Redesign CORBA Framework for distributed Medical Informatics model development

Ahmad Shukri M Noor and Md Yazid Mohd Saman
Department of Computer Science, Faculty of Science and Technology
University College of Science and Technology,
21030 Kuala Terengganu, Malaysia
096683159/09-6694660(FAX)
E-mail address: shukri@kyt.edu.my

Abstract—The distributed medical informatics (DMI) is a unique suite of multimedia telemedicine applications developed by our team. The applications support real-time patients’ data, image files, audio and video diagnosis annotation exchanges. The DMI enables joint collaboration between radiologists and physicians while they are at distant geographical locations. The DMI environment consists of heterogeneous, autonomous, and legacy resources. The Common Object Request Broker Architecture (CORBA), Java Database Connectivity (JDBC), and Java language provide the capability to combine the DMI resources into an integrated, interoperable, and scalable system. The underneath technology, including IDL ORB, Event Service, IIOP JDBC/ODBC, legacy system wrapping and Java implementation are explored. This distributed collaborative CORBA/JDBC based framework will challenge the advanced, medical information management requirements. It also makes the DMI both hardware and software technologically independent. As our research and development extend, we will continue to incorporate the latest advances in computer technology. DMI is not another new tool in teleMedicine, but rather a new paradigm for the delivery of health services that requires process reengineering, cultural changes, as well as organisational change. It is a whole new way of practicing in telemedicine. We believe that the DMI project has long-term, comprehensive solutions for today tomorrow-healthcare needs.

I. 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[1][2].

II. PROJECT BACKGROUND

This research will develop a framework of distributed medical informatics that can be used for multimedia data exchange. Framework can be expanded to any distributed object oriented, collaborative applications, for example, distance learning modeling and simulation.

A DMI system based on distributed object computing system. The system can be viewed as a set of object services and a set of client applications. Each client application has a defined, interactive user interface. The object services provide and manage the information for the DMI clients. The ultimate goal is to have a complete set of services with a single fine-grained framework. The DMI strategy is an approach towards a single architecture where hardware and software from multiple vendors coexist in harmony. This is achieved by categorizing information into components or services (object services) as they communicate, by passing the information via interface invocations of objects. These object services are manufactured by different vendors and can run on different computers on networks.

Clearly, the integration methodology used to support such architecture must have certain key characteristics:

1. **distributed:** it must support a service object model that is distributed across a regional area over LAN and WAN networks.

2. **platform independent:** it must support multiple computing platforms, from mainframes to servers to desktop PCs.

3. **heterogeneous:** it must support all different types and classes of medical equipment and software tools from many different vendors.

4. **location insensitive:** it must allow components in the system to be replaced, repaired, upgraded and changed without compromising its ability.

Obvious, interoperability is a key technology that allows this exchange to scale. Interoperability is also the ability to leverage and reuse system content and functionality to an end user or to another system.

III. APPROACH

The Common Object Request Broker Architecture (CORBA) of the Object Management Group (OMG) sets forth the open system, cross platform standard
which best meets DMI requirements. CORBA is based upon the notion of an Object Request Broker (ORB) that is used to register objects and methods across a network and is accessed by applications wishing to utilize or modify information over the network. CORBA-based component software is universally accessible and interoperates across boundaries of operating systems, networks, languages, development tools and applications, as well as interface styles. Furthermore, CORBA Interface Definition Language (IDL) is both platform and language independent and can be supported in virtually any distributed computing environment. This network-centric view of distributed computing is an ideal fit for the aforementioned requirements.

We have chosen CORBA, an object-oriented middleware protocols, for DMI development. By using CORBA gives us several benefits in the system distributed computing environment. For example, we are able to interface legacy database by developing CORBA wrapper that allows us to access the data structures in the database without disturbing the existing database. Interoperability and scalability are other benefits of using CORBA. Actually, in all cases a CORBA interface service maps a subset of a medical system component onto a set of object classes which have been defined, using CORBA IDL to represent the “customer interface” of that component. Each interface service incorporates an ORB.

Java language provides a simpler and newer way to develop, manage, and deploy Client/Server applications. It is simple, object-oriented, distributed, robust, secure, architecture-neutral, portable, multithreaded, and dynamic. Java offers tremendous flexibility for distributed application development, but it currently does not support a Client/Server paradigm. CORBA provides the missing link between the Java portable applications environment and the world of intergalactic back-end services [61]. CORBA lets Java objects communicate with any other objects. Java and CORBA are very complementary. The intersection of Java and CORBA object technologies are the natural next step in the evolution of the DMI architecture.

The project plans to integrate Java-based DMI application over CORBA middleware over health institutions networks to provide a research and development environment in real practice.

A. 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 bytecode 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 a knowledge of the CORBA architecture. CORBA should be part of a 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.

