

DESIGN AND IMPLEMENTATION OF A WORLD  
WIDE WEB BASED DISTRIBUTED  
COMPUTING MODEL

By

WENXIA PENG

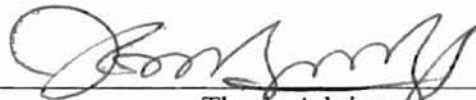
Master of Arts  
Peking University  
Beijing, China  
1995

Bachelor of Arts  
East China Normal University  
Shanghai, China  
1988

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
December, 2000

DESIGN AND IMPLEMENTATION OF A WORLD  
WIDE WEB BASED DISTRIBUTED  
COMPUTING MODEL

Thesis Approved:



---


Thesis Advisor



---



---



---

Dean of the Graduate College

## ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my major advisor Dr. K. M. George for his intelligent supervision, constructive guidance, and encouragement through my M. S. thesis work. My sincere appreciation extends to my thesis committee members, Dr. G. E. Hedrick and Dr. Nohpill Park for their valuable assistance and encouragement.

I wish to give my special appreciation to my husband, Qing Li, for his love, encouragement, and precious assistance in my study and life. I also wish to express my gratitude to my mother-in-law, Guohua Liu, for her significant support and understanding throughout the whole process. Finally, I would like to give my thanks to my darling baby girl, Grace Jiayun Li, and my precious newborn boy, Benjamin Jiayi Li, since they brighten my life.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
1.1 Distributed Computing and Implementation Models.....	1
1.2 A New Model.....	3
1.3 Objectives of This Thesis.....	4
1.4 Outline of Thesis.....	5
II. REVIEW OF LITERATURE AND RELATED WORK.....	6
2.1 Review of Distributed Systems and Distributed Computing.....	6
2.1.1 CORBA.....	8
2.1.2 Java RMI.....	9
2.1.3 DCOM.....	11
2.1.4 Mobile Agents.....	12
2.2 Review of Software Reuse and its Models.....	13
2.3 Review of WWW Applications.....	16
III. WDCM MODEL.....	18
3.1 WDCM Model Overview.....	18
3.2 WDCM Configuration.....	19
3.3 Composition Rules.....	19
3.3.1 Sequential Composition.....	21
3.3.2 Parallel Composition.....	21

Chapter	Page
3.3.3 Pipeline Composition.....	22
IV. WDCM PROGRAMMING SYSTEM DESIGN.....	24
4.1 Basic Operations of the WDCMPS.....	28
4.2 WDCMPS Web Server.....	31
4.3 Server Manager.....	31
4.4 Software Meta Library and the Library Manager.....	32
4.5 Application Managers and Application Programs.....	34
V. WDCMPS IMPLEMENTATION.....	35
5.1 WDCMPS Web Server.....	35
5.2 Server Manager.....	37
5.2.1 Implementation of Composition Rules.....	40
5.2.1.1 Sequential Composition.....	40
5.2.1.2 Parallel Composition.....	41
5.2.1.3 Pipeline Composition.....	43
5.2.2 Remote Call.....	44
5.3 Software Meta Library and Library Manager.....	45
5.4 Application Manager and Application Programs.....	46
5.4.1 Get Remote Input Data.....	46
5.4.2 Execution of Application Programs.....	47
5.4.3 Send Remote Output File.....	48
5.4.4 Return Execution Results to Server Manager.....	49
5.5 An Example.....	49

Chapter	Page
VI. CONCLUSION AND FUTURE WORK.....	58
REFERENCES.....	60
APPENDICES.....	63
APPENDIX A--Source Code for composer <i>P1.cgi</i> .....	63
APPENDIX B--Source Code for Final Composer <i>P2.cgi</i> .....	64
APPENDIX C--Source Code for Sequential Execution	
Subroutine <i>seqSub.cgi</i> .....	65
APPENDIX D--Source Code for Parallel Execution	
Subroutine <i>parSub.cgi</i> .....	66
APPENDIX E--Source Code for Pipeline Execution	
Subroutine <i>pipeSub.cgi</i> .....	68
APPENDIX F--Source Code for Application Manager <i>run.cgi</i> .....	71

## LIST OF TABLES

Table	Page
4.1 WDCMPS Six Components and Methods of Communication .....	26
4.2 Responsibilities of Each Component of WDCMPS.....	27

LIST OF FIGURES

Figure	Page
4.1 The System Architecture of WDCMPS.....	25
4.2 Template Structure in the Software Meta Library.....	33
5.1 Program Registry Form.....	36
5.2 Program Registry Continuation Form.....	37
5.3 Composition Selection Form.....	38
5.4 Sequential Program Build Form.....	39
5.5 Parallel Program Build Form.....	52
5.6 Run Form.....	53
5.7 Program Execution Results.....	54
5.8 out1: Remote Output File for Program TestSort.class.....	55
5.9 out2: Remote Output File for Program SumNum.class.....	56
5.10 out3: Remote Output File for Program Average.....	57



# CHAPTER I

## INTRODUCTION

### 1.1 Distributed Computing and Implementation Models

Distributed computing is an application of distributed systems. A distributed system is a set of autonomous computers linked by a network and equipped with distributed system software [1]. Distributed system software coordinates the activities of the computers in the network, and lets them share the resources of the system including hardware, software and data. In a well-designed distributed system, a user feels he/she is using a single integrated computer even though the system is implemented by many computers in different locations.

The usefulness and key characteristics of distributed systems include resource sharing, openness, concurrency, scalability, fault tolerance and transparency [1]. Users of a distributed system can share resources (hardware, software and data) in different machines in the distributed system through a resource manager, which is a software that manages a set of particular resources. Openness of a distributed system enables the distributed system to be extended in various ways. Concurrency provides the mechanism to allow several processes in the system to execute concurrently. Scalability indicates that distributed systems operate effectively and efficiently at many different scales. Fault tolerance means that the computer systems have the ability to deal with the system failure. Transparency enables the system to be viewed as a whole rather than as a collection of independent components even though components in a distributed system are separate.

There are two major types of distributed computing models: client/server model and object-based model [1]. A client/server model consists of servers and clients. A server acts as a resource manager, and a client requests access to resources. All resources are held and managed by servers.

Object-based model views every entity in a program as an object with an interface providing access to its operations [1]. Each shared resource is viewed as an object. Objects are uniquely identified and may be moved anywhere in the network without changing their identities. Therefore, in the object-based model, resource users can refer to all resources in a uniform manner. The interfaces of objects provide two useful properties: *interchangeability*, the ability of components to interchange, and *interoperability*, the ability of components to work together. CORBA (Common Object Request Broker Architecture), Java RMI (Remote Method Invocation), and DCOM (Distributed Component Object Model) are object-based models. These models will be reviewed in detail in Chapter II.

The three object-based models mentioned above have disadvantages in different aspects. CORBA and DCOM do not provide World Wide Web (WWW) based access or interface. Currently, the WWW is one of the major knowledge dissemination systems. It is an interactive hypermedia system built upon the Internet. It provides users convenient ways to access the information at anytime from anywhere. Although Java RMI does have the web-based access and interface because of its inheritance from Java's web language property, it lacks the capability of using legacy programs, which are implemented in programming languages other than Java. Therefore, Java RMI is not an ideal model for reusing legacy software.

## 1.2 A New Model

As the WWW becomes popular, there have been several approaches to harness the computing power of the Internet. Mobile agents are among the popular models. This thesis proposes a programming model called WDCM based on the World Wide Web (WWW). WDCM stands for WWW based Distributed Computing Model.

WDCM consists of four components:

- (1) An underlying network, which is WWW
- (2) Executable programs residing in the nodes of the network
- (3) Data distributed in the nodes of the network
- (4) A set of rules to compose new programs using existing ones

The existing programs may be written in any programming language, such as Java, C++, C, Ada, Fortran, etc. WDCM provides the accessibility to the remote input and output files. The input to each program can be located in any web-accessible machine. And the output file may be sent to a different machine. WDCM has the advantages of distributed computing and contains the web-based accessibility and software reusability.

In this thesis, we also design and implement a programming environment based on the WDCM called the WDCM programming system (WDCMPS) to (1) enhance the reusability of the existing, well-developed application programs located on different machines, (2) combine these distributed application programs to accomplish new tasks, and (3) make these existing distributed programs accessible to remote clients via WWW.

### 1.3 Objectives of This Thesis

This thesis aims to enhance the reusability of the existing programs by making them accessible to remote clients. Most application programs are designed to be used locally, usually not intended to be accessible and usable to remote clients. This thesis is to extend its usability to remote programs. It also aims to enhance the reusability of the existing programs to other programs written in different languages. Many stand-alone application programs have been well-developed and tested. They were written in the languages that are well suited to their specific problems. Other programs written in other programming languages usually cannot use these programs because they were written in different languages. This thesis enables the programs written in different languages to cooperate to accomplish new tasks.

WDCM simplifies the invocation of remote existing application programs by using WWW. This thesis will use common WWW browsers (Microsoft Internet Explorer, Netscape Navigator/Communicator, etc) to invoke remote existing programs. A web browser allows users to access the information managed by a web server anytime and anywhere without worrying about shifting from protocol to protocol and software incompatibility. WDCM combines the existing program's functionalities to create new tasks and perform new tasks easier by using existing programs on the WWW. It provides a mechanism to let the executing programs communicate with the input data files located at different machines, so that it looks, as if the input files were local. And the output files can be sent to any where on the web automatically.

## 1.4 Outline of Thesis

This thesis consists of six chapters. Chapter I introduces the basic concepts of distributed computing and several implementation models, provides the objectives and brief description of the proposed model, WDCM, and also gives the outline of this thesis. Chapter II reviews the literature and related work, including distributed systems and distributed computing, software reuse, and WWW applications. Chapter III describes the WDCM model, including the model overview, configuration, composition rules, and their Petri net representations. Chapter IV illustrates the WDCMPS system design issues, such as the basic operations of the system, and the responsibilities of its six components. Chapter V focuses on the implementation issues, and presents the implementations of three composition rules, and the implementations of each component of the WDCMPS system. Finally, conclusion and future work are included in Chapter VI. Some source code written in Perl scripts are listed in appendices.

## CHAPTER II

### REVIEW OF LITERATURE AND RELATED WORK

In this section, we review previous works that are related to this thesis topic, including distributed computing, software reuse, and WWW applications. Several models are studied.

#### 2.1 Review of Distributed Systems and Distributed Computing

A distributed system consists of a set of autonomous computers linked by a network and equipped with distributed system software [1]. The software enables computers to coordinate their activities and share the resources (hardware, software, and data) of the system. Resources in a distributed system are physically located in one of the computers, and other computers can only access them via communication. These resources are managed by resource managers, which constitute an important component of a distributed system. Therefore, in a distributed system, resource users communicate with resource managers to access the shared resources of the system [2].

The World Wide Web is a good example of a distributed system. Many web servers run on various computers, and server software is available for all operating systems, such as UNIX, Windows 95/98/NT, Macintosh and OS/2. Each server has a wide range of documents and information. These web servers act as resource managers [2]. Therefore, a web server is an appropriate communication media between resource users and the shared resources of a distributed system.

A web browser is the client in a WWW based distributed system. Its main function is to request a document available from a specific server through the Internet by using the URL (Uniform Resources Locator). The server on a remote machine returns the document to the web browser. Common Gateway Interface (CGI) supports execution of programs at the server side and returning result to the client web browser. The two most widely used browsers today are Microsoft's Internet Explorer (IE) and Netscape's Navigator/Communicator.

Distributed computing is one of the major types of applications of distributed systems [2]. Distributed computing has two basic implementation models: client/server model and object-based model.

The first technology, client/server, is a means for separating the functions of an application into distributed parts, each of which operates on a different computing platform. The client/server is a two-tiered architecture developed during the 1980s [3]. The client/server model is a form of distributed computing in which one program (the client) communicates with another program (the server) for the purpose of exchange of information. In client/server computing, the client communicates with the server through a protocol – a common language that they both understand. A protocol is a set of rules that computers in the distributed system must follow to exchange information.

