An IIOP Architecture for Web-Enabled Physiological Models

By

Shixin Zhang

B.E., Mechanical Engineering (1992)
B.E. (Minor), Electronics & Computer Technology (1992)
Tsinghua University

Submitted to the Department of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mechanical Engineering at the Massachusetts Institute of Technology

May 2001

© 2001 Massachusetts Institute of Technology
All rights reserved

Signature of Author..............................

Department of Mechanical Engineering
May 1, 2001

Certified by....................................

C. Forbes Dewey, Jr.
Professor of Mechanical Engineering and Bioengineering
Thesis Supervisor

Accepted by....................................

Ain A. Sonin
Chairman, Department Committee on Graduate Students
An IIOP Architecture for Web-Enabled Physiological Models

By

Shixin Zhang

Submitted to the Department of Mechanical Engineering on May 1, 2001 in partial fulfillment of the requirements for the Degree of Master of Science in Mechanical Engineering

ABSTRACT

This thesis developed a specific information architecture to serve complex physiological information models and a means of delivering these models in a manner that allows interactive and distributed use. By redesign of existing models for distributed use, the IIOP architecture provides general access across the Internet; the methods are replicable with many different types of physiological models that produce a variety of results. The concepts and software developed can be reusable by public domain.

This thesis defines and explains the complete architecture for the user interface, the model encapsulation, and the communication layer between the client and server and database by developing several common examples. Using the equivalent of interactive browsers to access remote models and display the results, the IIOP architecture is built up using platform-independent technology such as CORBA, Java and XML. The existing physiological models are first encapsulated by a suitable software language respect to the legacy models. Then CORBA IDL-XML interfaces are built accordingly as a broker interface connecting user interfaces to encapsulating interfaces. Therefore, the standard user interfaces on the browsers are easily built to access these models through the CORBA ORB and the encapsulating interfaces. This interface software is capable of interpreting and displaying very high-level descriptors and model output such that the amount of data required to be transmitted over the Internet is reduced.

Two example models using IIOP architecture are given in detail in this thesis. A 5D interpolation table was created for the cardiovascular model.


Title: Professor of Mechanical Engineering and Bioengineering
# Table of Contents

ABSTRACT ....................................................................................................................... 2

TABLE OF CONTENTS ............................................................................................... 3

LIST OF FIGURES ........................................................................................................ 3

LIST OF TABLES ........................................................................................................ 4

ACKNOWLEDGMENTS ............................................................................................... 5

CHAPTER 1. INTRODUCTION ................................................................................... 6

CHAPTER 2. BACKGROUND ....................................................................................... 8

2.1 HISTORY AND OPPORTUNITY ............................................................................. 8

2.2 TWO EXAMPLE MODELS ................................................................................... 9

CHAPTER 3. NETWORK TECHNOLOGY REVIEW ............................................... 11

3.1 NETWORK TECHNOLOGY AT PRESENT - JAVA, CORBA, AND XML ARE
REDEFINING THE FIELD OF INFORMATION SYSTEM ARCHITECTURE .................. 11

3.1.1 CGI ................................................................................................................. 11

3.1.2 CORBA and Java ........................................................................................... 13

3.1.3 XML .............................................................................................................. 16

3.2 APPLICATIONS AND COMPETITORS OF CORBA/IIOP ............................... 18

3.2.1 Applications of CORBA/IIOP ....................................................................... 18

3.2.2 Java RMI / EJB ............................................................................................. 19

3.2.3 DCOM .......................................................................................................... 20

3.3 ROLE OF CORBA ............................................................................................... 21

CHAPTER 4. THE IIOP ARCHITECTURE FOR WEB-ENABLED
PHYSIOLOGICAL MODELS ....................................................................................... 22

4.1 IIOP ARCHITECTURE ......................................................................................... 22

4.2 IMPLEMENTATIONS ............................................................................................ 24

4.3 ENCAPSULATING TECHNOLOGY ...................................................................... 25

4.4 ENCAPSULATION EXAMPLE - MATLAB MODEL .......................................... 26

4.5 SERVER INITIALIZATION .................................................................................. 27

4.6 CLIENT INITIALIZATION ................................................................................... 28

CHAPTER 5. CARDIOVASCULAR MODEL ......................................................... 29

5.1 INTRODUCTION ................................................................................................. 29

5.2 IMPLEMENTATIONS ........................................................................................... 30

5.2.1 Changes of the Existing Model Programs - Cut off the original GUI, keep the
computation sections, find out the inputs and outputs .......................................... 30

5.2.2 Definition of CORBA IDL Interfaces ............................................................. 30

5.2.3 Generation of Stub (Java in Client) and Skeleton Classes (C++ in Server)
Using the ORB's IDL Compiler ............................................................................. 31

5.2.4 Design and Implementation of the Server's Classes ..................................... 32
5.2.5 Design and Implementation of the Encapsulating Layer's Classes .................. 43
5.2.6 Design and Implementation of the Client Java Applet............................... 43

5.3 Interpolation Design of the Cardiovascular Model .................................. 45
  5.3.1 Construction of a Result Library (An Interpolation Table) ................... 46
  5.3.2 Interpolation in Five Dimensions ......................................................... 46
  5.3.3 Results ..................................................................................................... 49

5.4 Discussion .................................................................................................. 52

CHAPTER 6. CANINE VENTRICULAR CELL MODEL ........................................ 54
  6.1 An Information Architecture for Physiological Models, Clients and Databases .......................................................... 54
  6.2 Client Design .......................................................................................... 55
  6.3 Computation Server ................................................................................ 56
  6.4 Database Server / JDBC Class ................................................................. 59
  6.5 Discussion and Conclusions .................................................................... 60

CHAPTER 7. CONCLUSIONS ............................................................................ 62

ACRONYMS ......................................................................................................... 63

BIBLIOGRAPHY ................................................................................................ 64
List of Figures

Figure 1: The current X-windows user interface for the cardiovascular model based on MATLAB proprietary technology, 5 ............................................................... 12
Figure 2. Action potential model for canine ventricular cell (WRJ3), professor Zheng Li, University of Washington. Reference: http://nsr.bioeng.washington.edu/Software/DEMO/CANINE-AP/ .......... 14
Figure 3: A web-based architecture in CGI style for physiological model .................. 15
Figure 4: Evolution of Web architecture ........................................................................ 16
Figure 5: CORBA is the glue that binds disparate programming technologies, http://www.omg.org/library/wpjava.html ..................................................... 17
Figure 6: CORBA provides the bridge between different objects, http://www.omg.org/library/wpjava.html ................................................................. 18
Figure 7: A simple CORBA- and Java-based architecture ........................................... 19
Figure 8: Web-based Architecture of physiological modeling using Java, XML, and CORBA ......................................................................................................... 26
Figure 9: Architecture design: IIOP Web application architecture .............................. 27
Figure 10: Encapsulating technology ........................................................................... 28
Figure 11: The flowchart of server initialization class ............................................... 30
Figure 12: The flowchart of client initialization class .................................................. 34
Figure 13: The picture shows the first GUI when you run the cardiovascular model; there are two functions provided: one is simulation function and another is estimation function ............................................................................................................... 37
Figure 14: The current X-windows user interface for the cardiovascular model based on MATLAB proprietary technology ........................................... 38
Figure 15: The X-window for the estimation function ................................................. 39
Figure 16: The structure of the original MATLAB files of the cardiovascular model .... 40
Figure 17: User interface for the new cardiovascular model based on Java applet technology and IIOP architecture ................................................................. 44
Figure 18: The results response to the remote user request ........................................ 45
Figure 19: Rip out existing MATLAB model and replace with interpolation table ....... 45
Figure 20: Labeling of points used in the two-dimensional interpolation routines 47........ 47
Figure 21: Comparison of the result from interpolation with that from calculation (Pressure vs time) .......................................................................................... 49
Figure 22: Comparison of the result from interpolation with that from calculation (Velocity vs time) ......................................................................................... 50
Figure 23: Many more comparisons of interpolation results and the real results ........ 51
Figure 24: An information architecture for physiological models, clients and databases 54
Figure 25: The client Java applet window of canine ventricular cell model ................. 55
Figure 26: Details of the internal architecture of the computation server ................. 57
List of Tables

Table 1: Comparison of CORBA and DCOM\textsuperscript{44} ................................................................. 23
Table 2: Compare CORBA and DCOM from technical viewpoint\textsuperscript{44} ........................................ 24
Table 3: The number of tabulated points of each variable ................................................................. 46
Acknowledgments

Throughout this research I have received invaluable advice and support from several colleagues in our research group. I really appreciate their help.

I would like to take this opportunity first to thank Professor Forbes Dewey for his insight and guidance, continuing advice and support; Professor Roger D. Kamm and Ms Grace, for they providing me their complete model and data, also for their kind and helpful ideas; Professor Zheng Li, for his model and big help in programming; Ben Fu, for his thorough assistance and collaboration on numerous projects, for his databases and coding as well; Yuan Cheng, for giving many ideas and technique consulting; Jeannette Stephenson for technical assistance in writing the demonstration documents and developing our Website; Williams Chuang and Orion Richardson, for their assistance and advice on the approach; Ngon Dao and Patrick McCormick, for building the foundations of this project; and Donna Wilker, for her kindly administrative support.

I acknowledge the support of the National Simulation Resource Center at the University of Washington, which is supported by the National Institutes of Health, the International Consortium for Medical Imaging Technology, and the Defense Advanced Research Projects Agency. Our thanks also go to Raimond Winslow for assistance in the development of the original model of the WRJ3 cell computation.

Great thanks goes to Object Oriented Concepts, Inc. for their ORBacus CORBA development environment freely available on the Web, and to IBM, for providing the free Java developing tools, VisualAge for Java, Entry Editions.

Finally, I thank my family for their support over the past two years. Thanks to my Mom and Dad, and thanks to my wife, Xuhong, for making this experience a wonderful and happy one.
Chapter 1. Introduction

As we know, models of human physiology are at the heart of clinical prediction and the teaching process in biomedical engineering. These Models, developed over past 30 years, have become not only quantitative but also computationally intensive. The power of these models to illustrate and predict clinically relevant physiology has also become enormously important in health care delivery. But in most current embodiments, the user must sit at the same computer upon which the model runs or use X-window to simulate the environment.

The comprehensiveness and complexity of physiological models will increase dramatically during the next decade. Advances in computer power, biological pathway information, information on molecular mechanisms, and new genetic data are all driving factors for these changes. The physiological models of today are considerably more complex for two reasons. First, the questions being asked are much more demanding. Second, the level of interdependence of the component models has increased dramatically in order to serve complex physiological information models. Frequently the variety of models required to answer a single physiological question exceed the expertise of any individual group of researchers, and one is required to interface different models from many different sources with different local design rules.

