and MS degree in Computer Science from Iowa State University in December 1999. He is building systems to collecting information for financial institution via the Internet. His research interests are in distributed computing and Web-based applications.

Dr. Johnny S.K. Wong is a Full Professor of the Computer Science Department, Iowa State University at Ames, Iowa USA. His research interests include operating systems, distributed systems, telecommunication networks, broadband-integrated services digital networks, concurrency control and recovery; multimedia and hypermedia systems, intelligent multi-agents systems, intrusion detection. He has been an investigator for research contracts with Telecom Australia from 1983 to 1986, studying the performance of network protocols of the ISDN. During this period, he has contributed to the study and evaluation of the communication architecture and protocols of ISDN. From 1989 to 1990, he was the Principal Investigator for a research contract with Microwave Systems Corporation at Des Moines, Iowa. This involved the study of coordinated multimedia communication in ISDN. In Summers 1991 and 1992, Dr. Wong was supported by IBM corporation in Rochester. While at IBM, he worked on the distributed computing environment (DCE) for the application systems. This involved the integration of communication protocols and distributed database concepts. Dr. Wong is also involved in the coordinated multimedia system (COMS) in Courseware Matrix Software Project, funded by NSF synthesis Coalition project to enhance engineering education. From 1993–1996, he is the working on a research project on a knowledge-based system for energy conservation education using multimedia communication technology, funded by the Iowa Energy Center. From 1995–1996, he was supported by the Ames Laboratory of the Department of Energy (DOE), working in Middleware for Multidatabases system. He was involved in projects on intelligent multi-agents for intrusion detection and countermeasures funded the Department of Defense (DoD), database generating and X-ray displaying on the World-Wide Web applications funded by Mayo Foundation. Currently, he is working on the CISE Educational Innovation: Integrated Security...
Curricular Modules and NSF SFS Program on Information Assurance, both funded by the National Science Foundation (NSF).

William J. Schaller is a Senior Scientific Programmer at the Mayo Clinic in Rochester, MN. He has degrees in Mathematics and Physics. He is building systems to collecting information on Cancer patients and distribute summary data via the Internet.

Dr. Piet C. de Groen is consultant in Gastroenterology and Hepatology at the Mayo Clinic in Rochester, MN. He is an Associate Professor in Medicine and Gastroenterology at Mayo Medical School.

Dr. de Groen’s clinical research objective is to decrease the mortality due to cholangiocarcinoma (cancer of the bile ducts) by actively pursuing early detection of this disease in patients at risk, develop new or improve current diagnostic modalities, and finally investigate new treatment options. This is important as cholangiocarcinoma develops in a significant number of patients with diseases of the biliary tree, such as primary sclerosing cholangitis, intraductal gallstones, biliary tree strictures and choledochal cysts. Mayo Clinic is one of the largest referral centers for this type of tumor.

In addition, Dr. de Groen has an interest in bioinformatics. He has led the development of Web-based databases and applications for the Mayo Clinic Cancer Center in Rochester and currently is leading an effort toward creation of a comprehensive system allowing access to both clinical as well as genome- and proteome-derived data for use by all researchers at Mayo Clinic Rochester.

Dr. Bradley J. Erickson is a Neuroradiologist at Mayo Clinic with research interests in the technology and implementation the electronic medical practice, with a focus on the radiology component. Related interests are the impact of lossy image compression on diagnostic accuracy and computer-aided diagnosis.