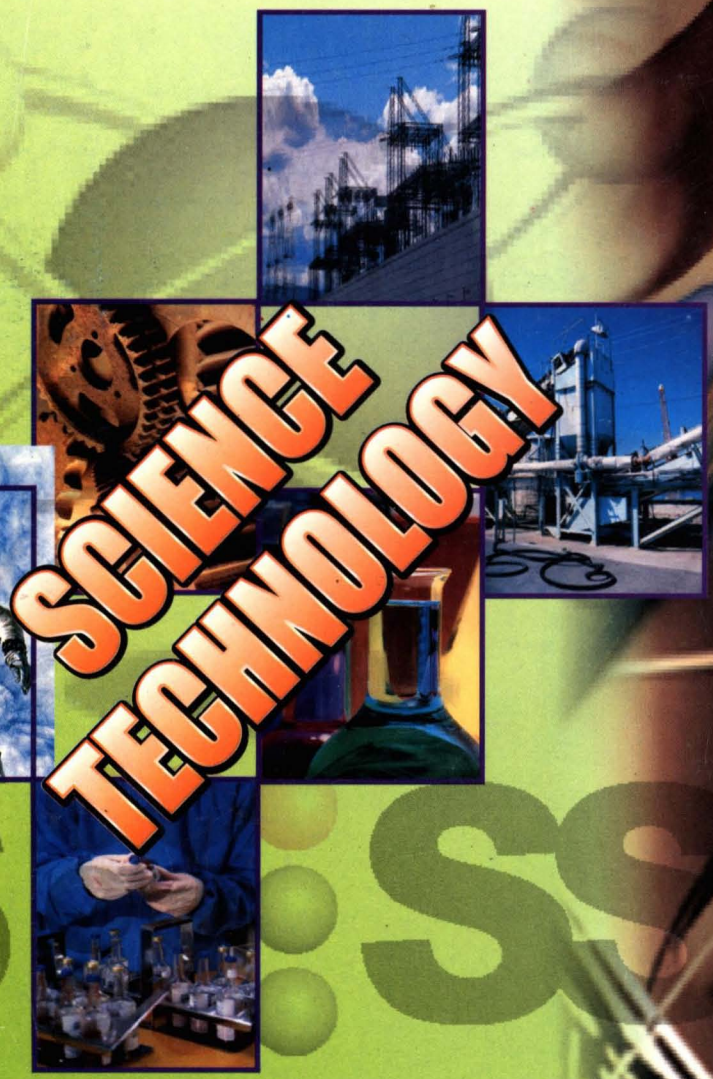
UNIVERSITI TEKNOLOGI MARA

ISO 9001:2000 No Sijil :0500132

Globalising Knowledge and Information



NATIONAL SEMINAR ON SCIENCE TECHNOLOGY & SOCIAL SCIENCES 2006

30-31 May 2006

Swiss Garden Resort & Spa

Kuantan, Pahang

Utilising Java Technology with CORBA for Medical Informatics

Ahmad Shukri M Noor
Md Yazid Mohd Saman
Mustafa M. Deris
Sarif A Manap

ABSTRACT

We are conducting research and developing a distributed multimedia database system for medical informatics to be used by the mobile physician. The ability the java based web application to distribute information to geographically distant users on a wide variety of computers. Makes it an obvious candidate for a technological solution for distributed medical informatics systems. This paper describes how to use Java tools and extensions, like the Java Media Framework (JMF), Java Swing, Java Foundation Class (JFC) and JDBC. This Java technology is utilized with Common Object Request Broker Architecture (CORBA) in order to develop a medical informatics systems model in distributed environments. This model is utilized java as client application and server application; java also can be used as client applet in web browsers. Furthermore we integrate java with CORBA as middleware that sit in middle-tier and IBM DB2 as relational database at the server side.