B. Java Media Framework (JMF)

JMF is a framework for handling streaming media in Java programs. JMF is an optional package of Java 2 standard platform. JMF provides a unified architecture and messaging protocol for managing the acquisition, processing and delivery of time-based media. JMF enables Java programs to:

(i) Present (playback) multimedia contents,
(ii) Capture audio through microphone and video through Camera,
(iii) Do real-time streaming of media over the Internet,
(iv) Process media
(v) Store media into a file.

JMF supports many popular media formats such as JPEG, MPEG-1, MPEG-2, QuickTime, AVI, WAV, MP3, GSM, G723, H263, and MIDI. JMF supports popular media access protocols such as file, HTTP, HTTPS, FTP, RTP, and RTSP. JMF uses a well-defined event reporting mechanism that follows the “Observer” design pattern. JMF uses the “Factory”
design pattern that simplifies the creation of JMF objects. The JMF support the reception and transmission of media streams using Real-time Transport Protocol (RTP) and JMF supports management of RTP sessions. JMF scales across different media data types, protocols and delivery mechanisms. JMF provides a plug-in architecture that allows JMF to be customized and extended. Technology providers can extend JMF to support additional media formats. High performance custom implementation of media players, or codecs possibly using hardware accelerators can be defined and integrated with the JMF.

JMF adopts the same model that is used by the consumer electronics industry in handling the media. According to the JMF model, the life cycle of the media starts from a media source, and ends in a media sink. In between the media is handled by media handlers. The media source can be a [3]

- a capture device, or
- a media file stored locally or remotely on the network database or
- a real-time media stream available on the network.

The media handlers process the media which may involve demultiplexing or multiplexing or encoding or decoding. The media processing can be implemented partly in hardware but mostly it is done by software. The media sink or destination can be rendering devices, or storage files or media streams

C. JAVA and 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 complements 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 version 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 Java™ standard for sharing medical imaging resources between heterogeneous and multi-vendor equipments (acquisition device, workstation, storage server, patient management system, etc.).

IV. DESIGN METHODOLOGY

The Application model have been designed and developed in the distributed computing environment based on corba-java based middleware and JDBC database server connectivity based on the architecture as depict in figure 1.

Fig 1. System architecture

The service object and components involved in the design are,

1. Graphical User Interface (GUI) Class
2. Textual Object Class
3. Video Object Class
4. Audio Object Class
5. Multimedia I/O Stream and Buffer Object Classes
6. DBAS (Database Archive System) Component
7. Multimedia storage Component
8. DICOM Server Component

The desire to ensure the reliability, robustness, interoperability, and maintainability of the DMI code...
has led out team to the implementations of all part (objects) of the DMI software in Java [141]. Java based DMI has been to use the existing Web for communication and remote consultations for medical documents.

- **GUI Object Class**

The Graphical User Interface (GUI) class represents the tools used for human-to-computer interaction. The GUI may include menus, buttons, graphics, textual and visual information, image annotation, full-duplex audio, and video information. The GUI class is responsible for collecting relevant information to be passed onto the user. The GUI class also distributes user input to objects whose states depend on the user. The objects in the class are linked with other objects in the system, e.g., management and control objects.

- **Audio Object Class**

The audio object class handles real-time and playback of audio sequences at the user workstation. Digital audio is encoded to sequences of 8-bit or 16-bit samples using standard telephone Coder/Decoder (CODEC) conversion. The audio sequences are two-way conversations between physicians or audio notes to be stored with a patient record.

- **Video Object Class**

The video object class handles real-time and playback video sequences. The video sequences are digital format using MPEG compression frames. Video sequences can be generated from digital camera or pre-recorded video on CD disk formats. Video sequences can be stored as video object classes in the multimedia database archive system.

- **Buffer Object Class**

The buffer object class controls jitter by queuing data until it is synchronized and ready for use. Buffering takes place on all data traversing the network stack to abstract the application from network timing idiosyncrasies and dependencies.

- **File Stream Object Class**

The file stream objects include files, e.g., images, which must be transferred between nodes over networks. These nodes can be workstations or the Database Archive System. The objects represent a flow of packets, and the objects inherit the format of their types, i.e., audio, video, annotation commands, etc.

- **Communication Object Class**

The communication object class represents a group of connectivity mechanism. For example, objects conduct communication via middleware APIs or TCP/IP sockets. These objects provide the communication paradigm to the data objects described above, i.e., annotation commands, file streams, etc., and link the multimedia information exchanging over the distributed computing environment.

- **Components**

The components in the DMI systems are either Client or Service Provider(server), which uses objects and classes. The Service Providers, e.g., image equipment, DBAS, are legacy objects in the CORBA framework. A application, module, or entry point can be encapsulated in a “wrapper” that defines an interface to the legacy code. Creating an object wrapper gives the legacy or existing code a CORBA-compliant interface, making it interoperable with other objects in a distributed computing environment. In addition, software objects that comply with the CORBA standard are portable. Thus, objects built on one platform can be deployed on any other supported platforms.

