2004

A decentralized code resource sharing model for grid computing.

Fan Wang
University of Windsor

Recommended Citation

Wang, Fan, "A decentralized code resource sharing model for grid computing." (2004). *Electronic Theses and Dissertations*. 586.
https://scholar.uwindsor.ca/etd/586

A Decentralized Code Resource Sharing Model For Grid Computing

By

Fan WANG

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2004

Your file Votre référence
ISBN: 0-612-92481-5
Our file Notre référence
ISBN: 0-612-92481-5

ABSTRACT

Grid computing is concerned with developing a conceptual infrastructure for resource sharing among geographically distributed virtual organizations. One important type of resource for Grid application developers is code resource, typically accessed through code repositories that store reusable software components. Current code resource sharing mechanisms use the client/server model, which does not support distributed code repositories. In this thesis, we approach the problem of distributed code resource sharing in Grid environment by proposing a new model that provides a decentralized, dynamic, scalable and heterogeneous solution.

In our model, we use Peer-to-Peer (P2P) technology to support code resource sharing in Grid context. Within our prototype software system, every computer is a Grid "servent" node, which acts as both server and client by providing and requesting for reusable software components. Thus, a distributed code resource sharing community is established indeed.

In this thesis we report on the design and implementation of a prototype code resource sharing software system. This system provides an effective way to aid Grid software developers to obtain access to distributed code resources in a decentralized way, featuring good economy and high autonomy within Grid networks.

DEDICATION

To my parents, for their endless love!

To my siblings, for their care and support!

ACKNOWLEDGEMENTS

My greatest thanks must be given to my supervisor, Dr. Robert D. Kent, for his continuous encouragement and guidance. Without his enlightened help and comments, this thesis work would never have been done.

I am also grateful to Dr. Chen and Dr. Zamani for their guidance and valuable suggestions. Thanks also go to Dr. Yuan for as the chair of the committee.

I would like to thank my friends and colleagues at University of Windsor for their earnest help during my thesis work.

Finally, I would like to acknowledge financial support in the form of research assistantships received during 2003 to 2004 from the Sharcnet Research Fellowships program through a grant to Dr. Robert D. Kent.

TABLE OF CONTENTS

ABSTRACT iii
DEDICATION iv
ACKNOWLEDGEMENTS v
LIST OF FIGURES viii
LIST OF TABLES x

1. INTRODUCTION 1

1.1 Overview of Grid 1
1.1.1 Grid Concept 1
1.1.2 Grid Architecture 3
1.1.3 Grid Characteristics 5
1.1.4 Relationship With Other Distributed Technologies 6
1.2 CodeNet Project 7
1.3 Thesis Statement 9
1.4 Organization of the Thesis 11

2. BACKGROUND 12

2.1 Overview of Peer-to-Peer 12
2.1.1 Peer-to-Peer Concept 12
2.1.2 Peer-to-Peer Architecture 13
2.1.3 Peer-to-Peer Discovery Approaches 15
2.1.4 Peer-to-Peer Characteristics 18
2.1.5 Peer-to-Peer Applications 20
2.1.6 JXTA Project 21
2.2 Overview of Software Reuse 23
2.3 Chapter Summary 27

3. RELATED WORK 28

3.1 Jini 28
3.1.1 Overview 28
3.1.2 Jini Resource Sharing Mechanism 29
3.1.3 Advantages and Disadvantages 32
3.2 Globus 33
3.2.1 Overview 33
3.2.2 Globus Resource Sharing Mechanism 34
3.2.3 Advantages and Disadvantages 37
3.3 Napster 38
3.4 Gnutella 40
3.5 Chapter Summary 42

4. PROTOTYPE DESIGN & IMPLEMENTATION 44

4.1 Motivation 44

4.2 System Design Goals 45
4.3 Prototype Model Overview 46
4.4 Prototype Model Design 47
4.4.1 Roles in Model 47
4.4.2 Working Modes in Model 50
4.4.3 Communication Messages in Model 53
4.4.4 Structures of System Roles 55
4.4.5 Scale Control in Model 58
4.4.6 Domain Split and Mergence in Model 59
4.5 Development Environment 60
4.6 System Design Specification 60
4.6.1 System Use Cases 61
4.6.2 System Class Diagrams 63
4.6.3 System Sequence Diagrams 67
5. SYSTEM DEPLOYMENT AND INTERFACE 76
5.1 System Setup 76
5.1.1 Setup for the role of Code Request Peer 76
5.1.2 Setup for the role of Code Service Peer 77
5.1.3 Setup for the role of Code Service SuperPeer 78
5.2 System Interface 79
5.2.1 Code Request Peer Interface 79
5.2.2 Code Service Peer Interface 83
5.2.3 Code Service SuperPeer Interface 83
6. SYSTEM EVALUATION 85
6.1 Interoperability 85
6.2 Performance 86
6.3 Scalability 90
6.4 Implementability 91
6.5 Limitations 91
7. CONCLUSION AND FUTURE WORK 93
7.1 Conclusion 93
7.1.1 Innovation 93
7.1.2 Achievements 93
7.2 Suggestions For Future Research 94
REFERENCES 96
VITA AUCTORIS 101