What is needed is a specific information architecture to serve complex physiological information models and a means of delivering these models in a manner that allows interactive and distributed use - the solution we should develop allows the reuse of existing models for distributed use, such as an Internet application. To find a means of delivering these models in a manner that allows interactive and distributed use, this requires: redesign of existing models for distributed use; providing general access across the Internet; making these models' power available to the students; the methods should be replicable with many different types of physiological models that produce a variety of results: graphical, numerical, and images (including moving images); The concepts and software developed can be reusable by public domain. This solution should be also a precursor to a large physiome project.

This need can be met by a web-based architecture that uses the equivalent of interactive browsers such as Netscape and Microsoft Explorer to access remote models and display the results. This access route can be built up using platform-independent technology such as HTML, XML, Java and CORBA. In a nutshell, the existing physiological modeling must first be encapsulated by a suitable software language, such as Java and C++ with respect to the legacy language and application. Then CORBA IDL-XML interfaces are built accordingly as a broker interface connecting user interfaces to encapsulating interfaces. Therefore, the standard user interfaces on the browsers can be easily built to access these models through the CORBA ORB and the encapsulating interfaces. This interface software is capable of interpreting and displaying very high-level descriptors and model output such that the amount of data required to be transmitted over the Internet is reduced.

The web-enabled information architecture and examples are given in detail in this thesis. This thesis defines and explains a complete architecture for the user interface, the
model encapsulation, and the communication layer between the client and server and database by developing several common examples.
Chapter 2. Background

2.1 History and Opportunity

Numerical simulation in science and engineering has progressed enormously since the first use of computational models in the 1950s and early 1960s. In many fields of mechanical, electrical, and chemical design, the use of physical experiments to test theoretical models has been postponed until the very end of the design cycle. This has had a profound economic effect on the development of computer chips, automobiles, aircraft, buildings, chemical processes, and mechanical devices of all types.

A similar revolution is coming in the fields influenced by bioengineering. The history of modeling research of human physiology goes back to the early 1960s. Our early pioneers, with engineering experience in more traditional disciplines such as fluid mechanics, electronics, materials science, and control theory, brought their disciplines to bear in research, teaching and clinic data analyzing. As clinic prediction developed, quantitative engineering models began to be used to explain the physiology. Among the concepts illustrated with simple models included wave propagation in the arteries, wave reflection at arterial branches, oscillating flow in a tube (Wormersley flow), countercurrent ion exchange in the nephrons of the kidney, the Krogh model of oxygen diffusion in tissue, and the Horst model of airway bifurcation.

The human physiology we study today remains unchanged, but the sophistication of our quantitative understanding of the physiology as well as our means of illustrating the concepts has evolved considerably. The capability to predict and model human physiology is advancing rapidly. These models fuel our ability to design everything from prosthetic devices to specialized pharmaceuticals. Ultimately, they will become powerful tools in managing medical therapy and treatment on an individualized basis, taking into account current condition, personal sensitivities to different drugs, and genetic background. The models will also yield new insight into the interaction of many individual factors in predicting outcomes for proposed therapies.

The models of today are more inclusive than the original simple linear uncoupled models. For examples, airway branching can be asymmetric and stochastic. Blood flow through curved vessels can be accurately calculated. Multiple solutes can be tracked in renal models. Left ventricle stroke volume can be derived from segmented ultrasound images. The effects of attachment proteins in mediating tissue response and leukocyte behavior can be demonstrated. In these and many other examples, the mathematical models themselves can be solved rapidly using the formidable computing power that is so ubiquitous today.

Facing to these existing powerful models, what we ask today is how to allow more doctors or researchers to access and use interactively these models and easily to get a "hands-on" appreciation of the changes that can occur when the physiological parameters are altered. That is, we need these models to become more open and reusable resources.

The key ingredient that remains to be supplied is a successful method of making these models power available to the remote users. Generally, the modeler is familiar with the computer language in which the model is written but is unfamiliar with how to
provide general access across the Internet. In most current embodiments, the user must sit at the same computer upon which the model runs. Solving communications and user interface problems for the first (or second or third) time on an ad hoc basis is very time consuming and frequently produces a fragile product. When the developer, leaves the laboratory, the model delivery mechanisms break and remain unrepairable because they were built as an afterthought and were not properly designed or documented.

Today, we have the opportunity and ability to begin the rational design of software systems to deliver modeling capabilities. The combination of the Internet, modern database technology, and object-oriented programming paradigms can be used effectively to build the infrastructure we need. Coupling this to the formidable computing power available today will provide the ability to deliver these models to everyone online.

This thesis is targeted to the development of information delivery software and architecture that will serve a wide variety of physiological models that can be used in biomedical engineering and clinic analyzing. With an appropriate software infrastructure, it should be very easy for a person developing a model to make it available to remote users over the Internet. The base network technology and industry software standards exist\(^5\), but the appropriate problem definition and software application programs are not currently available.

The following section introduces two example models of physiology that we will support on the Internet, and discusses the goal for our project.

### 2.2 Two Example Models

We have centered our project on two specific use cases where there are existing models that either use X-window interface or have an interface that is extremely limited. Our experience is that having concrete examples that range across the spectrum of possible applications is the best way to insure that the software architecture and its implementation will meet the general needs of users.

Details of each of these two example models and their current status are given below.


A non-linear, distributed model of the arterial system and heart has been developed which is based upon a numerical solution of the one-dimensional equations of motion in a geometrically accurate branching network of the arterial system. Inputs to the model are hemodynamic parameters such as Systemic Venous Resistance (SVR) which are critical parameters to evaluate the cardiovascular system but can usually be gained only through invasive measurements, outputs of the model are pressure and flow waveforms at all locations of arterial system.\(^3\) This computational model, which is written by using MATLAB and C language, is used to create a solution library consisting of an extensive collection of peripheral pressure and flow traces, each corresponding to a different set of system parameters. Then, after measuring pressure and/or velocity profiles at any one location of a patient’s arterial system noninvasively, the library can be used to give the estimation of the patient’s several hemodynamic parameters through a set
of parameter estimation routines. Thus, the painful invasive measurements for these parameters are not needed now.\textsuperscript{3,6} Figure 1 is a representation of the current user interface of the model. It is written using the proprietary mathematical package MATLAB and works only under the X-windows system. We shall define a general interface for the model that can be executed under conventional web browser technology on all computers, and change the model so it can be executed using a non-proprietary compiled and freely distributable executable. The connection software between the model and the browser will meet the same design rules as all the other models that we support.

Figure 1: The current X-windows user interface for the cardiovascular model based on MATLAB proprietary technology\textsuperscript{1,5}

where:

\begin{align*}
HR &= \text{Heart Rate (beats/min)}; \\
EDV &= \text{End Diastolic Volume (ml);} \\
SVR &= \text{Systemic Vascular Resistance (dyn/cm}^5\text{sec);} \\
ELV\text{Max} &= \text{Left Ventricle Elasticity (Contractility) (dyn/cm}^5\text{);} \\
\Omega &= \text{Ejection Period (Time systole/Time total);} \\
CO &= \text{Cardiac Output (L/min);} \\
SV &= \text{Stroke Volume (ml)}
\end{align*}

b. Action Potential Model for Canine Ventricular Cell (WRJ3), Professor Zheng Li, University of Washington
This Canine Ventricular Cell Model is one that integrates the calcium independent transient outward current Ito1 model into the Winslow-Rice-Jafri canine ventricular cell model. The Ito1 model shown below was recently developed by Joseph L. Greenstein. For a detailed description of this model, please see the online data supplement for this article, which can be found at http://www.circresaha.org.

The model of canine Ito1 is formulated as the combination of these Kv4.3 and Kv1.4 currents, and is incorporated in the Winslow-Rice-Jafri canine ventricular cell model. Two models are implemented: one is for Ito1 alone, and the other one integrates the Ito1 channel model into a cell model (WRJ3). The implementation of the two models are based on the above two papers and the integrative cell model corresponds to the Version 3 of the Canine Ventricular Cell Model originally developed at Dr. Raimond Winslow's laboratory. The group of Professor Zheng wrote model description files (ito1.mod and wrj3.mod) containing the published equations, model parameter values and initial conditions. The WRJ3 model program was then generated automatically with their new equation-based model configuration tool, J2XSIM. With J2XSIM, users can easily modify the model equations (ito1.mod and wrj3.mod) and generate new models without programming. Finally, their graphic simulation interface (X-window), XSIM, provides users easy and flexible runtime control (e.g., change parameter values, display of results, data analysis, sensitivity analysis, etc). In order to run the model, one needs to have a X-server installed and running on their system. One use of the model is the simulation of heart failure. The model is implemented so that two different conditions, normal and congestive heart failure (CHF), can be simulated by toggling a switch.

Our group used this model to develop an information architecture for physiological models, clients and databases to support complex physiological information models. In this information system, I shall also define a general interface for the model that can be executed under conventional web browser technology on all computers, and change the model so it can be executed using a non-proprietary compiled and freely distributable executable. The connection software between the model and the browser will meet the same design rules as all the other models that we support.

The following sections provide a summary of the technological building blocks that can be used to construct a modern solution, including the citation of prior efforts that attempted to address these needs and comparison of CGI and CORBA methods. In Chapter 4, I detail the proposed software architecture, and that I have completed successfully. Chapter 5 and 6 describe the details that the changes of each example model are made by using the IIOP architecture that can be used in supporting other models in a simple and standard manner.
Figure 2. Action potential model for canine ventricular cell (WRJ3), professor Zheng Li, University of Washington. Reference: http://nsr.bioeng.washington.edu/Software/DEMO/CANINE-AP/
Chapter 3. Network Technology Review

3.1 Network Technology at Present - Java, CORBA, and XML are redefining the field of information system architecture

Since we need to develop a solution that allows reusing these existing models for distributed use, we must mention the Web development in recent years. As the subtitle depicts, Java, CORBA, and XML, which are redefining the field of information system architecture, are the newest software technologies in recent several years. This architecture is considered more and more in many fields and applications. Here we start with basic knowledge of CGI, Java, CORBA, and XML to introduce the architecture.

3.1.1 CGI

The Web began as a giant unidirectional medium for publishing and broadcasting static electronic documents. Basically the Web was a giant URL-based file server. In late 1995, the Web evolved into a more interactive medium with the introduction of three-tier client/server, CGI style. CGI is now known to access every known server environment.