The second technology, object-based model, includes CORBA, DCOM, and Java RMI. They treat pieces of software located in remote nodes in the network as objects. Currently, they are the major standard distributed computing models used by applications [4-6]. The three basic elements of distributed architectures are: a communication protocol, an interface language, and a naming service [4].

The three object-based distributed models mentioned above and their three basic elements are reviewed below.

### 2.1.1 CORBA (Common Object Request Broker Architecture)

CORBA was developed as a distributed computing architecture in 1991, by the Object Management Group (OMG), a consortium of over 800 members that was founded in 1989 [4]. It was used by many large organizations for complex and large-scale distributed computing with UNIX servers and proprietary mainframes.

The interface language for CORBA is the Interface Definition Language (IDL). In a CORBA application, the IDL is written first, and then it is compiled into code in one of the supported languages. CORBA provides different translation tools for different languages from IDL. For example, `idltojava` is a translation tool from IDL module to java package. The supported languages for CORBA are C, C++, Smalltalk, Ada, Cobol, and Java [4].

Because of the IDL, CORBA is language neutral, which means that CORBA clients and servers may be implemented in any of the supporting languages and still be able to communicate with each other [4].

CORBA uses Internet Inter-ORB Protocol as its communication protocol. Everything in CORBA depends on an Object Request Broker (ORB), which serves as the object manager for CORBA. The ORB enables location transparency, which means that it allows objects to send and receive messages without worrying about whether they are local or remote [4]. Each CORBA server object has an interface and a set of methods. When a CORBA client requests a service, it acquires an object reference to a



CORBA server object, and then it can invoke the methods of the server objects by using the object reference. The ORB is responsible for finding the implementation of the CORBA server object. Then it returns the reply back to the client [7].

CORBA's naming service is composed of a pair of proxies for the client and the server. The proxy for the server is called *skeleton* and the proxy for the client is called *stub*. They are both generated when the IDL is started. The stub and skeleton can be in different languages, since all communication is done through the ORB [4].

CORBA has the property of interoperability for heterogeneous platforms. Interoperability means that distributed objects can invoke each other's methods even if such underlying platforms are different. Interoperability is significant because we cannot assume that every computing site in a distributed environment uses the same platforms [8]. CORBA can be used on various operating system platforms such as UNIX and Windows machines as long as there is an ORB for that platform.

The disadvantage of CORBA is the complexity of its code. CORBA is an extremely large and complex collection of specifications and protocols [4].

CORBA is suitable for client/server applications running on conventional local area networks, such as Ethernet and Token Ring [9]. However, currently, no web based CORBA applications are found.

### 2.1.2 Java RMI (Remote Method Invocation)

Sun developed RMI as a Java-based approach to distributed computing. The Java programming language has included Remote Method Invocation (RMI) as part of the standard Java libraries beginning with the Java Development Kit 1.1 (JDK1.1) [4].

RMI is an object-oriented type of Remote Procedure Call (RPC). RMI inherits the platform independence from Java programming language.

Java RMI uses TCP/IP (Transmission Control Protocol/Internet Protocol), the most common Internet protocol, as its communication protocol. Therefore, Java RMI is simpler than CORBA and DCOM in this sense.

Unlike CORBA and DCOM, Java RMI does not have interface language. Instead, it uses Java's own interface syntax as its interface language [10]. Java RMI imports the `java.rmi.Remote` interface as its interface, thus simplifies application design and implementation.

RMI uses a naming service mechanism called an `RMIRRegistry`, which is embedded into the `java.rmi.registry` package. `RMIRRegistry` allows clients to obtain references to remote server objects. The clients invoke the method implementations by passing arguments to the server objects as if the method implementations were local. This is done through Java's serialization facility.

RMI passes a remote stub for a remote server object to the client, rather than making a copy of the implementation object in the client machine. The stub acts as the local representative, or proxy, for the remote server object and basically is the remote reference to the client. When the client invokes a remote method call, it invokes the method on the local stub. The local stub is responsible for passing the method call on the remote object. A stub for a remote object implements the same set of remote interfaces that the remote object implements. This allows a stub to be cast to any of the interfaces that the remote object implements. That is, RMI uses the technique of

invoking a method of a remote object by using the same syntax used in local invocation. This is the unique characteristic of RMI and it supports the *location transparency*.

A client acquires a server object reference by using the URL. Since RMI relies on Java, which is a platform-neutral language, it can be used on diverse operating system platforms, such as UNIX, or Windows machines as long as there is a Java Virtual Machine (JVM) implementation for that platform, and no special software or drivers are needed to use RMI [7].

Like CORBA, RMI also has the interoperability property among heterogeneous platforms.

RMI has its limitation as a distributed computing architecture. It is a language-specific approach, and thus limits the extent to which RMI can deal with distributed objects implemented in languages other than Java. This results in the difficulties in incorporating legacy programs, which were implemented in other programming languages [4].

### 2.1.3 DCOM (Distributed Component Object Model)

DCOM was invented and mainly has been used by Microsoft as a distributed object system technology. It is heavily integrated with the Windows NT operating system [11].

DCOM uses ORPC (Object Remote Procedure Call) as its communication protocol, MIDL (Microsoft Interface Definition Language) as its interface, and Windows Registry as registration method for its naming service [5]. In the DCOM architecture, clients and servers communicate through ORPC protocol. A server object

is registered in Windows Registry. When a client wants to connect to a remote server object, the client specifies a class identifier (CLSID) and an interface identifier (IID) to obtain an interface pointer. The Service Control Manage (SCM) at the remote node looks up the registry and returns it a remote object reference. The client then can invoke methods of that interface [11].

DCOM is supported by many development tools, such as Microsoft's Visual Basic, Visual C++, C++ Builder, and Delphi [3]. In addition to these tools, Java integrates well with DCOM. For instance, Java classes can be treated like COM objects.

A DCOM interface MIDL is a collection of methods that define services. In fact, it uses an interface pointer to point to a *vtable*, which is a collection of pointers to methods [3].

The major disadvantage of DCOM is that it is still a Microsoft-only solution, and it was designed to use Microsoft's Active Directory to find components on the network [12]. In other words, no WWW based DCOM applications are found so far.

#### 2.1.4 Mobile Agents

“A mobile agent is a program that is able to change its location on its behalf and keeps its state (identity) across location changes” [13]. Mobile agent model provides a new paradigm for distributed computation, since mobile agents are very efficient compared with the traditional approaches to distributed computation. Mobile agents can reduce the network traffic by moving the code that handles the interaction more closely to the source of the data. Mobile agent model is suitable for large and complex

distributed applications [13]. One problem in mobile agent model is the security problem.

The DDAS (Dynamic Distributed Agents Server) System [14] proposes a secure, and adaptive distributed computation environment based on TCP/IP. It allows a user to build and to use a dynamic network system, which consists of domains accessible by a user with read-write-execute privileges. The DDAS system consists of two major parts, the DDAS and its Manager. The DDAS acts as a tool to distribute and execute subtasks. The Manager acts as the interface between the DDAS and a program. The DDAS system only allows the authorized machines to use its services, thus enhancing security [14].

The DDAS system is a user-centered distributed computing environment. It only supports the authorized users to do the distributed computing by using all the available machines in the system. It does not make use of the WWW.

## 2.2 Review of Software Reuse and its Models

The philosophy behind software reuse is to use the existing programs to build new useful programs. The goals of software reuse include the following [15]:

- Productivity: less costly than to build the software from scratch.
- Maintainability: error correction in reused components is of benefit to the software, which uses them.
- Reliability: reliability increases as programs run longer because errors are corrected.

- Extensibility: all extensions of a component must be able to be used without significant changes.
- Adaptability: capability of being adapted to a new context without affecting its other uses.

There are various ways for software reuse implementation. Systematic software reuse is one of them. It is considered to be a very effective way to significantly improve software reuse [16]. This is because both the creation of reusable components and their reuse are embedded into the software development processes so that everybody must adopt it in order to reuse the software components. Systematic reuse requires the following processes at company level:

- Reuse exploration and planning
- Domain analysis
- Software engineering with reuse
- Procurement of reusable components
- Component identification and extraction
- Component qualification and classification [17]

Another approach to software reuse is object-oriented technique. The object-oriented programming (OOP) languages provide a user with the techniques of reusable components by using the mechanisms supported by the OOP languages, including the following:

- Inheritance: a new class inherits all data and methods from its ancestor classes. Therefore, all the existing ancestor classes (software components) can be reused by all of the newly created descendant classes.

- **Extension:** new instances of a class can be created to communicate with other objects in the system, which are not affected by the new objects. This makes it easier for the reuser to adopt and extend a software component.
- **Polymorphism:** it means that an object can have many forms for method invocations. Different objects can respond differently to a method call. This enhances the reusability for the sending and receiving objects.
- **Encapsulation:** it means that implementation details are hidden from the user. This makes the reuse of the program easier.

Behle (1998) proposed a software reuse model called Internet-based Software Component Information System (SCIS) model [18]. It is supported by the WWW, the Internet, and the corresponding protocols. Internet services may be used to find software components. SCIS uses a software classification schema to generate HTML forms dynamically and to store the component data. The comprehensive information on all available software components is the critical point of the SCIS model. Since the SCIS is accessible to any project at different sites, the SCIS has to be placed in the Internet and a single Web browser is the only software necessary for access.

Furthermore, only registered users have access to SCIS. The SCIS is able to represent any information on different kinds of software components at different levels of granularity, such as classes, class libraries, and binary components. These software components can be developed using different programming languages.

Therefore, the SCIS can be called a global information place for software components. The user interface is completely provided by WWW access and HTML

forms. There are basically two ways to access the software components: search and navigation [18].

The proposed model WDCM will adopt parts of the SCIS's concepts to be used in the reuse model for the existing programs.

### 2.3 Review of WWW Applications

The Internet is a collection of computers and physical communication links. The links are operated by a standard set of communication protocols through which the computers throughout the world are interconnected. It began to be used as a research project called the ARPAnet (U.S. Defense Advanced Research Projects Agency) in 1969 [19]. Nowadays, the Internet becomes a network of networks. World Wide Web (WWW) is one part of the global Internet, but it has been growing extremely rapid because of its easy-to-use interface, graphical nature, and ability to integrate FTP, email, newsgroups,archie, gopher, and WAIS-type (Wide Area Information Service) services [19]. The Uniform Resource Locators (URLs) allow the WWW to provide access to files, directories, indices, data, email, pictures, audio, video, etc. When requesting information through the WWW, a web browser makes a request by using URL. HTTP (Hypertext Transform Protocol) is used to transmit the request from the browser to the server. The Internet provides the communications between the web browsers and the web servers [19].

There are various WWW applications regarding the navigation tools for the information retrieval. Persistent History Navigation Assistant (PHNA) [20] is one among those. PHNA defines a user-need based model for information retrieval, which



is different from the traditional stack based model widely used in the current browsers [20]. It maintains a complete list of the history of using BACK and FORWARD buttons. No sites are lost even if the user goes back or forward several sites. Therefore, PHNA provides the user with the freedom to move back and forward through the real sequence that was visited rather than through the route of the tree structure, which does not represent the user's real visit route. This is an advanced search tool compared with the current ones.

An advanced WWW based information retrieval technique is another goal for web application. Efficient search tools have been sought to meet the individuals' information need. A WWW resource discovery system [21] is to build and maintain an index database for keyword searching. In contrast to manual indexing, this approach is suitable for the size and the dynamical nature of the WWW. The index database can substantially reduce network load since users access only the index data in order to locate resources [21].

This approach also provides the capability of sharing one user's discovery with other users. To support such information sharing, the index server allows any user to save his query statement on the server's side so that other users can reuse it later [21].

Although a large number of WWW applications has appeared within these years, so far no publications on WWW based approaches to do distributed computing are found.

## CHAPTER III

### WDCM MODEL

WDCM (WWW based Distributed Computing Model) defines a programming model based on WWW technology. It consists of an underlying network, executable programs, data residing in the nodes of the network, and a set of rules to build new programs using existing ones. The WWW is adopted as the network of this model. This thesis defines three program-combining rules, namely sequential composition, parallel composition, and pipeline composition.