LIST OF FIGURES

Figure 1-1: The layered Grid architecture and its relationship to the Internet protocol architecture [Foster01] 3
Figure 2-1: A Simple Taxonomy of Computer Systems 13
Figure 2-2: P2P System Architecture 14
Figure 2-3: Centralized Directory Approach 16
Figure 2-4: Flooded Requests Approach 17
Figure 2-5: Document Routing Approach 18
Figure 2-6: The JXTA Protocol Suite 22
Figure 2-7: Reuse-oriented software development process with Software Reuse 26
Figure 3-1: Jini Resource Sharing Architecture and Workflow 30
Figure 3-2: The Globus Resource Management Architecture [Foster98] 35
Figure 3-3: Napster Content Sharing Mechanism 39
Figure 3-4: Gnutella Content Sharing Protocol 41
Figure 4-1: Overall Architecture of the Proposed Model 47
Figure 4-2: Multicasting Mode for Code Resource Searching 51
Figure 4-3: System Workflow in Multicasting Mode 51
Figure 4-4: System Workflow in Routing Mode 52
Figure 4-5: Communication Messages for CRP 54
Figure 4-6: Communication Messages for CSP 55
Figure 4-7: Communication Messages for CSS 55
Figure 4-8: Code Request Peer Structure 56
Figure 4-9: Code Service Peer Structure 57
Figure 4-10: Code Service SuperPeer Structure 58
Figure 4-11: Use Case Diagram for Code Request Peer 61
Figure 4-12: Use Case Diagram for Code Service Peer 62
Figure 4-13: Use Case Diagram for Code Service SuperPeer 62
Figure 4-14: Code Request Peer (CRP) Class Diagram 64
Figure 4-15: Code Service Peer (CSP) Class Diagram 65
Figure 4-16: Code Service SuperPeer (CSS) Class Diagram 66
Figure 4-17: CRP Broadcast Query Sequence Diagram 68
Figure 4-18: CRP Route Query Sequence Diagram [Within Domain] 69
Figure 4-19: CRP Route Query (to CSS) Sequence Diagram [Across Domain] 70
Figure 4-20: CSP Search and Retrieve Code Sequence Diagram 71
Figure 4-21: CSS Forward Query Sequence Diagram [Query from CRP] 72
Figure 4-22: CSS Forward Query Sequence Diagram [Query from CSS] 73
Figure 4-23: CSS Search Code Sequence Diagram 74
Figure 4-24: CSS Store Code Sequence Diagram 75
Figure 5-1: CRP Setup package in Windows98/2000 76
Figure 5-2: Microsoft Access^{TM} Database Setup in Windows 77
Figure 5-3: Startup MySQL Database Service in Windows 78

Figure 5-4: Domain Setting Dialog GUI........79
Figure 5-5: GUI for Code Request peer........80
Figure 5-6: GUI Example of Code Request Peer in multicasting mode........82
Figure 5-7: GUI Example of Code Request Peer in routing mode........82
Figure 5-8: GUI of Code Service Peer........83
Figure 5-9: GUI of Code Service SuperPeer........83
Figure 6-1: Chart for Performance Comparison on Windows2000 with Access^{TM}........88
Figure 6-2: Chart for Performance Comparison on Linux with MySQL........90

LIST OF TABLES

Table 3-1: Core Globus Services [Foster98] 33
Table 3-2: Comparison of the Existing Models with C/S Model 43
Table 6-1: Testing Result Table for performance on Windows2000 with Access™ 87
Table 6-2: Testing Result Table for performance on Linux with MySQL 89
Table 7-1: Comparison of the C/S model and proposed model 94

1. INTRODUCTION

In this thesis we report on the design and implementation of a prototype software system for distributed code resource sharing in the context of Grid computing. In this first chapter, we introduce the Grid environment in which our model applies, the CodeNet project, thesis statement and the organization of this thesis.