Figure 3, a physiological modeling which was distributed using Java and CGI, illustrates a typical CGI structure. This example uses Java applet communicating with CGI program, MATLAB.exe, which takes charge of transferring data between the programs in the background and Java applet in the client browser. The CGI programs deal with the request from Java applet and send back the result response to the requester. (This CGI model was developed by myself during the first year of my study in MIT.)

![Figure 3: A web-based architecture in CGI style for physiological model](image-url)
Although CGI has been an important technology and is appropriate in many circumstances, the Web with CGI is a slow, cumbersome, and stateless protocol. As the CGI scripts run on the server side, for servers that take thousands of hits a day, this can really slow down the system. CGI launches a new process to service each incoming client request. It can be counterintuitive for moderately to highly complex distributed applications. As a special example, figure 3, it needs 10 minutes to deal with one user request. CGI is unsuitable for writing modern client server applications and is no match for object-oriented Java clients. Furthermore, the CGI and Fast CGI protocols lack CORBA’s scalability for applications and applets with interactive user interfaces. While adaptable to many information retrieval and display applications, the retransmittal and regeneration of much of the user interface for each client/server invocation in CGI does not meet the responsiveness requirements of most interactive user interfaces. From an adaptability perspective, the implementation of a CGI script series to support a Web-based GUI to an enterprise server is a poor solution – a less obvious but important fact for enterprise applications. While CGI may be a common solution for data distribution to a Web browser, it is not a preferable middleware solution for interserver communication. To get around this limitation many vendors have introduced server extensions that have, as yet, ended up being nonstandard and, sometimes, platform-specific extensions.

The next generation of the Web - in its various forms of the Internet, Intranet, and Extranets - will evolve into a full-blown client server medium that can run business applications. The current HTTP/CGI paradigm cannot meet these requirements. To move to this next step, the Web needs distributed objects. The following figure shows this evolution.13

![Figure 4: Evolution of Web architecture](image-url)
3.1.2 CORBA and Java

One approach to creating this king of Web is with CORBA and Java. CORBA, or Component Object Request Broker Architecture, is an open standard set forth by the Object Management Group, OMG\textsuperscript{14}, for component-level language-independent communication. The CORBA architecture lets distributed objects communicate by use of object request brokers, or ORBs. It includes: RPC mechanism (remote procedure/process call); Naming service and Trading service (object service); IDL2XXX: reflection between different languages; Mutually operation protocol. CORBA - which is, in itself, a complete distributed object platform - extends the reach of Java application across networks, languages, component boundaries, and operating systems. Java on its part is a mobile object system and allows CORBA objects to run on everything from mainframes to network computers to mobile phones. Java's bytecode behavior simplifies distribution of code in large CORBA systems. Java, with its built-in multithreading, garbage collection, and error management, makes it easier to write robust networked objects. The bottom line is both are technologies that compliment each other well. Java begins where CORBA leaves off. CORBA deals with network transparency and Java deals with implementation transparency.

As the most widely-used middleware specification, CORBA is an integration technology, not a programming technology. The class of CORBA is neither part of an operating system nor an application, but is used to link together the various parts of a distributed application spread across geographically separated computers. It is specifically designed to be the glue that binds disparate programming technologies together. It does not exist as a point in the programming space; by design, it occupies the spaces between the peaks representing individual languages (see Figure 5)\textsuperscript{15}.

![Figure 5: CORBA is the glue that binds disparate programming technologies, http://www.omg.org/library/wpjava.html](http://www.omg.org/library/wpjava.html)

From our standpoint, one of CORBA's most valuable traits is that it is designed to be language and platform independent. CORBA is a standard and many different ORB implementations are available. Each of these provides bindings for one or more languages so that CORBA's functionality can be accessed from that language. For
example, Visigenic’s Visibroker\textsuperscript{16} ORB provides bindings for C++, Java, and COBOL. IBM’s, on the other hand, has bindings for C, C++, Ada, Java, COM, and COBOL. As one can see with IBM’s ORB\textsuperscript{17}, these bindings can even be to other component architectures like COM. The cross-platform and cross-language nature of CORBA is definitely its most powerful feature. For our purposes, where models are naturally distributed and are implemented with a variety of different languages, CORBA, we’ll show, is a powerful implementation tool.

When a Java client uses CORBA technology to communicate with a C++ object, for example, both the C++ programmer and the Java programmer work completely within their respective language environments. The CORBA ORB presents the Java client with a Java stub interface and the C++ programmer with a C++ skeleton interface (see Figure 6).\textsuperscript{15} CORBA takes care of the cross-language issues automatically. This picture reflects the fact that CORBA is specifically designed as an integration technology, not a programming technology.

![Figure 6: CORBA provides the bridge between different objects, http://www.omg.org/library/wpjava.html](http://www.omg.org/library/wpjava.html)

The medium that CORBA uses to perform this integration is OMG IDL (Interface Definition Language). IDL isn’t a programming language. It describes interfaces between distributed components. It doesn’t depend on any particular programming language technology. From IDL interface descriptions, an ORB product automatically generates code in the language of your choice to effect integration and distribution—the "glue" that connects components and manages communication between them.

In contrast with CGI, as figure 7 suggests,\textsuperscript{13} creating a CORBA-based API to an enterprise server results in Web-based GUI and other enterprise application accessibility for servers. Figure 7 illustrates the architectural flexibility and utility of a CORBA-based middleware solution by depicting a downloaded applet communicating with an enterprise server while that server interacts with a second non-Web based enterprise application. In general, CORBA is an adaptable inter-process communication infrastructure appropriate for most distributed systems—including the Web—while CGI is a protocol appropriate for less responsive and complex subsets of client/server interactions.
Figure 7 illustrates the architecture of a simple application built on CORBA and Java using a Web browser to provide the client-side Java virtual machine. As with any Java-enabled Web page, when the client browser parses the Web page’s applet tag, the lazy Java class loader downloads the minimum applet class files necessary to support the executing Java applet. Java class files (providing client-side ORB runtime functionality) may be included in this download sequence. Once the applet is initialized and executing on the client machine, it establishes an ORB-based connection to one or more ORB-enabled servers residing on the same host as the Web server that served up the applet. With the applet executing and connected to the server, Java objects in the applet may begin invoking functions on any ORB-enabled objects in the server application.

A number of ways exist in which CORBA objects can be accessed from Web-based applications.
Java applets are capable of directly accessing CORBA objects via IIOP. They can be downloaded directly as Web-based applications. A number of Java-based ORBS are available on the market. By introducing CORBA communication into a Java applet, arbitrary CORBA services can be accessed directly. These services can be developed in any language supported by CORBA or on top of any CORBA product that supports IIOP.

Web servers from Netscape and Oracle are beginning to support IIOP directly. This means, in addition to supporting HTTP, FTP access, and news groups access, they will be capable of accessing any CORBA object capable of supporting IIOP.

3.1.3 XML

So far, we haven’t mentioned XML. Then, what is XML, why it is worth we using it in physiological modeling? If you know HTML and SGML, you can easily understand it.

How can we reuse or combine different models and link multi databases in a meaningful way? Encapsulation is typically achieved using a strict software interface that continues to support a promised functionality while letting the underlying implementation change.

There are a number of communication protocols that are general-purpose enough to be used in a wide range of software applications. One of the most promising object communication protocols and data encapsulation mechanisms is CORBA and XML cooperation.\(^\text{18}\)

The basic means of exchanging data is to use structured text. In order to achieve this goal there must be a standard framework for describing the structure of the text. The eXtensible Markup Language (XML)\(^\text{19}\) standard has solidly emerged as the technology to solve this problem. XML is admirably well suited to the task of transferring data with the object-oriented structure required. In addition most databases can export directly to XML, and the results can be converted to objects in other languages. IBM's Alphaworks\(^\text{20}\) has provided alpha versions of their XML Lightweight Extractor (XLE) for free usage. Given an XML DTD, XLE allows users to annotate the DTD to associate various components with underlying data sources (databases). When requested, XLE will extract data from the data sources and assembles the data into XML documents conforming to the DTD. The mapping mechanisms in the annotated DTD allow XLE to assemble an XML document from relational tables related by foreign-key relationships or more advanced relationships. This technology is especially well suited to our design, as it melds the power of object-relational databases with the flexibility of XML as a data exchange medium.

As it is defined by the World Wide Web Consortium (W3C), the extensible Markup Language (XML) is a simplified subset of SGML specially designed for Web applications.\(^\text{19}\) Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. This subset retains the key SGML advantages of extensibility, structure, and validation in a language that is designed to be vastly easier to learn, use, and implement than full SGML.
XML is one solution to the problem of HTML’s lack of structure. XML differs from HTML in three major respects:

1. Information providers can define new tag and attribute names at will.
2. Document structures can be nested to any level of complexity.
3. Any XML document can contain an optional description of its grammar for use by applications that need to perform structural validation.

XML has been designed for maximum expressive power, maximum teachability, and maximum ease of implementation. It is being enthusiastically embraced in many application domains because there are a lot of applications that need to store data intended for human use, but which it will be useful to manipulate by machine. Many data and elements specification can easily be converted to XML, as can generic SGML documents and documents generated from databases.

XML and middleware are complimentary technologies. XML is intended for the storage and manipulation of text making up humane-readable documents like Web pages, while middleware solutions like CORBA tie together cooperating computer applications exchanging transient data that will probably never be directly read by anyone. Neither of these technologies will replace the other, but instead they will increasingly be used together.

In fact, many advanced experts have begun to use this information system architecture constructed by using Java, CORBA, and XML. Such as, Professor C. Forbes Dewey, Massachusetts Institute of Technology; V. J. Jagannathan and K. Srinivas, West Virginia University; European Bioinformatics Institute (EBI) and so on.

3.2 Applications and Competitors of CORBA/IIOP

3.2.1 Applications of CORBA/IIOP

Many research groups used or are considering using CORBA to implement many specific systems and functions, here lists only several impressive ones of them:

- Accessing to biological databases
  EBI used CORBA server to access EMBI sequence database;
- Combining with SNMP (CORBA/SNMP switch)
  Bell-lab, IBM, developed many research in CORBA/SNMP projects;
- Combining with SQL
  The packages of OAS in Oracle support CORBA event service;
  Oracle puts the object of CORBA to database;
  IONA has OTM to support ORB;
  As far as IBM DB2, Visualage has package to support CORBA;
- Improving the distribute management
  Building Telecommunications Management Applications with CORBA;
- Using CORBA/JAVA combination
  The Java Community Process Program and the Object Management Group increase cooperation;
As we discussed above, CORBA is an object-oriented client/server platform, it has many advantages: portable, open specification independence of vendors, supporting different language and platform and system, avoiding repeat complicate works during the programming of distributed computation, easier and less work than Java RMI. CORBA is using an open, standard, stable, and portable platform to supersede some specific mechanism such as Socket.

