INFORMATION TO USERS

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

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

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

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

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

Bell & Howell Information and Learning
300 North Zeib Road, Ann Arbor, MI 48106-1346 USA
800-521-0800

UMI®
A CORBA-BASED IMPLEMENTATION OF
VIRTUAL RADIOLOGY ENVIRONMENT (VRE) PROTOTYPE

Ferdinand Marinus Pardede

A Dissertation Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY
In the Graduate College
THE UNIVERSITY OF ARIZONA

1999
As members of the Final Examination Committee, we certify that we have read the dissertation prepared by FERDINAND MARINUS PARDEDE entitled A CORBA-BASED PROTOTYPE IMPLEMENTATION OF VIRTUAL RADIOLOGY ENVIRONMENT (VRE) and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of DOCTOR OF PHILOSOPHY.

Ralph Martinez, Ph.D.

JERZY W. ROZENBLIT, Ph.D.

JAY-COOK, M.D.

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director RALPH MARTINEZ, Ph.D.
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: [Signature]
ACKNOWLEDGEMENT

Working on this dissertation has been both challenging and rewarding for me. The struggles from within and without constantly came to my life as I tried to focus on finishing this project. However, I believe that these struggles are the means of molding my personality, forming my characters, and enabling me to see others and myself from a proper perspective.

I would like to thank Dr. Ralph Martinez, my research advisor, who has been patiently advising me about the project and provided me with financial assistantship without which I would never finish my Ph.D. program. Your encouragement and directions are deeply appreciated. In addition, I owe much help from the University of Arizona's Meta-Manager Team which includes Dr. Jerzy Rozenblit and Dr. Jay Cook, and Col. Anna K. Chacko MD from BAMC for providing this project environment.

I would also like to thank my friends from the Tucson chapter of Indonesian Christian Fellowship (ICF Tucson), both those who've gone before me and those who are still with me. Our Friday night meetings as well as others are the source of my inspiration of my constant curiosity about people and life in general. I appreciate and cherish the times we spent together and the memories we had in serving people around us and, especially, the LORD our God.
To my father and my mother.
# TABLE OF CONTENTS

LIST OF FIGURES................................................................................................................. 9  
LIST OF TABLES.................................................................................................................. 12  
LIST OF ABBREVIATIONS.................................................................................................. 13  
ABSTRACT.............................................................................................................................. 17  

## CHAPTER 1: INTRODUCTION .............................................................................................. 18  
1.1 BACKGROUND .................................................................................................. 18  
1.2 STATEMENT OF THE PROBLEM ........................................................................... 19  
1.3 OBJECTIVE ........................................................................................................ 22  
1.4 APPROACH............................................................................................................ 24  
1.5 DESIGN METHODOLOGY AND TOOLS .............................................................. 25  

## CHAPTER 2: VIRTUAL RADIOLOGY ENVIRONMENT (VRE) ...................................................... 28  
2.1 THE LEGACY OF DIN-PACS ................................................................................ 28  
2.1.1 Acquisition Pathway Subsystem ........................................................................ 29  
2.1.2 Display Subsystem ........................................................................................... 30  
2.1.3 Database and Archive Subsystem ....................................................................... 31  
2.1.4 Communication and Control Subsystem ............................................................ 32  
2.2 GLOBAL PACS .................................................................................................... 33  
2.3 DIGITAL IMAGING AND COMMUNICATIONS IN MEDICINE (DICOM) .............. 36  
2.3.1 DICOM Information Model ............................................................................. 37  
2.4 DEFINITION AND BACKGROUND ..................................................................... 41  
2.5 META-MANAGER ................................................................................................. 42  
2.6 VRE HIERARCHY .................................................................................................. 44  

## CHAPTER 3: COMMON OBJECT REQUEST BROKER ARCHITECTURE ................................. 46  
3.1 OVERVIEW OF DISTRIBUTED OBJECT & COMPONENT PARADIGM .................... 46  
3.2 INTRODUCTION TO CORBA 3 .............................................................................. 50  
3.2.1 Internet Integration .......................................................................................... 52  
3.2.1.1 Firewall Specification .................................................................................. 52  
3.2.1.2 Interoperable Name Service ....................................................................... 53  
3.2.2 Quality of Service (QoS) Control ..................................................................... 54  
3.2.2.1 Asynchronous Messaging and Quality of Service Control ....................... 54  
3.2.2.2 Minimum, Fault-Tolerant, and Real-Time CORBA .................................. 54  
3.2.3 CORBACOMPONENTS Package .................................................................... 55  
3.2.3.1 CORBACOMPONENTS and CORBASCRIPTING .................................. 55  
3.3 COMPARISONS OF CORBA, DCOM AND JAVA/RMI ........................................... 56  
3.3.1 Application Sample - The StockMarket Server and Client ............................... 58  
3.3.2 The IDL Interface ............................................................................................ 59  
3.3.3 Conclusion ....................................................................................................... 62  
3.4 VISIBROKER FOR JAVA 3.3 ................................................................................... 64
LIST OF FIGURES

Figure 1-1: Proposed VRE Sites ........................................................................... 21
Figure 1-2: Cable Bundling Initiative (CBI) Topology ........................................... 23
Figure 2-1: DIN-PACS Subsystem Overview ....................................................... 29
Figure 2-2: IBM Implementation of DIN-PACS .................................................... 30
Figure 2-3: DIN-PACS Model ............................................................................ 33
Figure 2-4: Global PACS Architecture ................................................................. 35
Figure 2-5: DICOM Protocol Stack ..................................................................... 38
Figure 2-6: A DICOM Entity-Relation Diagram .................................................. 40
Figure 2-7: VRE Hybrid Hierarchy ..................................................................... 45
Figure 3-1: The Structure of Object Request Broker Interfaces ............................. 51
Figure 3-2: Inprise VisiBroker Family .................................................................. 65
Figure 4-1: Meta-Manager 3-level Hierarchy ....................................................... 67
Figure 4-2: Relationships between VRE nodes .................................................... 71
Figure 4-3: Meta-Manager Specialization ............................................................ 81
Figure 4-4: VRE Object Model ........................................................................... 83
Figure 4-5: Pruning Stage 1 ................................................................................. 84
Figure 4-6: Pruning Stage 2 ................................................................................. 85
Figure 4-7: A more complete view of final instantiation ....................................... 86
Figure 4-8: Meta-Manager Entity Initialization State Diagram ............................ 87
Figure 4-9: Case Routing Algorithm in DIN-PACS .............................................. 96
Figure 4-10: Case Routing Algorithm in Client Meta-Manager ............................. 98
Figure 4-11: Case Routing Algorithm in Regional Meta-Manager ....................... 99
Figure 4-12: Case Routing Algorithm in Master Meta-Manager ........................................ 100
Figure 4-13: Process New Case Sequence Diagram – Local Read Scenario ..................... 102
Figure 4-14: Process New Case Sequence Diagram – Remote Read Scenario ................. 103
Figure 4-15: Process New Case Sequence Diagram – Slow Case Recall Scenario .......... 105
Figure 4-16: Process New Case Sequence Diagram – Lost Client Recall Scenario .......... 106
Figure 4-17: Monitor State Sequence Diagram – Master Ping Scenario ....................... 107
Figure 4-18: Monitor State Sequence Diagram – Client Update Scenario ...................... 108
Figure 4-19: Monitor State Sequence Diagram – Lost Client Scenario ......................... 110
Figure 4-20: Monitor State Sequence Diagram – Lost Master Scenario ....................... 111
Figure 4-21: UML package abstractions with CORBA IDL packages ............................ 112
Figure 4-22: UML package abstractions of specialized Meta-Manager Node packages ....... 113
Figure 5-1: VisiBrokerProduct Family ........................................................................... 123
Figure 5-2: Example of a multiple, nested, and federated naming context ...................... 127
Figure 5-3: Event Channel Architecture ........................................................................... 129
Figure 5-4: Java Platform Architecture .......................................................................... 132
Figure 5-5: Java Development and Runtime Environment ............................................. 133
Figure 5-6: Java Compilation and Interpretation .............................................................. 135
Figure 5-7: Java Bytecode: write once, run anywhere ..................................................... 136
Figure 5-8: Java Platform Layers .................................................................................... 137
Figure 5-9: JDBC Architecture ....................................................................................... 138
Figure 5-10: JDBC-ODBC Bridge Architecture ............................................................... 139
Figure 5-11: VRE IDL and Component Hierarchy .......................................................... 143
Figure 5-12: VRE Components and the ORB bus ......................................................... 144
Figure 5-13: DIN-PACS Class Diagram ......................................................................... 147
<table>
<thead>
<tr>
<th>Figure 5-14: Client Meta-Manager Class Diagram</th>
<th>148</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 5-15: Testing Environment Hierarchy</td>
<td>154</td>
</tr>
<tr>
<td>Figure 5-16: Testing Sites of VRE Prototype</td>
<td>155</td>
</tr>
<tr>
<td>Figure 5-17: Network Architecture of Local Area Testing Environment</td>
<td>156</td>
</tr>
<tr>
<td>Figure 5-18: Wide Area Testing Environment</td>
<td>157</td>
</tr>
<tr>
<td>Figure 5-19: VRE Component Icons</td>
<td>158</td>
</tr>
<tr>
<td>Figure 5-20: VRE DIN PACS window</td>
<td>159</td>
</tr>
<tr>
<td>Figure 5-21: VRE DIN PACS Simulate Step Dialog</td>
<td>160</td>
</tr>
<tr>
<td>Figure 5-22: VRE Client Meta-Manager Window</td>
<td>161</td>
</tr>
<tr>
<td>Figure 5-23: VRE Regional Meta-Manager Window</td>
<td>162</td>
</tr>
<tr>
<td>Figure 5-24: VRE Master Meta-Manager Window</td>
<td>163</td>
</tr>
<tr>
<td>Figure 6-1: An Example of G2 Environment Capability</td>
<td>168</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 3.1: Comparison of IDL from Various Architecture ............................................. 61
Table 3.2: Comparison of Features from Various Distributed Object Architecture .......... 62
Table 5.1: Bandwidth Requirements of Telemedicine Services ..................................... 116
Table 5.2: Modality Sizes of Telemedicine Services .................................................. 116
Table 5.3: Some of 1997 RSNA DICOM Files Used .................................................. 117
Table 5.4: CR Image IOD Modules ................................................................. 118
Table 5.5: Patient Identification Module Attributes ............................................... 118
Table 5.6: Patient Relationship Module Attributes ............................................... 119
Table 5.7: Patient Medical Module Attributes ....................................................... 119
Table 5.8: Patient Demographic Module Attributes ............................................... 120
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAL</td>
<td>ATM Adaptation Layer</td>
</tr>
<tr>
<td>ACK</td>
<td>ACKnowledgment</td>
</tr>
<tr>
<td>ACR</td>
<td>American College of Radiology</td>
</tr>
<tr>
<td>AE</td>
<td>Application Entity</td>
</tr>
<tr>
<td>AMC</td>
<td>Army Medical Center</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>ANSOC</td>
<td>Army Network Systems Operation Center</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ARTN</td>
<td>Arizona Rural Telemedicine Network</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>BAMC</td>
<td>Brooke Army Medical Center</td>
</tr>
<tr>
<td>BOA</td>
<td>Basic Object Adapter</td>
</tr>
<tr>
<td>BP</td>
<td>Blood Pressure</td>
</tr>
<tr>
<td>BRI</td>
<td>Basic Rate Interface</td>
</tr>
<tr>
<td>CATV</td>
<td>Cable TV</td>
</tr>
<tr>
<td>CBI</td>
<td>Cable Bundling Initiative</td>
</tr>
<tr>
<td>CDDI</td>
<td>Copper Distributed Data Interface</td>
</tr>
<tr>
<td>CECOM</td>
<td>Communications and Electronics Command</td>
</tr>
<tr>
<td>CEN</td>
<td>Comite Europeen de Normalisation</td>
</tr>
<tr>
<td>CERL</td>
<td>Computer Engineering Research Laboratory</td>
</tr>
<tr>
<td>CHCS</td>
<td>Composite Health Care System</td>
</tr>
<tr>
<td>CLSID</td>
<td>CLasS ID</td>
</tr>
<tr>
<td>CMM</td>
<td>Client Meta-Manager</td>
</tr>
<tr>
<td>CODEC</td>
<td>Coder / Decoder</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off-the-Shelf</td>
</tr>
<tr>
<td>CR</td>
<td>Computed Radiography</td>
</tr>
<tr>
<td>CT</td>
<td>Computed Tomography</td>
</tr>
<tr>
<td>DBAS</td>
<td>Database Archive System</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DCE</td>
<td>Distributed Computing Environment</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>DF</td>
<td>Digital Fluoroscopy</td>
</tr>
<tr>
<td>DICOM</td>
<td>Digital Imaging and Communications in Medicine</td>
</tr>
<tr>
<td>DII</td>
<td>Dynamic Interface Invocation</td>
</tr>
<tr>
<td>DII-COE</td>
<td>Defense Information Infrastructure - Common Operating Environment</td>
</tr>
<tr>
<td>DIMSE</td>
<td>DICOM Message Service Element</td>
</tr>
<tr>
<td>DIMSE-C</td>
<td>DICOM Message Service Element-Composite</td>
</tr>
<tr>
<td>DIMSE-N</td>
<td>DICOM Message Service Element-Normalized</td>
</tr>
<tr>
<td>DIN</td>
<td>Digital Imaging Network</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>DISA</td>
<td>Defense Information System Agency</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>DOS</td>
<td>Disk Operating System</td>
</tr>
<tr>
<td>DSA</td>
<td>Digital Subtraction Angiography</td>
</tr>
<tr>
<td>DSI</td>
<td>Dynamic Skeleton Interface</td>
</tr>
<tr>
<td>ECE</td>
<td>Electrical and Computer Engineering</td>
</tr>
<tr>
<td>ECG</td>
<td>Electrocardiogram</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBean</td>
</tr>
<tr>
<td>E-R</td>
<td>Entity-Relationship</td>
</tr>
<tr>
<td>FDDI</td>
<td>Fiber Distributed Data Interface</td>
</tr>
<tr>
<td>GFE</td>
<td>Government Furnished Equipment</td>
</tr>
<tr>
<td>GIOP</td>
<td>General Inter-ORB Protocol</td>
</tr>
<tr>
<td>GPACS</td>
<td>Global Picture Archiving and Communication System</td>
</tr>
<tr>
<td>GPRMC</td>
<td>Great Plain Regional Medical Command</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HIS</td>
<td>Hospital Information System</td>
</tr>
<tr>
<td>HISPP</td>
<td>Healthcare Information Standards Planning Panel</td>
</tr>
<tr>
<td>HL7</td>
<td>Health Level 7</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
<tr>
<td>IE</td>
<td>Imaging Equipment, Information Entity</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IID</td>
<td>Interface ID</td>
</tr>
<tr>
<td>IIOP</td>
<td>Internet Inter-ORB Protocol</td>
</tr>
<tr>
<td>IOD</td>
<td>Information Object Definition</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MIDL</td>
<td>Microsoft Interface Definition Language</td>
</tr>
<tr>
<td>MIU</td>
<td>Modality Interface Unit</td>
</tr>
<tr>
<td>MM</td>
<td>Meta-Manager</td>
</tr>
<tr>
<td>MMM</td>
<td>Master Meta-Manager</td>
</tr>
<tr>
<td>Mod</td>
<td>Modality</td>
</tr>
<tr>
<td>MPEG</td>
<td>Motion Picture Expert Group</td>
</tr>
<tr>
<td>MRI</td>
<td>Magnetic Resonance Imaging</td>
</tr>
<tr>
<td>MSDS</td>
<td>Healthcare Message Standard Developers Sub-Committee</td>
</tr>
<tr>
<td>MTF</td>
<td>Medical Treatment Facility</td>
</tr>
<tr>
<td>NEMA</td>
<td>National Electrical Manufacturers Association</td>
</tr>
<tr>
<td>NIH</td>
<td>National Institute of Health</td>
</tr>
<tr>
<td>NIPR</td>
<td>Non-IP Router</td>
</tr>
<tr>
<td>NM</td>
<td>Nuclear Medicine</td>
</tr>
<tr>
<td>NT</td>
<td>New Technology</td>
</tr>
<tr>
<td>OAD</td>
<td>Object Activation Daemon</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>OEM</td>
<td>Original Equipment Manufacturer</td>
</tr>
<tr>
<td>OLMMS</td>
<td>On-Line Medical Media Service</td>
</tr>
<tr>
<td>OMA</td>
<td>Object Management Architecture</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OMT</td>
<td>Object Modeling Technique</td>
</tr>
<tr>
<td>ORB</td>
<td>Object Request Broker</td>
</tr>
<tr>
<td>OSF</td>
<td>Open System Foundation</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>PACS</td>
<td>Picture Archiving and Communication System</td>
</tr>
<tr>
<td>POA</td>
<td>Portable Object Adapter</td>
</tr>
<tr>
<td>POTS</td>
<td>Plain Old Telephone System</td>
</tr>
<tr>
<td>QC</td>
<td>Quality Control</td>
</tr>
<tr>
<td>QtS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RCD</td>
<td>Remote Consultation and Diagnosis</td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments</td>
</tr>
<tr>
<td>RFP</td>
<td>Request for Proposal</td>
</tr>
<tr>
<td>RIS</td>
<td>Radiology Information System</td>
</tr>
<tr>
<td>RMC</td>
<td>Regional Medical Command</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RMM</td>
<td>Regional Meta-Manager</td>
</tr>
<tr>
<td>ROP</td>
<td>Rational Objectory Process</td>
</tr>
<tr>
<td>RSNA</td>
<td>Radiological Society of North America</td>
</tr>
<tr>
<td>SCP</td>
<td>Service Class Provider</td>
</tr>
<tr>
<td>SCU</td>
<td>Service Class User</td>
</tr>
<tr>
<td>SIPR</td>
<td>Secured IP Router</td>
</tr>
<tr>
<td>SMTPE</td>
<td>Society of Motion Picture and Test Engineers</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol</td>
</tr>
<tr>
<td>SONET</td>
<td>Synchronous Optical Network</td>
</tr>
<tr>
<td>SOP</td>
<td>Service-Object Pair</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>SUD</td>
<td>System Under Design</td>
</tr>
<tr>
<td>TARA</td>
<td>Technology Assessment and Requirement Analysis</td>
</tr>
</tbody>
</table>
TC 251  Technical Committee 251
TCP    Transmission Control Protocol
TIC    Technology Information Center
UAMC   University of Arizona Medical Center
UDP    User Datagram Protocol
UID    Unique Identifier
UML    Universal Modeling Language
UPS    Uninterruptible Power Supply
URL    Uniform Resource Locator
US     Ultrasound
USASC  United States Army Signal Command
USAVRE United States Army Virtual Radiology Environment
UUID   Universally Unique Identifier
VNR    Virtual Network Routing
VOI    Value of Interest
VRE    Virtual Radiology Environment
WAN    Wide Area Network
WG     Working Group
WS     Workstation
WWW    World Wide Web
XML    eXtensible Markup Language
Digital Imaging Network - Picture Archiving and Communication System (DIN-PACS) has been the cutting edge technology in telemedicine and teleradiology world. It enables sharing and processing of patient data and images in a networked environment. This sharing implies a distributed environment where doctors and other health practitioners do not need to be physically present in the data sources in order to access them. This advantage in itself is a very economical alternative in time and money. Furthermore, digital processing enables doctors and health practitioners to acquire, examine, and even digitally manipulate data (text, images, sound, and videos). A reduced rate of image loss probability and inexpensive and fast image storage further provide attractions to digital imaging network.

Until recent years, however, DIN-PACS technologies are implemented in a localized environment only, e.g., within a single hospital or clinic. The need to share and process these data in a wider area environment (e.g., inter-hospital, city-wide, state-wide, nation-wide, or even world-wide environment), especially in teleradiology discipline, prompted the concept Virtual Radiology Environment (VRE). Virtual Radiology Environment enables the networking of multiple DIN-PACS, which requires an integrated network design configured for both clinicians and radiologist needs. This dissertation presents the design specification and a CORBA-based prototype implementation of such environment.
CHAPTER 1: INTRODUCTION

This chapter provides the background of the project, the statement of the problem, the objectives, and the approaches used in the design and implementation of the Virtual Radiology Environment (VRE).

1.1 BACKGROUND

Current computing environment for medical imaging research includes a variety of software and hardware platforms. The operating systems vary from single-processor, single-tasking such as Microsoft Disk Operating System (DOS) to multi-processor, multi-tasking such as UNIX-based Solaris or OSF1. Furthermore, the underlying hardware in each machine also vary from Intel-based x86 machines to more complicated systems with multiple processing capability. These computer systems execute various analysis and image processing software package in support of clinical, biological and radiological tasks. It is clear that there is an interoperability problem between the existing systems. Image formats, database management systems, image processing software, and image acquisition systems currently have different standards, making it impossible to exchange database and images among the systems. The existing clinical, biological, and radiological systems do not interface with any electronic document management, document conversion, and electronic storage systems.

Earlier attempts, such as Medical Diagnosis Imaging Support System (MDIS) have only solved the local hospital problems. MDIS experiments have used point-to-point T1 connection in Korean telemedicine demonstration [30].

The successor to MDIS is the Picture Archiving and Communication System (PACS) network. PACS would normally reside in a hospital or laboratory setting with several medical
imaging acquisition systems. In radiology, the minimum sets of acquisition modalities are X-ray, Computed Tomography (CT) scan, Magnetic Resonance Imaging (MRI), Computed Radiography (CR), and Ultrasound systems. These represent static image modalities (i.e., X-ray, CT, MRI) or dynamic sequences (i.e., Ultrasound). PACS system procurement solves some of the problems with MDIS.

While these efforts have demonstrated contributions to the telemedicine technology, they are nevertheless only extensions of demonstration performed in the 1970's and 1980's. Little emphasis has been placed on the use of emerging technologies, such as Asynchronous Transfer Mode (ATM) network, Distributed Computing Environment (DCE), Java Programming language, as well as the ever more complicated Internet browsers.

Virtual Radiology Environment (VRE) is a new approach to alleviate the interoperability problems that present themselves within many Radiology departments today. It uses an Open Architecture standard-based approach for the research and development of medical imaging and archiving systems.

1.2 STATEMENT OF THE PROBLEM

Rapid advances in telecommunications and multimedia network technology and the requirements to improve health care in the Department of Defense (DoD) provides an opportunity for the U.S. Communications and Electronics Command (CECOM) and Medical Command (MEDCOM) to work towards a synergistic goal. CECOM and MEDCOM have been leaders in the development and deployment of new technology into DoD systems. The scope of this project covers the technology merger of high bandwidth telecommunications networks and tele-healthcare in the U.S. Army.
The U.S. Army Information Systems Engineering Command (ISEC) and the U.S. Army Medical Command have agreed in principle to develop a program initiative to develop a U.S. Army Virtual Radiology Environment (VRE). A high-speed backbone network will support the VRE across Continental U.S. and Hawaii. The VRE Intranet will interconnect major U.S. Army medical centers to medical treatment facilities (MTFs) located in Army Regional Medical Command (RMC) bases. The VRE will provide teleradiology services to address the issues of cost, improved quality, and medical access in the U.S. Army. This network will serve as a model for other DoD and industry developments for tele-healthcare Intranets. This project is proposed as a seed effort to start the development of a multimedia collaborative framework for teleradiology and distance learning systems. It is the goal of this project to demonstrate the feasibility of the VRE concepts and covers installation of testbeds and research and development activities in protocol frameworks for multimedia collaborative systems.

The VRE project is envisioned for two phases: the first phase is a Teleradiology Pilot Testbed consisting of five sites with the Brooke Army Medical Center (BAMC) at San Antonio, TX, as the major center for reading and diagnosis of radiology modality cases from the other four MTF sites. The four remote MTF sites are to be Ft. Huachuca, AZ, Ft. Hood, TX, Ft. Sill, OK, and Ft. Leavenworth, KS.

The Pilot Testbed will have a star topology representing a teleradiology environment. Cases will be acquired at the MTF sites and stored and forwarded to BAMC for diagnostic or consultative readings. The case reports will be returned to the originating MTF site. The Pilot Testbed will serve to evaluate new PACS (Picture Archiving and Communication System) vendor products and collect statistics that can be used for the long-term design and implementation of the VRE. Industry will provide the PACS networks and image databases at the four MTF sites. The cases will be read at BAMC on the MDIS (Meta-Data Interchange Specification) PACS.
workstations by BAMC radiologists. The Technology Integration Center (TIC) in ISEC will be part of the Pilot Testbed team and will provide the guidance for the telecommunications or ATM networks between the 5 sites. Research and development tasks are required to demonstrate the feasibility of both the Pilot Testbed and the long term VRE project.