Keywords: *Java, CORBA, DICOM, medical imagine, medical informatics, distributed object computing.*

Introduction

Digital medical images are commonly used in hospitals today, even outside the radiology department. Because of the interrelatedness of the radiology department and other departments, especially the intensive care unit and emergency department, the transmission of medical images has become a critical issue. The use of World Wide Web and network related technologies in radiology is not new. These technologies have been used in radiology teaching files to access information in multimedia integrated picture archiving and communication systems (PACS), for teleradiology purposes. Web technology has also been used to access the images stored in a Digital Imaging and Communications in Medicine (DICOM) archive in PACS environments (DICOM 2003, Mildenberger, P. and Eichelberg, M. Martin 2002).

Distributed Computing

Distributed Computing is also known as networked or client-server computing. It has been described as a logical computing model defined not by a specific technology or set of technologies, but by interoperability based on standards, by the hiding or encapsulation of complexity through documented, predictable interfaces, and a redistribution of processing based on business requirements. Simply put, parts of an application run on different computers simultaneously (Tari, Z. and Bukres, O. 2001, http://java.sun.com/j2ee/corba/index.html, http://java.sun.com/developer/technicalArticles/RMI/rmi_corba/).

A distributed application is an application whose software components reside on more than one computer in a network, with the network typically composed of diverse computers and operating systems (a heterogeneous network) (http://gsraj.tripod.com).

A distributed object-based system is a collection of objects that isolates the requesters of services (clients) from the providers of services (servers) by a well-defined encapsulating interface. In other words, clients are isolated from the implementation of services as data representations and executable code. This is one of the main differences that distinguishes the distributed object-based model from the pure client-server model (Tari and Bukres 2001).

In the distributed object technology, a client sends a message to an object, which in turns interprets the message to decide what service to perform. This service or method, selection could be performed by either the object or a broker

Distributed Object

At present, a lot of new applications are being developed based on the object-oriented philosophy and standards, together with distributed-object concepts. The design and implementation of these two concepts are difficult so that the applications developed could deliver business value to the business world, which mostly are driven by the information and technology. Other than that, information must be accessible across networks (either through local or

remote machines) and must accurate all the time. The software written must be able to run on a network where all the functionality components are distributed to machines in the network (different platforms) and use different programming languages for the components of the system. For example, Java maybe used at the user interface in front end and C++ maybe used at the main functionality components in back end. Anyway, the developed components of the system must also be integrated easily into new systems. With all these requirements and complexities, an application needs to have a three-tier distributed object- computing architecture that might be the answer for these problems.

The distributed and heterogeneous nature of today computing systems requires a middleware infrastructure capable of supporting a three-tier computing architecture such as Common Object Request Broker Architecture (CORBA). Business logic can be built, or existing applications encapsulated, into middle-tier components that interact with end users via standard interfaces such as web browsers and standard GUI desktops, and back-end data repositories (http://java.sun.com/developer/technicalArticles/RMI/rmi_corba/).

Common Object Request Broker Architecture (CORBA)

CORBA was introduced by OMG in 1991 to go a step beyond OMA to specify technologies for interoperable distributed OO systems. Figure 1 shows the CORBA in a client/server system. It is structured to allow integration of a wide variety of object systems. With the CORBA specification, a broad and consistent model for building distributed applications is defined:

i. An object-oriented based model for developing applications
ii. A common application programming objects in the network to be shared by client and server applications
iii. A syntax to define and describe the interfaces of objects used in the environment
iv. Support for multiple programming languages and platforms

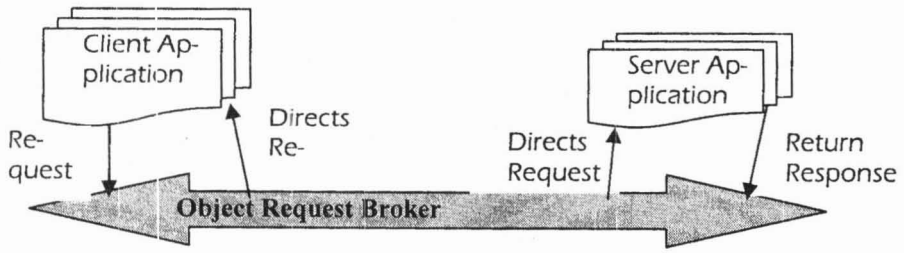


Fig. 1: CORBA in Client/Server Environment (Tari,and Bukres 2001)