Certainly, CORBA is one choice for the distributed computing systems, but and is it the most suitable and are there any other competitors? In any new and big future area, there's always serious competition. So it should be no surprise that CORBA faces many competitors. The competitors have: DEC, DCOM, RPC (remote procedure calls), Shared memory based interaction, Named Pipe communication, Socket level programming, Message Queuing, and Other IPC (inter-process communication) mechanisms. I would like to introduce two more competitive tools in the following sections.

### 3.2.2 Java RMI / EJB

As a very strong candidate of distributed computing, Java also has a beautiful prospect. It is a portability, multi-platform, natively integration with Web browsers, ease for use and object-oriented programming language.

Remote Method Invocation (RMI) enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. A Java technology-based program can make a call on a remote object once it obtains a reference to the remote object, either by looking up the remote object in the bootstrap naming service provided by RMI or by receiving the reference as an argument or a return value. A client can call a remote object in a server, and that server can also be a client of other remote objects. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism.

EJB is the Industry-Backed Server-Side Component Architecture. Since its introduction over two years ago, Enterprise JavaBeans technology has maintained unprecedented momentum among platform providers and enterprise development teams alike. That's because the EJB server-side component model simplifies development of middleware components that are transactional, scalable, and portable. Enterprise JavaBeans servers reduce the complexity of developing middleware by providing automatic support for middleware services such as transactions, security, database connectivity, and more. Enterprise JavaBeans is the server-side component architecture for the J2EE platform. EJB enables rapid and simplified development of distributed, transactional, secure and portable Java applications.

The following technologies are for building enterprise-class server-side applications written in the Java programming language: Enterprise JavaBeans; CORBA
JavaServer Pages; JDBC Java Message Service; XML Transactions; JNDI J2EE Connector; Servlets.

Nevertheless, which one are more suitable for our need? I would like to cite the case: in the competitive world of e-commerce, which middleware architecture is right for you? CORBA or Enterprise Java Beans (EJBs)?

If we were to compare CORBA and EJBs then I'd right suggest that CORBA is an excellent candidate for legacy systems and EJBs are for the new multi-tier e-commerce applications that front-end applications. Also EJB containers are heavyweight application servers, but on the other hand ORBs are lightweight. The CORBA infrastructure ensures a solid foundation for today's distributed enterprise applications.

But in our physiological situation, as I discussed in the previous sections, since we need to deal with many legacy systems and different languages, CORBA has more competitive than Java RMI/EJB42.

3.2.3 DCOM

CORBA's major competitor is the Distributed Component Object Model (DCOM)43 created by Microsoft. Microsoft's product situation in the business market makes them a very real competitor, claiming that COM and DCOM are used in over 150 million systems worldwide.

The similarities and differences of CORBA and DCOM are summarized on a top lever in the following table:44

<table>
<thead>
<tr>
<th>Elements</th>
<th>CORBA</th>
<th>DCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>history</td>
<td>1990</td>
<td>1996.</td>
</tr>
<tr>
<td>core of architecture</td>
<td>Object Request Broker</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>language support</td>
<td>multiple languages, such as Java, VB, C++, etc.</td>
<td>multiple languages, such as Java, VB, C++, etc.</td>
</tr>
<tr>
<td>platform support</td>
<td>support Microsoft and non Microsoft operating systems: Unix, MVS, MS-DOS, OS/2, Sun OS, AS 400, Open VMS, Mac OS, Microsoft platforms, etc.</td>
<td>available on Windows NT and Windows 95B,</td>
</tr>
<tr>
<td>communication method</td>
<td>object-oriented RPC-style communications the client stub is called the stub the server stub is called the skeleton</td>
<td>object-oriented RPC-style communications the client stub is called the proxy the server stub is called the stub</td>
</tr>
</tbody>
</table>
Many researchers also compare these two distributed models from technical viewpoint.\(^{44}\)

**Table 2: Compare CORBA and DCOM from technical viewpoint\(^{44}\)**

<table>
<thead>
<tr>
<th>PROs</th>
<th>CORBA</th>
<th>DCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Broad platform support;</td>
<td>2. Strong programming language support;</td>
<td>1. High-quality development tools;</td>
</tr>
<tr>
<td>3. Local/remote transparency;</td>
<td>4. Static or dynamic method invocation</td>
<td>2. Handy Microsoft wizards;</td>
</tr>
<tr>
<td>5. Implementation inheritance;</td>
<td>6. Strong security;</td>
<td>3. Large selection of commercially available ActiveX components;</td>
</tr>
<tr>
<td>7. Reliable cross-ORB behavior;</td>
<td>8. Multithreaded server support;</td>
<td>4. Static or dynamic interface support;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6. Includes multithreaded server support;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONS</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Proprietary extensions to CORBA, vendors may add nonstandard features that may sacrifice interoperability and portability;</td>
<td>1. Limited operating system availability;</td>
<td></td>
</tr>
<tr>
<td>2. Lack of full specification support;</td>
<td>3. Large command sets;</td>
<td>2. Interface inheritance only;</td>
</tr>
<tr>
<td>4. Nonstandard command sets;</td>
<td>5. Lack of development tools;</td>
<td>3. Different interfaces required for static and dynamic invocation;</td>
</tr>
<tr>
<td>6. Less-than-straightforward ORB-to-ORB interoperation;</td>
<td></td>
<td>4. Initiation of COM libraries required for multithreading;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Lack of robust fault-tolerance and load-balancing features;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6. Steepler-than-expected learning curve;</td>
</tr>
</tbody>
</table>

From these two tables above, we know DCOM can only be used at windows platform at present; that is the big obstacle that prevents us from choosing DCOM in our project. Therefore, the choice of CORBA/IOP to develop the architecture for Web-enabled physiological models is the most suitable for us.

### 3.3 Role of CORBA

The legacy system usually was written by using old programming language, such as Fortran, Basic Pascal and so on, or is written by a special tools software, like
MATLAB and something else; Models need a data interchange environment. Then we need to find a bridge language to connect or encapsulate the existing models first. CORBA is one of the best choices. From the previous sections, we can see, CORBA provides bridge between many different languages, such as C++, JAVA, COBOL, PASCAL, PERL. And the biggest function of CORBA is its connection between the objects wherever in client/server. This feature is the development direction of tomorrow web architecture. We choose CORBA is also because it can be easy to reuse for other database and other objects so using CORBA we can easy develop a whole biomedical modeling network systems. The portable, platform-independent and object-oriented CORBA is exactly suitable for our need.

But we choose CORBA/IIOP, which is a lightweight server architecture, we only use it as a transport layer and a bridge connecting models with server, server with client. We won't design any class of CORBA itself.

The following sections will discuss detail about the IIOP architecture for Web-enabled physiological models and the implementations.
Chapter 4. The IIOP Architecture for Web-Enabled Physiological Models

4.1 IIOP Architecture

As we discussed in the previous sections, our need can be solved using a web-based architecture that uses the equivalent of interactive browsers to access remote models and display the results on the browser. This access route can be built up using platform-independent language and technology such as JAVA, CORBA and XML. The information system architecture I developed as figure 8 and figure 9 illustrate.

![Figure 8: Web-based Architecture of physiological modeling using Java, XML, and CORBA](image)

The data flow is: first, the client user loads the Java applet to his Web browser using HTTP protocol; then the applet sends a request to CORBA-ORB server program through IIOP protocol; the server program communicates with the encapsulated models and then sends back the results from models to Java applet in the client side through IIOP or HTTP. From this diagram we can find it is easy to build up an integrated system by using CORBA-ORB technology, such as the same ORB can be used to communicate with databases and other model objects without any change of the whole architecture. Another advantage of this architecture is that we can shift the legacy model to a new model (for example, a library), without any change of the architecture. The client user will never realize any difference. This process is just like buckling on/off a new chain on the backstage.

In this architecture, first, we need to build up an encapsulating layer for existing model with a suitable technology, according to the original language and application situation. Second, since we use IIOP protocol, we need to define CORBA IDL interfaces for each mode as a broker interface which connecting the client Java applet to the encapsulating layers. Third, we need to build up a standard user interface that will be loaded on the browser.
Figure 9: Architecture design: IIOP Web application architecture
4.2 Implementations

Implementation of the route of figure 8 and 9 involves the following activities usually:

1. We need to do a bit of changes and operations to the existing model, because it usually has its special GUI that we can’t deliver directly to Internet. So we need to understand which input variables and output variables the model uses and where they are. Then we cut off the GUI section and keep the computation sections.
2. Design and implement the encapsulating layer’s classes for the model. This is a key step and there are many techniques for wrapper coding. I will discuss this in the coming paragraph.
3. According to the input and output variables and structure, we need to design and define CORBA IDL interface.
4. Automatically generate the stub and skeleton classes and interfaces using the ORB’s IDL compiler.
5. Design and implement the server’s classes that implement the functionality defined by the server’s IDL interface and communicate with the encapsulating sections, including the server’s initialization code and the intersection of the server’s implementation classes, at the same time linking with the skeleton classes produced by the IDL compiler.
6. Design and implement the client Java applet according to the original GUI of models, and any other functionality necessary but not visible to the client applets and applications. We can write Java applet similarly with the GUI style of original model in order to keep the user interface unchanged.
7. Define XML elements to encapsulate the structure data during the transmitting process, in order to communicate efficiently with applet, databases and other models and enhance HTML at the same time. Complete encoding and decoding classes.

4.3 Encapsulating Technology

Figure 10: Encapsulating technology

Figure 10 indicates that in order to use IIOP architecture, we need to design the wrapper codes for the legacy system. There are many technology of wrapping:
• Using C link libraries provided by original language, such as FORTRAN using "cfortran.h", MATLAB etc;
• Using intersection programs to connect each other, such as .DLL .so or temporary text files; Using the third bridge language, such as Tcl/Tk working with files and programs;
• Using a translator, for example, IDL2XXX; xxx means other language, such as PASCAL, C++, JAVA, PERL;
• Using COM/OLE technology; OLE is Object Linking and Embedding;
• Using some special interface, such as Java RMI, socket.