#### 3.1 WDCM Model Overview

The WDCM distinguishes itself from previous models by being capable of integrating all the following functionalities:

- Supports the reuse of existing programs independent of their locations.
- Builds new programs by combining existing ones.
- Supports sequential, parallel, and pipeline compositions of programs located on different machines.
- Provides WWW based access and interfaces. WDCM provides the user with an HTML form to register the existing executable programs into the software meta library, and program build forms to create new programs from existing programs and previously defined programs named composers.

### 3.2 WDCM Configuration

The network model adopted in WDCM, in this thesis, is the WWW. The programs reside in nodes, which are the machines distributed in the network. WDCM provides the rules to combine the existing programs and previously defined composers to perform a new task. There are three rules: sequential composition, parallel composition, and pipeline composition. These are discussed in detail in section 3.3.

WDCM deals with input and output data according to the specific program combining rules. For instance, in the sequential composition and pipeline composition, the output of one program is the input of another program. Both the input and output data might be located in remote and different machines distributed in the network.

In this thesis, we build a programming environment based on WDCM called the WDCM programming system (WDCMPS). WDCMPS consists of WDCMPS Web server, server manager, software meta library, library manager, application managers and application programs. The design of the WDCMPS system is presented in Chapter IV, and its implementation is provided in Chapter V.

### 3.3 Composition Rules

The composition rules, or combining rules, specify the different methods to build new programs or tasks from existing programs and/or previously defined composers, which will be defined in Chapter IV. They specify the control flow and data flow of the computation. As mentioned previously, in this thesis three composition rules are defined – sequential composition, parallel composition, and pipeline composition. Sequential composition is used to specify that the component programs execute in a

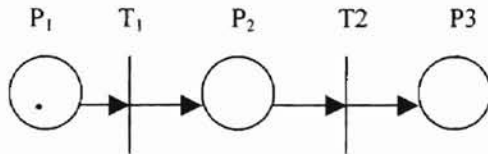
particular order, one after the other. Parallel composition is used to specify that component programs execute in any order, or simultaneously [22]. Pipeline composition is a combination of the sequential and parallel compositions. The component programs are executed in parallel as the new program's different stages, but the flow of data is sequential. Therefore, it is a special type of parallel computing. It is intended to achieve high performance.

The composition rules are illustrated below using Petri nets [23]. A Petri net is a bipartite diagram used to represent a flow of control [23]. A Petri net can be represented by a quadruple:  $(P, T, E, M_0)$ , where  $P$  is a set of places, which represents data for a program;  $T$  is a set of transitions, which represents a computation program;  $E$  is a set of directed edges (arcs) from a place to a transition or a transition to a place, and  $M_0$  denotes an initial token marking. Tokens control data flow and computation. Usually, places are represented by circles, transitions are represented by bars or rectangles, and tokens are represented by dots. A transition is enabled to fire or activated if each of its input places has at least one token. After the transition fired, the number of tokens in each input place is reduced by one, and the number of tokens in each output place is increased by one [23].

In the following three Petri net diagrams,  $P_i$  is used to denote a place,  $T_i$  is used to denote a transition, a dot is used to denote the token which controls the data flow, and an "x" is used to denote the token which controls the computation. A transition represents a computation program.

### 3.3.1 Sequential Composition

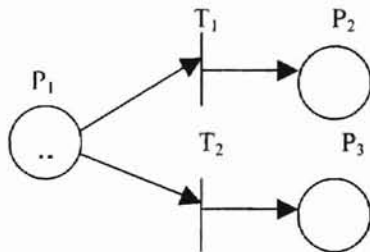
For sequential composition, assume that Prog3 is a new program constructed from existing programs Prog1 and Prog2 by sequential composition. The Petri net diagram representing the executing order is as follows:



In the diagram above, T<sub>1</sub> represents Prog1, and T<sub>2</sub> represents Prog2. Initially, only P<sub>1</sub> has a token, thus T<sub>1</sub> is enabled to fire, since only Prog1 is ready to be run. After T<sub>1</sub> fires, the token in P<sub>1</sub> is consumed and it is passed to P<sub>2</sub>, so T<sub>2</sub> is enabled. After T<sub>2</sub> fires, the token is passed to P<sub>3</sub>, which becomes the output of Prog3. The transition then stops.

### 3.3.2 Parallel Composition

For parallel composition, assume that Prog3 is constructed from programs Prog1, and Prog2 by parallel composition rule. The Petri net diagram is as follows:

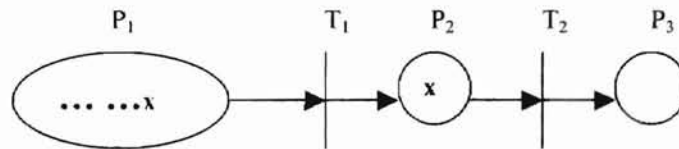


In this diagram, T<sub>1</sub> represents Prog1, and T<sub>2</sub> represents Prog2. Initially, P<sub>1</sub> has two tokens, thus T<sub>1</sub> and T<sub>2</sub> are enabled. Both of them can be fired concurrently. After

$T_1$  and  $T_2$  are fired, the tokens in  $P_1$  are consumed, and  $P_3$  and  $P_4$  each has a token, which became the outputs of Prog3. The transition then stops.

### 3.3.3 Pipeline Composition

For the pipeline composition, assume that Prog3 is a new program constructed from the existing programs Prog1 and Prog2. There exist the following relations between Prog1 and Prog2: for the flow of computation, they are parallel; for the flow of data, they are sequential. That is, Prog1 and Prog2 can be run concurrently as Prog3's different stages. The biggest advantage for pipelining is that all stages in pipelining can operate concurrently, thus enhancing the performance of the program. The corresponding Petri net diagram for pipeline program Prog3 is as follows:



In this diagram,  $T_1$  and  $T_2$  represent Prog1, Prog2 respectively. The dot tokens control the flow of data, and the "x" tokens control the flow of computation. Initially, only  $P_1$  has several dot tokens, but every place has an "x" token. In this case, the requirements for a transition to fire is that each input of the transition has at least one dot token and one "x" token. Only dot tokens are consumable, which means that after firing, only a dot token is consumed, and "x" token is not consumed. Therefore, initially, only  $T_1$  is enabled. After  $T_1$  fired, the number of dot tokens in  $P_1$  decreased by one, and  $P_2$  got one dot token. From this moment on,  $T_1$  and  $T_2$  can be fired

simultaneously. After  $T_2$  fired,  $P_3$  gets one dot token.  $T_1$  and  $T_2$  fire concurrently until the dot tokens in  $P_1$  have been consumed completely.

In the next chapter we describe the WDCM programming system (WDCMPS). WDCMPS is a programming system based on WDCM and is developed in this thesis.

## CHAPTER IV

### WDCM PROGRAMMING SYSTEM DESIGN

WDCM Programming System (WDCMPS) has six components: the WDCMPS web server, server manager, software meta library, library manager, application managers, and application programs. This chapter describes WDCMPS and its components.

WDCMPS provides an environment for a user to build new programs (called composers) using existing executable programs. Information about all component programs must be available in the software meta library. The system provides ways to add program information into the library as well as to build new programs using existing programs.

Figure 4.1 shows the system architecture of WDCMPS. Table 4.1 describes the six components of WDCMPS and the methods of communication between them. Table 4.2 illustrates the responsibilities of each component of WDCMPS.



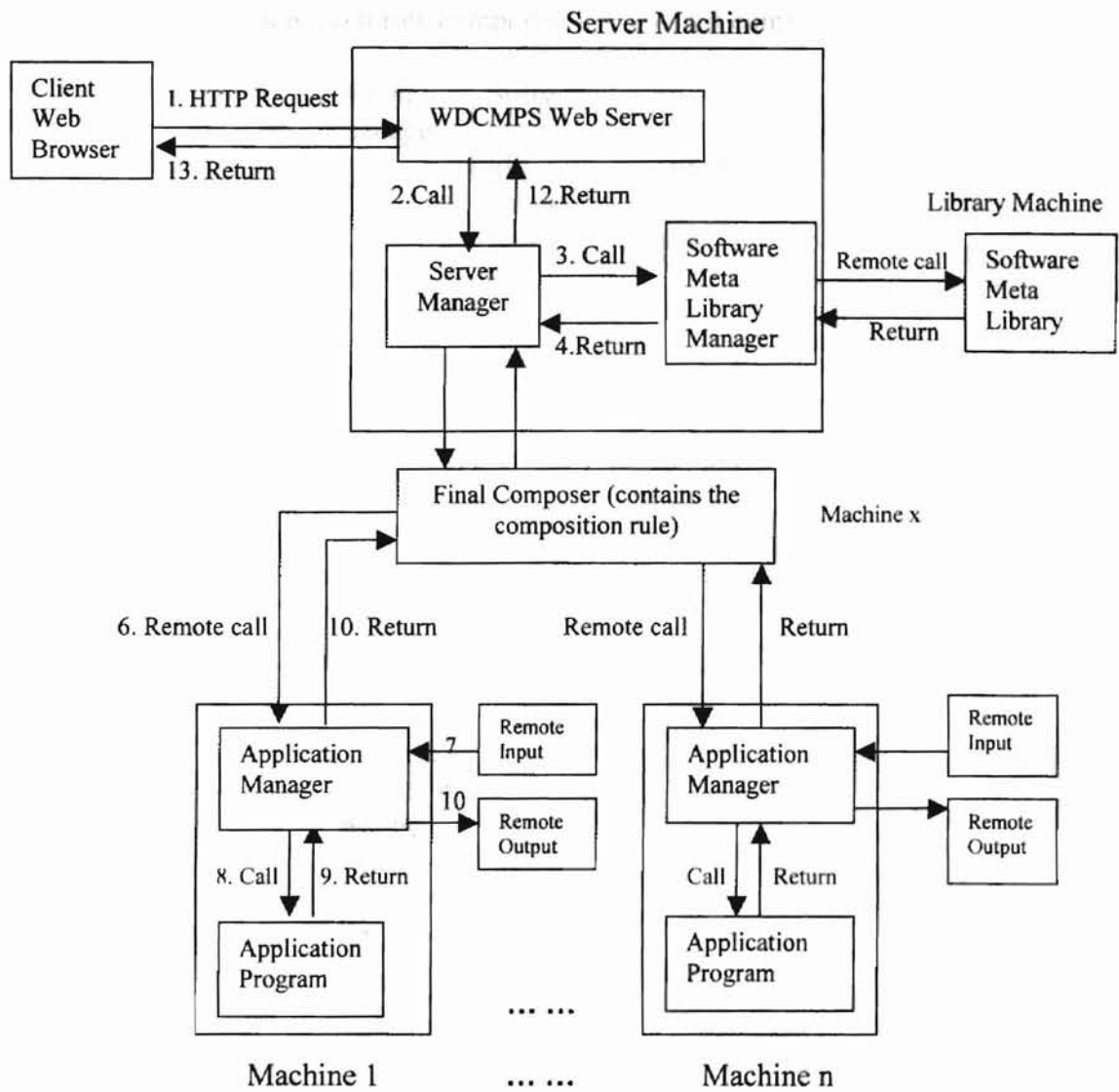


Figure 4.1 The System Architecture of WDCMPS

Table 4.1 WDCMPS Six Components and Methods of Communication

(The direction is from row components to column components)

	Web server	Server manager	Software meta library managers	Software meta library	Application manager	Application programs
Web server	N/A	Call	N/A	N/A	N/A	N/A
Server manager	Return results	N/A	Call	N/A	Remote call through final composer	N/A
Software meta library managers	N/A	Return search results	N/A	Remote call	N/A	N/A
Software meta library	N/A	N/A	Return results	N/A	N/A	N/A
Application manager	N/A	Return program execution results through final composer	N/A	N/A	N/A	Call to invoke execution
Application programs	N/A	N/A	N/A	N/A	Return execution results	N/A