Therefore, CORBA model formally separates the client and server portions of the application and also logically separates an application into objects that can perform certain functions. It also provides data marshaling to send and receive data with remote or local machine applications without direct knowledge of the information source or its location. In the CORBA environment, client and server applications communicate using Object Request Broker (ORB).

Java Programming

The Java programming language is a strongly typed, object-oriented language that borrows heavily most of its syntax from C and C++. Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high performance, multithreaded and dynamic language. This language was primarily used for developing applets-downloadable mini-applications that could be embedded inside Web pages and performed in browsers. However, since 1995, Java has emerged as a first-class programming language that is being used for everything from embedded devices to enterprise servers. Nowadays the Java language can be seen in use in a wider range of applications.

When an application is written and compiled in one place it can run on any machine under any operating system. Sometimes the "Write Once, Run Anywhere" slogan is called the synonym of Java. Anyway, platform independence is the ability of a program to move from one computer system to another. Java is platform independent at both the source and the binary level. The secret of the Java has been hidden into Java Virtual machine (JVM). Instead of creating machine dependent code, the Java compiler creates a byte code format, which can be run on any

Virtual Machine (VM). Somehow, Java makes programming easier because it is object-oriented and has automatic garbage collection.

Java offers tremendous flexibility for distributed application development, but it currently does not support a client/server paradigm. To do this, Java needs to be augmented with a distributed object infrastructure, which is where OMG's CORBA comes into the picture. Using CORBA requires more than just knowledge of the CORBA architecture. CORBA should be part of well designed system architecture.

CORBA technology as part of the Java 2 platform consists of an Object Request Broker (ORB) written in Java. Java IDL adds CORBA capability to the Java platform, providing standards-based interoperability and connectivity. Java IDL enables distributed Web-enabled Java applications to transparently invoke operations on remote network services using the industry standard OMG IDL (Interface Definition Language) and IIOP (Internet Inter-ORB Protocol) defined by the Object Management Group.

The Digital Imaging and Communications in Medicine (DICOM)

The Digital Imaging and Communications in Medicine (DICOM) standard was created by the National Electrical Manufacturers Association (NEMA) to aid the distribution and viewing of medical images, such as CT scans and ultrasound.

New technologies such as Java should always be used as a complement of the de facto standard in medical imagine, DICOM. DICOM allows the interchange of images from different modalities, archives, and workstations from different vendors. Java technology can be used to build a storage system and to make this service accessible for different clients. However, this storage service should also incorporate DICOM services to store and access examination data from DICOM workstations and DICOM modalities.

Since Java 1.4, the Java standard includes a specification for working with images stored in files and accessed across the network. This specification is called Java Image I/O. It provides a pluggable framework for easily adding support for alternate image formats using third-party plugins. The DICOM Image I/O Plug-in connects the DICOM® standard to the JavaTM standard. DICOM is the universal standard for sharing medical imaging resources between heterogeneous and multi-vendor equipments (acquisition device, workstation, storage server, patient management system, etc.).

Java Medical Imaging Application

Medical imaging applications are becoming more commonplace. An important and common stage in the development of such applications is the interpretation of medical image data. These data are generally stored in accordance with the Digital Imaging and Communications in Medicine (DICOM) standard (DICOM 2003) (summarized by Mildenberger et al 2002). Interpretation of medical images involves decoding the relevant DICOM data and making them readily available to the application developer for analysis and display. Our research group has developed a Java-based medical imaging application framework to facilitate the rapid development and deployment of medical imaging applications in a distributed environment.