The second phase of the VRE Project involves a larger domain of hospitals and Medical Treatment Facilities within the Army, as well as integration of many vendor-specific implementation of DIN-PACS.

The Computer Engineering Research Laboratory (CERL) at the University of Arizona is uniquely positioned to perform research and development in collaborative systems and multimedia protocols for the Pilot Testbed and the VRE. The CERL proposes to perform the research tasks with undergraduate and graduate students that are trained in state-of-the-art tele-healthcare systems, multimedia protocols, and telecommunications systems. The CERL has research relationships with the Arizona Health Sciences Center (AHSC), Radiology Department,
Biomedical Engineering Program, Virtual Universities, government agencies, distance learning systems, and teleradiology users. This experience base gives the CERL a state-of-the-art look at tele-healthcare and distance learning user requirements.

1.3 OBJECTIVE

In this project, the CERL will develop protocol components that can be integrated into new multimedia collaborative system prototypes. Specifically, the CERL has developed a Global Picture Archiving and Communications System (GPACS) prototype based on the Java programming language. Although, not an actual PACS for diagnostic readings, the remote consultation and diagnosis (RCD) system represents a multimedia collaborative system. In the proposed project, the Global PACS prototypes were used to demonstrate high speed multimedia tele-healthcare services over portions of the current ISEC intranet called the Army Circuit Bundling Initiative Network (CBI-Net). The Global PACS teleradiology services will be demonstrated in a testbed at the Technology Integration Center (TIC) at Ft. Huachuca, AZ, and the CERL.

The CERL has developed protocol middleware for distributed computing environments (DCE) based on the Common Object Request Broker Architecture (CORBA) standards developed by the Object Management Group. The goal of the CERL research is to develop a protocol framework for the development of multimedia collaborative systems, with applications to tele-healthcare and distance learning systems. The CORBA-based framework allows the implementation of object oriented applications over a heterogeneous internetworking environment. Since the TIC is deploying ATM technology to Army bases and developing the CBI-Net wide area network, we propose to use the CORBA middleware to support multimedia applications over high speed Army intranetworks. The CORBA components include an Object
Request Broker (ORB) that is used to discover object resources in a intranet and associate them with a client. The TIC is interested in multimedia object services over the CBI-Net, such as a video stream for a teleradiology ultrasound session or a synchronized stream of audio and image annotation objects.

![Figure 1-2: Cable Bundling Initiative (CBI) Topology](image)

The overall goal of this research project is to develop the next generation framework systems for tele-healthcare and distance learning systems in ATM-based environments. This is done by providing and developing a CORBA-based testbed of the Virtual Radiology Environment to test multimedia tele-healthcare protocols and services. The testbed will be implemented over high bandwidth ATM local and wide area networks between the Computer Engineering Research Laboratory (CERL) at the University of Arizona and the Technology Integration Center (TIC) at Ft. Huachuca, AZ.
1.4 APPROACH

There are three main subsystems of this project that allow the flexibility, interoperability and ease of the development. First, the Common Object Request Broker Architecture (CORBA) of the Object Management Group (OMG) sets forth the open system, cross platform standard which best meets the Virtual Radiology Environment 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 inter-operates 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. Interoperability and scalability are other benefits of using CORBA. Actually, in all cases a CORBA interface service maps a subset of a the Virtual Radiology Environment 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.

Second, 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 service [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 Virtual Radiology Environment
architecture. The project plans to integrate Java-based RCD software over CORBA middleware
over ATM networks to provide a research and development environment.

Finally, the Windows NT Workstation will serve as the platform of the design and
implementation of CORBA-based Virtual Radiology Environment. Windows NT Workstation
has proved itself to be a stable and reliable workstation suitable for Virtual Radiology
Environment. It claims to have the following benefits:

• Highest reliability of any member of the Windows desktop family
• Security for desktop workstation from unauthorized use and access to data
• Offers key mobile features for notebooks and laptops, including power management, PC
  Card support, and hot-docking solutions via third-party add-ons
• Broad software and hardware compatibility, running the leading business applications and
tested on over 9,000 PC brands and 6,500 PC peripherals and devices
• Excellent multi-purpose network client for Windows NT Server, Novell NetWare, and now
  UNIX.

1.5 DESIGN METHODOLOGY AND TOOLS

A systems engineering methodology approach was used for the design and specification of
the Meta-Manager. As a first step, Technology Assessment and Requirements Analysis
(TARA) was performed. TARA is an analysis of the workload in VRE radiology departments,
their equipment and their staffing. A team of biomedical engineers, medical maintenance
personnel, radiologists and logisticians performs this work. From the requirement analysis
phase, computer models and simulations of the VRE operation and processes were developed.
The design and implementation phase involves developing demonstration prototypes and specifications that can be given to vendors to build the final VRE components. The University of Arizona developed the model and simulation of the VRE, the Meta-Manager Design Specification, and a prototype implementation. This document describes the Meta-Manager Design Specification.

The Object Modeling Technique (OMT) developed by Rumbaugh et. al. was used as the design specification and modeling formalism [20]. OMT and its subsequent implementation in the Unified Modeling Language (UML) and supporting software tools, provides a set of uniform description constructs from which a complete design can be specified. More specifically, the OMT/UML notation enables the specification of three types of models:

- object model,
- functional model, and
- dynamic model of the systems under design (SUD).

The notational constructs used to describe the object model follow the classical object-oriented paradigm. They are classes, class instants and class associations with all the requisite mechanisms for class variables and methods specifications. The functional models are described as data flow diagrams while the dynamic models are specified using state charts. (In this document, the functional and dynamic models are aggregated into one notion of a behavioral model).

Enterprise Edition Rose Version 4.0 system development tools from Rational Software were used to facilitate Meta-Manager modeling [43]. The Rose product includes support for modeling processes, objects, and component-based applications. It also enables visual differencing and merging between models in different stages of design to facilitate parallel development [35].
The Rose software not only facilitates the creation and modification of design models but also automatically generates a code framework that corresponds to the models. Rose supports C++ and Java among others and generates CORBA Interface Definition Language (IDL) from the model set.
CHAPTER 2: VIRTUAL RADIOLOGY ENVIRONMENT (VRE)

This chapter covers the details of Virtual Radiology Environment (VRE). First, the architecture and background of the VRE is explained. It includes the notion of Meta-Manager as the agent that implement administration and routing of cases in VRE. Second, the hierarchy of VRE and Meta-Manager will be shown and described. This chapter concludes with a pointer toward the middleware, CORBA, to implement the system. CORBA and its details are covered in the next chapter.

2.1 THE LEGACY OF DIN-PACS

The Digital Imaging Network - Picture Archiving and Communications System (DIN-PACS) is an open system network of digital devices designed for the effective acquisition, transmission, display and management of diagnostic imaging studies. This network is primarily based on two international standards: Digital Imaging and Communications in Medicine (DICOM) and Health Level 7 (HL7, version 2.4), a standard for electronic data interchange in the healthcare environment [3].

DIN-PACS is considered to contain four major functional subsystems: Acquisition Pathway, Display, Database and Archive, and Communication and Control. Figure 2-1 depicts relationships between subsystems and external devices and systems. This construct of DIN-PACS reflects logical concepts and does not necessarily represent physical entities within the system. Figure 2-2 shows a particular implementation of DIN-PACS developed by IBM.
2.1.1 Acquisition Pathway Subsystem

The image acquisition pathway subsystem is defined as the physical and logical pathway from the image acquisition gateway to the safe storage of the image, and includes:

- Network Interfaces to DICOM Devices: to interface and integrate DICOM conforming equipment (both Government and Contractor furnished) with the DIN-PACS network.
- Interface to Non-DICOM Devices: to accept images from non-DICOM compliant modalities and to provide DICOM services (e.g., worklist management) via a user workstation in the physical vicinity of the modality.
- DICOM Modality Worklist Management Provider services to support scheduling and patient demographic information at the modality.
• Network Failure Protection and Recovery: for all Image Acquisition Network Interfaces. The system shall provide pathways and methods to bypass failing segments, nodes, and devices, in order to continue clinical operations and ensure safe image and data storage.

Figure 2-2: IBM Implementation of DIN-PACS

2.1.2 Display Subsystem

The image display subsystem includes:

• Soft Copy Image and Report Display Workstations: to display and manipulate digital images (gray-scale and color where applicable) and associated overlays (i.e., textual and graphical annotations) and reports. This capability is defined for three classes of display workstations, one for primary image interpretation, the “diagnostic workstation”, one for secondary clinical review, the “review workstation”, and finally one for Government furnished personal computers (PC’s).
Video Projector System: to project images onto a larger format backdrop for viewing in conference and teaching environments.

Large Screen Monitor: to display images in a conferencing and teaching environment.

Laser Imager: to print images to film in multiple formats for translumination. DIN-PACS vendor usually provides integrated laser imager for both wet and dry process printing as well as DICOM-compliant interfaces to government-furnished laser imaging devices which may be connected as dedicated DIN-PACS network devices or as components of modality equipment groups (e.g., mini-PACS).

2.1.3 Database and Archive Subsystem

The image database and archive subsystem supports all DICOM services, which includes:

- Database Management System: to maintain the information integrity of the system ensuring the proper transfer of images and data into, within, and out of, the imaging network and DIN-PACS.

- Image Examination Storage: to provide configurable, seamless, and autonomous, temporary short-term storage, temporary intermediate storage and permanent long-term archive of images as well as associated overlays and reports.

- Interface to the Composite Health Care System (CHCS): to provide a bi-directional interface to CHCS to facilitate the efficient retrieval and transmission of patient demographic data, patient examination data, and reports. A subset of this information is exchanged between the DIN-PACS and the modalities in the form of modality worklists. CHCS is the Department of Defense’s computerized healthcare information system that is
deployed worldwide, across all levels of medical treatment facilities both on land and at sea.

2.1.4 Communication and Control Subsystem

The Communication and Control Subsystem is conceptually defined as those devices that support and control the communication of images and associated information. The physical arrangement and functioning of the devices is dependent upon the vendor's proposed architecture. The control component of this subsystem provides the intelligence needed to configure and manage system components, operation, and flow of information, in order to achieve predetermined performance requirements.

Figure 2-3 shows the DICOM conformance requirements for the DIN-PACS system. It specifies the interfaces under "DICOM In" and "DICOM Out". This requirement specification is seen from an "outside perspective" looking into the DIN-PACS system, i.e., based on the function that has to be provided via the DICOM interface viewed from the viewpoint of the devices that are connected.

The "DICOM In" interface contains acquisition or DICOM modalities, Teleradiology, and DICOM Workstation. The "DICOM Out" interface connects DIN-PACS with other DICOM Modalities and Workstation, Laser Printers, Government Furnished Equipment (GFE) PCs, and Teleradiology connection.

There are other modules connected to DIN-PACS that are not yet DICOM compliant. These modules communicate directly to DIN-PACS without DICOM interfaces: RIS Terminal, HL7 - CHCS Interface, Voice Recognition, Open Database Connectivity (ODBC) Interface, Failover Recovery Server, Laser Imager, Diagnostic Workstations, Clinical Workstation, and Quality Control (QC) Workstation.
2.2 GLOBAL PACS

A more advanced system that enables DIN-PACS to operate in a larger environment is called GPACS (Global Picture Archiving and Communications System). It is a medical imaging system that enables doctors and clinicians to capture, archive, and retrieve medical images over wide area networks. A Global PACS environment provides new and beneficial operations between radiologists and physicians when they are located in different geographical locations. A prototype multimedia collaboration system for the PACS area, called Remote Consultation and
Diagnosis (RCD) system Version 1.0, has been developed by the Computer Engineering Research Laboratory (CERL) of the University of Arizona [29], [33].

A Picture Archiving and Communication System (PACS) environment has been defined to combine Imaging Equipment (IE), viewing Workstation (WS), Database Archive System (DBAS) and a high-speed fiber optic backbone network on a state and national scale. The Global PACS environment enables new user scenarios and collaboration in radiology that improve the delivery and speed of medical diagnosis. In this project, distributed system software, called Remote Consultation and Diagnosis (RCD), has been developed. RCD provides operations with which two radiologists at different geographical locations can perform diagnosis on the same patient images. The RCD includes real-time voice interaction during the RCD session. The RCD session is stored in the patient folder and can be played back later.

The system consists of three major parts, the Local Workstation (WS), remote WS Consultant, and Database Archive System (DBAS). The viewing workstations provide a simple X-Window based user interface, and communicate with the DBAS send and receive an image set for diagnosis. The radiologists annotate the static image with framing information at the Remote and Local WS. The framing information or image annotation commands include a fixed size frame, a variable size frame, and a free-hand drawing, which are used to point out image features.

A prototype Global PACS system has been implemented and tested between SUN Sparc and DEC Alpha workstations between a Local and Remote PACS connected via the national Internet. The remote site on the Internet is the Bowman Gray School of Medicine at Wake Forest University, Winston-Salem, North Carolina. Performance tests over the Internet has shown that the real-time image annotation command information (400-byte packet) round trip response times across country are below an average of 188 milliseconds. This includes an acknowledgment packet from the Remote WS. Real-time voice packets are delivered in less than 200 milliseconds.
Images of size 1 MB are transferred in less than 2 minutes. In other parts of the Internet, the response time may be faster or slower depending on the site location and the communications links to that site.

Figure 2-4: Global PACS Architecture
The Global PACS distributed software for remote consultation and diagnosis is implemented in Unix using an X-windows user interface, Remote Procedure Calls (RPC), File Transfer Protocol (FTP), and a distributed file management system. The software includes connections to rural sites that are not on the Internet through the use of a SLIP telephone line over 28.8 Kbps modems. A video conferencing feature has been added to the remote consultation and diagnosis software, which allows RCD session participants to view each other or allow a live operation or diagnosis to take place in the session. The video window is overlaid on the static image display screen.

In addition, the system software has been re-designed using the Open Software Foundation Distributed Computing Environment (OSF DCE) services. These services will allow security features and scalability to 1000's of Global PACS workstations nodes and database archive systems. The OSF DCE environment will allow interoperability and security features among heterogeneous platforms in the Global PACS environment. The collaboration features of the RCD software system can be applied to other application areas. For example, in distance learning systems students can collaborate with the instructor using multimedia workstation during the lecture. Another example is in planetary sciences where planetary spacecraft images are acquired and stored in an archive center. In these applications, image collaboration and visualization through distributed system software over gigabit wide area networks is possible.

2.3 DIGITAL IMAGING AND COMMUNICATIONS IN MEDICINE (DICOM)

The American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) formed a joint committee to develop a Standard for Digital Imaging and
Communications in Medicine (DICOM). This DICOM Standard was developed according to the NEMA Procedures.

With the introduction of computed tomography (CT) followed by other digital diagnostic imaging modalities in the 1970's, and the increasing use of computers in clinical applications, the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) recognized the emerging need for a standard method for transferring images and associated information between devices manufactured by various vendors. These devices produce a variety of digital image formats.

The American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) formed a joint committee in 1983 to develop a standard to:

- Promote communication of digital image information, regardless of device manufacturer
- Facilitate the development and expansion of picture archiving and communication systems (PACS) that can also interface with other systems of hospital information
- Allow the creation of diagnostic information data bases that can be interrogated by a wide variety of devices distributed geographically.

This standard, now designated Digital Imaging and Communications in Medicine (DICOM) Version 3.0, specified a hardware interface, a minimum set of software commands, and a consistent set of data formats.

2.3.1 DICOM Information Model

A thorough examination of the types of services needed to communicate over different networks showed that defining a basic service would allow the top layer of the communications process (the application layer) to talk to a number of different network protocols. These protocols are modeled as a series of layers, often referred to as "stacks." Figure 2-5 shows a diagram of the
communication model developed. The basic design philosophy was that a given medical imaging application (which is outside of the scope of the standard) could communicate over any of the stacks to another device that used the same stack. With adherence to the standard, it would be possible to switch the communications stacks without having to rewrite the computer programs of the application.

DICOM relies on explicit and detailed models of how the “things” (patients, images, reports, etc.) involved in radiology operations are described and how they are related. These models are called entity-relationship (or E-R) models and are a way to be sure that manufacturers and users understand the basis for developing the data structures used in DICOM.

Figure 2-5: DICOM Protocol Stack
Figure 2-6 shows an example of an E-R diagram, the overall application model defend for DICOM. This modeling process was begun in one of the working groups created as DICOM was being developed by Working Group (WG) VIII. This group began with the task of defining the interface requirements between a picture archiving and communications system (PACS) network and a hospital or radiology information system (HIS or RIS). This definition process required that the operations in radiology be properly modeled so that what was needed from an HIS or RIS could be determined along with what would be done with that information in the PACS. The basic E-R diagram for radiology department function served as the basis for most of the additional modeling done by WG VI in developing DICOM.

The advantage of these models is that they clearly show both the data items required in a given scenario being modeled and how these items interact and are related. In looking at an E-R diagram it is important to note that it is not a flowchart that describes the steps of information movement; rather, it shows the relationships and hierarchies of information elements. Arrows are added to diagrams so that the direction of relationships is not misinterpreted. These diagrams are widely used throughout the DICOM standard as they clearly show the assumptions made in developing the components of the standard.
Figure 2-6: A DICOM Entity-Relation Diagram

1 The rectangular boxes represent the entities that singly, or in combination, form the information objects. The diamond-shaped boxes are the relationships. The arrows represent the connections between entities and relationships and are shown with arrows to give some idea of the hierarchy not necessarily the movement of information. The "1's" and "N's" indicate whether the relationship involves one entity to one entity, one to N, N to one, or N to N entities. IOD - information object definition, VOI - value of interest, LUT - look-up table, Mod - modality.
2.4 DEFINITION AND BACKGROUND

Defined in simple terms, Virtual Radiology Environment (VRE) is an interoperable solution at various levels of global teleradiology and PACS (Picture Archiving and Communication System) environments. It is a solution to interoperability problems in existing systems as well as a successor of hospital PACS. It also provides links to HIS (Hospital Information System) and Telemedicine environment.

The Virtual Radiology Environment (VRE) is a concept that takes advantage of multiple Picture Archiving & Communications Systems (PACS) via vast and robust networks. The VRE Intranet will interconnect major U.S. Army medical centers to Medical Treatment Facilities (MTFs) located in Army Remote Medical Command (RMC) bases. The purpose of the VRE is to move patient cases between MTF sites in order to make efficient use of radiology resources. The VRE will provide Teleradiology services to address the issues of cost, improved quality, and medical access in the U.S. Army.

It requires a high bandwidth backbone network, such as ATM, to interconnect US Army Medical Centers and Medical Treatment Facilities (MTFs) for radiology services. It consists of US Army Medical Centers and Clinical MTFs that have medical specialty expertise to share, and uses store-forward and interactive consultation technology for multimedia sessions between referring & specialist physicians. Furthermore, it also provides interoperability to US Army Medical legacy systems. VRE uses multimedia health standard to store and archive patient medical case and optimizes use of radiology resources.

There are three classes of service that can be obtained from VRE:

• Interpretative
  • Image Acquisition & Storage
  • Store & Forward
• Consultative
  • Interactive Tele-consultation With Image Annotation & Interactive Voice
  • Video Teleconferencing
• Interventional
  • Video Proctoring & Mentoring
  • Real-Time Vital Signs Monitoring

Some of the benefits of using and implementing VRE includes:

• Improve Access
  • 24 hour Staffing
  • Decrease Report Turn Around Times
• Improve Quality
  • Subspecialty Expertise
  • Resident Staffing
  • CME
  • Enhanced Capabilities Through Higher Bandwidth Access
• Diminish Costs
  • Workload Redistribution
  • Contract Decrease or Shift to Lower Cost Areas

2.5 META-MANAGER

In implementing its operation and administration, VRE relies on an integral agent called the Meta-Manager. Meta-Manager (MM) is an intelligent set of algorithms used by VRE for image acquisition, image transfer, network monitoring, case routing, mission control, fail-safe, and resource optimization. It allows instant accessibility to multiple data streams and the
streamlining of workflow throughout the entire entity. Meta-Manager understands the "state" of the VRE system at any time. This state of the system is a collection of VRE system-wide variables, which include network connectivity, site workload, radiologists schedule, and resource availability.

The primary purpose of the Meta-Manager system is to perform intelligent routing of patient cases between the Picture Archiving and Communication System (PACS) hospitals in the VRE network. The routing of the cases is based on dynamic information within each DIN-PACS node and the status of the communications network. The Meta-Manager enables the communications network to represent a virtual radiology department in the Army. The Meta-Manager Team of the Computer Engineering Research Laboratory (CERL) has used an object-oriented systems engineering methodology, the Rational Rose's using Rational Objectory Process (ROP), to model the functionality, design and prototype of the Meta-Manager system.

Meta-Manager is "intelligent" in the sense that it is distributed among the participating sites of VRE. A Meta-Manager in each site maintains a site-state, communicates to each other to inform about the changing in its site-sites. The aggregation of these site-states constitutes the state of the VRE system. Therefore, Meta-Manager knows what is going on in each site, and makes decisions accordingly.

In addition, the intelligence of the Meta-Manager comes into play when it routes cases in the system. Since each case has its own value for urgency, preferred radiologists, subspecialty required, Meta-Manager tried to find the best destination of the case using these variables. Priorities are set among the variables to determine their importance in making decision. The operations of Meta-Manager includes:

- Surveying system connectivity & delay
- Detecting radiologist on-line availability
• Routing and rerouting cases (images and data)
• Identifying site workload and reading speed
• Storing radiologist subspecialty
• Consulting radiologist schedule
• Maintaining fail-safe operation and reliability
• Controlling Mission Control GUI

2.6 VRE HIERARCHY

VRE is a large-scale real-time distributed system. It is desirable to take advantage of the size of the system to provide new levels of interaction between remote physicians. At the same time the ARMY would like the new system to adhere to their existing organizational structure for medical operations. This introduces a communications hierarchy into the VRE system.

The hierarchy of the VRE system contains the DIN-PACS nodes, Client Meta-Manager (CMM) nodes, Regional Meta-Manager (RMM) nodes, and Master Meta-Manager (MMM) node. There must be one and only one Master Meta-Manager node in the system that behaves as the root of the hierarchy tree. And for each DIN-PACS node, there must be a Client Meta-Manager node associated with it.

The system is hybrid because it contains distributed and centralized parts. Within a region, the system is centralized toward its Regional Meta-Manager. The Regional Meta-Manager aggregates the knowledge (database) from all Client Meta-Managers underneath it, and makes decisions of all the inter-DIN-PACS case routing within the region. The Client Meta-Manager is simply the representation the DIN-PACS underneath it. There must be one Client Meta-Manager for each DIN-PACS in the system, and it must receives a notification from DIN-PACS for all new cases to be routed.
In the event the Regional Meta-Manager cannot determine an optimum destination for a new case, it passes the case information to the Master Meta-Manager. The Master Meta-Manager, which has a list of other Regional Meta-Managers in the systems, sends the case information to all Regional Meta-Managers. The Meta-Manager that can handle the request then send a notification message to the Master Meta-Manager, which then sends the results to the requesting Regional Meta-Manager.

Figure 2-7: VRE Hybrid Hierarchy
CHAPTER 3: COMMON OBJECT REQUEST BROKER ARCHITECTURE

This chapter covers a brief introduction to Common Object Request Broker (CORBA). A particular implementation of CORBA, called VisiBroker, is explained in detail.

3.1 OVERVIEW OF DISTRIBUTED OBJECT & COMPONENT PARADIGM

The rapidly proliferating number of hardware and software products available today as well as the exponential growth of the Internet has opened the door to technologies that are changing the way we use computers. Some of the biggest changes are in object-oriented technology, where software developers and scientists are giving new meaning to distributed computing. Instead of the traditional client/server network, the new era of distributed computing brings with it enterprise systems whose objects are distributed across multiple computers that can all communicate, regardless of operating system, platform, or programming language.

Object-orientation is a new technology based on objects and classes. It is presently the most widely used methodological framework for software engineering and its pragmatics provides the foundation for a systematic engineering discipline. By providing first class support for the objects and classes of objects of an application domain, the object-oriented paradigm precepts better modeling and implementation of systems. Objects provide a canonical focus throughout analysis, design, and implementation by emphasizing the state, behavior, and interaction of objects in its models, providing the desirable property of seamlessness between activities.