Table 4.2 Responsibilities of Each Component of WDCMPS

	Responsibilities
WDCMPS Web server	<ol style="list-style-type: none"> <li>1. Provide Web accessible interfaces (an HTML registry form and HTML program build forms) for user inputs.</li> <li>2. Forward user inputs to server manager for processing.</li> <li>3. Forward execution results, which are returned from application managers to the final composer, to the user's web browser through the server manager.</li> </ol>
Server manager	<ol style="list-style-type: none"> <li>1. Parse the user inputs from the WDCMPS web server.</li> <li>2. Call the registry manager of the software meta library to register and store the existing programs or previously defined composers into the software meta library's templates.</li> <li>3. Call the search manager of the software meta library to retrieve template information about the requested existing programs or composers.</li> <li>4. Create a new executable program file (called composer), which combines the existing programs or previously defined composers in the execution order according to composition rule specified by the user. The composer contains code to execute the user requested existing programs or composers in sequential, parallel or pipeline order, and may also contain code to make remote calls to the remote application managers to request executions of the remote application programs. The server manager may send this composer to a remote machine. Then the server manager calls library registry manager to register this composer into the software meta library. After registration, the composer can be called the same way as other existing application programs in the library.</li> <li>5. Forward the execution results of the distributed application programs to the WDCMPS Web server.</li> </ol>
Software meta library	<p>Store existing and previously defined composers' information into templates in the library. Each program corresponds to a template in the library.</p>
Software meta library managers	<ol style="list-style-type: none"> <li>1. Library registry manager is responsible for registering the existing programs and composers into the library.</li> <li>2. Library search manager retrieves the user requested program's template information from the library and returns it to the server manager.</li> </ol>
Application manager	<ol style="list-style-type: none"> <li>1. Receive remote requests from the composer that the server manager creates.</li> <li>2. Get the remote input data file.</li> <li>3. Do the data conversion if necessary.</li> <li>4. Invoke the execution of the application program(s)</li> <li>5. Return the execution results to server manager and/or send the output to a remote machine specified by the user.</li> </ol>

Application program	An application program is an executable program residing in a node of the network. It is either an existing program or a composer. It is called by the application manager.
---------------------	---

#### 4.1 Basic Operations of the WDCMPS

WDCMPS web server provides a user HTML forms as user interfaces, which include a registry form and program build forms. The registry form allows the user to fill in the information about the existing executable programs and register them into the software meta library. The server manager parses the user-input and calls the registry manager of the software meta library to register these programs into the templates in the library.

The software meta library stores the information pertaining to the application programs into templates. A template is an object which contains information about an application program. Each program corresponds to a template in the library. A template consists of the following information:

- (1) Program name
- (2) Program location
- (3) Input file name
- (4) Input file location
- (5) User-specified input data format
- (6) Program-required input data format
- (7) Output file name
- (8) Output file location
- (9) User-specified output data format

#### (10) Program-required output data format

Among the information above, items (5), (6), (9), and (10) have the same format, which consist of three pieces of information: number of columns, data type of each column, and the delimiters.

The library has two managers, which are registry manager and search manager. The registry manager is responsible for registering the programs into the library according to the template structure, and the search manager is responsible for searching the program template information from the library.

The web server also provides HTML program build forms to allow the user to provide information to compose new programs. The user specifies, on the program build form, the name of the new program being composed, the names of the application programs (existing programs or previously defined composers) to be used, and the composition rule to combine them.

The server manager then calls the search manager of the software meta library to search in the library for the template information about each individual application program requested by the user for composing the new program. After having obtained the information of each individual program from the search manager, the server manager combines these individual application programs according to the composition rule, and creates a new executable program file to store the newly created program. This new program file contains code to execute each individual existing program or previously defined composers in the order specified by the composition rule provided by the user to perform a new task. Therefore, the new task can be carried out by just calling this newly created program, which is the composer.

The server manager may save the composer in local machine or send it to a remote machine. The server manager then calls the library registry manager to register the composer into the meta library. The information about the composer is stored in the meta library with the same template as any other existing program, and therefore can be called the same way as any other existing program in the library.

This program building process can be repeated as many times as the user wants. After finishing composing all programs on the program build form, the user requests the server manager to execute the final composer, which may contain several layers of previously defined composers. By executing the final composer, all existing programs and previously defined composers in all layers will be executed. The composers may make remote calls to the application managers to execute the application programs in remote machines.

The application manager first gets input data from the remote input data file. Then it checks to see whether the user-specified input format matches the program-required input format or not. If they match, which means that both the contents and the formats match, then the application manager invokes the execution of the application programs. If they do not match, which means that either the contents or the formats do not match, then the application manager needs to call a data conversion program to convert the user-specified input into a format required by the application program to be executed, and then invokes the execution of the application program. After execution, the application manager returns the results to the calling composer. It may also send the results to the remote machine specified by the user.

The process of the input data format checking and conversion applies to the output as well. If the program-required output format does not match the user-specified output format, then the data conversion program is called to convert it into the user-specified format. Otherwise, no conversion will be done. The results will be sent to the remote machine specified by the user as well as to the server manager and then forwarded to the user's web browser.

#### 4.2 WDCMPS Web Server

The WDCMPS web server provides the user WWW-accessible interfaces: an HTML registry form to allow the user to fill in the information about the existing programs, and several HTML program build forms to allow the user to compose programs. A user can use any web browser from anywhere at anytime to connect to the WDCMPS, and register existing programs.

The WDCMPS web server passes the user input to server manager for processing. Finally, it forwards the results of the application programs (from server manager) to the user's web browser.

#### 4.3 Server Manager

The server manager calls registry manager to register the user specified existing programs into the templates in the library. Then, after the user have selected a composition rule by using the composition selection form provided by the web server, the server manager provides an appropriate HTML program build form according to user's selection to request the user to fill in the names of the executable programs to be

combined, and the name for the new composed program to be created. The server manager then calls the library search manager to retrieve the template information about the user requested programs on the program build forms.

Using the retrieved information of user requested programs and the composition rule, the server manager creates a new executable program file, called composer, which combines the executable programs in the execution order according to the composition rule specified by the user. The composer contains code to execute the user requested programs in sequential, parallel or pipeline order, and may also contain code to make remote calls to the remote application managers to request execution of the remote application programs (see the example composer files P1.cgi in Appendix A and P2.cgi in Appendix B). The server manager may send this composer to a remote machine. Then the server manager calls library registry manager to register this composer into the template in the software meta library. After registration, the composer can be called the same way as other existing application programs in the library.

#### 4.4 Software Meta Library and Library Manager

The software meta library stores information about application programs (existing programs or composers) in templates. The library stores the information about the application programs rather than the program files themselves. The program files may reside at different remote machines. All the information is stored as templates, which contain all the information required to execute programs, such as program names, locations, input and output locations, names, and format information. Each program



corresponds to a template in the library. The library may be located in a different machine from the machine where the web server or server manager resides.

The structure for a template is as follows:

PN	PL	InN	InL	UInF	PInF	OutN	OutL	UOutF	POutF
----	----	-----	-----	------	------	------	------	-------	-------

Figure 4.2 Template Structure in the Software Meta Library

In Figure 4.2, PN stands for program name, PL stands for program location, InN stands for input name, InL stands for input location, UInF stands for user-specified input format, PInF stands for program-required input format, OutN stands for output name, OutL stands for output location, UOutF stands for user-specified output format, and POutF stands for program-required output format. In this thesis, we assume that UInF, PInF, UOutF, and POutF have the same format, which consists of the following information: (1) number of columns (2) data type of each column, and (3) delimiters. (Implementation of arbitrary formats will not be feasible.)

Each template distinguishes itself from others by its program name, which means that no two templates have the same program name. The software meta library is a collection of the templates.

The software meta library has two library managers, called registry manager and search manager, which are located in the same machine as the server manager so that the server manager can communicate with library managers easily. Through JDBC driver, the library managers can administrate the meta library remotely. The registry manager

is responsible for registering the existing programs into the library from the HTML registry form that the web server provides the user. It is also responsible for registering the dynamically created programs – composers – into the library. This registration process is done through the insertion of the templates into the library by the registry manager.

The search manager is responsible for searching the template information of the user requested programs in the software meta library. The server manager uses the retrieved information to create the composers.

#### 4.5 Application Managers and Application Programs

Each machine in the network has an application manager to administrate the application programs. An application manager is called by the composer, which is created by the server manager. It executes the application programs located in the local machine, and returns the results to both a remote machine as an output file and to the calling composer. Thus, an application manager contains the mechanism to handle the remote inputs and outputs. For detailed information related to implementation, refer to the implementation code of application manager, which is called run.cgi in Appendix F.

Application programs are previously compiled programs (existing programs) or composers. Application programs and data files are the basic elements of the WDCM. They are physical files located at the machines accessible through the HTTP protocol.

## CHAPTER V

### WDCMPS IMPLEMENTATION

This chapter describes an implementation of WDCMPS as a proof of concept. Implementation of each component and interfaces is described. This implementation assumes that the User-specified input format matches the program-required input format, and the User-specified output format matches the program-required output format, so there is no need to do the data conversion.

#### 5.1 WDCMPS Web Server

An HTTP web server with Common Gateway Interface (CGI) is used as the WDCMPS web server. In fact, any web server with CGI capacity can be used as the WDCMPS web server.

When a user wants to use WDCMPS to build programs, he/she uses any web browser to connect to the WDCMPS web server through a URL address of the WDCMPS web server. The web server first provides the user with an interface, which is a HTML registry form to allow the user to register into the software meta library the existing programs that are being used.

The registry form interface contains the following input fields to allow the user to fill in:

1. Location (URL) of the existing application program
2. Name of the program
3. Location of the input data file

4. Name of the input data file
5. Location of the output file
6. Name of the output file

After the user fills in the information about a program and clicks the “Register now” button, the server manager calls the registry manager of the software meta library to register the information of the user requested program into its template in the library. Figure 5.1 displays a registry form. Then, the user is presented with the interface shown in Figure 5.2. The user will have the option to register more programs or go to the phase of building programs.