The medical imaging application developers interface(API), which will be referred to as the NeatMed interface (API), was developed using the Java programming language (Sun Microsystems, Mountain View, Calif).

An extension API is a set of classes that can be instantiated by a programmer to create a particular type of application, thus facilitating software reuse. NeatMed is an example of an extension API that can be used for the development of applications that deal with off-line medical image data. NeatMed currently provides support for the Digital Imaging and Communications in Medicine and Analyze medical image file formats. The NeatMed API is a group of core and support classes that can be used to interpret, represent, and manipulate images and related data that are stored in DICOM-compliant files. The central class in the API is the DICOMImage class. A DICOMImage object can be instantiated by specifying a reference to a suitable data source in the constructor. The constructor will accept data from a number of sources (eg, local file, data stream, and remote uniform resource locator [URL]). Once constructed, a DICOMImage object provides direct access to all of the data elements stored within the specified DICOM source. Other classes in the API are used to represent individual components within a DICOM RadioGraphics.

The NeatMed API can be used to develop a wide variety of medical imaging applications. This section describes a number of sample applications that demonstrate the power, flexibility, and ease of use of the NeatMed API (Figure 2).

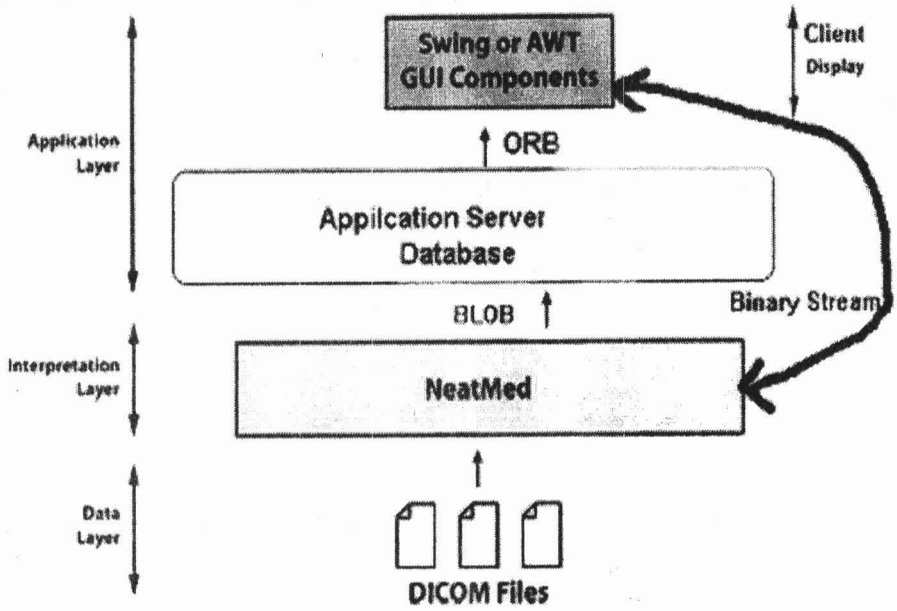


Fig. 2: Flexibility, and Ease of Use of the NeatMed API in Java Environment

The choice of Java for implementing the NeatMed API was also influenced by a number of its key features:

Ease of Use

Java is a modern programming language that was designed with simplicity in mind. Many of the complexities that are associated with other programming languages have been omitted, whereas much of the power and flexibility has been retained. This makes Java very easy to learn and use, particularly in the case of novice programmers.

Level of Support

Although Java is a relatively new programming language, there is a significant amount of support material available. Numerous texts have been written dealing with all aspects of the language. In addition, tutorials, sample source code, API documentation, and freely available integrated development environments (IDEs) can all be accessed via the Internet.