Even more amazing is that these distributed objects can be components of a single application or of the dozens of applications that form the enterprise system. Technologies that enable distributed computing across heterogeneous systems are the Common Object Request
Broker Architecture (CORBA). CORBA specifies the complete architecture necessary for communication between distributed objects. CORBA is a distributed object architecture that allows objects to interoperate across networks regardless of the language in which they were written or the platform on which they are deployed. It allows developers to write applications that are more flexible and future-proof, to wrap legacy systems, and to code in the language they know best. The Object Request Broker (ORB) is the middleware that handles the communication details between the objects. The CORBA 2.0 standard, adopted in December of 1994, defines true interoperability by specifying how ORBs from different vendors can communicate using a common protocol.

Component-oriented programming has been described as the natural extension of object-oriented programming to the realm of independently extensible systems. Component-oriented programming aims at producing software components for a component market and for late composition. Composers are third parties, possibly the end user, who are not able or willing to change components. This requires standards to allow independently created components to interoperate, and specifications that put the composer into the position to decide what can be composed under which conditions.

Distributed object computing is a computing paradigm that allows objects to be distributed across a heterogeneous network, and allows each of the components to interoperate as a unified whole. To an application built in a distributed object environment, and as expressed in Sun Microsystem's slogan, the network is the computer. Object orientation can radically simplify systems development. Distributed object models and tools extend an object-oriented programming system. The objects may be distributed on different computers throughout a network, living within their own dynamic library outside of an application, and yet appear as
though they were local within the application. This is the essence of plug-and-play software. Several technical advantages result from a distributed object environment.

(1) Legacy assets can be leveraged.

(2) Programmers have the ability to distribute components of an application to computers that best fit the task of each object without having to change the rest of the application using these objects.

(3) Since objects appear to be local to their clients, a client does not know what machine, or even what kind of machine, an object resides on.

(4) Systems integration can be performed to a higher degree. The overall technical goal of distributed object computing is clear: to advance distributed information technologies so that they may be more efficient and flexible, yet less complex. The benefits of distributed objects are indeed solutions to the problems with existing, monolithic client/server paradigms.

*Component-based development* represents the ‘industrialization’ of software development. When any manufacturing process evolves to the point where it can be based on pre-built components and subassemblies, product quality, quantity, and speed of delivery soar. This principle applies equally to software systems development, allowing unprecedented quality, speed of development, and highly effective change management. However, a fundamental mindset change toward components is necessary to usher in the industrial era of software development.

COM (Component Object Model) provides the component technology for Microsoft Windows Distributed interNet Applications (Windows DNA) architecture, which enables developers to integrate Web-based and client/server applications in a single, unified architecture. Using COM, developers can create distributed components that are written in any language and that can interact over any network. COM+ will make it even easier for developers to create
software components in any language using any tool. COM+ builds on the factors that have made today's COM the choice of developers worldwide, including the following:

1. The richest integrated services, including transactions, security, message queuing and database access to support the broadest range of application scenarios.
2. The widest choice of tools from multiple vendors using multiple development languages.
3. The largest customer base for customizable applications and reusable components.
4. Proven interoperability with users' and developers' existing investments.

The Common Object Request Broker Architecture (CORBA), developed by the Object Management Group (OMG) in 1990, enables invocations of methods on distributed objects residing anywhere on a network, just as if they were local objects. A CORBA implementation employs Object Request Brokers (ORBs), located on both the client and the server, to create and manage client/server communications between objects. ORBs are the key to the CORBA distributed object architecture. They allow objects on the client side to make requests of objects on the server side without any prior knowledge of where those objects exist, what language they are in, or what operating system they are running on. To facilitate these requests and provide ORB interoperability, the CORBA 2.0 specification outlines a protocol named Internet Inter-ORB Protocol, which has quickly been embraced by industry leaders.

JavaBeans is the platform-neutral component architecture for Java and has proven to be invaluable in the development of network-aware applications. A software developer can make any Java class into a bean just by changing the class to adhere to the JavaBeans specification. It's up to the developer to decide what to design as a bean and what to design as a Java class: It's a good idea to leave all the library classes as Java classes, but graphical user interface (GUI) elements can be designed to be beans. Some beans - such as beans on a server - can be non-GUI-related. Regardless of its functionality, every bean should support the following
characteristics and behavior: Persistence, Visual manipulation, Introspection, Events and Customization.

*Enterprise JavaBeans* takes the remarkably successful JavaBeans component architecture released in JDK 1.1 to the next level by providing an API optimized for building scalable business applications as reusable server components. With Enterprise JavaBeans, developers can design and re-use small program elements to build powerful corporate applications. These 'componentized' applications can run manufacturing, financial, inventory management, and data processing on any system or platform that is Java-enabled.

3.2 INTRODUCTION TO CORBA 3

Common Object Request Broker Architecture (CORBA) is a specification adopted by the Object Management Group (OMG) to address the complexity and high cost of developing object-oriented, distributed, heterogeneous applications. CORBA uses an object-oriented approach for creating reusable and shareable software components. Each object encapsulates the details of its inner workings and presents a well-defined interface with which other object communicate. This approach reduces not only the complexity of the application, but also the cost of development, because once an object is implemented and tested, it can be used over and over again. In other words, CORBA allows system developers to write a future-proof applications and wrap legacy systems.

Distributed applications can have objects located anywhere in a computer network, i.e., LAN, WAN, or Internet. Heterogeneous implies different parts of the system that use different programming languages, operating systems, hardware, as well as networking protocols. In other words, CORBA automates many common network programming tasks such as object registration, location, activation, and so on.
CORBA Object Services define a wide variety of common system services. These services, defined independently of operating environments, provide the system developer with standard interfaces to functionality that is commonly used.

CORBA 3 is the latest CORBA specifications standard defined by the Object Model Group (OMG). It is an umbrella term that actually refers to a suite of specifications which, taken together, add a new dimension of capability and ease-of-use to CORBA. Most of the specifications in the suite are available now; the last few will be available in draft form around the beginning of August and will start their final adoption votes at the OMG meeting in San Jose, California, on August 27, 1999.

Under a new OMG procedure, these specifications will start out classified as "pre-production", and transform to ‘available’ (that is, an official OMG specification representing the current CORBA specification) only after they have finished a round of maintenance revision and
the first products become available. Even though this won't happen until mid to late 2000, the specifications are available now; the revisions can only change technical details and not affect the structure of any part of the new release.

The specifications included in the designation CORBA 3 divide neatly into three major categories:

- Internet Integration;
- Quality of Service Control; and
- The CORBA component architecture

3.2.1 Internet Integration

The following specifications enhance CORBA integration with the increasingly popular Internet:

3.2.1.1 Firewall Specification

The CORBA 3 Firewall Specification defines transport-level firewalls, application-level firewalls, and (perhaps most interesting) a bi-directional GIOP connection useful for callbacks and event notifications.

Transport-level firewalls work at the TCP level. By defining (courtesy of IANA) well-known ports 683 for IIOP and 684 for IIOP over SSL, the specification allows administrators to configure firewalls to cope with CORBA traffic over the IIOP protocol. There is also a specification for CORBA over SOCKS.

In CORBA, objects frequently need to call back or notify the client that invoked them; for this, the objects act as clients and the client-side module instantiates an object that is called back in a reverse-direction invocation. Because standard CORBA connections carry invocations
only one way, a callback typically requires the establishing of a second TCP connection for this traffic heading in the other direction, which is a no-no to virtually every firewall in existence. Under the new specification, an IIOP connection is allowed to carry invocations in the reverse direction under certain restrictive conditions that don't compromise the security at either end of the connection.

3.2.1.2 Interoperable Name Service

The CORBA object reference is a cornerstone of the architecture. Because the computer-readable IOR was (until recently) the only way to reach an instance and invoke it, there was no way to reach a remote instance - even if you knew its location and that it was up and running - unless you could get access to its object reference. The easiest way to do that was to get a reference to its Naming Service; however, many problems arise when a reference for the object did not exist in the local system.

The Interoperable Name Service defines one URL-format object reference, iioploc, that can be typed into a program to reach defined services at a remote location, including the Naming Service. A second URL format, iiopname, actually invokes the remote Naming Service using the name that the user appends to the URL, and retrieves the IOR of the named object. For example, an iioploc identifier iioploc://www.omg.org/NameService would resolve to the CORBA Naming Service running on the machine whose IP address corresponded to the domain name www.omg.org (if we had a Name Server running here at OMG).
3.2.2 Quality of Service (QoS) Control

3.2.2.1 Asynchronous Messaging and Quality of Service Control

The new Messaging Specification defines a number of asynchronous and time-independent invocation modes for CORBA, and allows both static and dynamic invocations to use every mode. Asynchronous invocations' results may be retrieved by either polling or callback, with the choice made by the form used by the client in the original invocation. Policies allow control of Quality of Service of invocations. Clients and objects may control ordering (by time, priority, or deadline); set priority, deadlines, and time-to-live; set a start time and end time for time-sensitive invocations, and control routing policy and network routing hop count.

3.2.2.2 Minimum, Fault-Tolerant, and Real-Time CORBA

Minimum CORBA is primarily intended for embedded systems. Embedded systems, once they are finalized and burned into chips for production, are fixed, and their interactions with the outside network are predictable. They have no need for the dynamic aspects of CORBA, such as the Dynamic Invocation Interface or the Interface Repository that supports it, which are therefore not included in Minimum CORBA.

Real-time CORBA standardizes resource control - threads, protocols, connections, and so on - using priority models to achieve predictable behavior for both hard and statistical realtime environments. Dynamic scheduling, not a part of the current specification, is being added via a separate RFP.

Fault-tolerance for CORBA is being addressed by an RFP, also in process, for a standard based on entity redundancy, and fault management control.²

---

² See [http://www.omg.org/techprocess/meetings/schedule/Fault_Tolerance_RFP.html](http://www.omg.org/techprocess/meetings/schedule/Fault_Tolerance_RFP.html) for all information on this RFP.
3.2.3 CORBAComponents Package

3.2.3.1 CORBAcomponents and CORBAscripting

One of the most exciting developments to come out of OMG since the IIOP protocol defined in CORBA 2, CORBAcomponents represent a multi-pronged advance with benefits for programmers, users, and consumers of component software. The three major parts of CORBAcomponents are:

- A container environment that packages transactionality, security, and persistence, and provides interface and event resolution;
- Integration with Enterprise JavaBeans; and
- A software distribution format that enables a CORBAcomponent software marketplace.

The CORBAcomponents container environment is persistent, transactional, and secure. For the programmer, these functions are pre-packaged and provided at a higher level of abstraction than the CORBAservices provide. This leverages the skills of business programmers who are not necessarily skilled at building transactional or secure applications, who can now use their talents to produce business applications that acquire these necessary attributes automatically.

Containers keep track of event types emitted and consumed by components, and provide event channels to carry events. The containers also keep track of interfaces provided and required by the components they contain, and connect one to another where they fit. CORBAcomponents support multiple interfaces, and the architecture supports navigation among them.

Enterprise JavaBeans (EJBs) will act as CORBAcomponents, and can be installed in a CORBAcomponents container. Unlike EBJs, of course, CORBAcomponents can be written in multiple languages and support multiple interfaces.
The specification defines a multi-platform software distribution format, including an installer and XML-based configuration tool, and a separate Scripting specification will map CORBA and component assembly to a number of established scripting languages.

After ten years of cooperative work by OMG members, the base CORBA infrastructure is complete and in constant use at thousands of sites. The extensions bundled under the banner CORBA 3 bring ease-of-use and precise control to our established architecture. These additions will ensure that CORBA continues to play an ever-increasing role in the computing world of the future.

3.3 COMPARISONS OF CORBA, DCOM AND JAVA/RMI

Three of the most popular distributed object paradigms are Microsoft's Distributed Component Object Model (DCOM), OMG's Common Object Request Broker Architecture (CORBA) and JavaSoft's Java/Remote Method Invocation (Java/RMI). In this article, let us examine the differences between these three models from a programmer's standpoint and an architectural standpoint. At the end of this article, you will be able to better appreciate the merits and innards of each of the distributed object paradigms.

CORBA relies on a protocol called the Internet Inter-ORB Protocol (IIOP) for remoting objects. Everything in the CORBA architecture depends on an Object Request Broker (ORB). The ORB acts as a central Object Bus over which each CORBA object interacts transparently with other CORBA objects located either locally or remotely. Each CORBA server object has an interface and exposes a set of methods. To request a service, a CORBA client acquires an object reference to a CORBA server object. The client can now make method calls on the object

---

3 This section is an excerpt of a more detailed comparison study conducted by Gopalan Suresh Raj at http://www.execpc.com/~gopalan/misc/compare.html.
reference as if the CORBA server object resided in the client's address space. The ORB is responsible for finding a CORBA object's implementation, preparing it to receive requests, communicating requests to it and carry the reply back to the client. A CORBA object interacts with the ORB either through the ORB interface or through an Object Adapter - either a Basic Object Adapter (BOA) or a Portable Object Adapter (POA). Since CORBA is just a specification, it can be used on diverse operating system platforms from mainframes to UNIX boxes to Windows machines to handheld devices as long as there is an ORB implementation for that platform. Major ORB vendors like Inprise have CORBA ORB implementations through their VisiBroker product for Windows, UNIX and mainframe platforms and Iona through their Orbix product.

DCOM, which is often called 'COM on the wire', supports remoting objects by running on a protocol called the Object Remote Procedure Call (ORPC). This ORPC layer is built on top of DCE's RPC and interacts with COM's run-time services. A DCOM server is a body of code that is capable of serving up objects of a particular type at runtime. Each DCOM server object can support multiple interfaces each representing a different behavior of the object. A DCOM client calls into the exposed methods of a DCOM server by acquiring a pointer to one of the server object's interfaces. The client object then starts calling the server object's exposed methods through the acquired interface pointer as if the server object resided in the client's address space. As specified by COM, a server object's memory layout conforms to the C++ vtable layout. Since the COM specification is at the binary level it allows DCOM server components to be written in diverse programming languages like C++, Java, Object Pascal (Delphi), Visual Basic and even COBOL. As long as a platform supports COM services, DCOM can be used on that platform. DCOM is now heavily used on the Windows platform. Companies like Software AG provide COM service implementations through their EntireX product for UNIX, Linux and mainframe platforms; Digital for the Open VMS platform and Microsoft for Windows and Solaris platforms.
Java/RMI relies on a protocol called the Java Remote Method Protocol (JRMP). Java relies heavily on Java Object Serialization, which allows objects to be marshaled (or transmitted) as a stream. Since Java Object Serialization is specific to Java, both the Java/RMI server object and the client object have to be written in Java. Each Java/RMI Server object defines an interface which can be used to access the server object outside of the current Java Virtual Machine (JVM) and on another machine's JVM. The interface exposes a set of methods that are indicative of the services offered by the server object. For a client to locate a server object for the first time, RMI depends on a naming mechanism called an RMIRegistry that runs on the Server machine and holds information about available Server Objects. A Java/RMI client acquires an object reference to a Java/RMI server object by doing a lookup for a Server Object reference and invokes methods on the Server Object as if the Java/RMI server object resided in the client's address space. Java/RMI server objects are named using URLs and for a client to acquire a server object reference, it should specify the URL of the server object as you would with the URL to a HTML page. Since Java/RMI relies on Java, it can be used on diverse operating system platforms from mainframes to UNIX boxes to Windows machines to handheld devices as long as there is a Java Virtual Machine (JVM) implementation for that platform. In addition to Javasoft and Microsoft, a lot of other companies have announced Java Virtual Machine ports.

3.3.1 Application Sample - The StockMarket Server and Client

The StockMarket server reports the stock price of any given symbol. It has a method called get_price() to get the stock value of a particular symbol. This test application was written in Java.

Each of these implementations defines an IStockMarket interface. They expose a get_price() method that returns a float value indicating the stock value of the symbol passed in.
We list the sources from four files. The first set of files are the IDL and Java files that define the interface and its exposed methods. The second sets of files show how the client invokes methods on these interfaces by acquiring references to the server object. The third sets of files show the Server object implementations. The fourth sets of files show the main program implementations that start up the Remote Server objects for CORBA and Java/RMI. No main program implementation is shown for DCOM since the JavaReg program takes up the role of invoking the DCOM Server object on the Server machine. This means you have to also ensure that JavaReg is present on your server machine.

3.3.2 The IDL Interface

Whenever a client needs some service from a remote distributed object, it invokes a method implemented by the remote object. The service that the remote distributed object (Server) provides is encapsulated as an object and the remote object's interface is described in an Interface Definition Language (IDL). The interfaces specified in the IDL file serve as a contract between a remote object server and its clients. Clients can thus interact with these remote object servers by invoking methods defined in the IDL.

DCOM

The DCOM IDL file shows that our DCOM server implements a dual interface. COM supports both static and dynamic invocation of objects. It is a bit different than how CORBA does through its Dynamic Invocation Interface (DII) or Java does with Reflection. For the static invocation to work, The Microsoft IDL (MIDL) compiler creates the proxy and stub code when run on the IDL file. These are registered in the systems registry to allow greater flexibility of their use. This is the vtable method of invoking objects. For dynamic invocation to work, COM objects
implement an interface called IDispatch. As with CORBA or Java/RMI, to allow for dynamic invocation, there has to be some way to describe the object methods and their parameters. Type libraries are files that describe the object, and COM provides interfaces, obtained through the IDispatch interface, to query an Object’s type library.

In COM, an object whose methods are dynamically invoked must be written to support IDispatch. This is unlike CORBA where any object can be invoked with DII as long as the object information is in the Implementation Repository. The DCOM IDL file also associates the IStockMarket interface with an object class StockMarket as shown in the coclass block. Also notice that in DCOM, each interface is assigned a Universally Unique Identifier (UUID) called the Interface ID (IID). Similarly, each object class is assigned a unique UUID called a Class ID (CLSID). COM gives up on multiple inheritance to provide a binary standard for object implementations. Instead of supporting multiple inheritance, COM uses the notion of an object having multiple interfaces to achieve the same purpose. This also allows for some flexible forms of programming.

CORBA

Both CORBA and Java/RMI support multiple inheritance at the IDL or interface level. One difference between CORBA (and Java/RMI) IDLs and COM IDLs is that CORBA (and Java/RMI) can specify exceptions in the IDLs while DCOM does not. In CORBA, the IDL compiler generates type information for each method in an interface and stores it in the Interface Repository (IR). A client can thus query the IR to get run-time information about a particular interface and then use that information to create and invoke a method on the remote CORBA server object dynamically through the Dynamic Invocation Interface (DII).
Similarly, on the server side, the Dynamic Skeleton Interface (DSI) allows a client to invoke an operation of a remote CORBA Server object that has no compile time knowledge of the type of object it is implementing. The CORBA IDL file shows the StockMarket interface with the get_price() method. When an IDL compiler compiles this IDL file it generates files for stubs and skeletons.

Table 3.1: Comparison of IDL from Various Architecture

<table>
<thead>
<tr>
<th>DCOM - IDL</th>
<th>CORBA - IDL</th>
<th>Java/RMI - Interface definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>library SimpleStocks</td>
<td>module SimpleStocks</td>
<td>package SimpleStocks;</td>
</tr>
<tr>
<td>import &quot;stdole32.tlb&quot;;</td>
<td>{</td>
<td>import java.rmi.*;</td>
</tr>
<tr>
<td>uuid(7371a240-2e51-11d0-b4c1-444553540000), version(1.0)</td>
<td>interface StockMarket</td>
<td>import java.util.*;</td>
</tr>
<tr>
<td>library SimpleStocks</td>
<td>{</td>
<td>public interface StockMarket</td>
</tr>
<tr>
<td>interface IStockMarket : Idispatch</td>
<td>{</td>
<td>extends java.rmi.Remote</td>
</tr>
<tr>
<td>HRESULT get_price([in] BSTR pl, [out, retval] float * rtn);</td>
<td>float get_price( String symbol );</td>
<td>{</td>
</tr>
<tr>
<td>{</td>
<td>}</td>
<td>float get_price( String</td>
</tr>
<tr>
<td>uuid(BC4C0AB3-5A45-11d2-99C5-00A02414C655),</td>
<td>}</td>
<td>symbol ) throws RemoteException;</td>
</tr>
<tr>
<td>coclass StockMarket</td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>{</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>interface IStockMarket;</td>
<td>};</td>
<td>File: StockMarketLib.idl</td>
</tr>
<tr>
<td>};</td>
<td>module SimpleStocks</td>
<td>File: StockMarket.idl</td>
</tr>
<tr>
<td></td>
<td>{</td>
<td>File: StockMarket.java</td>
</tr>
<tr>
<td></td>
<td>interface StockMarket</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td>float get_price( String symbol ) throws RemoteException;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

Java/RMI

Notice that unlike the other two, Java/RMI uses a .java file to define its remote interface. This interface will ensure type consistency between the Java/RMI client and the Java/RMI Server Object. Every remotable server object in Java/RMI has to extend the
java.rmi.Remote class. Similarly, any method that can be remotely invoked in Java/RMI may throw a java.rmi.RemoteException. The java.rmi.RemoteException is the superclass of many more RMI specific exception classes. We define an interface called StockMarket which extends the java.rmi.Remote class. Also notice that the get_price() method throws a java.rmi.RemoteException.

3.3.3 Conclusion

The architectures of CORBA, DCOM and Java/RMI provide mechanisms for transparent invocation and accessing of remote distributed objects. Though the mechanisms that they employ to achieve remoting may be different, the approach each of them take is more or less similar.