4.4 Encapsulation Example - MATLAB Model

There are two ways to encapsulate a MATLAB model:
• Use C/C++ language to call the MATLAB engine libraries (C/C++) provided by MATLAB, such as libmx.lib libeng.lib libmat.lib
• Use MATLAB C/C++ compiler (mcc.exe) to convert MATLAB files (.m files) to C/C++ programs first, then link with MATLAB C/C++ Math&Graphics Library

The code of first method is like the following:

```c
#include "engine.h"
class run(...)
{
    Engine *ep; mxArray *input = NULL, *result = NULL;
    double inp[12] = { HR, EMax, EDV, Omega, SVR, PV, EMin, LVVis, ArtWallStif,
                      WindkesselCs, Lenghtscale, TransP };
    ...
    if (!(ep = engOpen("\O"))) {
        fprintf(stderr, "Can't start MATLAB engine\n"); _result = 0; return _result;
    }
    ...
    input = mxCreateDoubleMatrix(1, 12, mxREAL);
    mxSetName(input, "input");
    memcpy((void *)mxGetPr(input), (void *)inp, sizeof(inp));
    engPutArray(ep, input);
    engEvalString(ep, "cd\"C:\MATLABR11\"\work\corba\"");
    engEvalString(ep, "outvalue=flag1(input)\"");
    if ((result = engGetArray(ep,"outvalue")) == NULL)
    {
        ...
    }
    ...
}
```

First create a matrix/array ("input") and open an engine pointer ("ep"), then transfer this array to MATLAB environments; then execute MATLAB command (such as "cd C:\MATLABR11\work\corba" and "outvalue=flag1(input)"), we will get the results in an array "result" using "engGetArray" class.
4.5 Server Initialization

Figure 11 shows simply the flow scheme of server initialization class.

![Flowchart of server initialization class](image)

First initialize the ORB and BOA or POA, then create a new implementation object; after that, export the IOR reference in text format for client application or in HTML for client Java applet; then wait for incoming requests.

For ORBacus 3.2, which uses BOA, the server initialization code in C++ is listed following (extracted from the cardiovascular model project):

```cpp
int main(int argc, char* argv[], char**[])
{
    try {
        // Initialize the ORB and BOA
        CORBA_ORB_var orb = CORBA_ORB_init(argc, argv);
        CORBA_BOA_var boa = orb->BOA_init(argc, argv);

        // Create a new simulation object.
        simulationFlag1_var p=new simulationFlag1Impl(orb);

        // Export the newly created object.
        CORBA_String_var s=orb->object_to_string(p);
        const char* refFile="simulationFlag1.ref";
        ofstream out(refFile);
        out<<s<<endl;
    }
}
```
The initialization code in Java is like (extracted from the canine ventricular cell model project):

```java
public static void main(String[] args) {
    Properties props = System.getProperties();
    props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
    props.put("org.omg.CORBA.ORBSingletonClass", "com.ooc.CORBA.ORBSingleton");
    try {
        // Create ORB and BOA
        ORB orb = ORB.init(args, props);
        BOA boa = ((com.ooc.CORBA.ORB)orb).BOA_init(args, props);
        // Create implementation object
        iServerDB_impl p = new iServerDB_impl();
        // Save reference
        try {
            String ref = orb.object_to_string(p);
            String refFile = "/export/home/sxzhang/pmcd5/iServerDB.ref";
            FileOutputStream file = new FileOutputStream(refFile);
            PrintWriter out = new PrintWriter(file);
            out.println(ref);
            out.flush();
            file.close();
        } catch (IOException ex) {
        }
        // Save reference as html
        try {
            String ref = orb.object_to_string(p);
            String refFile = "iServerDB.html";
            FileOutputStream file = new FileOutputStream(refFile);
            PrintWriter out = new PrintWriter(file);
            out.println("<applet codebase="/classes" code="iServerDB.class" "
                width=500 height=300>");
            out.println("<param name=ior value="" + ref + ">");
            out.println("<param name=org.omg.CORBA.ORBClass ") +
```
If we use ORBacus 4.0.46, which uses POA, then the server code in Java is like the following (extracted from the experimental database project):

```java
public class Server {
    static int run(org.omg.CORBA.ORB orb, String[] args) throws org.omg.CORBA.UserException {
        try {
            String ref = orb.object_to_string(_serverDB);
            String refFile = "ServerDB.ref";
            java.io.FileOutputStream file = new java.io.FileOutputStream(refFile);
            java.io.PrintWriter out = new java.io.PrintWriter(file);
            out.println(ref);
            out.flush();
            file.close();
        } catch(java.io.IOException ex) { ... }
        // Save reference as html
    }
}
```
try {
    String ref = orb.object_to_string(_serverDB);
    String refFile = "ServerDB.html";
    java.io.FileOutputStream file =
        new java.io.FileOutputStream(refFile);
    java.io.PrintWriter out = new java.io.PrintWriter(file);
    out.println("<applet codebase="classes" "+
        "code="_serverDB/Client.class" "+
        "width=500 height=300">);
    out.println("<param name=ior value="" + ref + ">");
    ...  
    out.flush();
    file.close();
}

catch(java.io.IOException ex) { ... }

// Run implementation
manager.activate();
orb.run();
return 0;
}

The main class is very standard program that is the same for each program.

public static void main(String args[]) {
    java.util.Properties props = System.getProperties();
    props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
    props.put("org.omg.CORBA.ORBSingletonClass", "com.ooc.CORBA.ORBSingleton");
    int status = 0;
    org.omg.CORBA.ORB orb = null;
    try {
        orb = org.omg.CORBA.ORB.init(args, props);
        status = run(orb, args);
    } catch(Exception ex) {
        ex.printStackTrace();
        status = 1;
    }
    if(orb != null) {
        // Since the standard ORB.destroy() method is not present in
        // JDK 1.2.x, we must cast to com.ooc.CORBA.ORB so that this
// will compile with all JDK versions
try {
    ((com.ooc.CORBA.ORB)orb).destroy();
} catch(Exception ex) {
    ex.printStackTrace();
    status = 1;
}
System.exit(status);

These initialization codes are similar for each project and can be reused in every case.

4.6 Client Initialization

Figure 12 shows simply the flow scheme of client initialization class.

First initialize the ORB and BOA/POA, read IOR reference file or get IOR parameter from HTML; then create this CORBA object, execute narrow function to build up the communication between client and server object; finally call various functions in server to get results.
The C++ code (ORBacus 3.3) is the following (extracted from canine ventricular cell project):

```cpp
int main(int argc, char* argv[], char**[]) {
  try {
    cout << "Initializing the ORB and BOA" << endl;
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    const char* refFile="iClientDB.ref";
    ifstream in; in.open(refFile);
    if(in.fail()) {
      ... return 1;
    } char s[1000]; in>>s; in.close();
    CORBA::Object_var obj=orb->string_to_object(s);
    assert(!CORBA_is_nil(obj));
    cout << "Connecting to the servers" << endl;
    iClientDB_var theClientDB =iClientDB::_narrow(obj);
    assert(!CORBA_is_nil(theClientDB));
    //implementation code
    ...
    CORBA::Short i_main=theClientDB->CDBmain(file_name);
    ...
    outData=theClientDB->getOutput(file_name);
    ...
  } catch(CORBA::SystemException& ex) { 
    OBPrintException(ex);
    return 1;
  }
  return 0;
}
```

The Java code in ORBacus 3.3 is the following (extracted from cardiovascular model project):

```java
ORB orb = ORB.init(this, new java.util.Properties());
try {
  URL url=new URL(location);
  try {
```
URLConnection conn=url.openConnection();
BufferedReader in=new BufferedReader(new InputStreamReader(conn.getInputStream()));
String ref=in.readLine();
in.close();
getoutput().setText("ref="+ref);
org.omg.CORBA.Object object=orb.string_to_object(ref);
flag1=simulationFlag1Helper.narrow(object);
}
  catch (Exception e) { ... }
}
  catch (MalformedURLException e) { ... }
...

The client initialization code with ORBacus 4.0 is the following:

// Applet initialization
public void init()
{
  String ior = getParameter("ior");
  // Create ORB
  org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(this, null);
  // Create client object
  org.omg.CORBA.Object obj = orb.string_to_object(ior);
  if(obj == null)
    throw new RuntimeException();

  _serverDB_ = ServerDBHelper.narrow(obj);
  ...
  ...
}

We can see these codes are similar to each other; they can be reused for each different case. I will use two example models in the following chapters to discuss more detail about the implementations of the IIOP architecture.
Chapter 5. Cardiovascular Model

5.1 Introduction

As a typical physiological model, we use professor Roger D. Kamm’s project to explain the IIOP Web-based architecture. This project is “A Systemic Circulation Lumped-Parameter Model: Quantitative Physiology: Organ Transport Systems”. The introduction is in the section 2.2 “Two Example Models”.

Figure 13: The picture shows the first GUI when you run the cardiovascular model; there are two functions provided: one is simulation function and another is estimation function.

The original model runs in the following steps: first, in the MATLAB environment, type “gui” which is the main executed file, there will pop up a window (figure 13) which has two buttons of the functions: simulation and estimation. Then, choose one you want to launch, such as the simulation function showing as figure 14; the estimation function showing as figure 15.
Figure 14: The current X-windows user interface for the cardiovascular model based on MATLAB proprietary technology

where:

\[ \text{HR} = \text{Heart Rate (beats/min)}; \]
\[ \text{EDV} = \text{End Diastolic Volume (ml)}; \]
\[ \text{SVR} = \text{Systemic Vascular Resistance (dyn/cm}^5\text{sec)}; \]
\[ \text{ELV}_{\text{Max}} = \text{Left Ventricle Elasticity (Contractility) (dyn/cm}^5\text{)}; \]
\[ \text{Omega} = \text{Ejection Period (TimeSystole/TimeTotal)}; \]
\[ \text{CO} = \text{Cardiac Output (L/min)}; \]
\[ \text{SV} = \text{Stroke Volume (ml)} \]

As figure 14 shows, if we input some parameters, such as HR, EDV, SVR, ELV\_Max and so on, we can get results like CO and SV at the left bottom of the window after clicking “run” button. Then we can click any blue point on the cardiovascular system to show the pressure and velocity profiles of that point.
5.2 Implementations

I will explain how to convert this legacy model to the new Web-enabled model using IIOP architecture from the common seven-implementation steps (see the section 4.2).

5.2.1 Changes of the Existing Model Programs - Cut off the original GUI, keep the computation sections, find out the inputs and outputs Since those X-window interfaces cannot be delivered directly to the browser, we need do a little bit of operations on the model. First we need to understand the structure of the legacy model and find which are the inputs and outputs and where they are. Then we need to disable the GUI sections and keep the computation sections.
Figure 16: The structure of the original MATLAB files of the cardiovascular model
Figure 16 illustrates the structure of the cardiovascular model (MATLAB files). “GUI.m” is the first executed file, and then it loads “g1.m” and “g2.m” which is the simulation class and estimation class respectively. These files are the GUIs designed using MATLAB programming language, which can only run under MATLAB environment, so we need to cut them off. “eval1.m” and “eval2.m” are implementation classes. In the file of “eval1.m”, the classes of flag 2, 3, 4, 6, 7, 8, 9, and 10 are the functions of the buttons of “save”, “load”, “reset”, “quit help”, “finish”, “parameter selection”, “library” and so on. As far as these classes, we need to redesign them and then put them into the Java applet in client side. But the class of flag1 is the “run” button function (i.e. the calculation function) and the class of flag11 is the “plot figure” function, we need to keep both in the server side, and so does as flag 5.