Portability

Java is a multiplatform programming language. This means that a Java application developed on one operating system (eg, MacOS [Apple Computer, Cupertino, Calif]) can be deployed on a number of other different operating systems (eg, Windows [Microsoft, Redmond, Wash], Solaris [Sun Microsystems], and Linux). This is sometimes referred to as the "write once, run anywhere" paradigm. In theory, the use of a multiplatform programming language significantly increases the potential user base with little or no extra development overhead.

Core Functionality

The core Java libraries encapsulate an extensive range of functionality that can be easily reused to create reliable, diverse, and powerful applications. Some of the main capabilities supported by the core libraries include networking, file input/output, image processing, database access, and graphical user interface (GUI) development.

Extension APIs and Toolkits

There is a large number of extension APIs and toolkits that can be used in conjunction with the core Java libraries. These extensions usually deal with a particular specialty such as advanced image processing, three-dimensional graphics, or speech recognition. The concept of a Java-based medical imaging toolkit is not entirely new.

System Framework

Many multimedia applications, such as recording and playback of motion video and audio, slide presentations, and video conferencing, require real-time presentation of the media data stream and the synchronized display of multiple media data streams. Because of these time-related requirements, the allocation of various system resources becomes the most important aspect of the system design. Specifically, a successful system must maintain balanced tradeoffs between granting client requests and the overall performance of the system. On the one hand, when clients make their requests and expect these requests to be granted by the server.

To realize the above design, the System Framework follows the generalized CORBA-based three-tier architecture as shown in Figure 3. This helps resolve the complexity of the development of the distributed multimedia applications. The system architecture and protocols must be designed to achieve the above goals. Some functions may be needed and implemented at both client and server sides to achieve the best system performance. At the client side, the synchronized presentation of multiple media streams must be maintained.

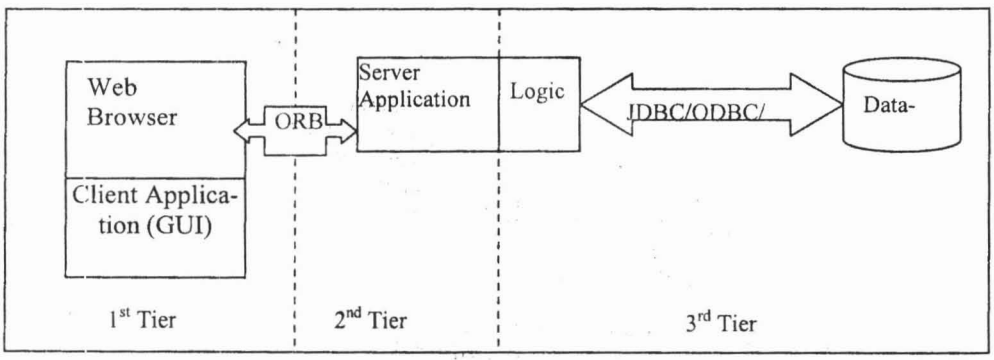


Fig. 3: Integrating CORBA and JAVA For Developing Distributed Objects Medical Imagine

The CORBA/JAVA-based distributed object adopts the three-tier client-server architecture. The distributed object utilises a Java applet and java application as a client for accessing a remote database server. The traditional method utilised a two-tier architecture in which all client functions are implemented entirely in Java. This method requires extensive programming effort because a Java client has to be implemented at the protocol level for vendor-specific database. Current implementation of database clients using two-tier architecture should provide extensive API class libraries so that most database functions with all parameters can be set and issued by the client. The result is that the system could become quite complex and the learning curve is so high for developers have to deal with a complete new set of APIs and protocols.

CORBA Object Request Brokers (ORB) provides remote instantiation of an object. That is, the ORB allows a Java class on a client machine to instantiate a class on a separate machine such as server machine that communicates via its methods. The ORB makes the fact that a method call in fact traverses machines transparent. The model inherits advantages of both CORBA and JAVA, so it can perfectly solve problems of the traditional HTTP/CGI model.