**A. Rework CORBA Framework**

The multimedia, audio and video stream system frameworks are developed based on figure 2 above. This framework established to Provide multimedia exchange between server(service provider) and client. This framework contains audio and video devices as depict above. A user can start or stop (control) the services to exchange either text, video or audio streams over LAN and WAN.

In the case of using ORB as transfer the multimedia data transfer layer, the benchmark incurs additional performance overhead. This overhead arises from excessive data-copying and memory allocation per-request, which increases packet latency [6] and inefficient marshaling-demarshaling in distributed object middleware decreases streaming data throughput [7].

Thus, the frameworks use the corba middleware to
locate and manage multimedia devices, the audio and video streaming data are exchange through UDP/Ip protocol.

This called hybrid model A model that use pair of application protocol comprise of CORBA and UDP for Streaming Service establishment protocol. In this model Java Media Framework (JMF) is the programming API for displaying multimedia file and TCP for data transfer protocol.

In general, this model performed better then traditional CORBA(multimedia over ORB) model due to various optimization, such as minimizing process overhead for streaming multimedia data. Thus, Hybrid Model is the ideal model since there is no additional ORB-related or presentation layer overhead for streaming multimedia data.

V. MULTIMEDIA STREAM THROUGHPUT COMPARISON

The aim of this experiment is to illustrate that hybrid model Streaming service does not introduce appreciable overhead in transporting data. To demonstrate this, a hybrid model and traditional CORBA model throughput have been tested. To measure the throughput, the number of bytes per second sent by the server to the client has been calculated. Table 1 and Figure 3 show the table for throughput and it graph respectively.

<table>
<thead>
<tr>
<th>Bi</th>
<th>File Name (.mpg)</th>
<th>Size (MB)</th>
<th>Traditional CORBA Model</th>
<th>Hybrid Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Splash</td>
<td>0.66</td>
<td>0.65</td>
<td>0.16</td>
</tr>
<tr>
<td>2</td>
<td>Popcorn</td>
<td>4.24</td>
<td>0.73</td>
<td>2.05</td>
</tr>
<tr>
<td>3</td>
<td>Bean12</td>
<td>12.3</td>
<td>0.98</td>
<td>2.10</td>
</tr>
<tr>
<td>4</td>
<td>Bean26</td>
<td>26.1</td>
<td>1.13</td>
<td>6.42</td>
</tr>
<tr>
<td>5</td>
<td>Bean43</td>
<td>42.9</td>
<td>1.34</td>
<td>4.86</td>
</tr>
<tr>
<td>6</td>
<td>Bean56</td>
<td>56.0</td>
<td>1.30</td>
<td>17.38</td>
</tr>
<tr>
<td>7</td>
<td>Bean68</td>
<td>68.3</td>
<td>1.37</td>
<td>25.73</td>
</tr>
</tbody>
</table>

The results depicted in Table 2 and Figure 3 indicate that as expected the hybrid model does not introduce any appreciable overhead to streaming multimedia data.

VI. IMPLEMENTATION

In this application, client patient’s detail screen and medical image in DICOM format [Sadleir et al 2002] screen are resided in separated windows. For the query purposes, Next, the system pop-up the medical image as shown figure 4. The, The implementation scenario of the architecture as discussed in design methodology Figure 1, is illustrated as follows:

i. The client request patient data by providing patient case retrieving procedures such as patient id and notifies the server(services Perovider) via corba event service .(figure 4)

Fig 4. Client Request

ii. The Server retrieve a patient demographic data and image(s) from client and send the DBA via JDBC Server using JDBC connectivity to the DBMS, e.g., IBM DB2 database. As in figure 5

Fig 5. Server response
iii. The Server passes the patient demographic data and image file identification (where those files reside in remote storage) to the client via ORB.

iv. The remote Client fetches the demographic data and display then details on client GUI screen as depict in figure 6. Then it point the patient image(s) based on the given file identification. Then, the system pop-up the medical image as illustrated figure 7 below

![Fig 6. Screenshot of patient’s data](image1)

![Fig 7. Patient’s image in DICOM Format](image2)

VII. CONCLUSION

This research illustrates an approach to building standards-based, flexible, adaptive, multimedia streaming applications using CORBA. The models implementation indicates that the standard CORBA specification using ORB and JMF defines a flexible and efficient model for developing flexible and high-performance distributed object for multimedia streaming applications.

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 potential success and utility of the framework lies mainly with how well it can be adapted to changing circumstances and meet the requirements of new medical, multimedia and distributed technology.

The key question here is whether the design captures essential abstractions well enough to provide reusability in future scenarios. As the intention was to create standards for distributed object-oriented medical informatics. More than on implementing a production level system, the focus was placed on assessing the feasibility of the task and identifying critical design and implementation issues.

REFERENCES