“Net.exe” is the heart program of this model, whose code is about 80 pages long. It is written using C language; basically, the algorithm is solving differential equations and using iteration method. So it is the reason why this model needs tens minutes or several hours to complete the calculation of one case. Of course, we don’t need to spend much time on this “net.c” class, but the only thing we need to do is just hooking it to our server class. Other files in the structure diagram need to be kept on the server side.

We notice here, there are many temporary files in this model, which is designed only for single user to use. So how to deal with these temporary files, i.e., how to convert single user version to multi-users version, it is a little tricky problem. Those files “netout”, “aortic”, etc, are output data files. They are very important results in this model; we need to deal with them carefully. I will discuss those in the section of discussion.

The input variables are hemodynamic parameters, such as Heart Rate (HR), End Diastolic Volume (EDV), Systemic Venous Resistance (SVR), Left Ventricle Elasticity (Contractility) (ELV Max), Ejection Period (Omega) and etc.; Outputs are Cardiac Output (CO), Stroke Volume (SV), pressure and flow waveforms at all locations of arterial system.

5.2.2 Definition of CORBA IDL Interfaces

According to the input and output variables and structure of the model, we need to design and define the IDL interfaces. After analyzed thoroughly the model structure, I summarized into two classes.

One is the computing class - flag 1: the input variables are HR, EMax, EDV, Omega, SVR, PV, EMin, LVVis, ArtWallStif, WindkessalCs, Lengthscale and TransP. The output variables are CO, EDP, SV and the file "netout" (text format data file). I uploaded this “netout” to the memory of the client in order to keep it private for the client user and also because it is results file that the client user needs to keep. Another is the image loading class - flag 11: the input variables are the number of the point (location), the selection of Pressure and Velocity and Area, and the file "netout" (We send this back to the server to be as the inputs for plotting classes); the output variables are the name of the result image file (We transport the image files using HTTP or through IIOP to Java applet.)

According to the inputs and outputs, the IDL definition is:

typedef double netout[100][130];
typedef double size[2];
interface simulationFlag1 {
    short Run(
        in double HR,
        in double Emax,
        in double EDV,
        in double Omega,
        in double SVR,
        in double PV,
        in double Emin,
        in double LVVis,
        in double ArtWallStif,
        in double WindkessalCs,
        in double Lengthscale,
        in double TransP,
        out double CO,
        out double EDP,
        out double SV,
        out netout outArg,
        out size netoutsizex);
    short ok(in netout outArg,
        in size netoutsizex,
        in short point,
        in short PVAnum,
        out long Filenum);
};

5.2.3 Generation of Stub (Java in Client) and Skeleton Classes (C++ in Server) Using the ORB’s IDL Compiler.

Now we can use the ORBacus IDL compiler to compile the above IDL file; just type “idl simulationFlag1.idl” in the command line to produce C++ stub and skeleton classes; or type “jidl simulationFlag1.idl” to produce Java stub and skeleton classes. It produces automatically four files using version 4.0 (C++): simulationFlag1.cpp, simulationFlag1.h, simulationFlag1_skel.h, and simulationFlag1_skel.cpp; if compile to Java version, the files are:
  netoutHelper.java
  netoutHolder.java
  simulationFlag1.java
  simulationFlag1Helper.java
  simulationFlag1Holder.java
  simulationFlag1Operations.java
  simulationFlag1POA.java
  sizeHelper.java
  sizeHolder.java

42
5.2.4 Design and Implementation of the Server's Classes

This step implements the functionality defined by the IDL, including the server's initialization code; any other functionality necessary; the class and interface used to communicate with encapsulating layer. In my solutions, these files are “simulationFlag1_srvimpl.cpp” and “simulationFlag1_srvimpl.h”. The detail codes see the attachment. There are four classes in it:

Main class is the server initialization code;

“Run” class is a class to response the “run” button in the client applet, including the encapsulating class (this layer calls the computing classes of the legacy system).

“Interpolation” class is a new function/model that uses interpolation technique to acquire the results from the results library.

“Ok” class is the plotting class that loads the plotting function of the legacy model.

5.2.5 Design and Implementation of the Encapsulating Layer's Classes

This is an important step as I discussed before (in the section 4.4). We need to build up an encapsulating layer to communicate with the model and intersection implementation classes. As I introduced in previous sections, the better way encapsulating a MATLAB model is using C/C++ to call the MATLAB engine C/C++ libraries, which is provided by MATLAB. The library is “engine.h”. The code is like one showed in the section 4.4 “Encapsulation example – MATLAB model”; the detail code see the attachment. We can create a matrix/array and open an engine pointer first, then transfer this array to MATLAB environments; then execute MATLAB command, we will get the results in an array “netout”. This is the easy and less work way to wrap the legacy model. We don’t need to research thoroughly about the 80 pages code of the original model.

Another way we can use is using MATLAB C/C++ compiler (mcc.exe) to convert MATLAB files (.m files) to C/C++ programs and then link these programs to CORBA server/client programs using MATLAB C/C++ Math library (& Graphics library). But this method needs additional software tools sold by MATLAB or other third company. Comparatively, this method will take more time than the first one.

5.2.6 Design and Implementation of the Client Java Applet

As same function as the original model, this new model with web-based architecture gives out correct results as figure 17 illustrates. We can type the parameters in or use drag bars to choose the values. Clicking the “run” button will call the real calculation model and display the results on the left-bottom of the window; clicking the “interpolation” button will get the results from the interpolation table. If you click the
blue points on the diagram of cardiovascular system after calculating, a window with three choices (pressure, velocity and area) will be popped up; then according to your choice, the program will give out the graphs of the pressure, velocity or area of that point (location), as shown in figure 18. There is an “output window” that I used it to display the intermediate results.

This client Java applet also includes the client initialization code as I listed in the previous sections (section 4.6).

Figure 17: User interface for the new cardiovascular model based on Java applet technology and IIOP architecture
5.3 Interpolation Design of the Cardiovascular Model

After the works of section 5.2, there is still one problem left: the model is not interactive; the legacy model takes several hours/case. Each calculation of our model spent from half an hour to several hours on my Pentium III 400MHz machine. Nobody can wait so long time on the Internet.

Solution:

How to get the result during short online time? My answer is using result library and interpolation technique instead of the real calculation (Figure 17). This solution can save time from several hours to half a minute.

I will discuss the interpolation model (results library) from the following sections.
5.3.1 Construction of a Result Library (An Interpolation Table)

Actually in the legacy system, there are only five valuable variables (after Ms. Grace’s thesis work, four variables are left). So the interpolation table consists of five variables, there are \((5 \times 6 \times 6 \times 16 \times 16) = 45855\) tabulated values/points, listed in the table 3.

<table>
<thead>
<tr>
<th>Variables</th>
<th>The number of tabulated points</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td>5</td>
</tr>
<tr>
<td>SVR</td>
<td>6</td>
</tr>
<tr>
<td>EMax</td>
<td>6</td>
</tr>
<tr>
<td>EDV</td>
<td>16</td>
</tr>
<tr>
<td>Omega</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 3: The number of tabulated points of each variable

5.3.2 Interpolation in Five Dimensions

In our five dimensional interpolation, we seek an estimate of \(y(x_1; x_2; x_3, x_4, x_5)\) from an 5-dimensional grid of tabulated values \(y\) and 5 one-dimensional vectors giving the tabulated values of each of the independent variables \(x_1, x_2, x_3, x_4, x_5\). For clarity, we use only the case of two dimensions to explain the method explicitly\(^{47}\); the cases of five dimensions being analogous in every way.

In two dimensions, we imagine that we are given an array \(x_{1a}[1..m]\), an array \(x_{2a}[1..n]\) and a matrix of functional values \(y_a[1..m][1..n]\). The relation of these input quantities to an underlying function \(y(x_1; x_2)\) is

\[
y_a[j][k] = y(x_{1a}[j]; x_{2a}[k])
\]

We want to estimate, by interpolation, the function \(y\) at some untabulated point \((x_1, x_2)\). Suppose the point \((x_1, x_2)\) falls in a grid square, which the four tabulated points that surround the desired interior point. For convenience, we will number these points from 1 to 4, counterclockwise starting from the lower left (see figure 20). I.e., if

\[
\begin{align*}
x_{1a}[j] & \leq x_1 \leq x_{1a}[j+1] \\
x_{2a}[k] & \leq x_2 \leq x_{2a}[k+1]
\end{align*}
\]

defines \(j\) and \(k\), then

\[
\begin{align*}
y_1 &= y_a[j][k] \\
y_2 &= y_a[j+1][k] \\
y_3 &= y_a[j+1][k+1] \\
y_4 &= y_a[j][k+1]
\end{align*}
\]
The simplest interpolation in two dimensions is bilinear interpolation on the grid square. Its formulas are:

\[
\begin{align*}
t &= (x_1 - x_{1a}[j])/(x_{1a}[j+1] - x_{1a}[j]) \\
u &= (x_2 - x_{2a}[k])/(x_{2a}[k+1] - x_{2a}[k])
\end{align*}
\] (5.4)

(so that \(t\) and \(u\) each lie between 0 and 1), and

\[
y(x_1; x_2) = (1-t)*(1-u)*y_1 + t*(1-u)*y_2 + t*u*y_3 + (1-t)*u*y_4
\] (5.5)

As the interpolating point wanders from grid square to grid square, the interpolated function value changes continuously. However, the gradient of the interpolated function changes discontinuously at the boundaries of each grid square.