The CORBA and Java integration can construct distributed heterogeneous applications and implement distributed-object computing system easily. CORBA and Java provide different methods and different viewpoints to solve the task. Essentially, Java objects are not distributed. But the neutralism and dynamic expandability of Java make it a good candidate to construct distributed heterogeneous applications. In fact, CORBA and Java are complement instead of competing with each other. CORBA provides Java with distributed object framework and enables applets to communicate with remote objects that programmed in other languages over different address spaces. Moreover, CORBA provides many distributed object services, such as transaction service, security service, yellow pages service and persistence service, to enhance Java functions.

Database Connectivity

The Java Development Kit (JDK) has already supplied together with the Java Database Connectivity (JDBC) module to allow Java programmers connecting and querying remote databases. That is, JDBC is a set of Java classes to provide a Java object interface to SQL databases. JDBC can be viewed from a high-level abstract view or from a low-level database specific view. Figure 4 shows the relationship between JDBC API and database driver. In brevity, the JDBC API is implemented via a Database vendor Native driver or Object Database Connectivity ODBC.

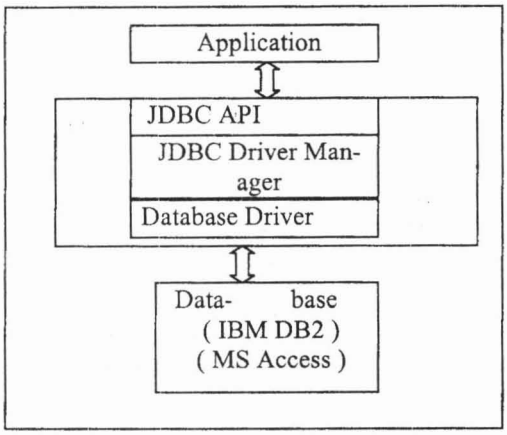


Fig. 4: JDBC API and Database Driver Relationship

JDBC API high-level view, the JDBC application programming interface (API) defines Java classes to represent database connections, SQL statements, result sets, database metadata, and so on. It allows a Java programmer to issue SQL statements and process the results. JDBC is the primary API for database access in Java. In short, the JDBC API provides methods allowing an application to connect, query, and manipulate databases. At low-level view (JDBC Driver), a database specific implementation of the JDBC abstract classes, called a JDBC driver, must be provided in order for the Java database programmer to access the database. JDBC drivers can either be entirely written in Java, so that they can be downloaded as part of an applet, or they can be implemented using native methods to bridge existing database access libraries.

So, a database application obtaining database access through the JDBC API will work with any data source providing a JDBC driver. IBM DB2 database has been selected for this application due its capability to handle Binary Large Objects or Large Objects (BLOBs) files. BLOBs in general, are used to store unstructured data. Unstructured data is data that cannot be decomposed into a relational schema. Examples are pictures in any format like GIF and JPEG, written documents like Microsoft Word, WordPerfect, or multimedia content as audio and video filesx since the IBM DB2 Database supports the storage of data BLOBs, all the medical images data can be stored in their native binary format in a particular column of the database table.

Implementation

In this edical imagine record system, patient's detail screen and medical image screen are reside in separated windows. For the query purposes, Firstly the details record must be retrieved as shown in Figure 5 below then system automatically search medical image for that record. Next, the system pop-up the medical image as shown Figure 6.