Table 3.2: Comparison of Features from Various Distributed Object Architecture

<table>
<thead>
<tr>
<th>DCOM</th>
<th>CORBA</th>
<th>Java/RMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports multiple interfaces for objects and uses the QueryInterface() method to navigate among interfaces. A client proxy dynamically loads multiple server stubs in the remoting layer depending on the number of interfaces being used.</td>
<td>Supports multiple inheritance at the interface level</td>
<td>Supports multiple inheritance at the interface level</td>
</tr>
<tr>
<td>Every object implements IUnknown.</td>
<td>Every interface inherits from CORBA.Object</td>
<td>Every server object implements java.rmi.Remote</td>
</tr>
<tr>
<td>Uniquely identifies a remote server object through its interface pointer, which serves as the object handle at run-time.</td>
<td>Uniquely identifies remote server objects through object references, which serves as the object handle at run-time. These references can be externalized (persistified) into strings which can then be converted back to an objref.</td>
<td>Uniquely identifies remote server objects with the ObjID, which serves as the object handle at runtime.</td>
</tr>
<tr>
<td>Uniquely identifies an interface using the concept of Interface IDs (IID) and uniquely identifies a named implementation of the server object using the concept of Class IDs (CLSID) the mapping of</td>
<td>Uniquely identifies an interface using the interface name and uniquely identifies a named implementation of the server object by its mapping to a name in the Implementation Repository</td>
<td>Uniquely identifies an interface using the interface name and uniquely identifies a named implementation of the server object by its mapping to a URL in the Registry</td>
</tr>
<tr>
<td>which is found in the registry.                                                                 conveyor engine</td>
<td>The remote server object reference generation is performed by the call to the method UnicastRemoteObject.exportObject (this)</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>The remote server object reference generation is performed on the wire protocol by the Object Adapter</td>
<td>The constructor implicitly performs common tasks like object registration, skeleton instantiation etc.</td>
<td>The RMI Registry performs common tasks like object registration through the Naming class.</td>
</tr>
<tr>
<td>Tasks like object registration, skeleton instantiation etc. are either explicitly performed by the server program or handled dynamically by the COM run-time system.</td>
<td>Uses the Object Remote Procedure Call (ORPC) as its underlying remoting protocol</td>
<td>Uses the Java Remote Method Protocol (JRMP) as its underlying remoting protocol</td>
</tr>
<tr>
<td>Uses the Object Remote Procedure Call (ORPC) as its underlying remoting protocol</td>
<td>When a client object needs to activate a server object, it can do a CoCreateInstance().</td>
<td>When a client object needs a server object reference, it has to do a lookup() on the remote server object's URL name.</td>
</tr>
<tr>
<td>The object handle that the client uses is the interface pointer</td>
<td>The object handle that the client uses is the Object Reference</td>
<td>The object handle that the client uses is the Object Reference</td>
</tr>
<tr>
<td>The mapping of Object Name to its Implementation is handled by the Registry</td>
<td>The mapping of Object Name to its Implementation is handled by the Implementation Repository</td>
<td>The mapping of Object Name to its Implementation is handled by the RMI Registry</td>
</tr>
<tr>
<td>The type information for methods is held in the Type Library</td>
<td>The type information for methods is held in the Interface Repository</td>
<td>The type information is held by the Object itself which can be queried using Reflection and Introspection</td>
</tr>
<tr>
<td>The responsibility of locating an object implementation falls on the Service Control Manager (SCM)</td>
<td>The responsibility of locating an object implementation falls on the Object Request Broker (ORB)</td>
<td>The responsibility of locating an object implementation falls on the Java Virtual Machine (JVM)</td>
</tr>
<tr>
<td>The responsibility of activating an object implementation falls on the Service Control Manager (SCM)</td>
<td>The responsibility of locating an object implementation falls on the Object Adapter (OA) - either the Basic Object Adapter (BOA) or the Portable Object Adapter (POA)</td>
<td>The responsibility of activating an object implementation falls on the Java Virtual Machine (JVM)</td>
</tr>
<tr>
<td>The client side stub is called a proxy</td>
<td>The client side stub is called a proxy or stub</td>
<td>The client side stub is called a proxy or stub</td>
</tr>
<tr>
<td>The server side stub is called stub</td>
<td>The server side stub is called a skeleton</td>
<td>The server side stub is called a skeleton</td>
</tr>
<tr>
<td>All parameters passed between the client and server objects are defined in the Interface Definition file. Hence, depending on what the IDL specifies, parameters are passed either by value or by reference.</td>
<td>When passing parameters between the client and the remote server object, all interface types are passed by reference. All other objects are passed by value including highly complex data types</td>
<td>When passing parameters between the client and the remote server object, all objects implementing interfaces extending java.rmi.Remote are passed by remote reference. All other objects are passed by value</td>
</tr>
<tr>
<td>Attempts to perform distributed garbage collection on the wire by pinging. The DCOM wire protocol uses a Pinging mechanism to garbage collect remote server</td>
<td>Does not attempt to perform general-purpose distributed garbage collection.</td>
<td>Attempts to perform distributed garbage collection of remote server objects using the mechanisms bundled in the JVM</td>
</tr>
<tr>
<td>object references.</td>
<td>Complex types that will cross interface boundaries must be declared in the IDL.</td>
<td>Any Serializable Java object can be passed as a parameter across processes.</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Allow the definition of arbitrarily complex structs, discriminated unions and conformant arrays in IDL and pass these as method parameters.</td>
<td>Will run on any platform as long as there is a COM Service implementation for that platform (like Software AG’s EntireX)</td>
<td>Will run on any platform as long as there is a CORBA ORB implementation for that platform (like Inprise’s VisiBroker)</td>
</tr>
<tr>
<td>Will run on any platform as long as there is a COM Service implementation for that platform (like Software AG’s EntireX)</td>
<td>Since the specification is at the binary level, diverse programming languages like C++, Java, Object Pascal (Delphi), Visual Basic and even COBOL can be used to code these objects</td>
<td>Since this is just a specification, diverse programming languages can be used to code these objects as long as there are ORB libraries you can use to code in that language</td>
</tr>
<tr>
<td>Since the specification is at the binary level, diverse programming languages like C++, Java, Object Pascal (Delphi), Visual Basic and even COBOL can be used to code these objects</td>
<td>Each method call returns a well-defined “flat” structure of type HRESULT, whose bit settings encode the return status. For richer exception handling it uses Error Objects (of type IErrorInfo), and the server object has to implement the ISupportErrorInfo interface.</td>
<td>Exception handling is taken care of by Exception Objects. When a distributed object throws an exception object, the ORB transparently serializes and marshals it across the wire.</td>
</tr>
<tr>
<td>Since this is just a specification, diverse programming languages can be used to code these objects as long as there are ORB libraries you can use to code in that language</td>
<td>Since it relies heavily on Java Object Serialization, these objects can only be coded in the Java language</td>
<td>Allows throwing exceptions which are then serialized and marshaled across the wire.</td>
</tr>
</tbody>
</table>

### 3.4 VISIBROKER FOR JAVA 3.3

The implementation of Virtual Radiology Environment uses a particular CORBA implementation called VisiBroker for Java version 3.3. VisiBroker is a market-leading CORBA ORB implementation developed by Inprise Corporation, the company that combined Borland and Visigenic, the original VisiBroker developer.

VisiBroker provides the ease-of-use, the scalability and the flexibility required to confront the many challenges of today’s complex, heterogeneous application environments. As the leading deployed CORBA implementation, VisiBroker makes good on the promise of the Object Management Group’s (OMG’s) CORBA standard: leveraging emerging Internet
technologies, enabling application re-use to improve time-to-market and providing a solid, industry standard foundation for mission-critical applications.

Figure 3-2: Inprise VisiBroker Family

The Inprise VisiBroker family of products provides an integrated suite of tools and services to enable the development, deployment, and management of flexible, scalable, and secure distributed object applications throughout an organization, an intranet, and across the Internet. Figure shows the Inprise VisiBroker family.
CHAPTER 4: VRE SYSTEM SPECIFICATIONS

This chapter covers the specifications of Virtual Radiology Environment (VRE), as defined by the Meta-Manager Team [30]. At the time of this writing, the specification is in still work-in-progress status, as new features and requirements may be added and existing ones modified or even eliminated.

Three major sections of this chapter cover the System Architecture, Requirement analysis and Domain analysis. The requirement analysis is further divided into Operational analysis, Functional analysis, and Performance analysis. The Domain analysis is further divided into Static Domain analysis and Dynamic Domain analysis.

4.1 SYSTEM ARCHITECTURE

As mentioned in section 2.6: VRE Hierarchy. VRE has a hybrid centralized-distributed architectural design. In the lowest level, the Client Meta-Manager level, the architecture is centralized, where all decisions within a particular DIN-PACS are made in a local centralized database system. Further up in the hierarchy, however, the VRE system becomes more distributed where Regional Meta-Manager and Master Meta-Manager are cooperating in tandem to find solutions or decisions for cases forwarded by DIN-PACS.

The lowest level of administration in the VRE system occurs at the individual Medical Treatment Facility (MTF) or DIN-PACS. Each site currently administers their own physicians, workstations and patient databases. In the VRE system this would still be the case, although for uniformity between sites, physician schedules are maintained as part of the MTF state.

Multiple MTF sites are grouped to form a regional VRE hierarchy. The sites in a VRE region will be determined by shared geographic, administrative and network connectivity
properties between MTF sites. These regional groupings correspond to Army Regional Medical Commands. Each VRE hierarchy level has one Master Command Center responsible for overseeing the operation and administration of all Client sites within its domain. The domain for a Master Command Center consists of all VRE components beneath it in the hierarchy tree.

Regional VRE hierarchies can also be grouped together to form a hierarchy of regions managed at a Command Center. In this way the regions can maintain administrative control over their domain while gaining access to the resources in other regions.

Currently, only these three levels of hierarchy are envisioned for the initial VRE system but we generalize the hierarchy to \( N \) levels in the Meta-Manager system to maintain scalability and modularity in the design. A three-level Meta-Manager hierarchy is depicted in Figure 4-1.

Each level of the Meta-Manager hierarchy will adhere to the following design rules:

1. Each Meta-Manager node shall route, store, and track patient cases introduced from within its domain.
2. A new case shall only be passed up the hierarchy tree if it cannot be adequately processed in the local domain.

![Figure 4-1: Meta-Manager 3-level Hierarchy](image-url)
3. In the absence of a higher controlling authority the current domain shall provide best-effort Meta-Manager functionality. This means the Meta-Manager shall do the best it can to provide normal VRE functionality given that it cannot defer cases to a higher authority and it cannot adequately process the case in the local domain.

4. A Meta-Manager command center shall have prior knowledge of all entities under its control. While this imposes a management burden on the administrators of the system it adds a level of security against false (spoofing) DIN-PACS sites gaining entry to the system.

5. The Meta-Manager shall send messages to the DIN-PACS networks to collect a case and physician knowledge base in a regional medical command domain.

6. The Meta-Manager shall provide a fail-safe operation for managing and controlling the flow of patient cases.

7. The Meta-Manager shall provide a graphical user interface (GUI) that allows operators and administrators to observe the status of the VRE and cases that are being routed.

8. The Meta-Manager shall provide the capability to interface CHCS messages to generic RIS messages via an application gateway function.

9. The Meta-Manager shall provide message transformation between two DIN-PACS nodes that contain variations in their CHCS messages and formats.

A Meta-Manager node that represents the Command Center for a given level of system hierarchy will be referred to as a Master Meta-Manager (MMM). Those entities that are being managed by a MMM are labeled Client Meta-Managers (CMM).

Keep in mind, the hierarchy being discussed here is a communications hierarchy and not a physical one. Each MTF site will have a single Meta-Manager node.
the VRE system may be acting as one or more levels in the Meta-Manager communications hierarchy. If a Meta-Manager node is configured as a Regional Meta-Manager it will establish communications links with all CMM nodes in the regional domain to form a communications hierarchy. Similarly, if a Meta-Manager node is configured as the Master Meta-Manager it will establish connections to the Regional CMM nodes in its domain.

4.2 REQUIREMENTS ANALYSIS

The requirements for the VRE system and the Meta-Manager include operational, functional, performance, and interface sections. The meaning of this requirements breakdown is:

Operational requirements describe the day-to-day operation and management of the Meta-Manager in the VRE. The operational requirements describe the user interfaces in the VRE, mission control functions, fault tolerant schemes, methods for replicating control functions, and security mechanisms. The mean-time-between-failure (MTBF) for the VRE Mission Control functions are specified. Reporting methods for the VRE operations and control are defined.

Functional requirements describe the processing functions in the VRE in terms of hierarchical object classes and methods. Algorithms for case routing, knowledge base collection, mobile agent cloning and routing, system security, databases, and user interfaces are specified. Functional requirements are presented as objects, object classes, object methods, and relationship between object classes. These specifications were used to develop a prototype of the Meta-Manager.

Performance requirements describe the parameters required in the Meta-Manager that determines processing speeds of algorithms, I/O bandwidth, and database sizes. From our model and simulation of the Meta-Manager, we can determine some of these parameters. Others are
derived as best-guess estimates. The parameters describe response times, network latency, and I/O throughput required in the Meta-Manager.

4.2.1 Operational Requirements

The Meta-Manager shall have a 3-level hierarchy as described in section 4.1. In this hierarchy the Client Meta-Manager (CMM) shall interface to the DIN-PACS networks in order to send and receive information on cases and case routing. The Regional Meta-Manager (RMM) shall be responsible for routing cases within Regional Medical Commands (RMCs). The Master Meta-Manager (MMM) shall serve the function of the operation control center for the VRE and shall route cases between RMCs, when necessary. The Client, Regional, and Master Meta-Manager shall provide fail-safe and persistent operation so the VRE services are available 24 hours a day, 7 days a week. DIN-PACS nodes shall be available in the same periods. Security services for VRE component access shall include control, authentication, communications, and secure storage. Network and DIN-PACS node availability shall be monitored by the Meta-Manager via the use of CORBA operating system agents between the CMM, RMM, and MMM nodes. The MM graphical user interface (GUI) shall provide operator monitoring and intervention of the case routing and the overall operation functions of the VRE system. The Meta-Manager operators and administrators shall monitor the operation of the VRE system and shall intervene only when case routing cannot be resolved. The operators and administrators shall have the ability to initialize and modify VRE and Meta-Manager parameters from the Meta-Manager GUI.
4.2.2 Functional Requirements

Most of the functional requirements of Virtual Radiology Environment is described using a Universal Modeling Language construct called *use case*. A use-case is defined as "a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transaction or sequence of interrelated events" [12].

Figure 4-2: Relationships between VRE nodes

Figure 4-2 shows a UML use-case representation of the Meta-Manager within the VRE system. A use-case diagram shows the relationships between major functions in the Meta-Manager. These use-cases are provided to the entities (actors) which use the Meta-Manager.
4.2.2.1 *Meta-Manager use-case actors:*

An actor in a UML use-case diagram is any entity outside of the Meta-Manager that shall use the Meta-Manager system. Actors can be functional processes, VRE components, or persons. The actors in this system are listed below with a concise description of their roles in the Meta-Manager.

**Local DIN-PACS**

The Local DIN-PACS actor is any DIN-PACS in the VRE system that shall initiate contact with the Meta-Manager. Contact shall be initiated when a Local DIN-PACS acquires a case to be processed or it has new CONNECTIVITY, SCHEDULE or SECURITY information to report to the Meta-Manager. The Local DIN-PACS actor has a *client* relationship with the Meta-Manager.

**Remote DIN-PACS**

The Remote DIN-PACS actor has a *server* relationship with the Meta-Manager. The Meta-Manager shall initiate contact with a Remote DIN-PACS when it has a case to be processed or it needs additional STATE, SCHEDULE or SECURITY information to manage the system. Most DIN-PACS networks within the system routinely act as both Remote and Local DIN-PACS actors during the normal operation of the system.

**Command Center**

The Command Center actor shall use the Meta-Manager to view the CONNECTIVITY (network connectivity and load), SCHEDULE, and SECURITY status of the VRE system. The Command Center shall modify the status of these parameters in the Meta-Manager. Each
Command Center shall support the use of the Meta-Manager system by individual actors responsible for auditing, scheduling, network management and security in each VRE region. A Command Center actor shall exist for each level of hierarchy in the Meta-Manager system.

**Operator**

The Operator actor handles the day-to-day operations of the Meta-Manager system. The Command Center shall permit the Operator to view the state of the VRE system. The Operator shall intervene when necessary to modify system parameters. In the event of a system failure, the Operator shall act to preserve system functionality while maintaining an audit trail as a permanent record of the system. The Operator shall work extensively with the SCHEDULE and CONNECTIVITY parameters to ensure the proper operation of the Meta-Manager system.

**Administrator**

The Administrator actor handles static system changes in the VRE system. The Administrator shall have the capability to add and remove VRE nodes within the administrative domain of the Command Center. Administrators shall be in charge of enforcing the security policy for their domain. It shall be the responsibility of the Administrator to keep the SECURITY and CONNECTIVITY parameters properly configured.

### 4.2.2.2 Meta-Manager use-cases:

Meta-Manager use-cases define the functionality of the Meta-Manager system. Each use-case scenario is described below. Each scenario is also documented with a flow of events. The flow of events for a use-case is a description of the events needed to accomplish the required behavior of each use case scenario.
Process New Case

This is the steady-state function of the Meta-Manager system. This function means, when a Meta-Manager node is up and running normally, that it is ready to process a new case generated within the VRE system. Local DIN-PACS actors with new cases to be processed refer the Case to the Meta-Manager system which shall determine the best Remote DIN-PACS to process the Case. The Meta-Manager system shall route the Case based on system parameters CONNECTIVITY and SCHEDULE and Case parameters (such as Preferred Physician, Urgency, Modality, and Body Part). The flow of events for processing a New Case in the Meta-Manager system is discussed later in this document.

Maintain VRE State

Command Centers shall use the Meta-Manager to view and modify VRE parameters. Client Command Centers track Meta-State information for their respective regions. Master Command Centers shall maintain the same Meta-State information but use Aggregate State from Client Command Centers. The flow of events for this use-case is discussed later in this document. Aggregate State is a generic class that contains any of the VRE parameters that can change with time. The major class specializations of Aggregate State include network connectivity and load (CONNECTIVITY), schedule (SCHEDULE), case load (CASE LOAD), and security (SECURITY).

Monitor VRE CONNECTIVITY

Operators at a Command Center shall view the VRE network connectivity and load. Administrators shall have the additional capability to modify the CONNECTIVITY parameters
for all elements being maintained at the control center. When a new site is added to the VRE network, an Administrator must specifically add the site before the Meta-Manager system can recognize it.

**Maintain VRE SCHEDULES**

The Meta-Manager shall collect a radiologist work SCHEDULE information from the local DIN-PACS and shall store it as the VRE system Meta-SCHEDULE. This schedule comes from DIN-PACS. An Operator at a Command Center shall view the Meta-SCHEDULE and identify low coverage areas by specialty or area. When a scheduling conflict is discovered, the scheduler shall contact the Local DIN-PACS to correct the problem. Only the Local DIN-PACS shall have write access to the SCHEDULES. When Local schedules change, an update message shall be sent to the Meta-Manager that will update the VRE Meta-SCHEDULE.

**Maintain VRE SECURITY**

Each Command Center acts on behalf of a Security Administrator to view and modify VRE SECURITY parameters. Within the Department of the Army, security policy is defined as the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information. As a system being designed for the U.S. Army, the VRE must comply with all federal and DoD requirements that apply to automated information systems and their security. Given that the VRE handles sensitive but unclassified (SBU) information, several security policies and requirements apply. Several Department of Defense regulations pertaining to automated information system security must be met by the VRE. Among the most applicable are

1) The Orange Book (DoD Trusted Computer System Evaluation Criteria),

3) AR 380-19 (Army Regulation 380-19: Information Systems Security), and


The Meta-Manager system shall implement the security policy for the VRE system [21]. The Meta-Manager SECURITY parameters specify who can use the system and limit the user’s activity to a predetermined set of use-cases. The use-case model of the system determines the target system requirements imposed by the users of the system. Objects in the VRE systems are derived from the use-case model. They will interact with the Meta-Manager component. Each relationship line from an actor to a Meta-Manager use case signifies an external interface to the system

Identification and Authentication

In general, identification and authentication is the requirement that each user identify himself and then prove his identity (e.g. UserlD and password). Identification is used to prevent unauthorized system access and to make users responsible for their actions on the system. Each action taken can be traced back to a particular person so that, if a security violation takes place, the responsible party can be determined. Requiring users to log on to the system with a unique ID is generally considered sufficient identification

Authentication is a key point in providing the security for teleradiology systems. Many of the other required security services are based upon the principal (human user) and system objects being authenticated by some means. Peer entity authentication is required so that users
are certain that the entity on the other end of the data transfer is who it claims to be. For instance, an end user sending an image to the data archive should be certain that the archive is on the other end of the transfer and not some entity masquerading as the archive. Similarly, the database should not divulge information unless it is sure, to some degree, that the data is being transferred to a user with the clearance to see that data. Security services such as access control and non-repudiation are based upon proper authentication of entities within the system. Peer entity authentication as well as user identification and authentication are required security services for teleradiology systems such as the VRE.

Access Control

Access control services provide a method by which access to a given system or the objects within the system can be controlled. Generally, authentication is used to prove the identity of the user and then some rules are applied to decide whether access should be granted or denied. Access control can be based on an individual's identity or on the group to which an individual belongs. It can be used to control access to system resources, specific objects in an object-oriented system, or certain types of data.

In teleradiology systems, access control should be used to limit access to system resources and to teleradiology data [22]. Because the transfer of digital images takes up a significant amount of network bandwidth, only authorized users should be allowed to ask for images to be transferred. The protection of teleradiology data is of extreme importance in the VRE Project. Patient privacy must be protected and only authorized personnel should have access to patient information (including medical data). Any unauthorized alteration or even viewing of patient data must be prevented. Access control can be used to protect the integrity and confidentiality of a patient's records. Integrity is the assurance that electronic data has not been unknowingly altered. Confidentiality ensures that an unauthorized party has not read the
information. Both integrity and confidentiality of information in a teleradiology system must be protected. Access control mechanisms can be used to help ensure their protection.

4.2.3 Performance Requirements

The general performance requirements of the VRE and Meta-Manager systems are:

1. The system shall facilitate diagnosis and retrieval of cases by radiologists without significant inconvenience to the radiologist. Inconveniences include tedious steps in the interfacing to the system and lengthy computing or transfer delays in the case diagnosis process.

2. The system shall support the concept of case urgency. Several case urgencies shall be recognized and satisfied by the system. An urgency designation of STAT denotes that the referring physician must route a case, have it diagnosed, and available for review within 30 minutes. An ASAP urgency designation implies a time constraint of maximum one hour but should be diagnosed in the shortest amount of time possible without interfering with STAT case requirements. Finally, the ROUTINE case urgency indicates that time is not the primary consideration in handling this case. The system shall be free to seek a diagnosis on ROUTINE cases based on other parameters such as preferred physician, modality or network state.

3. The Meta-Manager at each level shall contain a local disk capacity of at least 20 Gbytes of data storage.

4. The Meta-Manager databases at each level shall be persistent storage.

5. Client Meta-Managers shall receive processed messages (up to 5 Kbytes) from the DIN-PACS node in less than 500 milliseconds.

6. Each level of the Meta-Manager database shall be updated in less than 300 milliseconds.

7. Case routing decisions by the Meta-Manager shall be performed in less than 30 seconds.
8. Patient case headers, demographics, and images shall be routed to remote DIN-PACS for reading in less than 10 minutes.

9. The Meta-Manager shall have the capability to route 60 patient cases per minute.

10. The Meta-Manager at each level shall report the failure of communications link or nearest node to the mission control operators in less than 1 second of the occurrence of the failure.

11. The Meta-Manager at each level shall have the capability to perform program and data garbage collection.

12. The Meta-Manager levels shall operate without interruption by failures for 24 hours per day, 7 days a week.

13. Each level of the Meta-Manager shall not be down for maintenance for more than 1 hour.

14. The Meta-Manager at each level shall have a mean-time-between-failure (MTBF) of 166 hours in a 1 week time period.

15. Each Meta-Manager level shall not be down more than 2 hours.

16. The Meta-Manager at each level shall have the capability to reboot in less then 1 hour from the failure.

4.3 DOMAIN ANALYSIS

An analysis also itemizes the system domain, the key object classes in the Meta-Manager system. A domain analysis looks at which “concepts” should be handled by the system based on the results of the requirement analysis.

4.3.1 Static Domain Analysis

A class diagram is a model type, specifically a static, UML type. A class diagram describes the view of a system in terms of classes and relationships among the classes. To create a class
In this section, we specify the VRE domain object model, i.e., a structured model that shows the domain object classes and their relationships, from which specific instances of a VRE system can be instantiated. The model follows the classical UML approach in which an object hierarchy diagram describes the view of a system in terms of classes and relationships among the classes. To create the diagram, the key system classes have to be identified and described; then they can be related to one another by using the concept of associations.

The primary class relationships used in building the object model are aggregation (from a systems-theoretic perspective, this relation reflects the inverse of decomposition) and generalization. The aggregation relation demonstrates how a system component represented by an object can be decomposed into a set of sub-components (in the UML notation, this relation is depicted by the diamond symbol). The generalization relation captures the types of variants that are perceived for a component represented by a parent class. This relation is depicted in UML through the triangle symbol. These two relations afford us to construct a generic VRE object model in which we can perceive how systems are decomposed into subsystems and what instances of subsystems can be selected when instantiating a specific VRE environment. This instantiation process is called pruning [17]. Pruning consists in identifying such selections given the system requirements and constraints.

4.3.1.1 Discovering Static VRE Classes

The requirements analysis and the functional description of the Meta-Manager system presented in the previous section make it possible to identify a core set of candidate classes that will form the Meta-Manager system.
**Meta-Manager (MM):**

The Meta-Manager system is comprised of multiple Meta-Manager Nodes in a distributed computing environment. Each site that participates in the VRE system will have a Meta-Manager Node. At each node, System Administrators can access the VRE system to view or modify the system parameters via the Command Center GUI.

As shown in Figure 4-3, Meta-Manager Node Class is specialized into three classes: a) Master Meta-Manager, Regional Meta-Manager, and Client Meta-Manager.

![Figure 4-3: Meta-Manager Specialization](image)

**Client Meta-Manager:**

Client Meta-Manager is the lowest level of the Meta-Manager class hierarchy, which directly interfaces with the DIN-PACS to extract the DIN-PACS database parameters required for case routing decisions. The Client Meta-Manager to DIN-PACS interface is defined by an IDL test of parameters. This Client Meta-Manager reports to the higher level (i.e. Regional Meta-Manager) for routing the cases generated from the DIN-PACS it interfaces with.
Regional Meta-Manager:

Regional Meta-Manager is the middle level of the Meta-Manager class hierarchy, which is responsible for routing of the cases within a specific region or between DIN-PACS nodes. A region can contain many DIN-PACS sites and Client Meta-Managers. A Regional Meta-Manager thus acts as a master to the Client Meta-Manager and a servant to the higher level of Meta-Manager class, i.e. the Master-Meta-Manager.