Similarly, the five dimensional interpolation formulas are the following. Analogously, define 32 points \(y_0 \sim y_{31}\) first:

\[
\begin{align*}
y_0 &= y_{a[i][j][k][m][n]} \\
y_1 &= y_{a[i][j][k][m][n+1]} \\
y_2 &= y_{a[i][j][k][m+1][n]} \\
y_3 &= y_{a[i][j][k][m+1][n+1]} \\
y_4 &= y_{a[i][j][k+1][m][n]} \\
y_5 &= y_{a[i][j][k+1][m][n+1]} \\
y_6 &= y_{a[i][j][k+1][m+1][n]} \\
y_7 &= y_{a[i][j][k+1][m+1][n+1]}
\end{align*}
\]
The formulas are:

\[
p = \frac{(x_1 - x_{1a[i]})}{(x_{1a[i+1]} - x_{1a[i]})};
\]
\[
q = \frac{(x_2 - x_{2a[j]})}{(x_{2a[j+1]} - x_{2a[j]})};
\]
\[
r = \frac{(x_3 - x_{3a[k]})}{(x_{3a[k+1]} - x_{3a[k]})};
\]
\[
s = \frac{(x_4 - x_{4a[m]})}{(x_{4a[m+1]} - x_{4a[m]})};
\]
\[
t = \frac{(x_5 - x_{5a[n]})}{(x_{5a[n+1]} - x_{5a[n]})};
\]

(5.6)

(p, q, r, s and t each lie between 0 and 1), and

\[
y = (1-p)*(1-q)\cdot(1-r)\cdot(1-s)\cdot(1-t)\cdot y_0 + (1-p)*(1-q)*(1-r)*(1-s)\cdot( t)\cdot y_1 \\
+ (1-p)*(1-q)*(1-r)\cdot( s)\cdot(1-t)\cdot y_2 + (1-p)*(1-q)*(1-r)\cdot( s)\cdot( t)\cdot y_3 \\
+ (1-p)*(1-q)\cdot( r)\cdot(1-s)\cdot(1-t)\cdot y_4 + (1-p)*(1-q)\cdot( r)\cdot(1-s)\cdot( t)\cdot y_5 \\
+ (1-p)*(1-q)\cdot( r)\cdot( s)\cdot(1-t)\cdot y_6 + (1-p)*(1-q)\cdot( r)\cdot( s)\cdot( t)\cdot y_7 \\
+ (1-p)\cdot( q)\cdot(1-r)\cdot(1-s)\cdot(1-t)\cdot y_8 + (1-p)\cdot( q)\cdot(1-r)\cdot(1-s)\cdot( t)\cdot y_9 \\
+ (1-p)\cdot( q)\cdot( r)\cdot(1-s)\cdot(1-t)\cdot y_{10} + (1-p)\cdot( q)\cdot( r)\cdot(1-s)\cdot( t)\cdot y_{11} \\
+ (1-p)\cdot( q)\cdot( r)\cdot( s)\cdot(1-t)\cdot y_{12} + (1-p)\cdot( q)\cdot( r)\cdot( s)\cdot( t)\cdot y_{13} \\
+ (1-p)\cdot( q)\cdot( r)\cdot( s)\cdot(1-t)\cdot y_{14} + (1-p)\cdot( q)\cdot( r)\cdot( s)\cdot( t)\cdot y_{15} \\
+ ( p)\cdot(1-q)\cdot(1-r)\cdot(1-s)\cdot(1-t)\cdot y_{16} + ( p)\cdot(1-q)\cdot(1-r)\cdot(1-s)\cdot( t)\cdot y_{17} \\
+ ( p)\cdot(1-q)\cdot(1-r)\cdot(1-s)\cdot( s)\cdot(1-t)\cdot y_{18} + ( p)\cdot(1-q)\cdot(1-r)\cdot(1-s)\cdot( s)\cdot( t)\cdot y_{19} \\
+ ( p)\cdot(1-q)\cdot(1-r)\cdot(1-s)\cdot( t)\cdot y_{20} + ( p)\cdot(1-q)\cdot(1-r)\cdot(1-s)\cdot( t)\cdot y_{21} \\
+ ( p)\cdot(1-q)\cdot(1-r)\cdot(1-s)\cdot( s)\cdot(1-t)\cdot y_{22} + ( p)\cdot(1-q)\cdot(1-r)\cdot( s)\cdot( t)\cdot y_{23}
\]

(5.7)
We can get, by interpolation, the function $y$ at some untabulated point $(x_1, x_2, x_3, x_4, x_5)$ using equation 5.6, 5.7, 5.8.

Since it isn’t true that each set of variables in our selection domain is suitable for the legacy model, i.e., some tabulated data are void in our cardiovascular model. There is little tricky in searching the tabulated data in our interpolation table. This issue causes many troubles in searching the tabulated data and interpolation. So the class of searching is not easy to design, please see my code in the attachment.

5.3.3 Results

New interpolation model runs in seconds without any change in client or architecture.
Figure 22: Comparison of the result from interpolation with that from calculation (Velocity vs time)

Figure 21 and 22 illustrate that the results from interpolation has good accuracy although the error is about five percent. It is good enough for roughly clinical prediction and observations. Figure 23 shows more results compared between two models.

There are many ways that could increase the accuracy:
- Increase the amount of tabulated values in the library;
- Smooth each curve of one set of variables in the library first, but this will increase the time of computing;
- Use nonlinear interpolation and higher order interpolation.
Figure 23: Many more comparisons of interpolation results and the real results
5.4 Discussion

- **How to deal with the temporary files / How to convert single-user version to multi-user version.**

  First, my way is as I discussed before, I uploaded important results to the memory of client. Since the result file is only about 100kb, it won’t get any trouble. Or, the second choice for me is using temporary file technology like Window system does. That is to say, the results file or temporary files are private and secured; they should be locked so that any other users cannot access and edit them.

- **Image file loading technique**

  In the cardiovascular model, I used Java image loading classes to upload them through HTTP. This way is easier to be implemented. The second way is that use binary file transferring technique, i.e., use sequence of octet of CORBA, and then decode image data in Java applet; this method needs to design a Java class to convert the sequence of octet to image data in client. The third choice is that convert other image format data to raw image data in server, and then transport these large raw data to client applet in the format of the sequence of octet; then use memory copy class to create the image object (Toolkit.createImage(byte[]) class). In the third method, the problem is that it will waste large time to transmit the raw data to client. The code of binary file transferring technique is like the following:

```java
interface simulationFlag
{
    typedef sequence<octet> BinaryFile;
    BinaryFile fetch(in string file_name);
};

virtual BinaryFile* fetch(const char* file_name)
{
    const char* refFile="zsx.jpg";
    int fd;
    fd=open(refFile,O_RDONLY);
    struct stat st;
    fstat(fd,&st);
    CORBA_Octet* buf;
    buf=BinaryFile::allocbuf(st.stsize);
    read(fd,buf,st.stsize);
    BinaryFile* image_seq= new BinaryFile(st.stsize,st.stsize,buf,1);
    close(fd);
    return image_seq;
}
```

In this chapter, I implemented the new Web-enable cardiovascular model and interpolation model as well. As shown in figure 17 and 18, I retained the “run” function (legacy computing core) and added new “interpolation” model in the GUI of Java applet.
Thus, there are two choices for the client user: either loads the legacy computing core (if he has much time online and want more accurate results); or uses the interpolation to get the results in seconds. The error of this result is within five percent roughly. By these implementations of two models, it testifies that the IIOP architecture is suitable for legacy physiological models and lightweight.
Chapter 6. Canine Ventricular Cell model

6.1 An Information Architecture for Physiological Models, Clients and Databases

The introduction of the original canine ventricular cell model design by professor Zheng Li's group is in the section 2.2 “Two Example Models”. Professor Dewey used this model developing “An Information Architecture for Physiological Models, Clients and Databases” 49. The architecture is showing in the figure 24.

The system architecture consists of three essential parts: a networked Client that generates a request for information; a Computation Server that performs the calculation; and a Model Database that provides detailed parameters to the Computation Server to facilitate solution. This system is applied to a complex contemporary model of the electrophysiology of cardiac myocyte cells [the WU model; see reference 7,8 and 9]. The resulting architecture has been demonstrated over the Internet [ICMIT.MIT.EDU/myocyte_model.html].
In this project, the red color shows the work that I did. Similarly, I used IIOP/CORBA architecture implementing the computational server, client and database server. Here I won’t introduce too detail about the steps since it is the same as Chapter 4 and 5. But there are still several differences. One point needs to be mentioned is that I changed the output format of the legacy model from file format stored in hard drive to string format in memory in order to transfer the data between the client and computational server, the computational server and the database server. Moreover, in this project, there are two cases that using our IIOP/CORBA architecture; one is between the client Java applet and the computational server; another is between the computational server and the database server program, in which I treated the computational server as the client side of the database server. The third point is that JDBC class; this project also illustrated that IIOP/CORBA architecture can cooperate very well with JDBC class, which can combine databases together to form an integrated physiological system. This is a good example.

6.2 Client Design

The Client Java applet window (figure 25) is showing all of the key components of the client functionality.
The Client is a "thin client" built around industry-standard browser technology [Netscape Navigator and Microsoft Internet Explorer]. By "thin client" we mean that the Client programs that are executed on the browser are downloaded over the Internet from the Server. The browsers support the Java language that is employed to build the user interface. The user interface receives and checks the syntax of the user commands and provides graphing capabilities for viewing the computational output. An example of the Java-based client window is given in figure 25. Has CORBA stub, Java capabilities for display, parsing and treating the data, and generating graphical output. Also use Java to have an interactive interface (fill-in boxes, some type checking, etc.). You can choose “Model name”, “flag parameters”, “filename of parameters” and “results of variables”; also you can plot a graph of any two variables.

6.3 Computation Server

The server consists of three parts (see Figure 24). The first part is called the Request Interpreter; it handles the communication tasks and parses the input from the Client. The second part is the Computation Engine. It compiles and executes the run-time code from a list of appropriate equations, constants (reaction rates, boundary conditions, initial conditions, etc.), and computational parameters (convergence, time steps, etc.) supplied by the Request Interpreter. The final part is the Results Output module that opens communication channels with the Client and the Case Database and delivers results formatted for storage and display.

The Computation Server is assumed to service many models. This is the most general case and the one implemented here. Most current physiological modeling software runs on a single system and services only one model. In the future, a more general compute server architecture will be the rule rather than the exception.

The example that was used to represent the next generation of such servers is the computational engine developed by the National Simulation Resource at the University of Washington. This engine represents a set of routines for solving coupled ordinary differential equations and compiling the run-time code from a list of the appropriate equations, constants (reaction rates, boundary conditions, initial conditions, etc.), and computational parameters (convergence, time steps, etc.). It has served more than 40 different models over the course of the last five years.49

The specific model that was chosen for implementation was the recent action potential model for the canine cardiac myocyte published recently by Greenfield and colleagues7, 9 and is incorporated into the Winslow-Rice-Jaffri canine ventricular cell model.8 The physiology defined by these two papers is called the WJR3 model. The default case consists of the published equations, model parameter values, and initial conditions. The modeling software and the WRJ3 model may be freely downloaded. See http://nsr.bioeng.washington.edu/Software/DEMO/CANINE-AP.
Figure 26 shows the details of the internal architecture of the Computation Server. The request interpreter handles the construction of the run-time parameters by finding the identity of the model, extracting the default parameters and other information from the Model Database, and replacing the defaults with selected alternate (override) values sent by the Client.