1.1 Overview of Grid

1.1.1 Grid Concept

With the rapid development of computer technology, more and more computers located at different sites are connected together to form a Local Area Network (LAN). The demand for powerful distributed systems, which can take advantage of geographically distributed computing resource, increases accordingly. "...A distributed system consists of a collection of autonomous computers linked by a computer network and equipped with distributed system software, which enables computers to coordinate their activities and to share the resources of the system -- hardware, software, and data..." [Coulou94]. In recent years we have seen a rapid evolution of technologies in distributed systems. From network protocols to distributed system infrastructures, new concepts have been introduced. However, as the paradigm of distributed system expands, new interconnectivity and interoperability challenges arise. The conventional distributed computing technologies can not fully overcome those challenges.

As a result, the concept of Grid has evolved to describe a set of resources distributed over wide-area networks that can support large-scale distributed applications. Grid computing is defined as a "...hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" [Foster99]. Since its creation, the term "Grid" has been used in so many contexts that it has become difficult to get a clear picture of what Grid really means. In [Foster01], Foster, Kesselman, and Tuecke attempted to erase the confusion by defining the Grid problems as "...coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations." In [Foster02a], Foster set up a three-point check list for determining whether a distributed system is a Grid application, those check points include: (1) coordinated resources that are not subject to centralized control. (2) use of standard, open, general-purpose protocols and interfaces. (3) delivery of nontrivial qualities of service.

A Grid is an infrastructure for globally sharing data or compute-intensive resources such as large-scale data sets, supercomputers or computational clusters. Different from conventional distributed systems, Grids focus on the interoperability between different platforms and dynamic large-scale resource sharing among multiple virtual organizations (VOs). The concept of VO is central to Grid computing. In a simple view, a VO is a set of participants with various relationships that wish to share resources to perform some specific tasks. VO examples include: a group of researchers from computer science and life science work together to decipher information from an immense amount of data that is generated by gene research of life science [Mewes00]; several consultants engaged by a car manufacturer to perform scenario evaluation during planning for a new factory. As these examples suggest, the Grid problem is central not only to science research, but also to industry, where the coordination of distributed resources both within and across enterprises is increasingly important.

The resource sharing that Grid enables among VOs is not only file exchange but also direct access to Grid networking resources, as is required by a number of collaborative problem-solving and resource brokering strategies emerging in science, industry and engineering. This kind of sharing is highly controlled with resource sharing policies defined clearly by autonomous resource providers that govern access to resources. In Grid, there exist multiple types of resource, including computational resource, storage resource, network resource, code repositories and scientific instruments to name but a few. In this thesis, we focus on sharing mechanisms for code repositories.

1.1.2 Grid Architecture

In Grid environments, VOs vary tremendously in their size, duration, purpose, structure, community and supporting device platforms. Since the traditional distributed technologies do not support an integrated approach to the wide variety of required services and resources, and also lack of flexibility and control needed for enabling the types of resource sharing necessary, there is a need to define a Grid software infrastructure to support heterogenous VOs. In [Foster01], the Grid architecture presented places a large emphasis on interoperability as it is fundamental to ensure that VO participants can share heterogeneous resources dynamically. The architecture organizes components into layers, as shown in Figure 1-1 [Foster01]. Components within each layer share common characteristics, but can build on capabilities and behaviors provided by any lower layer.

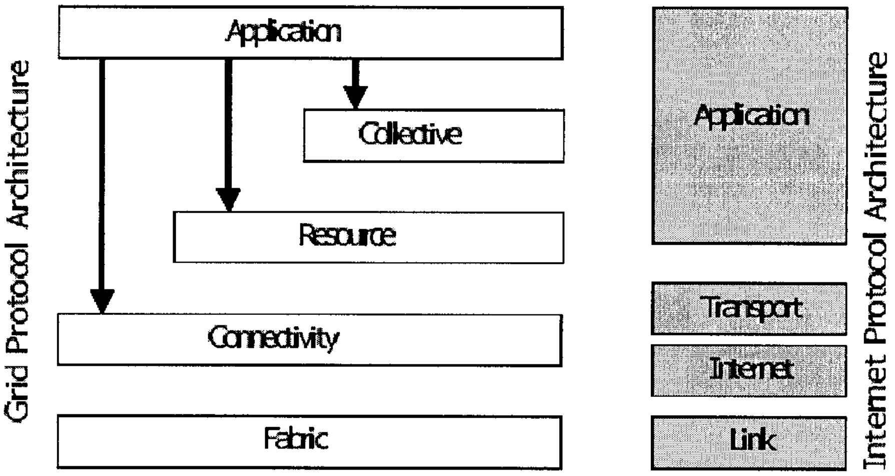


Figure 1-1: The layered Grid architecture and its relationship to the Internet protocol architecture [Foster01]

The Grid fabric layer provides the lowest level of access to actual native resource and implements the low-level mechanisms that allow those resources to be accessed and used. More specifically, those mechanisms must include at least, state enquiry and resource management mechanisms, each of which must be implemented for a large number of local systems.