Master Meta-Manager:

Master Meta-Manager is the topmost level of the Meta-Manager class hierarchy. When the case-routing threshold of a region exceeds, i.e. the Regional Meta-Manager is no more able to route the cases within its region, it asks this Master Meta-Manager to route the case for it. Master Meta-Manager sits on the top of each Regional Meta-Manager and thus can route the case within the two regions in such a scenario.

Case:

The Case class is an object class that represents all information pertinent to a radiology case in the Meta-Manager system. Case objects contain patient information and image sets in DICOM formats. Cases are generated, retrieved, diagnosed and routed in the VRE system.

DIN-PACS:

DIN-PACS is the source and destination of a patient case which is routed within the VRE system. This class is implemented by the specific vendor and thus is not a direct part of Meta-Manager environment, though it's the building block of the VRE system as a whole. The DIN-PACS nodes are responsible for providing case object information to the Client Meta-Manager.
System Administrator:

This is the actor class, which represents the operators and administrators for the VRE system. An operator and/or administrator, are the humans who constantly monitor the VRE system for its STATE and occasionally adjust the system parameters.

Figure 4-4: VRE Object Model

Command Center GUI:

Command Center GUI is the graphical user interface that an operator or administrator uses to interact with the VRE Meta-Manager node. It shows the current STATE of the VRE system and allows the administrator/operator to change the system parameters, if required.

The main object classes identified in Figure 4-3 are related through the association relations shown in the following Figure 4-4. The diagram depicted in this figure is the object model that
can underlie an instance of VRE. In the following section, we demonstrate the transition from this domain model to a specific VRE design object model.

4.3.1.2 Instantiating VRE Object Model

This section illustrates the process of pruning the VRE Object Model. Recall that several instances of an object class can be generated from a class object. Thus, we can generate as many Meta-Manager class instances as it may be required by the project's specification. For our project at hand, BAMC is to serve as the highest level manager. Thus, we select the Master Meta-Manager Object as the Meta-manager that servers as the root level manager in our design hierarchy. This is depicted in Figure 4-5 as Master Meta-Manager (BAMC). This manager is to serve in a supervisory mode for two regional mangers (which act in a Client model, i.e., the Regional Meta-Manager 1 (Madigan AMC) and Regional Meta-Manager 2 (Tripler AMC). Notice that the Client Meta-Managers have not been assigned to any specific managers yet.

![Figure 4-5: Pruning Stage 1](image)

Each regional manager acts in a supervisory mode to client Meta-Managers. To reflect this in the object hierarchy, we convert the Client Meta-Manager(s) specialization to a decomposition relation as shown in Figure 4-6. The resulting object model specifies an instance of the
underlying design structure. In the ensuing sections, we discuss the dynamic and functional aspects of our design. Figure 4-7 shows a more complete view of final instantiation of VRE Meta-Managers after two pruning stages.

![Diagram of VRE Meta-Managers after two pruning stages]

Figure 4-6: Pruning Stage 2

One purpose of the class diagram is to define a foundation for other diagrams where other aspects of the system are shown, such as the states of the objects and collaboration between objects shown in dynamic diagrams.
4.3.2 Dynamic Domain Analysis

Our static domain analysis introduced a set of candidate classes that will exist in our system and their relationships with other static classes (Figure 4-3). However, this is only one view of our proposed system. Having analyzed the functional requirements of our system through a Use-
Case diagram, we can now expand our domain analysis to capture a dynamic view of the VRE system.

4.3.2.1 System Initialization

The VRE system is a dynamic, multi-entity system with real-time events affecting membership in the system. The class diagram shows the generic entities in our Meta-Manager system.

Figure 4-8: Meta-Manager Entity Initialization State Diagram

When a Meta-Manager entity is prepared to run it does not know where it resides in the system hierarchy. Individual entities in the system should initialize as independent systems and not depend on a higher controlling authority. Once entities reach a self-sufficient state they may be contacted by a higher entity in the Meta-Manager hierarchy. The entity states during initialization are shown in Figure 4-8 followed by a brief discussion about each state.
System Login

All users on systems running Meta-Manager entities are verified at the local station using the traditional Level C-2 user authentication. System administrators set directory permissions to facilitate Meta-Manager use by those with prior authorization to use the Meta-Manager system. The local system administrator in most cases will also serve as the Security Administrator for local systems running Meta-Manager entities. Users are automatically logged out after a predetermined period of inactivity. See the Meta-Manager Security Policy for additional information [Jara 98].

Meta-Manager User Authentication

Any of the VRE system actors have access to run the Meta-Manager Command Control software. An additional security check is used to control access to the various functions inside of the Meta-Manager system. For example, only the security administrator is given access to view and modify the encryption keys. Each of the VRE actors at a Meta-Manager site have separate accounts within the Meta-Manager system and are given access to Meta-Manager features according to their account privileges.

Edit Mode

Once a Meta-Manager user has been authenticated they can view and edit system parameters according to the access control permissions for that user.

- List of Client Meta-Manager nodes in the domain of this node: Administrator
- Private Key and Keys of CMM nodes: Security Administrator
- Local Schedule of Radiologist by hour: Scheduler
The Meta-Manager administrator for that site has the ability to take the local node on and off-line with respect to Meta-Manager case routing operation.

Initialization

When a local Meta-Manager node goes on-line it prepares itself for processing cases locally and as part of the global Meta-Manager system. The steps taken to initialize a local Meta-Manager entity are:

- Traverse Client List to Determine Connectivity
- Build Meta-State for the local node and all Clients in the domain
- Build Patient Database (DB) for all local and Client Cases
- Clock Synchronization

When the initialization is complete the node begins operation in Master Mode.

Master Mode

As Master, a Meta-Manager node is the final authority in all routing requests it receives for new cases. A Master maintains the Meta-STATE for all Clients on its Client list. Based on the Meta-State it uses Clients on the list to process cases and retrieve patient history cases. A Master will dynamically become a Client as new levels of hierarchy are added to the system. When a verified message is received from a new Master node the local node enters Client mode.

Client Mode

A Client Meta-Manager is available as a resource for the Master. It receives new cases from the Master to be processed on behalf of other nodes in the system. A Client is also available to route cases within the local domain to other nodes needing patient history information. In return
for this service, a Client Meta-Manager can ask the Master to find a more suitable Client to process new cases. Before asking the Master to route a new case the Client makes a reasonable attempt to find a suitable reader within its own domain. A Master may deny requests for case routing and require the Client to process the case to the best of its abilities. Given the hierarchical nature of our system, all Client Meta-Manager will still act as Master to Client nodes within their domain. Only the Meta-Manager node at the highest level of the hierarchy will remain in Master mode indefinitely.

The fail-safe operation of the system is also considered in the initialization state diagram. A Meta-Manager Node in Client mode may have to return to Master mode in the event that communications are lost with its Master. This is an important feature since portions of the VRE network that get isolated from the rest of the network should not lose the functionality of the Meta-Manager system.

Dynamic Class Discovery

The Rational Objectory Process advocates class discovery by partitioning the system being developed into three types of classes – control, entity and boundary [1].

VRE Control Classes

Control classes model sequencing behavior specific to one or more use cases. Control classes coordinate the events needed to realize the behavior specified in the use case.

To discover control classes in the VRE system we create a class for each actor::use case pair in our VRE Use-Case diagram. (Error! Reference source not found.) The resulting control classes are listed below. The actor::use case pair that generated the class is shown in parenthesis next to the class name.
• **Submit_Case** *(Local DIN-PACS :: Process New Case)*: Handles the interaction with the Local DIN-PACS when a new Case is submitted to the system.

• **Receive_Case** *(Remote DIN-PACS :: Process New Case)*: Guides the interaction with a Remote DIN-PACS when the Meta-Manager submits a case for processing.

• **Authenticate_User** *(Command Center :: User Authentication)*: Protocol interface between the Command Center and Authentication Server used to authenticate users at Command Centers.

• **Monitor_State** *(Command Center :: Monitor VRE State)*: Protocol between the Command Center and the Meta-Manager system used to query the state of the system for purposes of viewing the state at the command center. This class is a likely candidate for merging with the Query State class because it has similar functionality.

• **Update_State** *(Local DIN-PACS :: Monitor VRE State)*: Protocol used to send a state update from a Client Meta-Manager to a Master.

• **Query_State** *(Remote DIN-PACS :: Monitor VRE State)*: Protocol used by a Master to query the state of a Client Meta-Manager.

**VRE Entity Classes**

An entity class models information and associated behavior that is generally long lived. This type of class may reflect a real-world entity, or it may be needed to perform tasks internal to the system.

The first step to discovering entity classes is to examine the responsibilities (*i.e.* what the system must do) outlined in the flow of events for each use case. These event flows were presented in the previous section during the functional requirements analysis. If we examine the
noun phrases used to describe the functional requirements of the system we come up with many of the classes that were discovered as control classes and the new entity classes listed below.

- **Route_Case**: This internal class implements the case routing procedure of the Meta-Manager system.
- **Retrieve_Case**: Internal class that handles Case retrieval from a Remote DIN-PACS.
- **Assign_Meta-Case_Number**: Internal class that implements the global Meta-Case number scheme across the Meta-Manager system.
- **Meta-Manager_Node**: A Meta-Manager_Node is a generic class that represents the Meta-Manager component at each site in the system. This class was specialized into three derived classes in the domain analysis: Client_Meta-Manager, Regional_Meta-Manager and Master_Meta-Manager.
- **Master_Interface**: The Master_Interface defines the internal interface Meta-Manager_Nodes use to communicate with Client Meta-Manager nodes.
- **Client_Interface**: The Client_Interface defines the internal interface a Meta-Manager_Node in Client mode uses to communicate with its Master.
- **Meta-Manager-CC_Interface**: The Meta-Manager-CC_Interface defines the internal interface a Meta-Manager_Node uses to communicate with a Command Center.
- **CC-Meta-Manager_Interface**: The CC-Meta-Manager_Interface defines the internal interface a Command_Center uses to communicate with the Meta-Manager system.
- **Case**: The Case class contains information needed to track, route, diagnosis, store and retrieve Cases in the Meta-Manager system. The Case class is a superset of the case used by DIN-PACS networks. It includes all information needed by the DIN-PACS plus Meta-
Manager specific information like Meta-Case Number and the location of patient history stored at other DIN-PACS sites.

- **Patient:** The Patient class includes information that uniquely identifies each patient in the Meta-Manager system. The Meta-Manager system tracks patients so that Cases can be found by patient ID.

- **Radiologist:** The Radiology class contains information about the capabilities of each radiologist in the Meta-Manager system. The Radiologist is the resource being maximized by the Meta-Manager system so this class will be accessed for compiling system reports or for routing cases.

- **Schedule:** The Schedule class is a data class that tracks specialty coverage in the system by mapping individual radiologist schedules to a system schedule. The Schedule class can be queried to evaluate coverage at any given time in the Meta-Manager system.

- **Connection:** The Connection class holds state information for each logical connection in the Meta-Manager system. Connections have properties that will be used by the Meta-Manager system to make routing and configuration decisions.

**VRE Boundary Classes**

Boundary Classes handle the communication between the system surroundings and the inside of the system. They provide the interface to a user or another system (*i.e.* the interface to an actor.)

Each physical actor/system interface is assigned a boundary class. These classes really represent the goals of the first elaboration phase of our design since we identified the external interfaces as the highest priority for our design. For the VRE system these classes are:
• Comand_Center_GUI: The Command_Center_GUI enables the system operators (external actors) to view the internal state of the system and make changes from a console at the Command_Center.

• DIN-PACS_Interface: This class defines the interface between the Meta-Manager system and the proprietary DIN-PACS network. This class maps the relationships between DIN-PACS Networks and the Meta-Manager system into a well-defined set of methods. The relationships were defined earlier as control classes, specifically the Submit_Case, Receive_Case, Update_State, and Query_State classes.

4.3.2.2 Integrating Classes with the Static Domain Analysis

We originally used a static domain analysis to discover a set of candidate classes in our system. From figures 7 and 8 we see that many of the classes in the domain analysis were also discovered following the ROP class discovery process on the use-case diagram. The only unique classes come from the specialization of the Meta-Manager_Node class into derived classes. This discrepancy can be expected since the domain analysis captures the hierarchy in the system and the use-case diagram treats the Meta-Manager system as a flat entity.

4.3.3 Case Routing Algorithms

The dynamic models of each component of the VRE system (DIN-PACS, Client Meta-Manager, Regional Meta-Manager and Master Meta-Manager) are shown in the following diagrams. Initially, DIN PACS node receives a new case to be examined by any doctors in the VRE. It then delivers the header information of the new case to its Client Meta-Manager, which then decides whether to forward the case to the upper level Meta-Managers. If the Client Meta-
Manager decides that the case can be read locally, returns a destination address for DIN PACS to forward both the header information and the images associated with the case.

4.3.3.1 DIN-PACS:

As shown in Figure 4-9, a DIN-PACS can be in a 'receiving' mode or 'generating' mode. That means, it can either be getting a patient case from other DIN-PACS or it would be generating a patient case for diagnosis, respectively. Consider the first scenario, where the DIN-PACS under consideration is receiving a case from outside (i.e., from a remote DIN-PACS). As shown in the figure, in this mode, the DIN-PACS (i.e., Local DIN-PACS) receives a case and routes it within the system using its own routing decision making process. It then continues processing the diagnosis of the case unless it is cancelled by the Client Meta-Manager monitoring the current state of the processed case. If the case is cancelled, the DIN-PACS updates the database for cancellation of the case and waits for the new case. If the case is not cancelled, the DIN-PACS completes processing the case and sends back the transcription of the diagnosis to the remote DIN-PACS. It then updates its database for the case that was diagnosed. It then waits for the next event, which could either be 'receiving a case' or 'generating a case'.

In the second scenario, the DIN-PACS generates a new patient case. By default, it notifies the Client Meta-Manager about the generated case. Here, it just sends the case-header to the Client Meta-Manager, which then checks for the possibility of routing the case within the same Local DIN-PACS, by checking the DIN-PACS threshold. The threshold for a DIN-PACS is the limit beyond which the DIN-PACS is not able to process a case on its own. This threshold depends on the radiologist availability and the current status of their work-schedules for that DIN-PACS. The threshold can also be site configurable.
If the Client Meta-Manager finds that the threshold is not exceeded, it tells the DIN-PACS to diagnose the case locally and the DIN-PACS proceeds in the same way it does in the first scenario, except that it stores the results of the diagnosis locally.

If the Client Meta-Manager finds that the threshold is exceeded and the DIN-PACS cannot process the case locally, it contacts its superior node, the Regional Meta-Manager, to make a routing decision for it. Once the DIN-PACS gets the routing decision from the Client
Meta-Manager, it retrieves the patient case from the archive, updates the database as 'remote read' for that case and sends the patient case, i.e., the header with images, to the address of the remote DIN-PACS it got from the Client Meta-Manager. It then waits for the next set of events. The cycle then continues.

Once a patient case arrives at a remote DIN-PACS for diagnosis, the radiologist may require viewing prior patient history and image sets that were not routed to the remote DIN-PACS. In this case, the radiologist may request through the Client Meta-Manager this additional information to be sent immediately.

4.3.3.2 Client Meta-Manager:

The Client Meta-Manager stages are shown in Figure 4-10. The CMM always waits for the notification from the DIN-PACS about the generated cases. Once it gets such notification from the DIN-PACS, it updates its database and checks for the threshold for diagnosis. If it finds that the local DIN-PACS is able to route the case within itself (i.e. the local threshold is not exceeded), then it supplies that information to the DIN-PACS (i.e., asking the DIN-PACS to route it within the same system). It again updates the database, stamping it as a local read and waits for the next notification from the DIN-PACS.

If the threshold is exceeded, the CMM notifies the Regional Meta-Manager and waits for the routing information to be supplied by the Regional Meta-Manager. After it gets the routing information from the Regional Meta-Manager, it supplies that to the DIN-PACS, updates the local database, stamping it as a remote read, and waits for the next notification from the DIN-PACS. To acquire the updated radiologists' work-schedules from the DIN-PACS, CMM polls the DIN-PACS at regular intervals. It passes this updated information to the Regional Meta-Manager when it polls the CMM.
4.3.3.3 Regional Meta-Manager:

The Regional Meta-Manager (RMM) stages are shown in Figure 4-11. The operation of the RMM is somewhat similar to that of the CMM. The RMM waits for the notification from the CMM. Once the notification is received, it understands that the requesting CMM is not able to find the correct reader DIN-PACS site for the case, so it checks for the thresholds of the all DIN-PACS sites within that particular region. If it finds the best reader within the region (i.e., if the

Figure 4-10: Case Routing Algorithm in Client Meta-Manager
regional threshold is not exceeded), it supplies that information to the CMM and waits for the next notification.

If the regional threshold is exceeded, it contacts the upper level of hierarchy, i.e. the Master Meta-Manager, to make the routing decision. Once it gets the routing decision from the Master Meta-Manager, it passes it to the CMM after updating the database.

In a manner similar to the CMM, the RMM needs to be updated about the changes in the work-schedules of the radiologists in the DIN-PACS. Therefore, it polls the CMM at regular intervals in order to update its own database and then it notifies the Master Meta-Manager of this updated status.

![Figure 4-11: Case Routing Algorithm in Regional Meta-Manager](image-url)
4.3.3.4 Master Meta-Manager:

The stages of the Master Meta-Manager (MMM) are shown in Figure 17. The MMM waits for the notification from the RMM. Upon receiving the notification from the RMM, it understands that the RMM is not able to route the case within that region. Thus, the MMM checks all the regions to find the best possible reader for that case. Once it finds the best reader from some other region (or it might be from the same region, if the work-schedules are updated in that time-interval), it supplies that routing information to the RMM requesting it. It updates its database stamping the case as read, and then listens to the next request from the RMM.

Figure 4-12: Case Routing Algorithm in Master Meta-Manager
4.3.4 Time Sequence Diagrams for Use-Cases

The above Case Routing distributed algorithm gives a high-level view of the actors' behavior in the VRE System. To give more detailed view of some specific use-cases, sequence-charts are presented in the following section.

The following sequence diagrams combine instances of the classes in the class diagram and the actors in the use-case diagram to present a clear sequence of class interaction for each use-case.

4.3.4.1 The Process New Case Use-case

There are two distinct scenarios possible when a Local DIN-PACS initiates the Process New Case sequence. The most common case occurs when the Local DIN-PACS actor has the resources to diagnose the case itself. In this scenario the Local Client Meta-Manager notifies the Regional Meta-Manager that a case is being read.

The Regional Meta-Manager responds with a list of any previous cases in the system that have historical relevance to this case. The Regional Meta-Manager also sends Forward (Case) messages to the Remote DIN-PACS that archive the relevant cases. When relevant cases have been retrieved from the Remote archives, they are sent directly to the Local DIN-PACS so that historical cases can be referenced in the diagnosis. The sequence diagram corresponding to this scenario is shown in Figure 4-13.
Figure 4-13: Process New Case Sequence Diagram – Local Read Scenario

The other common scenario of the Process New Case sequence is shown in Figure 4-14. In this scenario, the case parameters require that a Remote DIN-PACS site diagnose the case.
Figure 4-14: Process New Case Sequence Diagram – Remote Read Scenario

The Local Client Meta-Manager node shall request a reader for the case from the Regional Meta-Manager. When a reader is found the Regional Meta-Manager sends the location of the reading DIN-PACS to the Local Client Meta-Manager which forwards the new case to the
reader. Historical cases shall also be forwarded to the reading DIN-PACS by Archive DIN-PACS. Finally, a text diagnosis shall be returned to the Local DIN-PACS and shall be added to the permanent case record.

The sequence diagrams in the above figures assume that the Master Meta-Manager is able to find a suitable reading location for the new Case and that the Case is processed successfully by the diagnosing radiologist. Given the dynamic nature of the system and the need for fail-safe operations as described previously, it can not always be assumed that things will go according to plan. It may be necessary for the Meta-Manager system to find an alternate means of processing a Case after the Case has been assigned to a radiologist. This situation is called Case Recall and it occurs when either of two events takes place:

1. The Master determines that the reading radiologist will not be able to adequately process the Case. This could occur because the network state changes, the radiologist logs off the DIN-PACS network or simply because the radiologist gets behind in his Case reading. The messages used to recall a case in this scenario are shown in Figure 4-15.

2. The Master loses communications with the Client Meta-Manager Node that was assigned to process a Case. This would occur if the communications network fails or if an Administrator at the Client Node takes the Client off-line. Figure 4-16 shows the messages needed to recall a Case in this scenario.
Figure 4-15: Process New Case Sequence Diagram – Slow Case Recall Scenario
Figure 4-16: Process New Case Sequence Diagram – Lost Client Recall Scenario
4.3.4.2 The Monitor VRE State Use-case

Monitoring State parameters in the Meta-Manager system requires two-way communication between Master and the Client Meta-Manager Nodes in its domain. The Master is responsible for tracking CONNECTIVITY for its domain. A periodic Ping message is sent to each active Client as shown in Figure 4-17.

![Sequence Diagram](image)

**Figure 4-17: Monitor State Sequence Diagram – Master Ping Scenario**

When a Client Meta-Manager receives a Ping message from the Master, it responds with a message containing a list of radiologists on-line in the Client's domain, and an estimated time before the radiologist can process a new Case. The Master updates its Meta-CONNECTIVITY parameters based on the response time of the message. The radiologists list and time to read information is used by the Master to update the Meta-CASE LOAD parameters.

Periodic polling from the Master Meta-Manager is well suited to monitor CONNECTIVITY and CASE LOAD but often Client Nodes have State information that needs to be added the Meta-State of the system. In this scenario the Client initiates communications with the Master as shown in Figure 4-18.
State updates are necessary when an Operator or Administrator at a Client Node modifies a State parameter. Updates are necessary when an event drastically alters the CONNECTIVITY or CASE LOAD State. When a threshold is crossed on these parameters the Client generates an Update message rather than wait for the next Ping from the Master.

When State monitoring messages fail to reach their intended destination the integrity of the system is compromised. Because of the importance of correctly monitoring state in the Meta-Manager and to reduce traffic on the network, the State Monitoring messages with fail-safe operation of the Meta-Manager system have been integrated.

Masters are in charge of tracking CONNECTIVITY with all of the Client Meta-Managers in their domain. The latency and contents of the Ping message response measure CONNECTIVITY. The Master will set a timer when the Ping message is sent and assume the
Ping message was not received at the Client only after the timer expires. Since packet switched networks do experience lost messages and the system is not limited to specific technologies, it is assumed that Ping messages can be dropped by the network during normal operation up to a lost message threshold value. After this threshold is crossed the Master assumes that the Client is no longer reachable.

This scenario is depicted in Figure 4-19. Once the Client is determined to be unreachable, Cases being processed by the Client on behalf of other Clients are recalled and assigned to new Clients.

Client Meta-Managers also monitor the CONNECTIVITY of their Master so that they can recover when a Master fails. Clients expect a Ping message from their Master at predetermined intervals. When this Ping does not arrive within a reasonable latency window the Client assumes the Ping message has been lost and sends its own Ping message to the Master. If the Master does not respond to the message, another Ping message is sent until the lost message threshold is crossed. At that point the Client assumes that the Master is unreachable and becomes the Master of its domain. This scenario is displayed in the sequence diagram of Figure 4-20.
Figure 4-19: Monitor State Sequence Diagram – Lost Client Scenario
Figure 4-20: Monitor State Sequence Diagram – Lost Master Scenario
4.4 INTEGRATING ORB AND UML

When we integrate a generic Object Request Broker (ORB) architecture into the UML system architecture we outlined so far, we have a better picture of how the Meta-Manager system will look in a distributed computing environment, as shown in Figure 4-21.

Since the packages represent distributed elements in our system, each package will have an IDL for each interface it will support. These IDL’s have been added as packages within each of the existing packages.

![Figure 4-21: UML package abstractions with CORBA IDL packages](image)

The Command Center package now has an IDL package that will interface with Meta-Manager Nodes. In the case of the Meta-Manager Node package, four IDL’s are needed: one supporting an interface to a DIN-PACS network, one interfacing with the Command Center, and
one each for communications with other Meta-Manager Nodes up and down the Meta-Manager hierarchy tree.