Figure 26 presents the dataflow architecture of the computation server. Originally, the computation engine was a monolithic entity that was coupled directly to a X-windows based user interface. One could run the application only by logging in to the server from a
remote machine that supported X windows. All of the interaction at the remote site was transmitted directly to the server for parsing and interpretation. This was modified to support browser-based client programs that communicate with the server on a peer-to-peer basis using CORBA. Figure 26 illustrates that the same CORBA architecture is used to establish a communication link between the Computation Server and the Model Database. The Model Database can be located on the same physical machine as the Computation Server, or it can be located remotely and the requests and data transmitted over the Internet. In all cases, the same CORBA interface architecture is used. This allows the different parts of the software to be developed and maintained separately. One Model Database could serve more than one Computation Server and vice-versa.

Figure 24 and figure 26 describes a network-based computer architecture for supporting complex physiological models. It is based on the premise that these models will be sufficiently complex and difficult to maintain that they will be run only at selected sites on the Internet. Therefore, access by many researchers will demand robust means for interacting with the Computation Server to initiate solutions and receive results. Current network browser software, including down-loadable Java programs, fits this requirement well. I have chosen to implement the communication path between Client and Computation Server using Java applets, CORBA communication utilities, and XML formatting for the data transmitted between the two. This prototype can be run across the network from any Internet browser by going to ICMIT.MIT.EDU/myocyte_model.html.

To satisfy the data format during the transmitting from legacy models to the computational server, I changed the format of the output of the model. Some code of the changes is listed following:

```c
//I change the output from a file to a string
char * write_parm_simple(run, parm)
RunData *run;
Parm *parm;
{
  int ct, i, j;
  double dat;
  Domain *t;
  char fp[1024];
  char *fp2;
  if (parm->x && !parm->tvary)
    Fatal("Static array output (for %s) not yet implemented\n", parm->name);
  /* t-invariantparms */
  if (!parm->tvary) {
    switch (parm->type) {
      case PUBREAL: dat = run->pubreal[parm->loc-1]; break;
      case PUBINT: dat = run->pubint[parm->loc-1]; break;
      default: Fatal("Invalid parm->type\n");
    }
    sprintf(fp, ">%s = %0.4g;\n", parm->name, dat);
    fp2=(char *)malloc(strlen(fp)+1);
```
strcpy(fp2,fp);
return fp2;
}

/* t-varying parms */
t = run->dom + 0;
ct = parm->x ? parm->x->ct : 1;
fp2=(char *)malloc(t->ct*ct*15+255);
sprintf(fp2, ">%s = {", parm->name);

for (i=0; i<ct->ct; i++) {
    for (j=0; j<ct; j++) {
        sprintf(fp, " %0.6g", parm->tstore[ct*i + j]);
        strcat(fp2,fp);
    }
    sprintf(fp, "\n");
    strcat(fp2,fp);
}

sprintf(fp, "}\n");
strcat(fp2,fp);
return fp2;
}

6.4 Database server / JDBC class

Here a JDBC class was need to be used to communicate with Model database. Above JDBC, I used the same IIOP/architecture between the computational server and model database server. Sample code of JDBC, i.e., the database server implementation, is the following (Thank Ben Fu):

```java
private static String getModelParameters(String modelName)
{
    String cmd = null;
    ResultSet modelResultSet, solutionResultSet, geometricResultSet, computResultSet, initResultSet;
    Connection conn = null;
    String newUrl = "jdbc:informix-sqli://18.66.0.25:1013/canine_ap:INFORMIXSERVER=ICMIT;user=informix;password=*********";
    //The connection login is _hardcoded_ above. This needs to be changed eventually.
    StringBuffer sb = new StringBuffer();
    try
    {
        Class.forName("com.informix.jdbc.IfxDriver");
        //Load Informix JDBC driver
```
catch (Exception e) { ... }
try {
    conn = DriverManager.getConnection(newUrl);
    //Make the connection to the DB thru the URL
}
catch (SQLException e) { ... }
try {
    //IDs to lookup parameters
    int solution_par_id;
    ...  
    Statement stmt = conn.createStatement();
    cmd = "select solution_par_id, geometric_par_id, "
    +"comput_par_id, init_cond_par_id, "
    +"parameter_docs_id, equation_docs_id"
    +"from model where name = " + modelName + "";
    modelResultSet = stmt.executeQuery(cmd);
    modelResultSet.next();  //rc.first() does not work this way
    solution_par_id = modelResultSet.getInt(1);
    ...
    //comput_par
    cmd = "select parameter, value from comput_par where id = "
    +comput_par_id+"";
    computResultSet = stmt.executeQuery(cmd);
    while(computResultSet.next())
    {
        sb.append(computResultSet.getString(1));
        ...
    }
    computResultSet.close();
    ...
    stmt.close();
    return sb.toString();
}
catch (SQLException e) { ... }
return "EXCEPTION ERRORS!";

6.5 Discussion and Conclusions

This project describes a network-based computer architecture for supporting complex physiological models. It is based on the premise that these models will be sufficiently complex and difficult to maintain that they will be run only at selected sites on the Internet. Therefore, access by many researchers will demand robust means for
interacting with the Computation Server to initiate solutions and receive results. Current network browser software, including down-loadable Java programs, fits this requirement well. Our group has chosen to implement the communication path between Client and Computation Server using Java applets, CORBA communication utilities, and XML formatting for the data transmitted between the two. This prototype can be run across the network from any Internet browser by going to ICMIT.MIT.EDU/myocyte_model.html.

Once the primary information on a specific case is passed to the Computation Server, the server must assemble all of the data that are required for the computation. In this implementation, this is achieved with a layered architecture. The first layer parses the commands and assembles the required information by acting as a client of a Model Database that is the repository of all of the information about the many models that are supported by the Computation Server. This design allows the data in the Model Database to be assembled, searched, corrected, and maintained independently of the Computation Server itself. In some cases, the Model Database and Computation Server might reside on the same computer, but that is certainly not necessary.

CORBA, JAVA and XML represent the best tools for developing distributed computing architectures at this time and hopefully the near future. These components could be replaced by others without changing the basic design features that we find attractive. Java is used because it is relatively fast for the tasks that we expect of it, and it is easy to use. It is a complete object-oriented programming language that includes graphics tools and database capabilities. CORBA has managed to survive as an important middleware standard for half a dozen years, and there are both freeware implementations and commercial implementations of the standard. The network transport mechanisms are based on TCP/IP and can be built easily and reused.

I believe that the architecture described above will be useful not only for physiological models but for molecular, genetic, and biological models as well. The medical and biological sciences are already awash in information, and this state can only become more pronounced with time.
Chapter 7. Conclusions

With CORBA being a strong distributed technology and with the growth of the Internet in a stupendous fashion, seeing a large number of resources spent on developing Web-based healthcare and physiological applications using CORBA as the underlying technology is not surprising. Automated memory management, built-in thread support, and applet mobility make Java an ideal choice to implement CORBA client and servers. With a backing of major players like Sun Microsystems, Netscape, and Oracle, and so on, the combined use of Java and CORBA will become commonplace in healthcare and physiological, even in enterprise systems. Because of wide support for IIOP in firewall proxies, availability of free or low cost ORB implementations, and built-in support for IIOP in server-side Web applications such as data blades and Web servers, however, IIOP may not become the standard Internet protocol. But based on recognition of CORBA’s strengths and wide industry acceptance, a growing number of adaptable and robust systems built on CORBA and Java will appear. XML, retaining the key SGML advantages of extensibility, structure and validation in a language, has been designed for maximum expressive power, maximum teach ability, and maximum ease of implementation. Being intended for the storage and manipulation of text making up human-readable documents like Web pages, it is already applied in many fields like DCOM today. As we expected, this technical route using CORBA/IIOP, Java, (JDBC) and XML is replicable with many different types of physiological models that produce a variety of results: graphical, numerical, and images. This concepts and software should be reusable by others. It is a good means of delivering the existing physiological models that allows interactive and distributed use.

This thesis described an IIOP architecture for supporting Web-enabled complex physiological models. Being demonstrated its feasibility by four implementations, IIOP architecture is replicable, Web-enabled and light-weight. This thesis discussed encapsulating technique and others; implemented two projects (four cases): first, built up new Web-enabled models, made them interactive (using interpolation table); second, in the development of information architecture of models, clients and database, the same IIOP architecture was applied to the client and computational server, the computational server and database server. As the request of this thesis project, this IIOP architecture can be as a precursor to a large Physiome project and DARPA project, it would be suitable for many projects such as the one which I will do in the Ph.D. research: experimental data—models—databases.
Acronyms

• IIOP (Internet Inter-ORB Protocol)
• ORB (Object Request Broker)
• CORBA (Common Object Request Broker Architecture)
• IOR (Interoperable Object References)
• IDL (Interface Definition Language)
• HTTP (HyperText Transfer Protocol)
• CGI (Common Gateway Interface)
• TCP/IP (Transmission Control Protocol/Internet Protocol)
• XML (eXtensible Markup Language)
• SGML (Standard Generalized Markup Language)
• SNMP (Simple Network Management Protocol)
• TMN (Telecommunications Management Applications)
• OSI (Open System Interconnection)
• GIS (Geographic Information Systems)
• COM (Component Object Model)
• DCOM (Data Center Operations Management)
• RMI (Remote Method Invocation)
• EJB (Enterprise JavaBeans)
Bibliography


3 C. Forbes Dewey, Jr. Empowering Physiological Modeling in Biomedical Engineering Education, Proposal to be submitted to the Whitaker Foundation Preliminary Description. 4-5, 7, 30 October 1999.


20 XML Lightweight Extractor: another alphaWorks technology,
http://www.alphaworks.ibm.com/tech/xle

28 European Bioinformatics Institute (EBI), CORBA at EBI, http://corba.ebi.ac.uk/
38 Bernd Amann, Integrating GIS Components with Mediators and CORBA (Correct) (3 citations), http://citeseer.nj.nec.com/amann96integrating.html, 1996


http://sern.ucalgary.ca/courses/seng/691/F97/qianw/691-4.html

ORBacus for C++ and Java (Version 3.2). Object Oriented Concepts, Inc. 16-21, 1999

ORBacus for C++ and Java (Version 4.0.4). Object Oriented Concepts, Inc. 17-30, 2000


Zheng LI¹, Ben Fu², Shixin Zhang², Ngon Dao², and C. Forbes Dewey, Jr.². An Information Architecture for Physiological Models, Clients and Databases.¹ Department of Bioengineering, University of Washington, Seattle, WA USA; ² Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. 3-4, 2001