The Grid connectivity layer defines communication and authentication protocols for network transactions between resources. Communication protocols enable data exchange between Grid users and resources. Authentication protocols provide cryptographically secure mechanisms for verifying the identifying Grid users and resources. Many communication protocols in the connectivity layer are drawn from TCP/IP protocols stack such as IP, ICMP, TCP, UDP, DNS and so on. In this thesis, our prototype system utilizes UDP, IP and DNS protocols for communication.

The Grid resource layer builds on the connectivity layer to implement protocols that enable the use and sharing of individual resource. More specifically, two fundamental components in this layer are (1) information protocols for querying the state of a resource and (2) management protocols to negotiate access to a resource.

The Grid collective layer focuses on the coordination of multiple resources. Examples of functionalities include resource discovery, co-allocation, scheduling and monitoring. This is the layer where a wide variety of challenging distributed computing questions must be answered.

Finally, the application layer is where VO applications are implemented and may use several of the previous layers. As shown in Figure 1-1, the application layer is not required to be based strictly on collective layer. For our prototype system in this thesis, the application is based on Grid connectivity layer directly.

The description of this architecture in [Foster01] (Page 6-13) uses components defined in the Globus toolkit [Globus] as concrete examples of Grid layer implementations. Globus toolkit provides a set of tools and basic services that are currently leveraged by many Grid efforts. Beyond the Globus toolkit, the Global Grid Forum [GGF] is a consortium of over 1000 academic researchers and industrial partners whose goal is to make recommendations for protocols and system design standards development in Grid infrastructure.

1.1.3 Grid Characteristics

Distinguished from conventional distributed systems, Grid has many unique characteristics as described in [Foster98] and [Foster02a]. We summarize those characteristics as follows:

1. Decentralized Control: A Grid integrates and coordinates resource and users that live within different control domains. Usually a Grid domain physically represents a LAN connected with other LANs. In every domain there exists the native resource management policy. There is no universal centralized control mechanism to rule over all the components in all Grid domains.

2. Heterogeneity: A Grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. Since a Grid involves many administrative domains, which may vary tremendously in their size, purpose, structure and supporting platforms, these protocols and interfaces must be standard and open to achieve interoperability among heterogenous domains.

3. Dynamism: A Grid contains a great number of large-scale resources, which may be in geographically distributed national or international groups. In each group, resource and users can join or leave Grid community dynamically without affecting the function of other Grid components.

4. High Performance: A Grid allows its constituent resource to be used in a coordinated style to deliver various qualities of service, so the utility of the combined system is greater than that of the sum of its parts.

5. Scalability: A Grid may increase in its size when new resource joins or new domains incorporate. This kind of change will not lower the performance of whole Gird system, but increase the capability of Grid and make Grid more powerful to delivery higher quality of services.

Because of the above distinctive and demanding characteristics, many powerful Grid application systems have been produced in recent years. These systems display great capability in areas including data-intensive computing, distributed computing, collaborative work and remote access to scientific facilities.

1.1.4 Relationship With Other Distributed Technologies

The concept of controlled, dynamic sharing within VOs is so fundamental that some problems and challenges in Grid also appear partially or completely in other distributed systems. Therefore, Grid system can work with other existing technologies for distributed resource sharing. For example, Grid can integrate web technologies such as Java Server Pages (JSP) to build "VO Portals" that provide thin client interfaces to sophisticated VO applications, WebOS addresses some of these issues as described in [Vahdat98]; A company that has built an enterprise system using Java 2 Enterprise Edition (J2EE) and CORBA can use Grid technology to integrate its local network with other external systems of other companies to form an enterprise market.

In recent years, there has been a strong convergence of interests among Grid computing, Peer-to-Peer (P2P), Internet and Enterprise computing. A typical example is the concept of Open Grid Services Architecture (OGSA) [Tuecke03] [Foster02b] [Foster02c], which is an area of intensive interest for Grid research now. In essence, OGSA is the combination of concepts and technologies from the Grid and Web services communities. In this thesis work, we focus on the integration of P2P technology with Grid computing.