A special package is needed to represent the IDL that a DIN-PACS Network will use to interface with the Meta-Manager Nodes. This package is unique because it represents a vendor specific component of the Meta-Manager system. Individual DIN-PACS vendors will need to develop proprietary IDLs to interface with the Meta-Manager system. This gives DIN-PACS vendors additional flexibility in how they implement the interaction to the Meta-Manager system but it also implies that the vendors are proficient in developing IDL's for distributed computing environments. From the perspective of the Meta-Manager design, proprietary IDL's allows us to support any DIN-PACS network that conforms to the IDL specification.

Figure 4-22: UML package abstractions of specialized Meta-Manager Node packages
Note that there is still a looping dependency in Figure 4-22 between the IDL packages in the Meta-Manager Node. The loop signifies that Meta-Manager Node instances will depend on each other in the system. We can go one step further and say that Meta-Manager Nodes will depend on Meta-Manager Nodes above and below themselves in the Meta-Manager hierarchy.

The dependencies between Meta-Manager Node packages is more clearly shown in, where instances of Meta-Manager Node packages exist at different levels in the communications hierarchy.
CHAPTER 5: VRE PILOT DESIGN AND IMPLEMENTATION

This chapter covers detail implementation of the pilot application for the VRE, beginning with the system requirements, to system architecture, to the database subsystem, and some workaround required that are not covered in the initial specifications.

5.1 DATA

The data used for the pilot implementation of VRE was obtained from the 1997 RSNA DICOM Image Set. RSNA is a society whose purpose is to promote and develop the highest standards of radiology and related sciences through education and research. It seeks to provide radiologists and allied health scientists with educational programs and materials of the highest quality, and to constantly improve the content and value of these educational activities. The Society seeks to promote research in all aspects of Radiology and related sciences, including basic clinical research in the promotion of quality healthcare.

The cases consist of 35 patients and 7 modalities that came from 9 Vendor participating in the Society. There are roughly 150 DICOM images buried within these cases. Table 5.3 shows some of the descriptions of the cases.

In addition to the adaptation of RSNA data as its data source, this project followed the DICOM standards for database structures for its tables. Table 5.4 - Table 5.7 are some examples of DICOM standards of patient data. As seen in the tables, DICOM decomposes patient information into four tables: Patient Relationship, Patient Identification, Patient Demographic, and Patient Medical modules.

The uncompressed message sizes and average number of images per case for the various radiological image modalities are shown below. It is assumed that a 12-bit pixel is contained in a
2-byte (byte-packing) storage for ease of storage, display, and manipulation, rather than in a 1.5-byte (bit-packing) which is done for storage compaction.

**Table 5.1: Bandwidth Requirements of Telemedicine Services**

<table>
<thead>
<tr>
<th>Services and Operations</th>
<th>Type</th>
<th>Data Structure</th>
<th>Delivery Time</th>
<th>Required Bandwidth (Kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vital</td>
<td>BP</td>
<td>2 bytes</td>
<td>1 sec</td>
<td>3.2</td>
</tr>
<tr>
<td>Sign</td>
<td>HR</td>
<td>2 bytes</td>
<td>1 sec</td>
<td>3.2</td>
</tr>
<tr>
<td>Transfer</td>
<td>TEM</td>
<td>2 bytes</td>
<td>1 sec</td>
<td>3.2</td>
</tr>
<tr>
<td>ECG</td>
<td></td>
<td>2 Kbytes</td>
<td>1 sec</td>
<td>24.0</td>
</tr>
<tr>
<td>Store</td>
<td>CR</td>
<td>512 x 512 x 8</td>
<td>30 sec</td>
<td>17.48</td>
</tr>
<tr>
<td>Forward</td>
<td>CT</td>
<td>1K x 1K x 8</td>
<td>30 sec</td>
<td>66.67</td>
</tr>
<tr>
<td>Static Image</td>
<td>X-RAY</td>
<td>2K x 2K x 12</td>
<td>30 sec</td>
<td>400.0</td>
</tr>
<tr>
<td>Video</td>
<td>Digital</td>
<td>320 x 280 x 24 MPEG</td>
<td>30 frame/sec</td>
<td>45,200.0</td>
</tr>
<tr>
<td>Conferencing</td>
<td>Analog</td>
<td>-</td>
<td>-</td>
<td>8 MHz channel</td>
</tr>
<tr>
<td>Real-Time</td>
<td>Analog</td>
<td>-</td>
<td>-</td>
<td>1 channel @ 3-8 KHz</td>
</tr>
<tr>
<td>Voice</td>
<td>Digital</td>
<td>-</td>
<td>-</td>
<td>2 channels @ 64 Kbps</td>
</tr>
<tr>
<td>Patient Information</td>
<td>Text</td>
<td>4 Kbytes</td>
<td>30 sec</td>
<td>1.067</td>
</tr>
<tr>
<td>On-Line Med. Info</td>
<td>Hypertext</td>
<td>1 Mbytes</td>
<td>30 sec</td>
<td>267.0</td>
</tr>
<tr>
<td>Image Annotation &amp; Pointing</td>
<td>Real-Time Date</td>
<td>400 bytes</td>
<td>200 millisecond</td>
<td>16.0</td>
</tr>
<tr>
<td>RCD</td>
<td>Multimedia</td>
<td>8 Mbytes of image 128 Kbytes of voice 800 bytes annotation</td>
<td>120 sec. 1 sec. 1 sec.</td>
<td>197.87</td>
</tr>
</tbody>
</table>

**Table 5.2: Modality Sizes of Telemedicine Services**

<table>
<thead>
<tr>
<th>Modality</th>
<th>Image Size</th>
<th>Images per Case</th>
<th>Case Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Ray (CR)</td>
<td>10 Mbytes</td>
<td>3</td>
<td>30 Mbytes</td>
</tr>
<tr>
<td>CT Scan (512x512x12)</td>
<td>524,288 bytes</td>
<td>40</td>
<td>20.97 Mbytes</td>
</tr>
<tr>
<td>MRI (256x256x12)</td>
<td>131,072 bytes</td>
<td>80</td>
<td>10.49 Mbytes</td>
</tr>
<tr>
<td>US (512x512x12)</td>
<td>524,288 bytes</td>
<td>6</td>
<td>3.15 Mbytes</td>
</tr>
</tbody>
</table>
Table 5.3: Some of 1997 RSNA DICOM Files Used

<table>
<thead>
<tr>
<th>Case ID</th>
<th>Vendor</th>
<th>Modality</th>
<th>Examination</th>
<th>History</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>96-01</td>
<td>Picker</td>
<td>NM</td>
<td>Renal Scan</td>
<td>Question renal function abnormality. Prior renal calculi.</td>
<td>Following the administration of Tc-99m DTPA dynamic renal scan images were acquired.</td>
</tr>
<tr>
<td>96-02</td>
<td>Picker</td>
<td>NM</td>
<td>Total body bone scan</td>
<td>65 year old man recently diagnosed with prostate carcinoma</td>
<td>Following the administration of Tc-99m MDP total body bone scan images were acquired.</td>
</tr>
<tr>
<td>960-3</td>
<td>Picker</td>
<td>CT</td>
<td>CT: Spiral Angiography of the Thorax</td>
<td>46-year-old male presented to the emergency room with parasternal and epigastric pain. A chest X-ray showed widening of the mediastinum. R/O thoracic aneurysm.</td>
<td></td>
</tr>
<tr>
<td>96-04</td>
<td>Picker</td>
<td>MR</td>
<td>MRI: Knee</td>
<td>16-year-old with right knee pain after an injury playing basketball</td>
<td></td>
</tr>
<tr>
<td>96-05</td>
<td>Picker</td>
<td>MR</td>
<td>MRI - Cervical Spine</td>
<td>Right arm pain following car accident</td>
<td></td>
</tr>
<tr>
<td>96-06</td>
<td>Picker</td>
<td>MR</td>
<td>MRI - L-spine</td>
<td>Low back and bilateral hip pain which goes down into the right hip.</td>
<td></td>
</tr>
<tr>
<td>96-07</td>
<td>Kodak</td>
<td>CR</td>
<td>Portable chest x-ray dated 07/30/95 at 0603 hours</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96-08</td>
<td>Kodak</td>
<td>CR</td>
<td>Portable chest x-ray</td>
<td>Six month old, with a history of tetralogy of Fallot.</td>
<td></td>
</tr>
<tr>
<td>96-09</td>
<td>Acuson</td>
<td>US</td>
<td>Ultrasound: Right lower quadrant</td>
<td>Right lower quadrant pain r/o appendicitis</td>
<td></td>
</tr>
</tbody>
</table>
Table 5.4: CR Image IOD Modules

<table>
<thead>
<tr>
<th>IE</th>
<th>Module</th>
<th>Reference</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient</td>
<td>Patient</td>
<td>C.7.1.1</td>
<td>M</td>
</tr>
<tr>
<td>Study</td>
<td>General Study</td>
<td>C.7.2.1</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Patient Study</td>
<td>C.7.2.2</td>
<td>U</td>
</tr>
<tr>
<td>Series</td>
<td>General Series</td>
<td>C.7.3.1</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>CR Series</td>
<td>C.8.1.1</td>
<td>M</td>
</tr>
<tr>
<td>Equipment</td>
<td>General Equipment</td>
<td>C.7.5.1</td>
<td>M</td>
</tr>
<tr>
<td>Image</td>
<td>General Image</td>
<td>C.7.6.1</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Image Pixel</td>
<td>C.7.6.3</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Contrast/bolus</td>
<td>C.7.6.5</td>
<td>C Required if contrast media was used in this image</td>
</tr>
<tr>
<td></td>
<td>CR Image</td>
<td>C.8.1.2</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Overlay Plane</td>
<td>C.9.2</td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>Curve</td>
<td>C.10.2</td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>Modality LUT</td>
<td>C.11.1</td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>VOI LUT</td>
<td>C.11.2</td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>SOP Common</td>
<td>C.12.1</td>
<td>M</td>
</tr>
</tbody>
</table>

Table 5.5: Patient Identification Module Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Tag</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient’s Name</td>
<td>(0010,0010)</td>
<td>Patient’s full legal name</td>
</tr>
<tr>
<td>Patient ID</td>
<td>(0010,0020)</td>
<td>Primary hospital identification number or code for the patient</td>
</tr>
<tr>
<td>Issuer of Patient ID</td>
<td>(0010,0021)</td>
<td>Name of healthcare provider which issued the Patient ID</td>
</tr>
<tr>
<td>Other Patient Ids</td>
<td>(0010,1000)</td>
<td>Other identification numbers or codes used to identify the patient</td>
</tr>
<tr>
<td>Other Patient Names</td>
<td>(0010,1001)</td>
<td>Other names used to identify the patient</td>
</tr>
<tr>
<td>Patient’s Birth Name</td>
<td>(0010,1005)</td>
<td>Patient’s birth name</td>
</tr>
<tr>
<td>Patient’s Mother’s Birth Name</td>
<td>(0010,1060)</td>
<td>Birth name of patient’s mother</td>
</tr>
<tr>
<td>Medical Record Locator</td>
<td>(0010,1090)</td>
<td>An identifier used to find the patient’s existing medical record (e.g. film jacket)</td>
</tr>
</tbody>
</table>
### Table 5.6: Patient Relationship Module Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Tag</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referenced Visit Sequence</td>
<td>(0008,1125)</td>
<td>Uniquely identifies the Visit SOP Instances associated with this Patient SOP Instance.</td>
</tr>
<tr>
<td>&gt;Referenced SOP Class UID</td>
<td>(0008,1150)</td>
<td>Uniquely identifies the referenced SOP Class.</td>
</tr>
<tr>
<td>&gt;Referenced SOP Instance UID</td>
<td>(0008,1155)</td>
<td>Uniquely identifies the referenced SOP Instance.</td>
</tr>
<tr>
<td>Referenced Study Sequence</td>
<td>(0008,1110)</td>
<td>Uniquely identifies the Study SOP Instances associated with the Patient SOP Instance.</td>
</tr>
<tr>
<td>&gt;Referenced SOP Class UID</td>
<td>(0008,1150)</td>
<td>Uniquely identifies the referenced SOP Class.</td>
</tr>
<tr>
<td>&gt;Referenced SOP Instance UID</td>
<td>(0008,1155)</td>
<td>Uniquely identifies the referenced SOP Instance.</td>
</tr>
<tr>
<td>Referenced Patient Alias Sequence</td>
<td>(0038,0004)</td>
<td>Uniquely identifies any Patient SOP Instances which also describe this patient. These SOP Instances are aliases.</td>
</tr>
<tr>
<td>&gt;Referenced SOP Class UID</td>
<td>(0008,1150)</td>
<td>Uniquely identifies the referenced SOP Class.</td>
</tr>
<tr>
<td>&gt;Referenced SOP Instance UID</td>
<td>(0008,1155)</td>
<td>Uniquely identifies the referenced SOP Instance.</td>
</tr>
</tbody>
</table>

### Table 5.7: Patient Medical Module Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Tag</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient State</td>
<td>(0038.0500)</td>
<td>Description of patient state (comatose, disoriented, vision impaired etc.)</td>
</tr>
<tr>
<td>Pregnancy Status</td>
<td>(0010.21C0)</td>
<td>Describes pregnancy state of patient. Enumerated Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0001 = not pregnant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0002 = possibly pregnant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0003 = definitely pregnant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0004 = unknown</td>
</tr>
<tr>
<td>Medical Alerts</td>
<td>(0010.2000)</td>
<td>Conditions to which medical staff should be alerted (e.g. contagious condition, drug allergies, etc.)</td>
</tr>
<tr>
<td>Contrast Allergies</td>
<td>(0010.2110)</td>
<td>Description of prior reaction to contrast agents.</td>
</tr>
<tr>
<td>Special Needs</td>
<td>(0038.0050)</td>
<td>Medical and social needs (e.g. wheelchair, oxygen, non-English-speaking etc.)</td>
</tr>
<tr>
<td>Last Menstrual Date</td>
<td>(0010.21D0)</td>
<td>Date of onset of last menstrual period</td>
</tr>
<tr>
<td>Smoking Status</td>
<td>(0010.21A0)</td>
<td>Indicates whether patient smokes. Enumerated Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>Additional Patient History</td>
<td>(0010.21B0)</td>
<td>Additional information about the patient’s medical history</td>
</tr>
</tbody>
</table>
### Table 5.8: Patient Demographic Module Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Tag</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient's Address</td>
<td>(0010,1040)</td>
<td>Legal address of the named patient</td>
</tr>
<tr>
<td>Region of Residence</td>
<td>(0010,2152)</td>
<td>Region within patient’s country of residence</td>
</tr>
<tr>
<td>Country of Residence</td>
<td>(0010,2150)</td>
<td>Country in which patient currently resides</td>
</tr>
<tr>
<td>Patient’s Telephone Numbers</td>
<td>(0010,2154)</td>
<td>Telephone numbers at which the patient can be reached</td>
</tr>
<tr>
<td>Patient’s Birth Date</td>
<td>(0010,0030)</td>
<td>Date of birth of the named patient</td>
</tr>
<tr>
<td>Patient’s Birth Time</td>
<td>(0010,0032)</td>
<td>Time of birth of the named patient</td>
</tr>
<tr>
<td>Ethnic Group</td>
<td>(0010,2160)</td>
<td>Ethnic group or race of patient</td>
</tr>
<tr>
<td>Patient’s Sex</td>
<td>(0010,0040)</td>
<td>Sex of the named patient. Enumerated Values: M = male F = female O = other</td>
</tr>
<tr>
<td>Patient’s Size</td>
<td>(0010,1020)</td>
<td>Patient’s height or length in meters</td>
</tr>
<tr>
<td>Patient’s Weight</td>
<td>(0010,1030)</td>
<td>Weight of the patient in kilograms</td>
</tr>
<tr>
<td>Military Rank</td>
<td>(0010,1080)</td>
<td>Military rank of patient</td>
</tr>
<tr>
<td>Branch of Service</td>
<td>(0010,1081)</td>
<td>Branch of the military. The country allegiance may also be included (e.g. U.S. Army)</td>
</tr>
<tr>
<td>Patient’s Insurance Plan Code Sequence</td>
<td>(0010,0050)</td>
<td>A sequence that conveys the patient’s insurance plan.</td>
</tr>
<tr>
<td>&gt;Code Value</td>
<td>(0008,0100)</td>
<td>The code value (defined by the coding scheme) that represents the patient’s insurance plan name.</td>
</tr>
<tr>
<td>&gt;Coding Scheme Designator</td>
<td>(0008,0102)</td>
<td>The code from table D-1 designating the coding scheme which maps the Code Value (0008,0100) onto the Code Meaning (0008,0104)</td>
</tr>
<tr>
<td>&gt;Code Meaning</td>
<td>(0008,0104)</td>
<td>The patient’s insurance plan name that is represented by the Code Value (0008,0100)</td>
</tr>
<tr>
<td>Patient’s Religious Preference</td>
<td>(0010,21F0)</td>
<td>The religious preference of the patient</td>
</tr>
<tr>
<td>Patient Comments</td>
<td>(0010,4000)</td>
<td>User-defined comments about the patient</td>
</tr>
</tbody>
</table>

### 5.2 REQUIREMENTS

This section discusses some of the requirements for the components of VRE system. It covers hardware (workstation and network infrastructure) and software (operating system, libraries, and other supporting packages.)
5.2.1 Hardware Requirements

For the workstation used in VRE system, a Pentium II 400 MHz with 256 MB RAM and 100 Mbps network card should be standard. A monitor of at least 19-inch in diameter and video card with 8MB RAM is also required for running VRE application. With the rapidly falling prices of personal computer and its peripherals, at the time of the completion of VRE prototype project, this workstation costs no more than $3000 each. The connection to at least an Ethernet local area network, and the installation of a network operating system in the workstation is assumed.

5.2.2 NT Workstation 4.0

Microsoft® Windows NT Workstation 4.0 is one of the most powerful 32-bit desktop operating systems available today, offering major benefits in terms of reliability, performance, security, and manageability. It takes full advantage of network capabilities, the latest technologies, and advances yet to come.

Many technical workstation users today keep two computers running at once: a UNIX box for technical tasks, and a Windows machine for e-mail and other productivity applications. Windows NT Workstation offers full support for standard UNIX protocols and easily interoperates with UNIX workstations and servers. Included are support for resource sharing via a client and server for Network File System (NFS), Telnet client and server, password synchronization, and UNIX shell and commands.

Furthermore, it takes two to three times as much for a UNIX box to get comparable performance available with Windows NT Workstation, while new workstations from third party vendors such as Hewlett-Packard, IBM, and Digital are designed exclusively for Windows NT.
These workstations offer better performance than leading UNIX workstations at a significantly lower price. Finally, a rapidly increasing number of software applications for both technical and productivity needs are available for Windows NT Workstation 4.0.

In June 1998, ISD (Integrated System Design) Magazine released the results of their latest platform performance comparison, pitting the newest 400Mhz Windows NT-based machines from IBM, HP, and Compaq against a 300Mhz Sun Ultra 60. In each of the five tests comprising the benchmark, the Sun Ultra 60 lagged the Windows NT platforms significantly, showing performance approximately 60 percent that of the Windows NT platforms, which cost approximately $8000 compared to Sun Ultra 60 at $27,000 [19]. Also, a study from Deloitte & Touche Consulting Group documents industry trends in the migration to Windows NT Workstation: This report documents the reasons why companies have migrated, best practices for moving to Windows NT Workstation, and how companies have overcome common transition challenges [4].

5.2.3 VisiBroker for Java 3.3

VisiBroker provides the ease-of-use, the scalability and the flexibility required to confront the many challenges of today's complex, heterogeneous application environments. As the leading deployed CORBA implementation, VisiBroker makes good on the promise of the Object Management Group's (OMG's) CORBA standard: leveraging emerging Internet technologies, enabling application re-use to improve time-to-market and providing a solid, industry standard foundation for mission-critical applications.
The Inprise VisiBroker family of products provides an integrated suite of tools and services to enable the development, deployment, and management of flexible, scalable, and secure distributed object applications throughout an organization, an intranet, and across the Internet. Before using VisiBroker for Java products, the environment variable CLASSPATH must be setup to point to the jar files provided by VisiBroker. In Windows NT, set it from Control Panel | System | Environment:

**5.2.3.1 Visibroker’s Smart Agent**

The Visibroker’s ORB Smart Agent (osagent) is a dynamic, distributed directory service that provides facilities for both client applications and object implementations. When a client application invokes the `bind` method on an object, the osagent locates the specified

---

4 Before using VisiBroker for Java products, the environment variable CLASSPATH must be setup to point to the jar files provided by VisiBroker. In Windows NT, set it from Control Panel | System | Environment:
implementation and objects so that a connection can be established between the client and the implementation. Similarly, object implementations register their objects with the osagent so that the client applications can locate and use those objects. When an object or implementation is destroyed, the osagent removes them from its list of available objects.

Agent Communication

An osagent may be started on any host. To locate an osagent, client applications and object implementations send a broadcast message, and the first osagent to respond will be used. Once an osagent has been located, an point-to-point User Datagram Protocol (UDP) communication is established for registration and look-up requests. The UDP protocol is used because it consumes fewer network resources than a TCP connection. All registration and locate requests are dynamic, so there are no required configuration files or mapping to maintain.

Agent-to-Agent Cooperation

When multiple instances of the osagent are on different hosts, each osagent will recognize a subset of the objects available and communicate with other osagents to locate objects it cannot find. If one of the osagent processes should terminate unexpectedly, all implementations registered with that agent will be notified and they will automatically re-register with another available osagent.

D:/Inprise/vbroker/lib/orb.jar;D:/Inprise/vbroker/lib/vjcosnm.jar;D:/Inprise/vbroker/lib/vjcoserv.jar;D:/Inprise/vbroker/lib/vbjtools.jar;D:/Inprise/vbroker/lib/vbjapp.jar

*Broadcast messages are usually used only to locate an osagent. All other communication with the makes use of a point-to-point communication.*
Agent Fault Tolerance

If there are more than one instance of the osagent on a local network and one of those agents becomes unavailable, all object implementations registered with that agent will be automatically re-registered with another agent. Likewise, client applications using an osagent that becomes unavailable will be automatically switched to another agent by VisiBroker. No special coding techniques are required to take advantage of this osagent fault-tolerance, as long as the osagent is running on more than one host in a local area network.

5.2.3.2 VisiBroker's Location Service

VisiBroker Location Service provides an enhanced object discovery that enables applications to find object instances based on particular attributes. Working with the Smart Agents, the Location Service notifies application program of what objects are presently accessible on the network, and where they reside. The Location Service is a VisiBroker extension to the CORBA specification, and is only useful to find objects implemented with VisiBroker.

The Location Service communicates directly with one Smart Agent which maintains a catalog, or a list of instances it knows, along with information it knows about the instances. When queried by the Location Service, a Smart Agent forwards the query to the other Smart Agents, and aggregates their replies in the result it returns to the Location Service. In this way, the Location Service "sees" all the instances of an object to which a client can bind.\(^6\) Smart Agents only know about instances with persistent object reference that are accessible. An accessible instance is either an active object – an object whose server is running and has invoked

\(^6\) Only instances with persistent object references are registered with the Smart Agent.
obj_is_ready() for the object – or an activable object – an object that has been registered with an Object Activation Daemon (OAD).

The Location Service is useful for purposes such as load balancing. Suppose that replicas of an object are located on several hosts. A bind interceptor could be deployed to maintain a cache of the host names that offer a replica, and each host’s recent load average. The interceptor updates its cache by asking the Location Service for the hosts currently offering instances of the object, and then queries the hosts to obtain their load averages. The interceptor then returns an object reference for the replica on the host with the lightest load.

5.2.3.3 VisiBroker’s Naming Service

VisiBroker Naming Service provides a standard-based naming service that allows meaningful association of names with objects, and simplifies the management of these names within an application. Conforming to the Object Naming Service specification from the Object Management Group (OMG), the CORBAservices-compliant VisiBroker Naming Service enables client applications to bind to objects using meaningful, logical names – without having to address any platform specific naming conventions.

To use the Name Service, at least one Naming Factory must be started. The Factory object lives within a server process, and is used to create NamingContext objects. When the default Factory server is started, it creates no NamingContexts. When it is asked to create NamingContexts, all such NamingContext objects created by a given Factory are located within the same process as that Factory. This, of course, is transparent to the CORBA client of the Name Service (but may be useful knowledge for the developer or Name Space administrator).

To enable rapid connections with objects, the VisiBroker Naming Service allows application developers to organize names in hierarchies, and create federated name spaces.
Multiple, nested, and federated contexts are all fully supported. Figure 5-2 shows an example of what VisiBroker Naming Service can do. There are several naming contexts in the diagram: Company A, Int’l Sales, U.S. Sales, and Northeast Territory. The Int’l Sales naming context is nested within the Company A naming context.