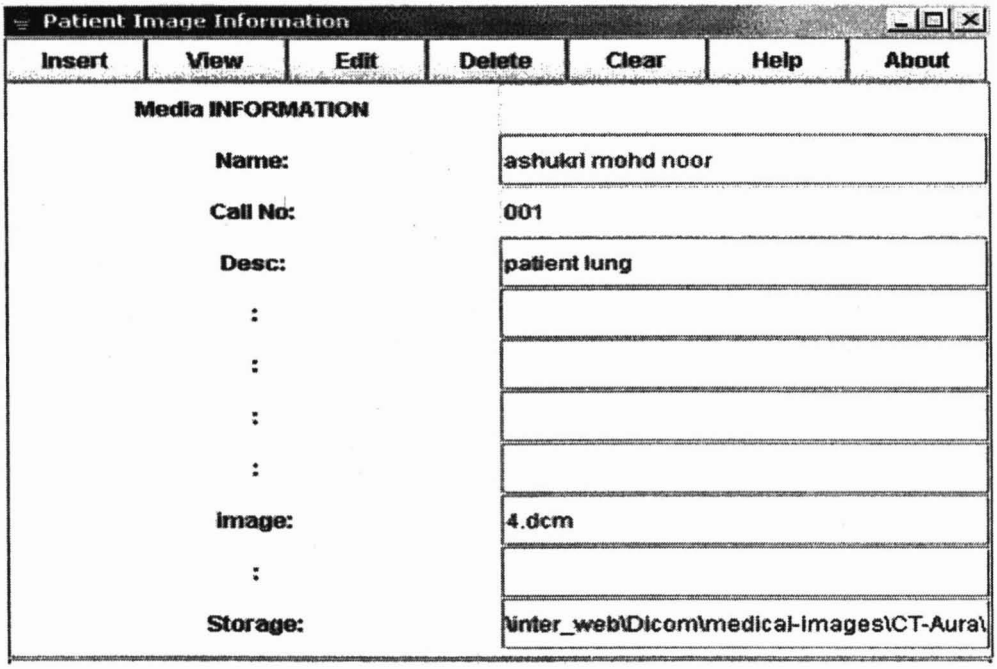


Fig. 5: Patient's Detail Screen Shot

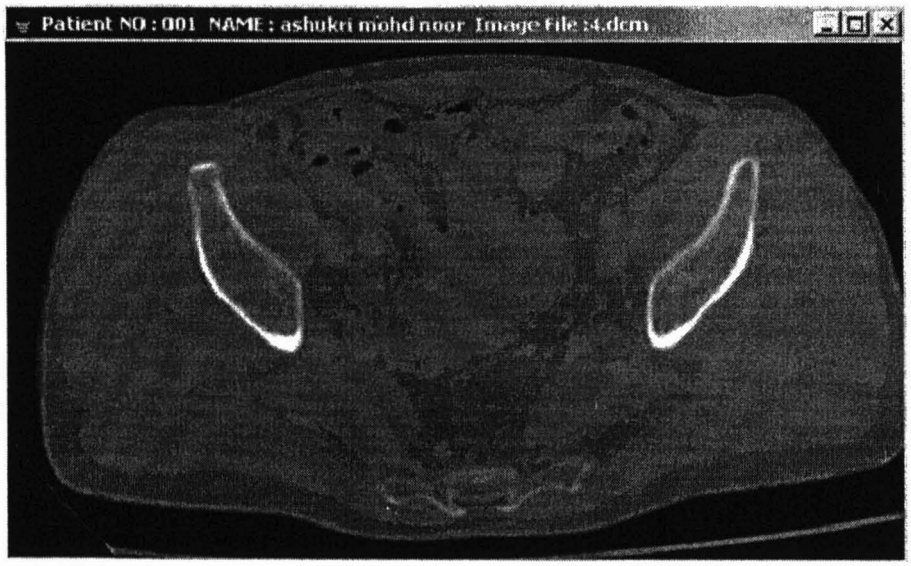


Fig. 6: Patient's Medical Image Record Screen.

Conclusion

The three-tier distributed medical imagine application allow the application interoperability and independence of platform, operating system, programming language and even of network and protocol
The application framework is an important and common stage in the development of any medical imaging application in distributed environment.

NeatMed removes the need to deal directly with encoded medical image data, thus increasing productivity and allowing the developer to concentrate on other aspects of application development. NeatMed is written in Java, a multiplatform programming language with a large amount of freely available support material that is straightforward

to learn and use.