Grid is a very broad and abstruse topic in its domain of application and raises research questions and challenges that span many areas of distributed systems, and even of computer science in general. In [Casanova02], some critical issues related to the future of Grid development are summarized. Resource dissemination and retrieval is one of those important issues. It is also what this thesis focuses on.

1.2 CodeNet Project

As more and more computers are connected together through network technology, users can solve complex problems by utilizing greater computing resources through networks. Among all the Grid resources, code resource is an important type of resource for software developers. It is formed by repositing existing software code components, which then can be used as building blocks to construct new software systems. Code resource sharing means making those reusable code components in code repository available for distributed users to search and retrieve.

At the present time software programming codes are being produced with exponential increase, but among all the produced codes, complete or partial redundancy exists, and the reuse of codes is rare even though software development is difficult, especially within parallel and Grid computing environments. One of the main reasons for this situation is: the shortage of good applications or tools to assist software developers to search for and retrieve suitable reusable software components from code repositories through networks. To change this situation, the concept of CodeNet is brought forward.

CodeNet is a research project conducted by Dr. R. Kent at the University of Windsor, and directed at investigating issues involving both software reuse and Grid computing. One objective of this project is to create Grid-enabled applications and tools that aim to provide efficiently accessible distributed repositories for code developers within the context of virtual communities served by Grid networks.

A lot of research and practice has been made on this project: In [Zhong00] and [Zhang00], a prototype system named DORLM (Distributed Object-based Software Reuse Library Module) was designed and constructed for software reuse and reuse-oriented program development in a distributed computing context; In [Yu01], a multi-server code service system model was suggested to solve the fundamental problem of work balance for efficiency and fail over for availability; In [Zhao02], object-relational database technology and the Java JINI concept were introduced into the CodeNet project to establish a centralized object–relational database-based code service retrieval system within a Grid test-bed environment.

In these works, the prototype systems all adopted a Client/Server (C/S) infrastructure, in which all the available reusable code components are collected together and stored in a server computer. Users must send requests to server computer and retrieve server-side code components through network. This model can provide good performance within LAN with limited clients. But, as the network becomes pervasive and large-scaled, this code-resource sharing model will display some vital limitations because of its high dependency on central server and increased cost of server maintenance. Besides, with the appearance of the advanced Grid computing technology, code repositories are increasingly prone to wide distribution, which is not fully supported by traditional C/S model. Therefore, a new code resource sharing mechanism is in urgent need for Grid networks environment. Based on the previous research on CodeNet Project, in this thesis we import Peer-to-Peer (P2P) technology to replace the traditional C/S code resource sharing model with a new approach.

To the extent that our research goal involves a peer-to-peer based approach to many possible, different types of content sharing, we have also included a discussion of how this approach is used to support sharing of code source files specifically, in the context of CodeNet as it evolves. Thus, in this thesis, we view code sharing to mean a generalization of more specific content sharing evidenced by some of the examples referenced later, such as Napster.

1.3 Thesis Statement

This thesis presents the design and implementation of a new decentralized code resource sharing model, in which peer-to-peer (P2P) technology is used to address typical known limitations of content sharing systems, as exemplified by the CodeNet project. These limitations include, but are not limited to:

1. High dependency: Current systems are too dependent on central code resource server. If the server fails over, the whole system will stop working.
2. High cost: The cost for purchasing and maintaining the hardware and software in server-side machine is too high.
3. Limited scalability: Each client must send requests to the central server for searching and retrieving reusable software components. As the number of clients increases, the load and bandwidth demands on the server also increases, eventually preventing the server from handling additional clients.
4. High centralization: Current code sharing systems only support centralized code resources, and do not support distributed code repositories among multiple Grid domains.

We design and implement a new model for support of distributed code resource sharing in Grid context. It overcomes the limitations of the conventional code resource sharing mechanism with improved decentralization, interoperability, scalability and dynamism.

In our proposed model, source code components can be stored into and retrieved from distributed code repositories in a decentralized manner, thus the whole system avoids dependency on central code repository server and reduces cost by sharing cost among all peers. Based on the previous research of CodeNet project, our thesis extends the CodeNet application system with a decentralized P2P based approach. With our model, we can further establish a Grid resource-sharing framework into which many existing Grid applications and services can be incorporated for the next step. Our system can also be taken as a code service to be integrated into Open Grid Services Architecture (OGSA) [Tuecke03] [Foster02b] [Foster02c]

The objectives of this thesis can be summarized as follows:

- Propose and present a new model that supports distributed code repository based sharing in Grid environments, where distributed code refers to generalized content data.
- Design and construct a code resource retrieval prototype system that works in a decentralized manner to overcome the shortcomings that exist in the previous CodeNet implementations and other Client-Server based infrastructure applications.