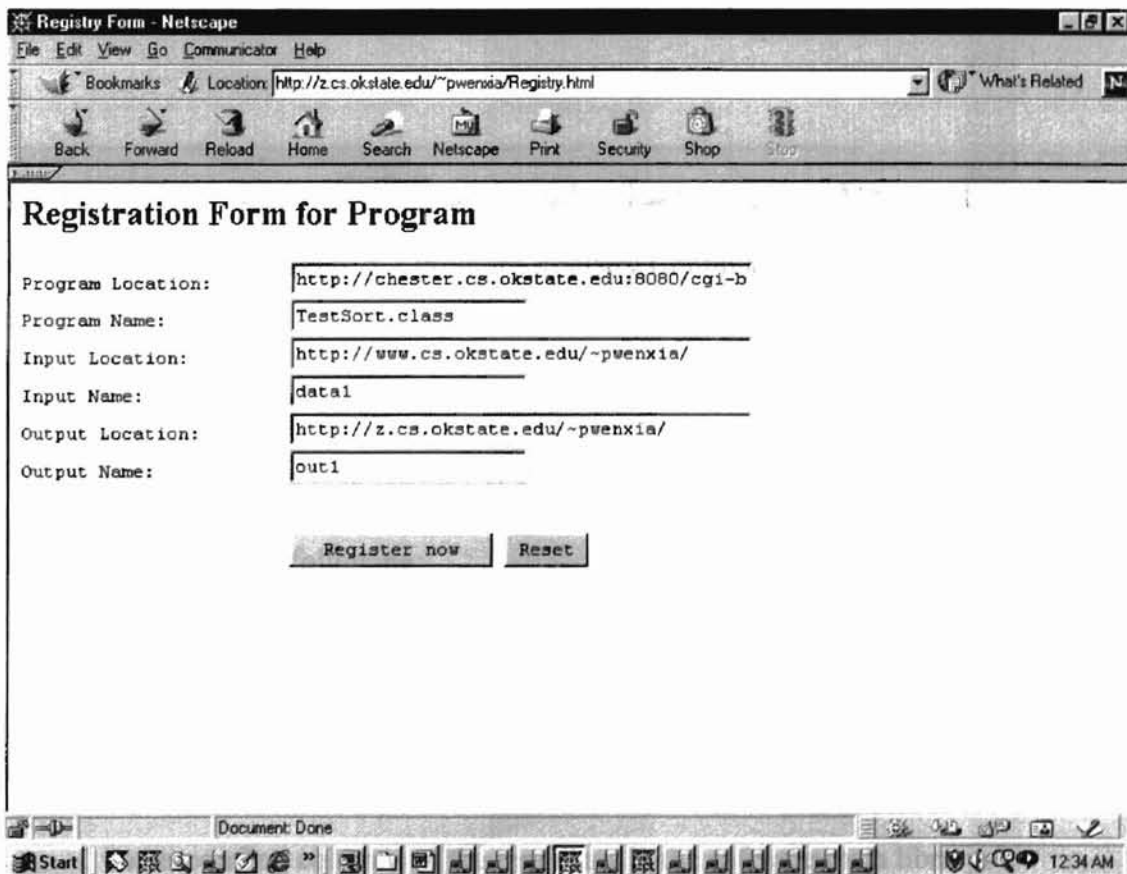


Figure 5.1 Program Registry Form

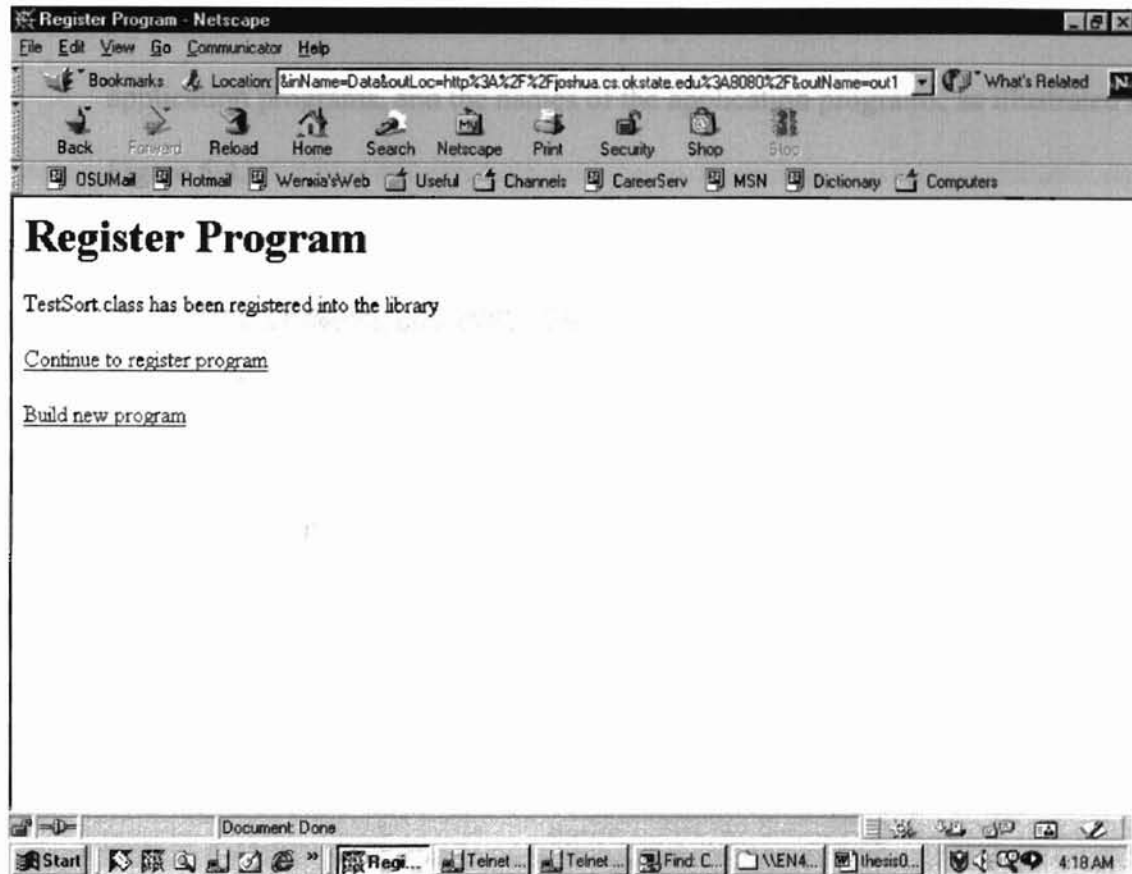


Figure 5.2 Program Registry Continuation Form

## 5.2 Server manager

The server manager consists of several CGI programs written in Perl. Those CGI programs do the following:

1. Register the existing programs' information from registry form into the software meta library. The CGI program calls the registry manager, which is a java program named Registry, to insert the fields from the registry form into the corresponding columns of the templates in the software meta library.
2. After the user selects a composition rule from the composition selection form, which is illustrated in Figure 5.3, it shows the user an appropriate program build

form to allow the user to fill in the new program's name, the number of the application programs, and the names of the application programs, as illustrated in Figure 5.4.

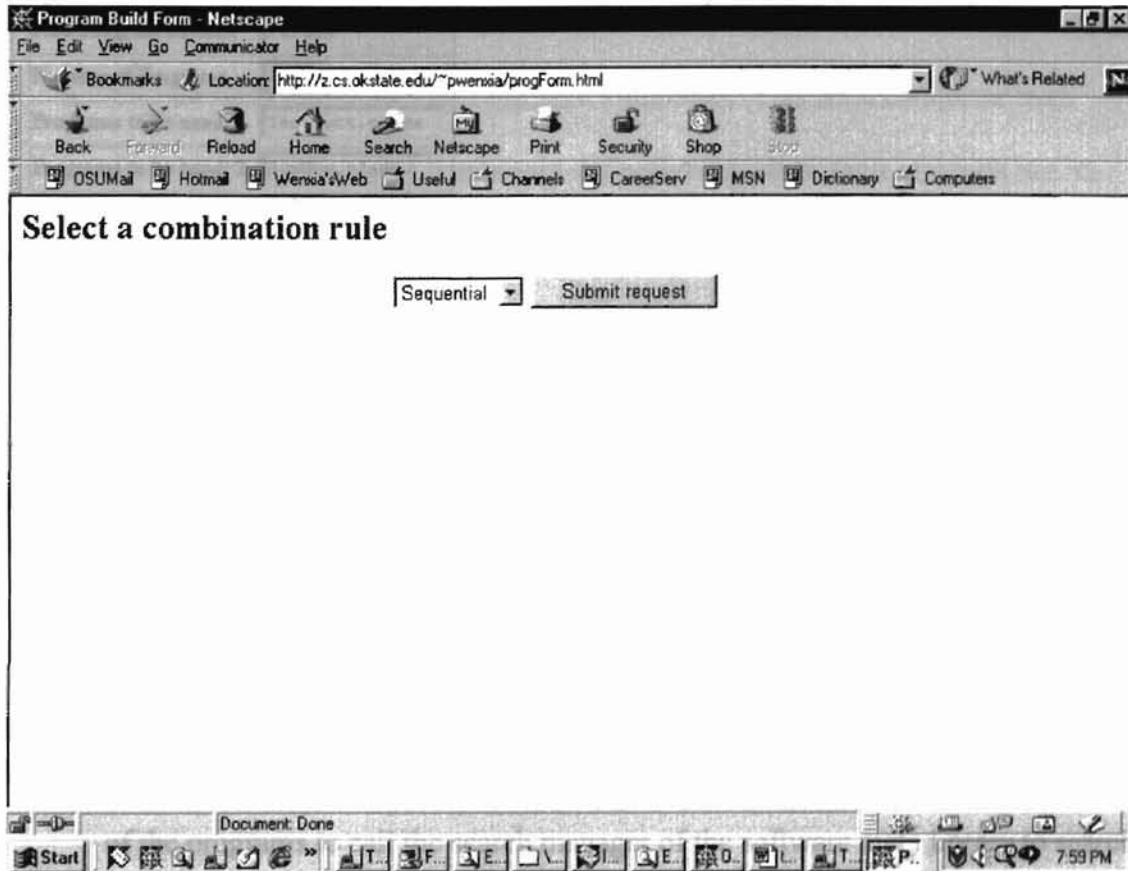


Figure 5.3 Composition Selection Form

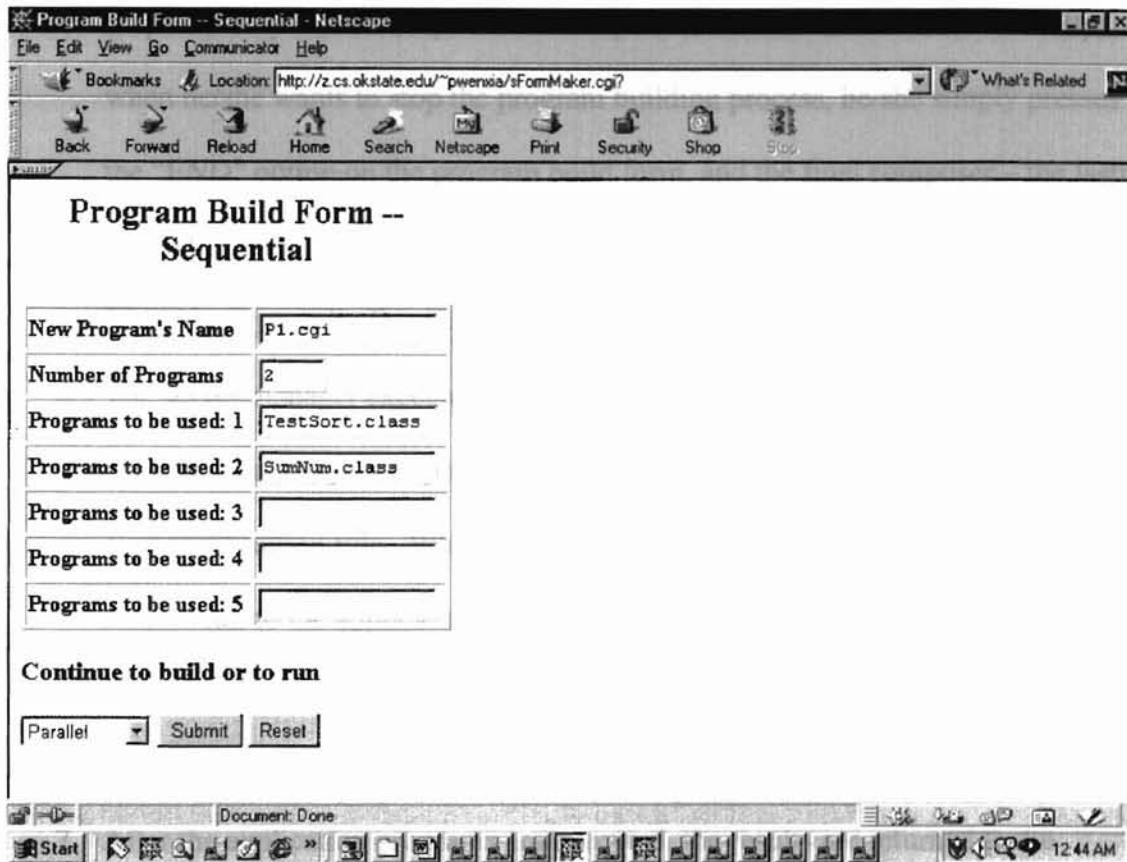


Figure 5.4 Sequential Program Build Form

3. Parse the information obtained from the program build form, and uses the names of the programs to call the search manager of the software meta library to search for template information about the registered programs.
4. After obtaining the template information of the programs from the search manager, the server manager (the CGI program) will use these programs' information to create a new program file (a composer), which contains the composition rule. Refer to the example of composer P1.cgi in Appendix A. A composer may be sent to a remote machine in the network.

5. Step 3 to Step 4 can be repeated as many times as the user wants. Eventually, when he/she wants to stop the program building process, he/she simply presses the “END” option on the program build form, and the final composer – the lastly created executable program is created. Refer to the example of final composer P2.cgi in Appendix B.
6. The server manager executes the final composer. This final composer may contain several layers of previously defined composers, each of which has its own composition rule. When the final composer is executed, it may make remote call to the application manager (also a CGI program) to invoke the execution of the application programs. Each machine containing distributed programs has one application manager.
7. After the application programs are executed, the results are returned to the calling composer. The results of the final composer are returned to the server manager, and the server manager then returns them to the user through the web server.