The VisiBroker Naming Service is the first implementation of a naming service in Java based on the OMG’s specification. With VisiBroker for Java and the VisiBroker Naming Service, one can deploy CORBA objects written in Java on any platform where a Java Virtual Machine (JVM) exists. By using VisiBroker technology to build distributed applications, developers can access the full ORB functionality that is integrated into leading Web application development environments.

Figure 5-2: Example of a multiple, nested, and federated naming context

A Persistent, Reliable, and Familiar Naming Service

The VisiBroker Naming Service provides persistence for name associations and naming contexts by logging all state information. The information is persistent across service shutdowns and start-ups, as well as recoverable in the event of an abnormal service failure. Additional, VisiBroker Smart Agent architecture provides high availability by starting a new naming service
server automatically if a running server becomes unavailable. With the combination of the VisiBroker ORB and its Smart Agent architecture, the VisiBroker Naming Service is a highly available, self-recovering service. If a naming service fails, the VisiBroker Smart Agent automatically reroutes service requests to a node that can activate a new, up-to-date naming service using VisiBroker’s Object Activation Daemon facility.

The VisiBroker Naming Service organizes names in much the same way that files are organized in a file system (e.g., with directory structure, where each directory may contain files or other directories). However, the naming service does not need a root naming-context, unlike a file system, which always has a root directory. This means that one set of naming contexts can be used by multiple groups. The naming contexts of these multiple groups can subsequently be federated into a company-wide naming context — resulting in arbitrarily nested naming contexts with no fixed root context. This added flexibility enables the manipulation of name contexts to serve the needs of the company or customers.

Enhances Object Navigation and Access

With the hierarchy offered by the VisiBroker Naming Service, all naming services can share their name databases. This means that all the objects in remote naming services can appear in results returned by a higher-level naming service. Using the VisiBroker ORB, any client program can navigate through these hierarchies in search of objects, regardless of domain locations. By making navigation of and access to objects easier, distributed applications are easier to implement and attain greater portability. Additionally, at runtime the VisiBroker Naming Service maintains all state information in memory, making it quicker for clients to look up object information.
5.2.3.4 VisiBroker Event Service

VisiBroker Event Service provides a standards-based event mechanism that allows developers to decouple communications between objects, and asynchronously distribute data to multiple objects through event channels. One or more suppliers send data to the event channel, and the channel asynchronously distributes the data to one or more consumers.

Conforming to the Object Event Service specification from the Object Management Group (OMG), the CORBA services-compliant VisiBroker Event Service supports both push and pull models for event communication. In the push model, a supplier sends an event to the event channel, from which consumers can obtain the event. In the pull model, the event channel solicits the event from the supplier. Like suppliers, consumers can also receive events by requesting the information from the event channel (pull), or by receiving the information automatically (push). Figure 5-3 shows the architecture of VisiBroker's Event Channel.

![Event Channel Architecture](image)

**Figure 5-3: Event Channel Architecture**
The VisiBroker Event Service is a collection of methods that enable developers to define event channels, determine the communication model to be used by suppliers and consumers (i.e., push or pull), and transfer data through the event channel. Additional configurations enable prevention of out-of-memory conditions by limiting the number of messages that can be queued for each consumer. As shown in Figure 5-3, the VisiBroker Event Service enables both push and pull suppliers, and push and pull consumers. For consumers, an event queue in the event channel keeps track of events that have not been communicated. The event queue ensures that slower clients will not lose information.

The VisiBroker Event Service can handle events in a variety of configurations and maintain event channels on multiple platforms. Events can be handled through either push or pull suppliers, as well as push or pull consumers. The ability to implement both models gives the flexibility to design the object architecture that best meets the application's needs. Furthermore, the VisiBroker Event Service provides ways to configure the number of outstanding messages that will be queued for each consumer — providing greater control over system resources.

With the combination of the VisiBroker ORB and its Smart Agent architecture, the VisiBroker Event Service is a highly available, self-recovering service. If an event service fails, the VisiBroker Smart Agent can automatically reroute the service request to a node that can activate a new, up-to-date event service using VisiBroker's Object Activation Daemon facility.

5.2.3.5 VisiBroker Object Activation Daemon (OAD)

The Object ActivationDaemon (OAD) is a deployment-time service that allows referenced CORBA implementations to be automatically started in response to client requests. During development, the individual processes that comprise the distributed application can be started manually. When it comes time for deployment, the installation can start the OAD, and
register with that daemon how all servers should be started upon a client request. The daemon
starts the application servers as they are needed.

When an implementation is registered with the OAD, the Daemon creates a Forwarder
that masquerades as the actual (as yet unspawned) server. As such, it registers with the
VisiBroker Smart Agent. When a client requests the object (for example, by calling the
generated_bind method for that interface), the following actions occur "under the covers"
before a reference is actually returned to the client:

- The client's ORB asks the Smart Agent for the location of a Provider that matches the bind
criteria. Unknown to the Agent or the client's ORB, the location actually refers to the
Forwarder within the OAD.

- The client's ORB generates an unbound stub that points to the Forwarder within the OAD. At
this point there has been no spawn and there is no knowledge that the obtained Provider
actually lives within the OAD. Upon the first method invocation by the client, the following
actions occur:

  - The client's ORB sends a request to the server to find out if the desired object is alive. The
    OAD responds by:

    - spawning the desired server according to its registration

    - waiting for the spawned server to call BOA::obj_is_ready on the requested object

- When the spawned server notifies the OAD that the requested object is active (implicitly as
  part of the BOA::obj_is_ready call within the spawned server), the OAD responds to the
original client with the new location of the desired object e. The client (now with the new/real
location of the desired object) makes all subsequent invocations on the real object. If the
spawned server were to subsequently exit, any future methods by the client would attempt to
rebind to the server (implicitly upon detection of the failure of the IIOP connection). The steps for rebind follow exactly the steps outlined above for the original bind.

5.2.4 Java Development Kit (JDK) and Java Platform

The Java programming language and environment is designed to solve a number of problems in modern programming practice. Java started as a part of a larger project to develop advanced software for consumer electronics. These devices are small, reliable, portable, distributed, and real-time embedded systems. When we started the project we intended to use C++, but encountered a number of problems. Initially these were just compiler technology problems, but as time passed more problems emerged that were best solved by changing the language.

Figure 5-4: Java Platform Architecture

The computer world currently has many platforms, among them Microsoft Windows, Macintosh, OS/2, UNIX® and NetWare®, software must be compiled separately to run on each
platform. The binary file for an application that runs on one platform cannot run on another platform, because the binary file is platform-specific.

The Java Platform is a new software platform for delivering and running highly interactive, dynamic, and secure applets and applications on networked computer systems. But what sets the Java Platform apart is that it sits on top of these other platforms, and executes bytecodes, which are not specific to any physical machine, but are machine instructions for a virtual machine. A program written in the Java Language compiles to a bytecode file that can run wherever the Java Platform is present, on any underlying operating system. In other words, the same exact file can run on any operating system that is running the Java Platform. This portability is possible because at the core of the Java Platform is the Java Virtual Machine.

Figure 5-5: Java Development and Runtime Environment
While each underlying platform has its own implementation of the Java Virtual Machine, there is only one virtual machine specification. Because of this, the Java Platform can provide a standard, uniform programming interface to applets and applications on any hardware. The Java Platform is therefore ideal for the Internet, where one program should be capable of running on any computer in the world. The Java Platform is designed to provide this "Write Once, Run Anywhere" capability.

Developers use the Java Language to write source code for Java-powered applications. They compile once to the Java Platform, rather than to the underlying system. Java Language source code compiles to an intermediate, portable form of bytecodes that will run anywhere the Java Platform is present.

Developers can write object-oriented, multithreaded, dynamically linked applications using the Java Language. The platform has built-in security, exception handling, and automatic garbage collection. Just-in-time compilers are available to speed up execution by converting Java bytecodes into machine language. From within the Java Language, developers can also write and call native methods—methods in C, C++ or another language, compiled to a specific underlying operating system—for speed or special functionality.

When people talk about Java, they usually refer to the Java Programming Language. However, this is only one side of the coin. Java is a two-sided coin: a programming language and a platform. As a high-level programming language, Java has a few of distinct characteristics, which includes: Simple, Architecture-neutral, Object-Oriented, Portable, Distributed, High-performance, Interpreted, Multithreaded, Robust, Dynamic, Secure.
5.2.4.1 Java as a Programming Language

Java is an unusual programming language in that each Java program is both compiled and interpreted. With a compiler, a Java program is translated or compiled into an intermediate language called Java bytecodes – the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java bytecode instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. Figure 5-6 illustrates how this works.

![Figure 5-6: Java Compilation and Interpretation](image)

Java bytecodes can be thought as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java bytecodes help make “write once, run anywhere” possible. A Java program can be compiled into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. For example, the same Java program can run on Windows NT, Solaris, and Macintosh.
5.2.4.2 Java as a Platform

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components:

- The Java Virtual Machine (Java VM)
- The Java Application Programming Interface (Java API)

The Java VM is the base for the Java platform and is ported onto various hardware-based platforms. The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (packages) of related components.

The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.
As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring Java’s performance close to that of native code without threatening portability.

5.2.5 JDBC

Java Database Connectivity (JDBC) is a standard SQL database access interface, providing uniform access to a wide range of relational databases. JDBC also provides a common base on which higher level tools and interfaces can be built. It consists of a set of classes and interfaces written in the Java programming language. JDBC provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API.

Using JDBC, it is easy to send SQL statements to virtually any relational database. In other words, with the JDBC API, it is not necessary to write one program to access a Sybase database, another program to access an Oracle database, another program to access an Informix database, and so on. One can write a single program using the JDBC API, and the program will be able to send SQL statements to the appropriate database. And, with an application written in
the Java programming language, one also doesn’t have to worry about writing different applications to run on different platforms. The combinations of Java and JDBC lets a programmer write it once and run it anywhere [25].

![Figure 5-9: JDBC Architecture](image)

5.2.5.1 JDBC - ODBC Bridge

To use JDBC with a particular database management system, a JDBC Driver is needed to mediate between JDBC and the database. Depending on various factors, a Driver might be written purely in Java, or in a mixture of Java and JNI native methods. The latest SDK includes the JDBC-ODBC Bridge. This JDBC Driver makes most Open Database Connectivity (ODBC) drivers available to JDBC programmers. The JDBC-ODBC Bridge Guide describes the current status of this software [42].

7 As a point of interest, JDBC is a trademarked name and is not an acronym; nevertheless, JDBC is often thought of as
The JDBC-ODBC Bridge is a JDBC driver that implements JDBC operations by translating them into ODBC operations. To ODBC it appears as a normal application program. The Bridge implements JDBC for any database for which an ODBC driver is available. The Bridge is implemented as the `sun.jdbc.odbc` Java package and contains a native library used to access ODBC. The Bridge is a joint development of Intersolv and JavaSoft.

The Bridge is used by opening a JDBC connection using a URL with the `odbc` sub-protocol. See below for URL examples. Before a connection can be established, the bridge driver class, `sun.jdbc.odbc.JdbcOdbcDriver`, must either be added to the `java.lang.System` property named `jdbc.drivers`, or it must be explicitly loaded using the Java class loader. When loaded, the ODBC driver (like all good JDBC drivers) creates an instance of itself and registers this with the JDBC driver manager. Explicit loading is done with the following line of code:

```java
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

standing for "Java Database Connectivity".
After the JDBC-ODBC Bridge driver has been loaded, the next step in establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea:

```java
Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");
```

If JDBC-ODBC Bridge driver is used, the JDBC URL will start with "jdbc:odbc:". The rest of the URL is generally the data source name or database system. In this project, ODBC is used to access a Microsoft Access ODBC data source called "pacs-1", so the JDBC URL is `jdbc:odbc:pacs-1`. In place of "myLogin", the string "sa" is used to reflect the System Administrator login name used to log in to the DBMS; in place of "myPassword", the password of the System Administrator is used.

If one of the loaded drivers recognizes the JDBC URL supplied to the method `DriverManager.getConnection`, that driver will establish a connection to the DBMS specified in the JDBC URL. The `DriverManager` class manages all of the details of establishing the connection behind the scenes. The connection that is returned by the method `DriverManager.getConnection` is an open connection that can be used to create JDBC statements that pass SQL statements to the DBMS.

### 5.2.5.2 JDBC as a Low-Level API

JDBC is a "low-level" interface, which means that it is used to invoke (or "call") SQL commands directly. It works very well in this capacity and is easier to use than other database connectivity APIs, but it was designed also to be a base upon which to build higher-level
interfaces and tools. A higher-level interface is "user-friendly," using a more understandable or more convenient API that is translated behind the scenes into a low-level interface such as JDBC. Currently, there are two kinds of higher-level APIs under development on top of JDBC:

- An embedded SQL for Java: At least one vendor plans to build this. DBMSs implement SQL, a language designed specifically for use with databases. JDBC requires that the SQL statements be passed as Strings to Java methods. An embedded SQL preprocessor allows a programmer to instead mix SQL statements directly with Java: for example, a Java variable can be used in a SQL statement to receive or provide SQL values. The embedded SQL preprocessor then translates this Java/SQL mix into Java with JDBC calls.

- A direct mapping of relational database tables to Java classes: JavaSoft and others have announced plans to implement this. In this "object/relational" mapping, each row of the table becomes an instance of that class, and each column value corresponds to an attribute of that instance. Programmers can then operate directly on Java objects; the required SQL calls to fetch and store data are automatically generated "beneath the covers." More sophisticated mappings are also provided, for example, where rows of multiple tables are combined in a Java class.

As interest in JDBC has grown, more developers have been working on JDBC-based tools to make building programs easier, as well. Programmers have also been writing applications that make accessing a database easier for the end user. For example, an application might present a menu of database tasks from which to choose. After a task is selected, the application presents prompts and blanks for filling in information needed to carry out the selected task. With the requested input typed in, the application then automatically invokes the necessary SQL
commands. With the help of such an application, users can perform database tasks even when they have little or no knowledge of SQL syntax.

5.3 IDL & ORB ARCHITECTURE

CORBA uses Interface Definition Language (IDL) to define the types and objects. These types and objects are specified by IDL through their interfaces. An interface consists of a set of named operations (methods) and the parameters to those operations. IDL is the means by which a particular ORB object implementation informs its potential clients of the available operations and how to invoke them. From the IDL definitions, it is possible to map any CORBA objects into particular programming languages.

MetaManager Interface Definition Language (IDL) constitutes an interface between different levels of hierarchy. DIN-PACS IDL defines interfaces between existing DIN-PACS systems and our Client Meta-Manager. ClientMM IDL defines interfaces to Regional Meta-Manager and DIN-PACS. RegionalMM IDL defines interfaces to Client Meta-Manager and Master MetaManager. Finally, MasterMM IDL defines its interfaces to Regional Meta-Manager.

In our hierarchy of MetaManager systems, the IDL resides at various levels of the hierarchy tree. Figure 5-11 shows the tree hierarchy of VRE components and how component talks to each other through its IDL.
In CORBA implementation, each of the components (nodes) in the hierarchy tree above will be implemented as an ORB object. Communication among these objects is done through the ORB itself. Figure 5-12 shows how the IDL modules of each component are connected to the Object Request Broker (ORB), depicted as a solid line from the node to the CORBA bus. Multiple implementations of DIN-PACS and Client Meta-Manager are shown here.

Although each node is connected to the ORB, communications are only allowed between certain pair of nodes as dictated by the VRE hierarchy. In Figure 5-12, the dotted lines between two nodes represent existing virtual communication channel in the system. As can be seen in the figure, for example, the communication from the Master Meta-Manager is only to and from the Regional Meta-Manager, although connections exist between Master Meta-Manager and Client Meta-Manager.
Another type of communication is also depicted in Figure 5-12, and that is the communication between DIN-PACS and its database system. Although this communication channel does not directly relate to the ORB architecture of the VRE system, it helps to know that the implementation DIN-PACS node in the VRE Prototype system has database JDBC-ODBC connections. These connections are enabled through the use of JDBC-ODBC driver.

5.3.1 IDL Methods

DIN-PACS as an ORB object will be required to provide methods and information for other ORB objects, mainly Client Meta-Manager, in the system. Client Meta-Manager must have interfaces to communicate with other ORB objects (DIN-PACS and Regional Meta-Manager) n the systems. Similarly, Regional Meta-Manager and Master Meta-Manager must also provide interfaces for communication with other objects. The rules and privilege of the communication
dynamics is dictated by the behavior of each component as set in the VRE Specification document. Several basic methods required from a component object are:

5.3.1.1 *Query and Results Methods*

1. `get_patient_db()`
2. `get_case_db()`
3. `get_visit_db()`
4. `get_study_db()`
5. `get_doctor_schedule_db()`
6. `get_prior_cases()`
7. `search_result()`

5.3.1.2 *Communications-related methods*

1.1 `are_you_alive()` - sent by Client MetaManager to ensure connectivity status
1.2 `site_workload()` - to obtain DIN-PACS workload (0 - 100%)
1.3 `ping_server()` - called by ClientMM to obtain ClientMM-DIN-PACS (downlink) bandwidth

5.3.1.3 *Administrative and Command-Control methods*

1.4 `Get_administrator()` - get DIN-PACS system administrator info
1.5 `Get_parameter()` - get DIN-PACS parameter current values
1.6 `Set_parameter()` - set DIN-PACS parameter values
1.7 Parameters include:

1.7.1 Site IP address
1.7.2 Permissions
5.3.2 VRE Prototype Objects and Classes

The VRE Prototype is designed as an object-oriented system. It uses many of the objects provided by the standard CORBA. In all CORBA application, the root of the object is set to be `org.omg.CORBA.Object`. All other objects are extensions (inheritance) of this object.\(^6\)

A CORBA IDL interface represents an object’s public specification and is mapped to a public Java interface with the same name. This interface extends, or inherits from, the `org.omg.CORBA.Object` and contains methods specifications for the mapped IDL operations and attribute accessor/mutator method.

In addition to Java interface, helper and holder classes are generated with a name constructed by appending the suffix Helper and Holder, respectively, to the interface name. The role of Helper class is to provide static methods that operate on the generated Java interface. One helper method needed quite often is `narrow`. The `narrow` method allows a generic `org.omg.CORBA.Object` to be narrowed down to the object reference of an extended class. The Holder class is a final public class (i.e., not extendable) that contains methods to read from and write to an Input Stream.

Figure 5-13 shows object relationship of DIN-PACS implementation. The main function of the applications resides in `DIN_PACSmain` object, which uses `DIN_PACSHolder`, `DIN_PACSHelper`, and `DIN_PACSImp1` objects.

---

\(^6\) In Java implementation of CORBA, however, there is still an ancestor to `org.omg.CORBA.Object` which is the `java.lang.Object`. This object is the root of all other objects in all Java application. Since CORBA for Java is actually a Java application (environment), it has to obey this rule.
DIN_PACSIimpl object constructs an ORB for communications with other ORB objects (ClientMM, RegionalMM, and MasterMM) in the system. DIN_PACSIimpl object is an inheritance of _DIN_PACSIimplBase object, an object automatically created by idl2java compiler. _DIN_PACSIimplBase object is an abstract class, so it cannot be instantiated directly. Rather, it needs to be extended to other objects before it can be instantiated. _DIN_PACSIimplBase object
implements the DIN_PACS interface and contains the basic information about the methods available in DIN_PACS interface.

Figure 5-14 shows the class diagram for Client Meta-Manager. The Regional Meta-Manager and Master Meta-Manager class diagrams are very similar to that of Client Meta-Manager, and are not shown.

Figure 5-14: Client Meta-Manager Class Diagram

5.4 VRE TABLES AND VARIABLES

To ease the management and retrieval of data, the information that will be needed from DIN-PACS is contained within SQL database tables. With this configuration, any SQL database program should be able to access (query, add, modify, delete) the data.
The VRE Prototype system follows the database information set by the DICOM standard, although there are many additional variables added to the system for control and communications. The DICOM tables consulted for the VRE Prototype system are described below.

### 5.4.1.1 DICOM Tables used in VRE Prototype

1. **Patient Tables:**
   - 1.1 Patient Relationship Module
   - 1.2 Patient Identification Module
   - 1.3 Patient Demographic Module
   - 1.4 Patient Medical Module

2. **Visit Tables:**
   - 2.1 Visit Relationship Module
   - 2.2 Visit Identification Module
   - 2.3 Visit Admission Module
   - 2.4 Visit Discharge Module
   - 2.5 Visit Scheduling Module

3. **Study Tables:**
   - 3.1 Study Relationship Module
   - 3.2 Study Identification Module
   - 3.3 Study Classification Module
   - 3.4 Study Scheduling Module
   - 3.5 Study Component Module
   - 3.6 Study Acquisition Module
3.7 Study Read Module
3.8 Study Component Relationship Module
3.9 Study Component Acquisition Module
3.10 Scheduled Procedure Step Module
3.11 Requested Procedure Module
3.12 Imaging Service Request Module
3.13 Performed Procedure Step Relationship Module
3.14 Performed Procedure Step Identification Module
3.15 Image Acquisition Result Module
3.16 Radiation Dose Module
3.17 Billing and Material Management Code Module

4. Result Tables:
   4.1 Results Relationship Module
   4.2 Results Identification Module
   4.3 Results Impression Module

5. Interpretation Tables:
   5.1 Interpretation Relationship Module
   5.2 Interpretation Identification Module
   5.3 Interpretation State Module
   5.4 Interpretation Recording Module
   5.5 Interpretation Transcription Module
   5.6 Interpretation Approval Module
5.4.1.2 Example Databases

The tables below are example tables of a DICOM-compliant system taken from Lumisys DI-2000 Scanner. These tables are useful because they are described in an SQL-like description:

**PATIENT TABLE:**

```sql
{
    "id" varchar(65) NOT NULL,
    "last_name" varchar(65) NOT NULL,
    "first_name" varchar(65) NULL,
    "middle_initial" char(2) NULL,
    "address_line_1" varchar(65) NULL,
    "address_line_2" varchar(65) NULL,
    "city" varchar(21) NULL,
    "state" varchar(4) NULL,
    "zipcode" varchar(16) NULL,
    "birth_date" datetime,
    "birth_time" char(17) NULL,
    "ethnic_group" varchar(31) NULL,
    "home_phone" varchar(17) NULL,
    "office_phone" varchar(17) NULL,
    "occupation" varchar(16) NULL,
    "sex" char(17) NULL,
    "size" varchar(17) NULL,
    "ssn" varchar(17) NULL,
    "weight" varchar(17) NULL,
    "date_entered" datetime,
    "date_modified" datetime,
    "history" varchar(241) NULL,
    PRIMARY KEY ("id")
}
```

**EXAM TABLE:**

```sql
{
    "id" varchar(65) NOT NULL,
    "ref_exam_id" varchar(65) NULL,
    "case_id" varchar(65) NULL,
    "institution_name" varchar(65) NULL,
    "completion_time" char(17) NULL,
    "diagnosis" varchar(241) NULL,
    "dispatch_time" char(17) NULL,
    "exam_date" datetime,
    "exam_description" varchar(241) NULL,
    "exam_priority" varchar(31) NULL,
    "exam_status" varchar(31) NULL,
    "exam_time" varchar(17) NULL,
    "exam_type" varchar(31) NULL,
    "patient_id" varchar(65) NOT NULL,
    "physician" varchar(65) NULL,
    "procedure_code" varchar(31) NULL,
    "reading_physician" varchar(65) NULL,
    "receipt_time" char(17) NULL,
    "referring_physician" varchar(65) NULL,
    "referring_physician_phone" varchar(17) NULL,
    "technologist" varchar(65) NULL,
    "date_entered" datetime,
    "date_modified" datetime,
    "institution_adr" varchar(241) NULL,
    "requesting_phys" varchar(65) NULL,
}
```
```sql
"referring_phys" varchar(65) NULL,
"req_phone" varchar(17) NULL,
"read_unread" char(2) NULL,
"comments" varchar(241) NULL,
PRIMARY KEY ("id")
)

SERIES (IMAGE_SETS) TABLE:
{
    "id" varchar(65) NOT NULL,
    "exam_id" varchar(65) NOT NULL,
    "acquisition_device_id" varchar(65) NULL,
    "modality" varchar(31) NULL,
    "bodypart" varchar(65) NULL,
    "patient_position" varchar(31) NULL,
    "date_entered" datetime,
    "date_modified" datetime,
    PRIMARY KEY ("id")
}