The qualities that a reasonable grid solution must possess include decentralized control, heterogeneity, dynamism, high performance and scalability, as stated previously. Also, a good solution must overcome the limitations of dependencies, costs, scalability and centralization. In order to meet all these goals it is necessary to develop a regimen of objective testing and evaluation; this is also a vital aspect of this thesis research.

We note that throughout the thesis we often refer specifically to the CodeNet project. The intention of this research project is to develop a laboratory framework and test bed to investigate and analyze various aspects of code sharing on many levels. As such, CodeNet is a generic representation of many other resource sharing systems and may be understood to provide a bridge to appreciate the problems and possible solutions that may arise in this research context.

1.4 Organization of the Thesis

This thesis is organized into six chapters:

Chapter 1 gives a brief overview of Grid-related concepts and technologies, takes a quick glance at CodeNet project, presents the statement and the organization of this thesis.

Chapter 2 introduces the background knowledge of the code resource sharing. In this chapter, we discuss the general concepts and technologies from two research areas: Peer-to-Peer and Software Reuse.

Chapter 3 discusses the related work on resource sharing mechanism. In this chapter, we describe four resource sharing models: Jini, Globus, Napster and Gnutella. We review the mechanism of each model and compare these four models at the end of this chapter in summary.

Chapter 4 defines the goals of new resource sharing model in Grid environment, and presents the detailed design and implementation of the prototype system that fulfills the model.

Chapter 5 discusses the steps required to deploy and use the prototype system.

Chapter 6 evaluates the features of the prototype system.

Chapter 7 draws conclusions from this thesis and suggests for future research directions.

2. BACKGROUND

In this chapter, we introduce the background concepts and technologies related with our proposed model. The chapter is organized into three sections. Section 2.1 discusses the background knowledge on Peer-to-Peer. Section 2.2 describes the background knowledge on Software Reuse. Section 2.3 summarizes this chapter.

2.1 Overview of Peer-to-Peer

2.1.1 Peer-to-Peer Concept

Peer-to-Peer (P2P) is a very controversial topic. There are several definitions of P2P that are being used. The Intel P2P working group defines it as "the sharing of computer resource and services by direct exchange between systems" [P2pwg01]. The Hewlett-Pachard corporate laboratory defines P2P as "a class of systems and applications that employ distributed resource to perform a critical function in a decentralized manner" [Milo02]. Clay Shirky defines P2P as "... a class of applications that takes advantage of resource – storages, cycles, content, human presence – available at the edges of the Internet. Because accessing these decentralized resource means operating in an environment of unstable connectivity and unpredictable IP address, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers" [Shirky01]. In a simple view, P2P is about sharing: giving to and obtaining from the peer community. A peer gives out some resource and obtains other resources in return.

The concept of P2P is not new. It is based on technology and techniques that have existed since the early days of Internet. But in recent years, P2P systems have grown dramatically and received much attention as a means of sharing and distributing information in distributed system. Conceptually, P2P is an alternative to the centralized and client-server models of computing, where there is typically a single computer or a small cluster of servers and many clients. In its purest form, the P2P system has no concept of server, and all participants are peers. A simple taxonomy of computer systems from the P2P perspective is presented in Figure 2-1. All computer systems can be classified into centralized systems and distributed systems. Distributed systems can be further classified into the client-server model and the P2P model. The P2P model can either be pure or be hybrid. In the hybrid P2P model, a server-like peer is approached first to obtain meta-information, such as the identity of the peer on which some resource is stored, after that, the P2P communication is performed. Examples of the hybrid model include Napster [OpenNap], Groove[Groove]. In the pure P2P model, there does not exist such a centralized peer, the communication between peers happens directly. Examples of the pure model include Gnutella [Gnutella] and Freenet [Freenet]. In this thesis, aspects of both of these two P2P models are adopted in our prototype system.