### 5.2.1 Implementation of Composition Rules

This section illustrates the implementation of the three composition rules.

#### 5.2.1.1 Sequential Composition

In sequential composition, the component programs of a new program are executed in sequential order, which means they run one after the other. Therefore, a for-



loop is used to control the execution order of the programs. The following pseudo-code implements sequential composition:

```
for($i=1; $i<=$num; $i++){
    $results[$i]=`java RemoteCall $pLoc[$i] $pName[$i]
    $inLoc[$i] $inName[$i] $outHost[$i] $outName[$i]`;
    `java chMode $outLoc[$i] $outName[$i]`;
}#for loop
```

In this pseudo-code, the \$num is the number of the component programs. If a new program uses three sequential component programs, we will use a for-loop with three iterations to do the sequential execution. In the first iteration, the first component program is executed; in the second iteration, the second component program is executed; in the third iteration, the third component program is executed. RemoteCall is a java program which will call the remote application manger run.cgi to invoke the executions of application programs. Refer to the complete implementation code, seqSub.cgi, the sequential composition rule in Appendix C.

#### 5.2.1.2 Parallel Composition

In parallel composition, the component programs of a new program are executed in parallel order. If they are located in different machines in the network, then they can be executed concurrently without conflict, since they can be executed under different CPUs. If they are located in the same machine, then they can be executed in any

arbitrary order, sharing one CPU with time slices. This can be done through the use of the system call *fork*. The following pseudo-code illustrates the use of fork function:

```
if (($n2-$n1)<=1){#base case
    if(fork){
        #in parent ... process P1
        $results1=`java RemoteCall $pLoc[$n1] $pName[$n1]
        $inLoc[$n1] $inName[$n1] $outHost[$n1]
        $outName[$n1]`;
        `java chMode $outLoc[$n1] $outName[$n1]`;
        wait;
    }#if
    else{
        #in child ... process P2
        $results2=`java RemoteCall $pLoc[$n2] $pName[$n2]
        $inLoc[$n2] $inName[$n2] $outHost[$n2]
        $outName[$n2]`;
        `java chMode $outLoc[$n2] $outName[$n2]`;
    }#else
}#if base case
```

The fork call creates a new process which is an exact copy of the original. The original process is called the *parent*, and the newly created process is called the *child*. The child has the same data and variable values as the parent. In the parent, the return

value of the fork call is the process ID of the newly created child. In the child, the fork call returns zero. Therefore, the fork call appears inside an *if* test, so that the parent and child can branch to different places. Both the parent and child processes continue to execute from a shared copy of the same code. Usually, when we create a child process, we do so because we want it to execute a totally separate program.

The *wait* function is used to synchronize the two processes. By using fork command, the parent and child processes can be executed concurrently. Therefore, this technique can be used to implement the parallel composition. For parallel general case, which means that there are more than two parallel processes to be executed at the same time, we can use a recursion to call the fork command. For detailed implementation, see the file called `parSub.cgi` in the Appendix D.

### 5.2.1.3 Pipeline Composition

In pipeline composition, the component programs of a new program are executed in pipeline order. Pipelining is the combination of sequential and parallel compositions. As for the flow of data, it is sequential, but as for the flow of computation, it is parallel. The component programs of the new program can be executed at the same time as the different stages of the new program. Therefore, pipeline composition also can use the fork command to implement the parallel flow of computation.

In addition, it uses a pipe function to implement the sequential flow of data. Because the forked process is an exact copy of its parent, it inherits the same filehandles as its parent. The parent sets up a pair of pipe file descriptors using the Perl function `pipe` before forking. The pipe has a read-end PRH and a write-end PWH. This pipe is

used to establish communications between the parent and child processes. The child process will inherit that same pipe file descriptors. The parent is the generator of data, so it calls the Perl function `close` on the redundant read filehandle PRH. The child is a reader, so it closes the redundant write filehandle PWH. Parent and child are now linked by a pipe communication channel. The child then copies the pipe filehandle to its STDIN filehandle, so STDIN will actually read from the pipe. It accomplishes this by calling `open` with PRH as its second argument. STDIN reads data from PRH. Therefore, the parent process can send its output to the PWH of the pipe, and the child process then can read its input from the PRH of the same pipe. PRH reads data from PWH of the same pipe. So, the child process can read its input data from the output of its parent. By using this technique, the data flow of pipeline program can be controlled.

In general pipeline case, which means that there are more than two component programs in the pipeline program, the pipelining can be achieved by using recursion to call the `fork` command and to create the pipe filehandles. Refer to the file `pipeSub.cgi` in the Appendix E for the detailed implementation.

### 5.2.2 Remote Call

When the final composer is executed by the server manager, all layers of the previously defined composers contained in this final composer are executed according to their own composition rules. When these composers are executed, the appropriate subroutines are called according to the composition rules. All three subroutines – `seqSun.cgi`, `parSub.cgi`, and `pipeSub.cgi`, which are CGI programs written in Perl, make a system call to a java program named *RemoteCall*, which invokes the remote

application manager – a CGI program called *run.cgi*. Through the application manager, the application programs can be invoked to execute.

### 5.3 Software Meta Library and Library Manager

The library stores the information about the application programs rather than the application program files themselves. The program files reside in nodes in the network. The software meta library is implemented as a relational database by using mysql database.

All information about the reusable application programs is stored in a table in the database. Each row in the table represents one template of one program. Each column stores a specific piece of information about a program, such as its name, its location, the name of its input, etc.

The library manager consists of two parts: a registry manager and a search manager, both of which are implemented as Java programs, which load the `mm.mysql.jdbc` driver as their JDBC drivers. Through JDBC driver, the library manager can administrate the library, which is located in a remote machine, where the mysql database resides.

The registry manager is responsible for inserting the application programs into templates (rows) in the library. The registry manager is implemented as a java program named Registry. Registry inserts the fields obtained from the registry form into their corresponding columns in the templates. The program name is assigned as the primary key. So, no two templates with the same program name are allowed.

The search manager is responsible for searching the application programs at the request of the server manager and returning the search results to the server manager.

#### 5.4 Application Manager and Application Program(s)

Each machine containing programs in the software meta library has an application manager. The application manager is also a CGI program called run.cgi. The parts of application manager implementation are described in the following sections.

##### 5.4.1 Get Remote Input Data

The application manager connects to a remote input file by calling a Java program named *GetInput*, which is able to access a remote text file or a remote database file. If the remote file is a text file, *GetInput* will use a URL object to connect to the text file in a remote machine. The URL class is in the java.net package. When a URL object is connected to the remote text file, the *GetInput* program can manipulate the file as if it were local. In this sense location transparency is achieved. The *GetInput* program gets the remote text file. If the remote file is a database file, then *GetInput* will use the JDBC, which is a standard Java database connectivity API. By using JDBC API and DBMS specific JDBC drivers, we are able to connect to many relational databases, such as Oracle, Sybase, Mysql, without rewriting the Java program. This thesis uses mm.mysql.jdbc driver, which is a Type 4 JDBC, a pure Java JDBC driver with direct connection to a mysql database. Within the *GetInput* program, the java.sql package must be imported in order to use a JDBC driver. These JDBC base classes contain the

necessary elements for properly instantiating JDBC driver. The JDBC driver `mm.mysql.jdbc` must be loaded in order to connect to the database `mysql`. To load the `mysql` JDBC driver, just one line of code is used:

```
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
```

where `org.gjt.mm.mysql.Driver` is the classpath of the `mm.mysql.jdbc` driver in the WDCMPS web server, and the method `newInstance()` is required by the `mysql` database connection. Next, the connection object must be explicitly created to connect the driver with the database, which can be accessed via its URL address. A valid driver must be registered with the JDBC `DriverManager` before attempting to create this connection.

[24]

The standard way to establish such a connection with a database is to call the method `DriverManager.getConnection`. This method takes a string containing the URL of the database. Once a connection is established, it is used to pass SQL statements to its underlying database. Then, the Java program `GetInput` can retrieve the data we want from the database.

#### 5.4.2 Execution of Application Programs

After getting the remote input data by using `GetInput` program, the application manager executes the local application program(s) by using the interprocess communication mechanism pipe. A command of the following type could be used to pass the input to an executable program:

```
java GetInput | java <name of the java bytecode>, or  
java GetInput | <name of the executable program  
written in other language than java>.
```

The application program, which might be written in any programming language, takes the standard output of the java GetInput program as its input. By this means, there is no need to save a local input file for the program. This method also assumes that all programs read input data from the standard input.

#### 5.4.3 Send Remote Output File

The application manager saves the output, which is a text file, on a remote machine by using Java RMI technology. The prerequisites for Remote Method Invocation (RMI) are as follows:

1. The server (the destination machine where the RMI program's output should be stored) side has to have the sever class, implementation class, which handles the remote method calls, and the skeleton class, which is the proxy of the server. The skeleton class is created through running the implementation program by rmic compiler.
2. The client (the machine where the RMI program resides) side has to have the interface class of the implementation class of the server class, the client class, which is to invoke the remote methods located in the server side, and the stub class, which is the proxy of the client to make request to the server. The stub



class in the client side, like the skeleton class in the server side, is also created by using the rmic compiler.

This remote method invocation takes place as follows:

- The server calls the registry to associate or bind a name with a remote object. Then the server is started up to be ready for the client to connect.
- The client looks up the remote object by its name in the server's registry and then invokes methods on it.
- The client then saves the output as a file in the server side machine as if it saves an output file on the local machine.

#### 5.4.4 Return the Results to the Server Manager

The application manager returns the execution results of the application program to the server manager after the program returns the results to the application manager.

#### 5.5 An Example

This section illustrates the implementation using an example. The user will compose a new program P1.cgi, which is constructed from two sequential programs TestSort.class and SumNum.class. Then he/she will compose a program P2.cgi, which is constructed from two parallel programs P1.cgi and Average. Then he/she ends the program building process. In this case, P1.cgi is a previously defined composer, and P2.cgi is the final composer. P2.cgi program file contains the program name of P1.cgi. Therefore, when the server manager executes P2.cgi program, P2.cgi calls the parallel program subroutine parSub.cgi, which allows P1.cgi and Average to execute at the same

time. Since P1.cgi is a composer which has sequential composition rule, when P1.cgi is executed, the sequential program subroutine seqSub.cgi is called. seqSub.cgi allows TestSort.class and SumNum.class to execute sequentially.

The following steps illustrate the process of the implementation of this program building process:

1. Fill in the registry form:

The user is presented with the registry form as shown in Figure 5.1. He/she fills in the information about the TestSort.class program. When the user clicks the “Register now” button, the server manager calls the registry manager of the library to insert the TestSort.class program into its template in the library. Then the user is presented with the interface shown in Figure 5.2. The user also registers program SumNum.class the same way. Then the user registers the third program Average into the library. After having registered these three programs, the user clicks the “Build the program” link to start building new programs.

2. Select a composition rule:

The composition selection form (Figure 5.3) first appears to allow the user to select a desired composition rule. Then an appropriate program build form, a sequential form, a parallel form, or a pipeline form appears according to the user’s selection. The sequential program build form corresponds to the sequential combination rule, the parallel program build form corresponds to the parallel combination rule, and the pipeline program build form corresponds to the pipeline combination rule. All of the three kinds of forms allow the user to provide the name of the new program, the number

of the executable programs, and the names of the executable programs. In this example, the user selects sequential composition rule.

3. Fill in the sequential program build form:

The sequential program build form, as Figure 5.4 illustrates, consists of two parts.

The first part allows the user to fill in the new program's name, the number of component programs, and the names of the component programs.