These and other features of Java make NeatMed accessible to a large group of potential users. NeatMed is well supported with documentation and sample code available through the NeatMed Web site. Most important, NeatMed is a freely available research tool whose ongoing development is driven by the needs and requirements of its users.

References

Fernandez-Bayo, J., Barbero, O., Rubies, C., Sentis, M. & Donoso, L. (2000). Distributing Medical Images with Internet Technologies: A DICOM Web Server and a DICOM Java Viewer. *RadioGraphics*, 20:pp. 581-591.

Knoll, B., Kenny, S., Mirzaei, K., Koriska, H. Kohn & Fernandez-Bayo, J. (2000, November 1). Application of a "Jini-enabled" Patient Data Storage System. *RadioGraphics*, 20(6):pp. 1817 - 1818.

Mildenberger, P., Eichelberg, M. & Martin, E. (2002). *Introduction to the DICOM standard.* Eur Radiol, 12:pp. 920-927. [Medline].

National Electrical Manufacturers Association. (2003). *Digital Imaging and Communications in Medicine (DICOM).* Rosslyn, Va: National Electrical Manufacturers Association, PS 3, 1-2003– 3.16-2003.

Nimsky,C. & Eberhardt, K. E. W. (2001, November 1). Local and Remote Visualization Techniques for Interactive Direct Volume Rendering in Neuroradiology. *RadioGraphics*, 21(6):pp. 1561 - 1572.

Robinson, K., Whelan, P.F. & Stack, J. (2003). Segmentation of the Biliary Tree in MRCP Data. *Proc SPIE*, 4877:192-200.

Rosset, A., Ratib, O., Geissbuhler, A. & Vallee, J.-P. (2002, November 1). Integration of a Multimedia Teaching and Reference Database in a PACS Environment. *RadioGraphics*, 22(6):pp. 1567 - 1577.

Sadleir, R. J. T., Whelan, P. F., MacMathuna, P. & Fenlon, H. M. (2004, July 1). Informatics in Radiology (infoRAD): Portable Toolkit for Providing Straightforward Access to Medical Image Data. *RadioGraphics*, 24 (4):pp. 1193 - 1202.

Sadleir, R.J. & Whelan, P.F. (2002). Colon Centreline Calculation for CT Colonography using Optimised 3D Topological Thinning. *Proceedings of the 1st International Symposium on 3D Data Processing Visualization and Transmission.* Washington, DC: IEEE Computer Society:pp. 800-803.

Sadleir, R.J. & Whelan, P.F., Bruzzi, J.F., Moss, A.C., Fenlon, H.M. & MacMathuna, P. (2002). A Novel Technique for Identifying the Individual Regions of the Human Colon at CT Colonography. *Proceedings of the IEE Seminar on Medical Applications of Signal Processing.* London, England: Institution of Electrical Engineers, 8/1-8/5.

Tari, Z. & Bukres, O.(2001). *Fundamentals of Distributed Object System, the CORBA Perspective.* John Wiley.

Tomandl, B. F., Hastreiter, P., Rezk-Salama, C., Engel, K., Ertl, T., Huk, W. J., Naraghi, R., Ganslandt, O.,

_____. (No date). *A Detailed Comparison of CORBA, DCOM and Java/RMI.* Retrieved on 23.1.2003 [Online] Available: http://gsraj.tripod.com

_____. (No date). Corba and RMI. Retrieved on 23.2.2003. http://java.sun.com/developer/technicalArticles/RMI/rmi_corba/

_____. (No date). *Java and Corba.* Retrieved on 23.1.2003. http://java.sun.com/j2ee/corba/index.html

---

AHMAD SHUKRI M NOOR, MD YAZID MOHD SAMAN, MUSTAFA M. DERIS & SARIF A MANAP, Department of Computer Science , Faculty of Science and Technology, University College of Science and Technology , 21030 Kuala Terengganu. shukri@kyt.edu.my