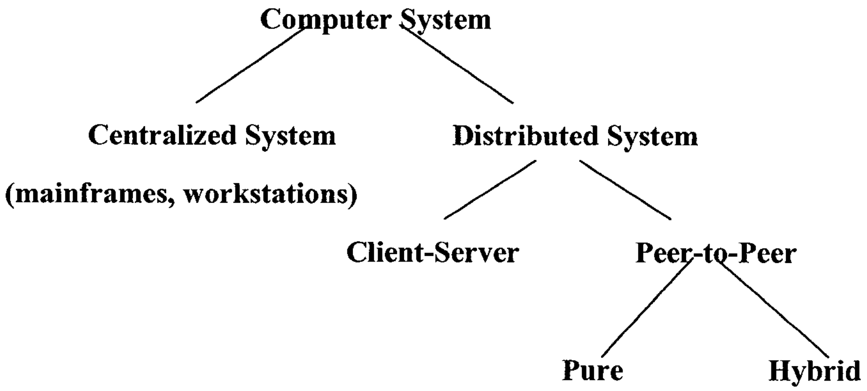


Figure 2-1: A Simple Taxonomy of Computer Systems

2.1.2 Peer-to-Peer Architecture

Figure 2-2 illustrates an abstract P2P system architecture. Theoretically speaking, a complete P2P system is constructed by 5 functional layers: Communication layer, Group management layer, Robustness layer, Class-specific layer and Application-specific layer. Every layer contains one or several components. The P2P architecture is described in [P2pwg01] (page 9-10).

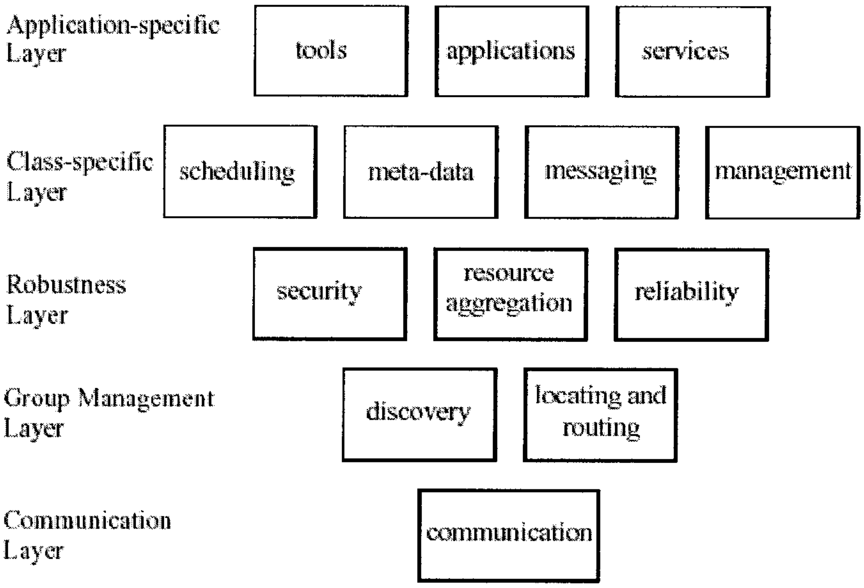


Figure 2-2: P2P System Architecture

Communication: The P2P model covers a wide spectrum of communication paradigms ranging from stable, high-speed links over the Internet to links through wireless medium. The fundamental challenge of communication layer is to overcome the problems related to the dynamic nature of P2P system.

Group Management: Peer group management includes two components: discovery of other peers in the community and location & routing between those peers. Discovery of peers can adopt highly centralized mechanism such as in Napster [OpenNap], highly distributed mechanism such as in Gnutella [Gnutella], or somewhere in between. Location and routing algorithms optimize the path of the messages traveling from one peer to another.

Robustness: Robustness layer contains three main components: security, resource aggregation and reliability, which are essential to maintaining robust P2P system. Security is one of the biggest challenges for P2P systems. One premise of P2P communication is that all the peers in peer community trust each other, which indeed poses a number of potential risks to the system. The P2P model provides the basis for aggregating resource available on P2P community. Classifying the architecture of a P2P resource aggregation component is very difficult because of the wide variety of resources that may be aggregated across peers. Reliability in P2P system is a challenging problem. It is difficult to guarantee reliable behavior because of the inherently distributed nature of peer networks makes. The most common solution to reliability across P2P system is to take advantage of scale and redundancy.

Class-Specific: In class-specific layer, scheduling applies to computer-intensive P2P applications since computer-intensive tasks need to be broken into pieces, which must be scheduled across the peer community. Metadata applies to content and file sharing applications because it describes the content stored across nodes of the peer community and may be consulted to determine the location and availability of desired information. Messaging is used by collaborative applications because messages sent between peers enable communication. Management is used to support managing the underlying P2P infrastructure.

Application-Specific: In this layer, tools, applications, and services implement application-specific functionality, which correspond to specific P2P applications running on top of the P2P infrastructure.

For a P2P system, not all the layers and components are necessary. In this thesis, our code resource sharing system is composed of components of communication, discovery, locating and routing, resource aggregation and messaging.

2.1.3 Peer-to-Peer Discovery Approaches

In P2P system, there are three common P2P approaches for source sharing and discovery in peer community.