The second part of the form has a drop down list, which allows the user to select the choices among the four options – Sequential, Parallel, Pipeline, and END. If the user chooses the “Sequential”, and clicks the “Submit” button, the sequential program build form appears again, allowing the user to continue with the sequential program building process. At this time, a new program (composer) with the user defined name has been created by the server manager, and the composition rule has been built inside this composer. This program is available for use as a component program. If the user chooses the “Parallel”, and clicks the “Submit” button, the parallel program build form appears, allowing the user to build parallel programs. If the user chooses the “Pipeline”, and clicks the “Submit” button, the pipeline program build form appears, allowing the user to build pipeline programs. If the user chooses the “END” button, the program building process will be complete, and the server will execute the lastly created program – final composer. When the final composer is executed, it calls the corresponding subroutine, and the subroutine calls the RemoteCall program to invoke the remote application manager run.cgi to execute the existing programs and the composers.

In this example, the user selects the parallel composition rule from the drop down list. Then Figure 5.5 parallel program build form shows up.

Program Build Form -- Parallel

New Program's Name	P2.cgi			
Number of Programs	2			
Programs to be used	P1.cgi	Average		

Continue to build or to run

END Submit Reset

Figure 5.5 Parallel Program Build Form

4. Fill in the parallel program build form:

The parallel program build form is similar to the sequential one, except that the formats are different. It also consists of two parts. The first part lets the user provide the new program's name, and the information about the executable programs. The second part allows the user to continue to build the programs with the three options, or allows the user to end the program building process with the "END" option. When the

user clicks the “END” option, it means that he/she wants the final composer execution to begin. In this example, the user no longer wants to build new programs, so the “RUN” option in the drop down list is selected. Then Figure 5.6 appears to allow the user to click the “RUN” button to execute the final composer, in this case, P2.cgi.

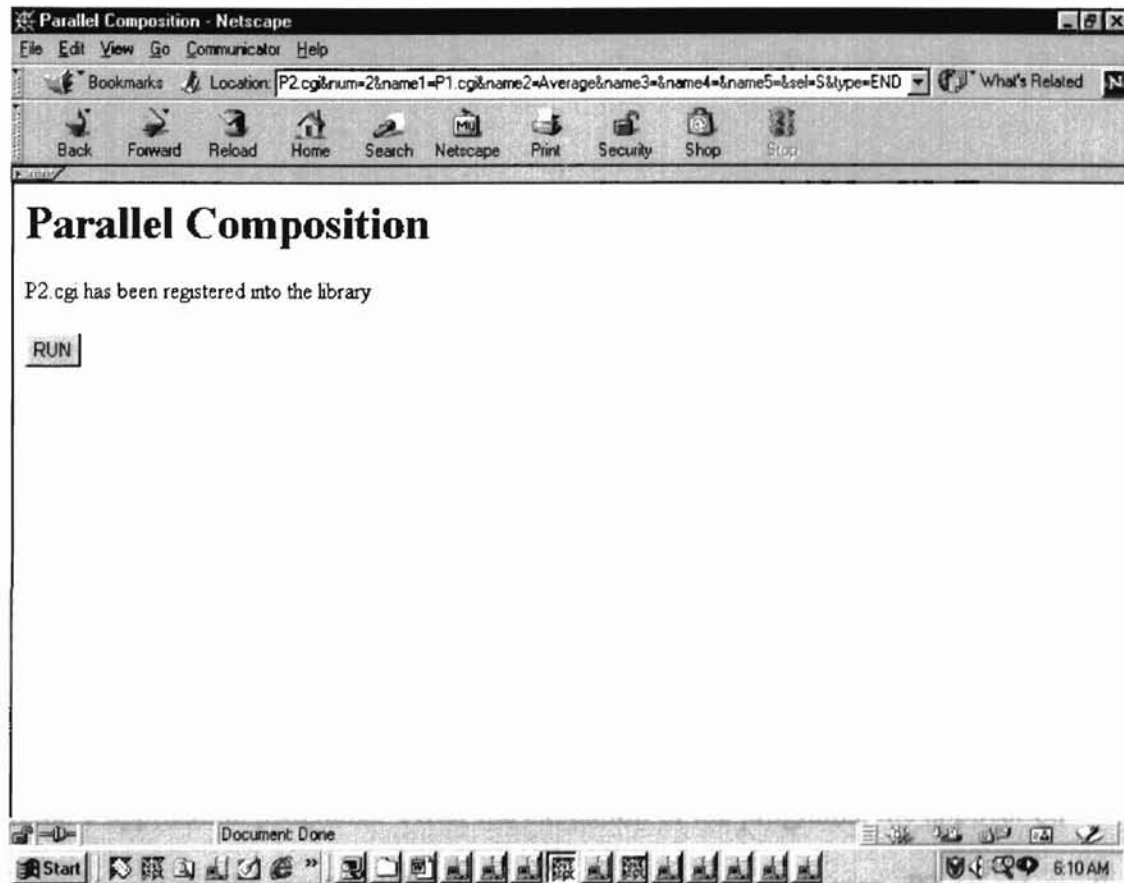


Figure 5.6 Run Form



Figure 5.7 Program Execution Results

Figure 5.7 shows the results after P2.cgi (the final composer) is executed. When it is executed, the remote application managers are called to invoke the execution of the remote application programs. After the execution of the programs, the application manager returns the results to the final composer, P2.cgi in this case. P2.cgi returns the results to the server manager, and the server manager then returns them to the user's web browser through the web server. Thus the results can be displayed on the user's web browser.

The program's execution results are also saved as a text file on a public-accessible remote machine specified by the user. When the user goes to the URL of this remote

text file, the contents of the file will be displayed on the user's web browser, as Figure 5.8, Figure 5.9, and Figure 5.10 illustrate.

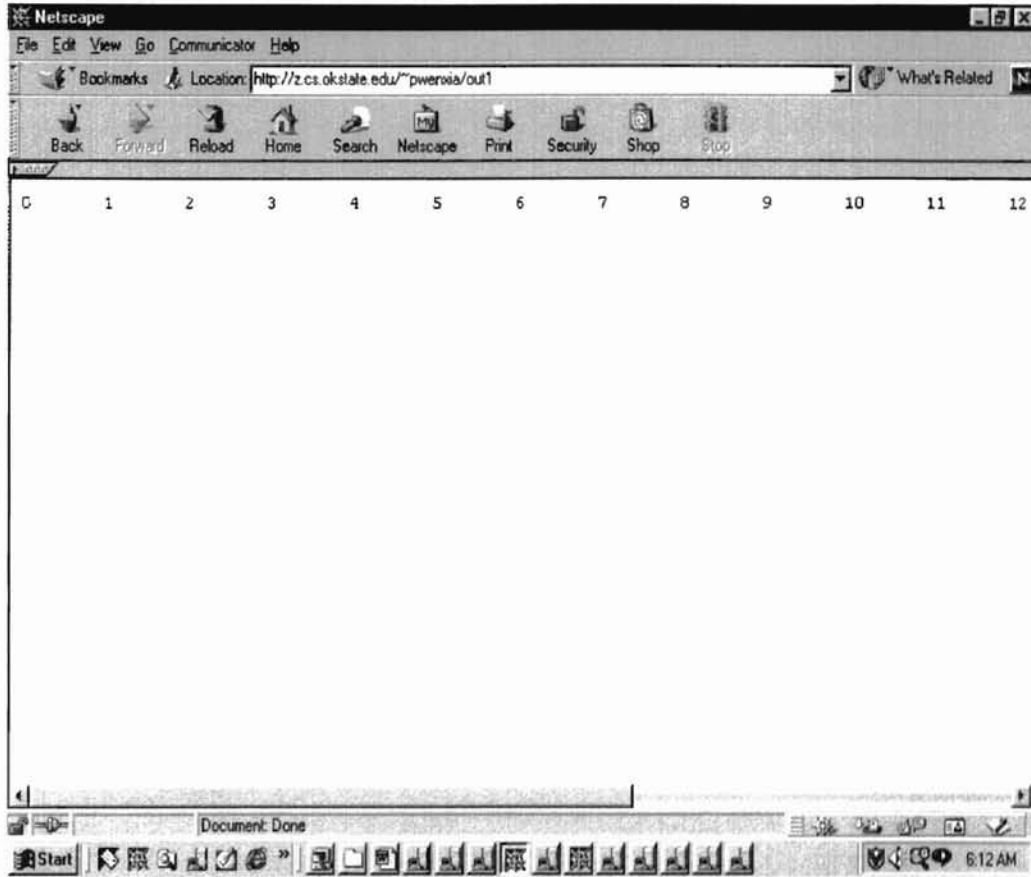


Figure 5.8 out1: Remote Output File for Program TestSort.class



Figure 5.9 out2: Remote Output File for Program SumNum.class





Figure 5.10 out3: Remote Output File for Program Average

## CHAPTER VI

### CONCLUSION AND FUTURE WORK

This thesis proposes a WWW based distributed computing model (WDCM). This model has three major advantages: it (1) provides the user WWW access and interfaces, which is an improvement over CORBA and DCOM, (2) makes use of the existing executable programs to perform new tasks through software meta library templates, and (3) provides the mechanisms to do distributed computing. WDCM combines all the three techniques – distributed computing, software reuse and WWW interface together, and makes meaningful contributions to the distributed computing models.

This thesis designs and implements a programming environment based on WDCM named WDCMPS. WDCMPS is implemented using two programming languages: Java and Perl.

The three composition rules defined in WDCM are implemented. However, the following aspects have not been implemented yet, and are left as future work:

1. A data conversion program
2. An improved software meta library
3. Remote implementation of a composer
4. A performance evaluation scheme. Queuing theory [25] can be used to evaluate the program performance. Due to the network traffic, the remote calls take longer time than local calls, and thus the performance measurement evaluation is

necessary to judge whether the distributed computing in some context is worth doing or not.

## REFERENCES

1. Coulouris, G.F., J. Dollimore, and T. Kindberg, *Distributed Systems Concepts and Design*. 2nd ed. 1994, Wokingham, England, Reading, Massachusetts: Addison-Wesley Publishing Company.
2. Mahmoud, Q.H., *Distributed Programming with Java*. 2000, Greenwich, CT: Manning Publications Co.
3. McCarty, B. and L. Cassady-Dorion, *Java Distributed Objects*. 1999, Indianapolis, Indiana: Sams.
4. Buss, A. and L. Jackson, *Distributed Simulation Modeling: A Comparison of HLA, CORBA, and RMI*. Proceedings of the 1998 Winter Simulation Conference, 1998.
5. Daniel, J., B. Traverson, and V. Vallee, *Active COM: An Inter-working Framework for CORBA and DCOM*. IEEE, 1999.
6. Thompson, D. and D. Watkins, *Comparison between CORBA and DCOM: Architectures for Distributed Computing*. IEEE, 1998.
7. Raj, G.S., *A Detailed Comparison of CORBA, DCOM and Java/RMI*, .
8. Kono, K. and T. Masuda, *Efficient RMI: Dynamic Specialization of Object Serialization*. IEEE, 2000.
9. Maffeits, S. and D. Schmidt, *Constructing Reliable Distributed Communication Systems with CORBA*. IEEE Communications Magazine, 1997.
10. Farley, J., *Java Distributed Computing*. 1998, Cambridge: O'Reilly.

11. Wang, Y.-M., O.P. Damani, and W.-J. Lee, *Reliability and Availability Issues In Distributed Component Object Model (DCOM)*. IEEE, 1997.
12. Hayes, F., *Distributed Component Object Model*. computerworld, 1999. **33**(8).
13. Berghoff, J., *et al.*, *Agent-based Configuration management of distributed applications*. Configurable Distributed Systems, 1996.
14. George, K.M. and K.-S. Kim, *An Adaptive Distributed Computation Support System*. International Conference on Parallel and Distributed Techniques and Applications, 1999.
15. Coulange, B., *Software Reuse*. 1998, London: Springer.
16. Griss, M.L., *Software Reuse Architecture, Process, and Organization for Business Success*. IEEE, 1997.
17. Doublait, S. and C. Lissoni, *Processes for Systematic Software Reuse*. IEEE, 1997.
18. Behle, A., *An Internet-based Information System for Cooperative Software Reuse*. IEEE, 1998.
19. Weaver, A.C., *Profiting from the Internet and the World Wide Web*. IEEE, 1998.
20. Grayson, R.A. and K.M. George, *A Persistent History Navigation Assistant*. Proceedings of the International Conference on Internet Computing IC'2000, 2000.
21. Lam, S.L.Y., *A World Wide Web Resource Discovery System*. Proc. of the 4th International World Wide Web Conference, 1995.
22. Chandy, K.M. and S. Taylor, *An Introduction to Parallel Programming*. 1992, Boston, MA: Jones and Bartlett Publishers.