CASE TABLE:
{
    "id" varchar(30) NULL,
    "admission_route" varchar(25) NULL,
    "admitting_date" datetime,
    "admitting_time" char(7) NULL,
    "admitting_diagnosis" varchar(240) NULL,
    "admitting_diagnosis_code" varchar(30) NULL,
    "discharge_date" datetime,
    "discharge_time" char(7) NULL,
    "discharge_diagnosis" varchar(240) NULL,
    "discharge_diagnosis_code" varchar(30) NULL,
    "location_id" varchar(30) NULL,
    "nurse" varchar(25) NULL,
    "patient_id" varchar(30) NULL,
    "station" varchar(25) NULL,
    "room" varchar(25) NULL,
    "date_entered" datetime,
    "date_modified" datetime
}

CPT TABLE:
{
    "cpt_code" varchar(30) NULL,
    "cpt_version" varchar(5) NULL,
    "description" varchar(240) NULL,
    "date_entered" datetime,
    "date_modified" datetime
}

DIAGNOSISCODE TABLE:
{
    "id" varchar(30) NULL,
    "scheme" varchar(30) NULL,
    "description" varchar(240) NULL,
    "date_entered" datetime,
    "date_modified" datetime
}

IMAGE TABLE:
{
    "acquisition_date" datetime,
    "image_sop" varchar(65) NOT NULL,
    "acquisition_time" char(17) NULL,
    "acquisition_number" integer NULL,
    "derivation_description" varchar(241) NULL,
```
5.5 TESTING ENVIRONMENT

The VRE Prototype system is tested in a minimal, controlled environment in the Computer Engineering Research Laboratory (CERL). The environment consists of one Master Meta Manager, two Regional Meta-Managers, three Client Meta-Managers, and three simulated DIN-PACS sites. These components are implemented in NT 4.0 workstations. This environment is so designed in order to provide the necessary system architecture that can show the proper working of local reading and remote reading. In local reading, a case is generated by a DIN-PACS, submitted to its Client Meta-Manager that determines that the case is to be read locally.

In a remote reading situation, a case is generated in a local DIN-PACS, submitted to its Client Meta-Manager. The Client Meta-Manager, however, determines that the case is to be forwarded (read) by a remote physician in a remote DIN-PACS. The Client Meta-Manager submits the case to its Regional Meta-Manager to determine if there is a physician available within its region to handle (read) the case. If there is one, then Regional Meta-Manager notifies the requesting Client Meta-Manager, which in turns notifies the DIN-PACS node. Upon receiving this answer, DIN-PACS forwarded the case header and accompanying images to the
destination DIN-PACS. When the case is finished being evaluated by the remote DIN-PACS, the case header will be returned to the local DIN-PACS along with the report. The images will be deleted in the remote DIN-PACS and not transferred back to the local DIN-PACS.

If the Regional Meta-Manager determines that no physician within its boundary can handle the case, then it will forward the case header to the Master Meta-Manager. The Master Meta-Manager holds all information from all DIN-PACS within the system. So it can be considered as the omniscient agent in the system, and therefore is able to make decision on all cases.

The Master Meta-Manager decides which site is suitable and optimum to handle the case, and return this information to the requesting Regional Meta-Manager. This information flows down to the requesting DIN-PACS, which then forwards the case to the destination DIN-PACS.
and waits for the report in reply. Figure 2-17 shows the environment hierarchy of the VRE Prototype system.

![Diagram of VRE Prototype hierarchy](image)

**Figure 5-16: Testing Sites of VRE Prototype**

Figure 5-16 shows the map of three sites that implements the testing environment hierarchy. Each site is handled by a separate NT workstation. Site 1 consists of the DIN-PACS I, Client Meta-Manager I, Regional Meta-Manager I, and Master Meta-Manager I. Site 2 consists of only DIN-PACS II and Client Meta-Manager II, and Site 3 consists of the DIN-PACS III, Client Meta-Manager III, and Regional Meta-Manager III.
5.5.1 Local Area Testing

A local testing is conducted at the Computer Engineering Research Laboratory (CERL), Electrical and Computer Engineering (ECE) Department, University of Arizona. The laboratory consists of 4 Sun Workstations, 1 DEC Workstation, 1 Windows NT Server, and 9 NT Workstations. The workstations are connected together to form an Ethernet Local Area Network (LAN) using a 3COM SuperStack II Dual Speed Hub. Almost all the workstations have a 100 Mbps network card attached, although some of the older workstations still have 10 Mbps network connection.

![Network Architecture of Local Area Testing Environment](image)

Figure 5-17: Network Architecture of Local Area Testing Environment

For the purpose of our test, however, all the workstations used have 100 Mbps card. Figure 5-17 shows the architecture of the testing environment. The workstation Pinehurst (pinehurst.ece.arizona.edu) acts as Site 1 in Figure 5-16 above; similarly, Sahalee workstation (sahalee.ece.arizona.edu) acts as Site 2 and Blackhorse (blackhorse.ece.arizona.edu) as Site 3.
All workstation has either 350 MHz Pentium II processor, 256 MB RAM, 8 MB video card, and Windows NT 4.0 operating system installed, with at least Service Pack 3.

5.5.2 Wide Area Testing

Wide area testing is also planned, but has not been implemented. The proposed network architecture for remote area testing environment is shown in Figure 5-18. Two of three workstations used in local area testing are still used in this scenario. However, the third one, Site 3, is proposed to be moved to Fort Huachuca, an ARMY based location about 90 miles southeast of the University.

![Diagram](image)

*Figure 5-18: Wide Area Testing Environment*

The communication between Site 1 and Site 2 is still up to 100 Mbps; the communication to Site 3 is now only up to 1.544 Mbps (T-1 line bandwidth). Since the actual test has not been
performed yet, there are no measurement data available to determine the feasibility of VRE system using 1.5 Mbps network connection.

5.5.3 Graphical User Interface

Figure 5-19 shows component icons of the VRE in the testing environment described in previous section. There are 3 DIN PACS systems, 3 Client Meta-Managers, 2 Regional Meta-Managers, and 1 Master Meta-Manager.

Figure 5-20, Figure 5-22, Figure 5-23, and Figure 5-24 shows the Graphical User Interface for DIN PACS, Client Meta-Manager, Regional Meta-Manager, and Master Meta-Manager, respectively.

![Diagram of VRE Component Icons]

Figure 5-19: VRE Component Icons

DIN PACS window shows the Case Table, Physician Table, and a monitor console at the bottom. The Case Table shows StudyID (or Case ID), Patient ID, Series ID, Date, Time, Priority, Status, Modality, and other pertinent information about a case.
The "Suspend" and "Resume" buttons are used to temporarily stop the running simulation thread. This is useful for examining the system's state or to debug the simulation flow. When the "Step" button is clicked, the simulation program interacts with the user to inform him about what is going on in the system, and to ask for user's responses. Figure 5-21 shows a dialog box that requests permission of the user to route a generated case whose information is displayed.

![VRE DIN PACS window](image)

Figure 5-20: VRE DIN PACS window
The “Restart” button can be used when something has gone wrong with the current simulation thread. It basically kills the currently running thread, release its system resources, create a new one, and run the simulation using the new thread.

There is also a set of toolbar buttons that control the flow of simulation of the VRE system. The VRE simulation is done through the use of threads. The “Start” button should be clicked when the simulation is initially started. This button creates a new thread to control the simulation, while the rest of DIN PACS system can run as before. The “Stop” button should be used to completely stop the thread. In addition to stopping the thread, this button also releases all the system resources used by the thread.

![Simulate Step](image)

**Figure 5-21: VRE DIN PACS Simulate Step Dialog**

All the Meta-Manager windows show the activities currently going on in the system. Client Meta-Manager window shows some initialization STATE transition when it first starts. It also shows the sizes of tables (Patient Table and Physician Table) that it has successfully acquired from the lower level DIN PACS system.
Regional and Master Meta-Manager windows show similar information, except that the number of lower level systems it communicates with could be more than one. For example, in both the local are and wide are testing described in previous section, there are two Regional Meta-Manager in the whole system. Furthermore, Regional Meta-Manager 1 has two Client Meta-Managers underneath it.
The aggregation of data from multiple lower-level systems is done using CORBA sequence mechanism. Each row of data is from the database is converted into a structure construct. The whole table of rows therefore is converted into a sequence of structures. Since there are many tables in the DIN PACS system, there are also many sequences of structures in Client Meta-Manager, Regional Meta-Manger, and Master Meta-Manager.
Server-side is ready
STATE = INITIALIZING

run_MMM(): Getting Database from RMM_2
RMM_2: Study Table Size = 11
RMM_2: Physician Table Size = 3
RMM_1: STATE = DB_READY

<table>
<thead>
<tr>
<th>PhysicianID</th>
<th>DIN-PACS</th>
<th>PhysicianName</th>
<th>Subspecialty</th>
</tr>
</thead>
<tbody>
<tr>
<td>44-2544</td>
<td>DIN_PACS_3</td>
<td>Monroe</td>
<td>Daniel</td>
</tr>
<tr>
<td>29-5412</td>
<td>DIN_PACS_3</td>
<td>Fairchild</td>
<td>Anita</td>
</tr>
<tr>
<td>63-5451</td>
<td>DIN_PACS_3</td>
<td>Gates</td>
<td>Sarah</td>
</tr>
</tbody>
</table>

Figure 5-24: VRE Master Meta-Manager Window
CHAPTER 6: SUMMARY AND FUTURE WORKS

This dissertation describes implementation notes on a CORBA-based prototype of Virtual Radiology Environment (VRE). It begins with some history and background of the major components of VRE which include Digital Image Network – Picture Archiving and Communication System (DIN-PACS), Digital Imaging and Communications in Medicine (DICOM). A brief overview of Common Object Request Broker (CORBA) was also presented to usher in the detailed implementation of the prototype. The VRE System Specifications, which were developed by the Meta-Manager Team of the Computer Engineering Research Laboratory at the University of Arizona and the BAMC VRE Team, were also briefly discussed.

Designing an intelligent case routing algorithm for a distributed network domain requires a good understanding of the existing network structure and workflow. A knowledge base of information forms the basic building block for the analysis and design. A object-oriented design helps in identifying the major players within the system as well as helps in making the software:

1) easy to build (reduced complexity).
2) easy to modify or enhance (increased scalability, re-usability and maintainability).
3) easy to troubleshoot.

6.1 A FEW REMAINING ISSUES

Despite all attempts to incorporate all knowledge and experience into the design specifications and a prototype implementation, there are still a few remaining issues that need to be considered in building a full fledged implementation of VRE. These issues concern more about how existing systems work, and how to open the legal & administrative walls of the
hospital to provide a true open-system Virtual Radiology Environment. Some of the most difficult issues include:

6.1.1 Constraints Imposed by Legacy DIN-PACS

The legacy DIN-PACS systems at each MTF will be the primary actors in the final system. As such it is necessary to identify the interface to these systems as early as possible. Adding risk to this process is the proprietary nature of the DIN-PACS architecture, which obscures many of the design details in DIN-PACS. Further complicating the process is the nature of the relationship between our customer, the U.S. Army, and the architects of the DIN-PACS systems currently in use. The manufacturer contractually protects DIN-PACS design details. Modifications to accommodate the Meta-Manager system in existing DIN-PACS systems must be arranged with the manufacturer through a contract bid process. Attacking this problem first will identify the legacy limitations imposed on the system and give manufacturers advance notice of the changes that will be needed to accommodate the Meta-Manager.

6.1.2 Security

Security is a concern for any system comprised of shared networking components. The growth of networked business applications has partially addressed the security problems faced in this design. Part of the elaboration of this risk will be a study of existing security mechanisms and their applicability to the Meta-Manager system.

In addition to the traditional security problems, the system will operate in an environment with several specialized security concerns. Patient confidentiality must be ensured by the Meta-Manager system. The implications of this requirement will be developed during a subsequent
elaboration phase. There are also security requirements imposed by the uniqueness of having the U.S. Army as the customer. Many guidelines have been set out by the Army to dictate a "secure" system by their definition.

6.1.3 Performance

Once the essential system properties have been determined and system construction has progressed to a working system prototype, an accurate performance analysis can be performed on the design. The purpose of performance analysis is twofold. First, stated performance criteria must be met by this design before much time, money, and effort is dedicated to full-scale implementation of the system. Second, the system designers need to understand the performance issues in the system and modify the design to correct or account for system bottlenecks. The design should not be drastically modified at this stage in the analysis but many of the parameters used to tune the system should be understood. The potential performance problems in the first or subsequent revisions of the product should be identified and a clear plan should be presented to correct problems that do arise. A modeling and simulation effort is required to model the VRE and its Meta-Manager algorithms to predict their performance.

6.2 FUTURE WORKS

A comprehensive, object-oriented environment for building and deploying mission-critical, intelligent applications that dramatically improve complex business operations such as Virtual Radiology Environment (VRE). A product called G2 from Gensym Corporation in an
example of such support environment. A proven technology in this area could provide a competitive advantage by helping to:

- optimize operating efficiencies,
- improve asset and service availability,
- better manage complex, time-critical operations,
- build and deploy operations management applications dramatically faster,
- minimize the costs of maintaining operations management applications,
- preserve, enhance, and leverage operations knowledge.

6.2.1 Streamlining Development through the Power of Objects

With a highly interactive and visual development environment we can simplify and speed the prototyping, development, and on-line deployment of intelligent systems. Objects can be made a powerful and intuitive way to represent the physical and abstract aspects of applications. Objects are organized in a hierarchical class structure, providing the flexibility of multiple inheritance so that an object inherits properties and behaviors from multiple object classes. Once an object - or class of objects - is defined, the work is immediately reusable. Any object or group of objects can be cloned repeatedly, and each cloned copy will inherit all the properties and behaviors of the original object. Objects, rules, and procedures can then be grouped into library modules that are shared by all G2 applications, allowing streamlined development of new applications.

9 Gensym Corporation is a leading supplier of software products and services for intelligent operations management that help organizations manage and optimize complex dynamic operations. Common applications include quality management, process optimization, dynamic scheduling, network fault management, energy and environmental management, and abnormal situation management.
VRE’s graphical user interface (GUI) can represent much more than pictures - they can represent the properties and behaviors of objects and the relationships among them. Developers can quickly model an application by graphically representing and connecting objects. These connections can be dynamically created, modified, and delete. Connected objects form powerful models that visually represent application processes - such as material flows, industrial processes, communication networks, transportation and logistics networks, information routings, and even logic flows. VRE’s graphical user interface (GUI) can also include built-in dialogs, graphs, charts, dials, tables, sliders, bitmaps, and meters to speed up the development of user interfaces. See Figure 6-1 below.

Figure 6-1: An Example of G2 Environment Capability
6.2.2 Capturing Knowledge with Rules, Procedures, and Models

Knowledge that is available in a variety of places in VRE system can be captured with rules, procedures, and models. With G2, VRE can capture knowledge and minimize development efforts to create generic rules, procedures, formulas, and relationships that apply across entire classes of objects. G2's structured natural language enables even non-programmers to read, understand, and modify applications. The interactive look-ahead editor helps edit rules, procedures, and models by displaying choices and checking for errors.

6.2.2.1 Rules

Expert knowledge is expressed using G2 rules, which work in real time and can mimic the human ability to focus on specific problems while maintaining a general awareness. Rules capture an expert's knowledge of how to reason about and respond to a given set of conditions.

G2 rules reason about real-time data and histories for time-critical analysis and action. They can be event-driven (through forward chaining) to automatically respond whenever new data arrives. They can also be data-driven (through backward chaining) to automatically invoke other rules, procedures, or formulas. And G2 can automatically monitor situations by regularly invoking rules that scan for possible circumstances and then take actions when defined thresholds are reached.

6.2.2.2 Concurrent Procedures for Real-Time Execution

G2's procedures work in real time and can be scheduled to the millisecond for non-stop execution. Procedures, rules, and models execute concurrently based on priorities. G2 procedures can be linked to object classes as methods to help developers effectively represent object
behaviors. Wait states and parallel execution threads can be specified in any procedure. As a result, organizations can build powerful real-time applications that are far more robust than those built with traditional programming tools. G2 also provides RPCs for executing procedures in other G2 applications throughout an enterprise or for creating interfaces to other real-time systems, databases, or applications.
APPENDIX: USE CASE FLOW OF EVENTS

B.1 THE PROCESS NEW CASE USE CASE

Preconditions:
The Meta-Manager Node must be in an Active Mode and Node Initialization must be complete before it can participate in Case processing activities.

Main Flow
This use case begins when a DIN-PACS Network has a Case that is ready to be processed.

M1. The Local DIN-PACS Network submits the new Case Header to the Meta-Manager.

M2. Meta-Manager finds all existing Cases in the VRE region that pertain to the history of the new case.

M3. Meta-Manager assigns a Case Number unique to the entire Meta-Manager system to the new Case and adds references to any relevant historical Cases discovered in M2. This new class is called a Meta-Case.

M4. The Meta-Manager passes the new Meta-Case header back to the Local DIN-PACS where it is added to the local Database Archive System.

M5. The Meta-Manager compares the reading requirements for the new Case with the reading capabilities of the Local DIN-PACS. If the Local DIN-PACS has the resources to adequately process the new Case subflow S1 is executed. Otherwise the Meta-Manager routing subflow in S2 is executed.
**Subflows**

S1. This subflow handles the scenario when the Local DIN-PACS processes the new Case.

  S1.1 The Meta-Manager signals the Local DIN-PACS that the new Case is to be processed locally.

  S1.2 Historical Cases listed in the Meta-Case Header are forwarded from their permanent locations to the Local DIN-PACS for reference.

  S1.3 The Case is processed as normal Case in the proprietary method of the Local DIN-PACS.

  S1.4 The Meta-Manager periodically monitors the status of the new Case to ensure that it is being processed according to the parameters specified in the header. If the Meta-Manager determines that the requirements of a Case are not being met it will invoke Alternative Flow E.2.

  S1.5 When the Case has been processed the Local DIN-PACS notifies the Meta-Manager of the new status of the Case.

S2. This subflow handles the scenario when the Local DIN-PACS is unable to adequately process a new Case.

  S2.1 The Meta-Manager discovers the most qualified Remote DIN-PACS to process the new Case according the following algorithm:

  \[
  \text{if} \ ( \text{Urgency} = \text{STAT or Urgency} = \text{ASAP} ) \\
  \quad \text{send to first available Radiologist for processing} \\
  \text{else if} \ ( \text{Preffered Radiologist is specified} ) \\
  \quad \text{if} \ ( \text{Radiologist available} )
  \]
send to specified Radiologist

else if ( Modality specified )

send to first available Specialist

else

send to specified Radiologist

else if ( Modality specified )

send to first available Specialist

else

send to first available Radiologist

A Specialist is a specialized Radiologist trained to read a specific Modality. The term available means that the Radiologist is currently active in the Meta-Manager system. The first available Radiologist is defined as the Radiologist who will be able to process the case the soonest based on their individual Case Queue length, Case Reading Rate and network transmission delays to deliver the Case.

S2.2 The Meta-Case header is sent to the Remote DIN-PACS signaling that there is a referral Case to be processed.

S2.3 The images for the new Case are sent from the Local DIN-PACS to the Remote DIN-PACS.

S2.4 Historical Cases listed in the Meta-Case Header are forwarded from their permanent locations to the Remote DIN-PACS for reference.

S2.5 The Case is diagnosed as normal Case in the proprietary method of the Remote DIN-PACS.
S2.6 The Meta-Manager periodically monitors the status of the new Case to ensure that it is being processed according to the parameters specified in the header. If the Meta-Manager determines that the requirements of a Case are not being met it will invoke Alternative Flow E.2.

S2.7 The Remote DIN-PACS notifies the Meta-Manager when the Case has been diagnosed.

S2.8 A text diagnosis back to the Local DIN-PACS to be kept with the permanent Case record.

Alternative Flows

E.1 A processing error case occurs when one or more Cases have been routed to a Remote DIN-PACS for diagnosis and the Meta-Manager system loses connectivity with the Remote site. In this case the following sequence of events takes place.

E.1.a The Remote DIN-PACS discovers that it has lost contact of the Meta-Manager system and drops all Cases it is reading on behalf of other sites.

E.1.b The Meta-Manager system updates the Connectivity status of the Remote DIN-PACS to "Unreachable".

E.1.c The Meta-Manager reroutes the Cases assigned to the lost Remote DIN-PACS starting with article M5 of the main flow for this Use-Case.

E.2 A similar but less extreme error occurs when the Meta-Manager system determines that the Case is not being processed according to the parameters set in the header for a Case. This determination causes the following sequence of events to take place:
E.2.a The Meta-Manager recalculates the optimal reader in the system according to article M5 of the Main Flow.

E.2.b If the Meta-Manager determines that the current reader is still best qualified to diagnosis this Case, no action is taken and the Case is read in a best-effort mode.

E.2.c If the Meta-Manager finds a more suitable reader it will send a cancel message to the DIN-PACS reading the case. The Case is rerouted to a new reader as dictated by the routine procedure in the Main Flow.

E.3 It is also possible that a Local DIN-PACS may become unreachable while it has Cases being read on it’s behalf at one or more Remote DIN-PACS. In this scenario, the following events take place:

E.3.a The Meta-Manager updates the Local DIN-PACS Connectivity as “Unreachable”.

E.3.b The Meta-Manager sends a cancel message to all Remote DIN-PACS for Cases they are reading on behalf of the lost Local DIN-PACS.

E.3.b The Local DIN-PACS processes all Cases that were being read remotely in a best-effort mode. If connectivity is reestablished with the Meta-Manager system before the cases have been processed they may be resubmitted for remote diagnosis.

B.1 THE MONITOR VRE STATE USE CASE

Preconditions

The Meta-Manager Node must be in an Active Mode and Node Initialization must be complete before it can participate in State monitoring activities.
Main Flow

The main flow for this use case is a periodic loop that polls all Client Meta-Manager's for state information at a predetermined interval.

M1. Select the first Client Meta-Manager on the Client List
M2. Set the retry counter to an initial value of 1 or greater.
M3. Send a State Query message to the selected Client.
M4. If the Client Meta-Manager responds with valid data in an executable amount of time proceed to M9.
M5. Decrement the retry counter.
M6. If the retry count > 0 go to M3.
M7. Update the Connectivity of the Client to "Unreachable".
M8. If the lost Client is involved in any remote processing, either as the Local or Remote DIN-PACS. invoke the appropriate exception in the Process New Case flow.
M9. Update the state of the selected Client Meta-Manager.
M10. If the current Client Meta-Manager is the last Client on the list wait the specified period and begin again at M1. Otherwise, select the next Client Meta-Manager on the Client List and proceed to step M2.

Subflows

S1. This sub-flow handles the scenario when a Client Meta-Manager sends a State Update to the Master without first receiving a State Query. This scenario can happen for two reasons. First, the Client Meta-Manager may have experienced a drastic state change and it needs to update the Master immediately. For example, when a Radiologist logs off of the system the Master should be
notified that the radiologist is no longer accepting cases. A Client may also send a State Query update because it has not received a State Query message from its Master within an expected time frame. In this case it is trying to reestablish communications with a Master that it has reason to believe is no longer reachable.

S1.1 Set the retry counter to an initial value of 1 or greater.
S1.2 Send a State Update message to the Master.
S1.3 If the Master responds with valid data in an executable amount of time exit this sub-flow.
S1.4 Decrement the retry counter.
S1.5 If the retry count > 0 go to S2.2.
S1.6 Update the Connectivity of the Master to "Unreachable".
S1.7 If the Master has assigned Cases for us to read, cancel the Cases.
S1.8 If we have Cases being read through the Master, reroute those Cases within this domain on a best-effort basis.

S2. This sub-flow handles the scenario a State Query message is received at a Meta-Manager node. State Query messages can come from Masters who are maintaining system state settings or they can come from Command Centers who are collecting state information for display through the CC GUI.

S2.1 Parse the State Query message for requested information.
S2.2 Collect the requested information from within the local domain.
S1.3 Send a State Query Response containing the requested information.

Alternative Flows
None.
REFERENCES


[31] Ralph Martinez, Ph.D., LTC Jay Cook, and William Chimiak, Ph.D., “Design of the U.S. Army Virtual Radiology Environment (USA VRE),” The University of Arizona, Brooke Army Medical Center, and Wake Forest University, 6 September 1997.