23. Lawson, H.W., *Parallel Processing in Industrial Real-Time Applications*. 1992, EngleWood Cliffs, New Jersey: Prentice Hall.
24. Haecke, B.V., *JDBC: Java Database Connectivity*. 1997, Foster City, CA: IDG Books Worldwide, Inc.
25. Singhal, M. and N.G. Shivaratri, *Advanced Concepts in Operating Systems*. 1994, New York, NY: McGraw-Hill, Inc.

## APPENDICES

### APPENDIX A

#### Source Code for Composer *Pl.cgi*

```
#!/usr/local/bin/perl
require 'seqSub.cgi';
print "Content-type: text/html\n\n";

$out[1]= "http://chester.cs.okstate.edu:8080/cgi-bin/ TestSort.class
http://www.cs.okstate.edu/~pwenxia/ data1 http://z.cs.okstate.edu/~pwenxia/
out1";
($pLoc[1], $pName[1], $inLoc[1], $inName[1], $outLoc[1],
$outName[1])=split(/t/, $out[1]);
($outProt[1], $rest[1])=split(/:VV/, $outLoc[1], 2);
($temp[1], $remain[1])=split(/V/, $rest[1]);
($outHost[1], $left[1])=split(/:./, $temp[1]);

$out[2]="http://chester.cs.okstate.edu:8080/cgi-bin/ SumNum.class
http://z.cs.okstate.edu/~pwenxia/ out1 http://z.cs.okstate.edu/~pwenxia/
out2";
($pLoc[2], $pName[2], $inLoc[2], $inName[2], $outLoc[2],
$outName[2])=split(/t/, $out[2]);
($outProt[2], $rest[2])=split(/:VV/, $outLoc[2], 2);
($temp[2], $remain[2])=split(/V/, $rest[2]);
($outHost[2], $left[2])=split(/:./, $temp[2]);

$num=2;

&SL($pLoc, $pName, $inLoc, $inName, $outLoc, $outHost, $outName, $num);
```

## APPENDIX B

### Source Code for Final Composer *P2.cgi*

```
#!/usr/local/bin/perl
require 'parSub.cgi';
print "Content-type: text/html\n\n";

$out[1]= "http://z.cs.okstate.edu/~pwenxia/  P1.cgi -1  -1  -1  -1";
($pLoc[1], $pName[1], $inLoc[1], $inName[1], $outLoc[1],
$outName[1])=split(/\t/, $out[1]);
($outProt[1], $rest[1])=split(/\:W/, $outLoc[1], 2);
($temp[1], $remain[1])=split(/\//, $rest[1]);
($outHost[1], $left[1])=split(/\:/, $temp[1]);

$out[2]="http://chester.cs.okstate.edu:8080/cgi-bin/  Average
http://www.cs.okstate.edu/~pwenxia/  data1 http://z.cs.okstate.edu/~pwenxia/
out3";
($pLoc[2], $pName[2], $inLoc[2], $inName[2], $outLoc[2],
$outName[2])=split(/\t/, $out[2]);
($outProt[2], $rest[2])=split(/\:W/, $outLoc[2], 2);
($temp[2], $remain[2])=split(/\//, $rest[2]);
($outHost[2], $left[2])=split(/\:/, $temp[2]);

$n1=1, $n2=2;

&PL($pLoc, $pName, $inLoc, $inName, $outLoc, $outHost, $outName, $n1,
$n2);
```



## APPENDIX C

### Source Code for Sequential Execution Subroutine *seqSub.cgi*

```
#!/usr/local/bin/perl
# seqSub.cgi
# is used to run the sequential programs
# It is called by a composer, such as P1.cgi

sub SL{
  for($i=1; $i<=$num; $i++){
    if($inLoc[$i] eq "-1"){
      `./$pName[$i]`;
    }#if
    else{
      $results[$i]=`java RemoteCall $pLoc[$i] $pName[$i] $inLoc[$i]
        $inName[$i] $outHost[$i1] $outName[$i]`;
      `java chMode $outLoc[$i] $outName[$i]`;

      print <<ETX;
      Result of $pName[$i]: $results[$i]
      <br>
    }#else
  }#for loop
}#SL
1;
```

## APPENDIX D

### Source Code for Parallel Execution Subroutine *parSub.cgi*

```
#!/usr/local/bin/perl
# parSub.cgi
# is used to run the parallel programs
# it is called by a composer, such as P2.cgi

sub PL{
  if(($n2-$n1)<=1){#base case
    if(fork){
      #in parent ... process P1
      if($inLoc[$n1] eq "-1"){
        $out[$n1]=`./$pName[$n1]`;
        print <<ETX;
        Result of $pName[$n1]: $out[$n1]
        <br>
ETX
      }#if
      else{
        $results1=`java RemoteCall $pLoc[$n1] $pName[$n1] $inLoc[$n1]
        $inName[$n1] $outHost[$n1] $outName[$n1]`;
        `java chMode $outLoc[$n1] $outName[$n1]`;

        print<<ETX;
        Result of $pName[$n1]: $results1
        <br>
ETX
      }#else
      wait;
    }#if
    else{
      #in child ... process P2
      if($inLoc[$n2] eq "-1"){
        $out2=`./$pName[$n2]`;
        print <<ETX;
        Result of $pName[$n2]: $out2
        <br>
ETX
      }#if
      else{
        $results2=`java RemoteCall $pLoc[$n2] $pName[$n2] $inLoc[$n2]
        $inName[$n2] $outHost[$n2] $outName[$n2]`;
        `java chMode $outLoc[$n2] $outName[$n2]`;
        print <<ETX;
```

```

        Result of $pName[$n2]: $results2
        <br>
ETX
        }#else
        }#else
    }#if base case
    else{#recursion case
        if(fork){
            #in parent ... process P1
            if($inLoc[$n1] eq "-1"){
                $out3=`./$pName[$n1]`;
                print <<ETX;
                Result of $pName[$n1]: $out3
                <br>
ETX
            }
            else{
                $results3=`java RemoteCall $pLoc[$n1] $pName[$n1] $inLoc[$n1]
                $inName[$n1] $outHost[$n1] $outName[$n1]`;
                `java chMode $outLoc[$n1] $outName[$n1]`;

                print <<ETX;
                Result of $pName[$n1]: $results3
                <br>
ETX
            }
            wait;
        }#if
        else{
            #in child ... call itself recursively
            &PL(@pLoc, @pName, @inLoc, @inName, @outLoc, @outHost,
            @outName, $n1++, $n2);
        }#else
    }#else recursion case
}#subroutine
1;

```

Source Code for Pipeline Execution Subroutine *pipeSub.cgi*

```
#!/usr/local/bin/perl
# pipeSub.cgi
# is used to run the pipeline programs
# It is called by a composer

sub PIPE{
    pipe("PRH$n1", "PWH$n1") || die "pipe $!";

    if(($n2-$n1)<=1){#base case
        if(fork){
            #in parent ... process P1

            # Writer close the pipe read side if $n1!=1
            if($n1==1){
                close("PRH$n1");
            }
            else{# Redirect STDIN
                close(STDIN);
                $k=$n1-1;
                open(STDIN, ">&PRH$k");
            }

            # Redirect STDOUT
            close(STDOUT);
            open(STDOUT, ">&PWH$n1");
            select(STDOUT); $|=1;

            # Pour STDIN down the pipe.
            if($inLoc[$n1] eq "-1"){
                `./$pName[$n1]`;
            }#if
            else{
                if($n1==1){
                    $tmpOut[$n1]=`java GetInput $inLoc[$n1]
                    $inName[$n1]|java RemoteCall1 $pLoc[$n1]
                    $pName[$n1]`;
                }#if
                else{
                    $tmpOut[$n1]=`java RemoteCall1 $pLoc[$n1]
                    $pName[$n1]`;
                }#else
            }#else
        }#else
    }
}
```

```

        print $tmpOut[$n1];
        print "end\n";
        close(STDOUT);
        wait;
        open(STDOUT, ">/dev/tty" );
        print "$0 existing\n";
        exit(0);
    }#if
    else{
        #in child ... process P2
        # Reader close write side
        close("PWH$n1");
        # Make read side of the pipe our STDIN
        close(STDIN);
        open(STDIN, ">&PRH$n1");
        select(STDIN); $|=1;

        if($inLoc[$n2] eq "-1"){
            `./$pName[$n2]`;
        }
        else{
            `java RemoteCall1 $pLoc[$n2] $pName[$n2]||java
            IORMIClient $outHost[$n2] $outName[$n2]`;
            `java chMode $outLoc[$n2] $outName[$n2]`;
        }#else
    }#else
}#if base case

else{#recursion case
    if(fork){
        #in parent ... P1

        # Writer close the pipe read side if $n1!=1
        if($n1==1){
            close("PRH$n1");
        }
        else{# Redirect STDIN
            close(STDIN);
            $k=$n1-1;
            open(STDIN, ">&PRH$k");
        }

        # Redirect STDOUT
        close(STDOUT);
        open(STDOUT, ">&PWH$n1");
        select(STDOUT); $|=1;
    }
}

```

```

# Pour STDIN down the pipe.
if($inLoc[$n1] eq "-1"){
    `./$pName[$n1]`;
}#if
else{
    if($n1==1){
        `java GetInput $inLoc[$n1] $inName[$n1]`;
        $tmpOut[$n1]=`cat file|java RemoteCall1 $pLoc[$n1]
        $pName[$n1]`;
    }#if
    else{
        $tmpOut[$n1]=`java RemoteCall1 $pLoc[$n1]
        $pName[$n1]`;
    }#else
}#else

print $tmpOut[$n1];
print "end\n";
close(STDOUT);
wait;
}#if
else{
    #in child ... call itself recursively
    &PIPE($pLoc, $pName, $inLoc, $inName, $outLoc, $outHost, $outName,
    $n1++, $n2);
}#else
}#else recursion case
}#subroutine
1;

```

## APPENDIX F

### Source Code for Application Manager *run.cgi*

```
#!/usr/local/bin/perl
print "Content-type: text/html\n\n";
@lines=<STDIN>;

($prog, $suffix)=split(/\./, $lines[0], 2);
chop $suffix;

for($j=1; $j<5; $j++){
    chop $lines[$j];
}
if($suffix eq "class"){
    $results=`java GetInput $lines[1] $lines[2]java $progjava IORMIClient
    $lines[3] $lines[4]`;
}
else{
    $results=`java GetInput $lines[1] $lines[2]|.$progjava IORMIClient
    $lines[3] $lines[4]`;
}

print $results;
exit 0;
```

VITA

Wenxia Peng

Candidate for the Degree of

Master of Science

Thesis: DESIGN AND IMPLEMENTATION OF A WORLD WIDE  
WED BASED DISTRIBUTED COMPUTING MODEL

Major Field: Computer Science

Biographical:

Education: Graduated from East China Normal University, Shanghai, China in July, 1988; received Bachelor of Arts degree in Foreign Languages and Literatures. Graduated from Peking University, Beijing, China in July, 1992; received Master of Arts degree in Oriental Languages and Literatures. Studied as Ph.D. Program student at UCLA (University of California, Los Angeles). Completed the requirements for the Master of Science degree at Oklahoma State University in December, 2000.

Professional Experience: Lecturer from 1988 to 1992 at Shanghai Petrochemical Institute, Shanghai, China. Teaching Assistant from 1995 to 1996 in Department of East Asian Languages and Cultures, UCLA (University of California, Los Angeles). Teaching Associate from 1996 to 1997 in Department of East Asian Languages and Cultures, UCLA. Instructor in 1997 at UCLA Extension. Teaching Assistant from 1999 to 2000 in Department of Computer Science, Oklahoma State University, Stillwater, Oklahoma.