Centralized directory approach. The typical and famous example of this approach is Napster [OpenNap]. In this approach, the peers of the community communicate with a central directory where they publish information about the resource they offer for sharing (shown in Figure 2-3). Upon receiving request from a peer, the central index server will match the request with the best peer in its directory. The best peer could be the one that is fastest, cheapest, or the most available. Then the data exchange will occur directly between the two peers. This model requires some managed infrastructure (the index server), which hosts information about all participants in the peer community. This may cause the system to develop some scalability limits, because it requires bigger server when the number of peers increases, and larger storage when the number of users and resources increases.

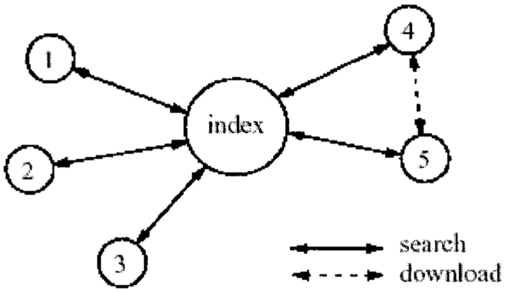


Figure 2-3: Centralized Directory Approach

Flooded requests approach. The flooding approach is a pure P2P model in which no advertisement of shared resource occurs. It is different from the central index approach. Each request from a peer is flooded (broadcast) to directly connected peers, which themselves flood their peers and so on, until the request is answered, as shown in Figure 2-4. The typical and famous example of this approach is Gnutella [Gnutella]. The flooded requests approach requires a lot of network bandwidth and, hence, does not prove to be very scalable; but it is efficient in limited communities such as a company network.

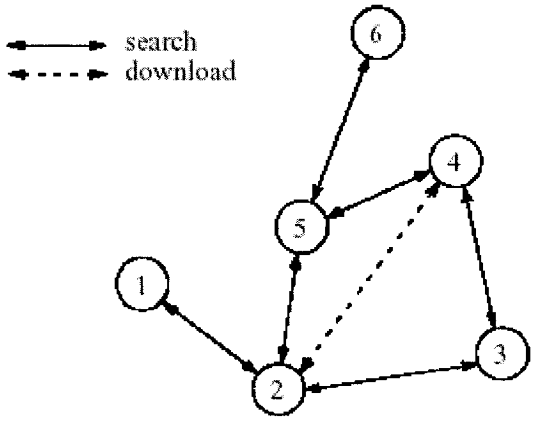


Figure 2-4: Flooded Requests Approach

Document routing approach. The typical and famous example of this approach is FreeNet [Freenet]. In this approach, each peer of the community is assigned a random peer ID and each peer also knows a given number of peers (see Figure 2-5). When a file is published (shared) on such a system, an ID based on a hash of the file's contents and its name is assigned to the file. Each peer will then route the file towards the peer with the peer ID that is most similar to the file ID. This process is repeated until the nearest peer ID is the current peer's ID. When a peer requests the file from the P2P community, the request will be sent to the peer with the ID most similar to the file ID until the file is found. The document routing model is very efficient for large, global community. However, it has the limitation that the document IDs must be known before requesting for a given document. Hence, it is more difficult to implement a search than in the flooded requests model. Also, network partitioning can lead to an islanding problem, where the community is split into independent sub-communities that don't have links to each other.

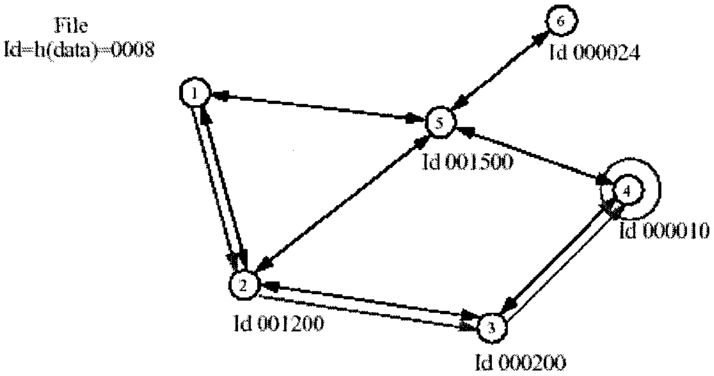


Figure 2-5: Document Routing Approach

In this thesis, a variation of the flooded requests approach is adopted in our prototype system for code resource searching.

2.1.4 Peer-to-Peer Characteristics

Compared with the traditional centralized and client/server systems, P2P approaches have many characteristics benefits, as follows:

(1) Cost sharing/reduction. Centralized systems or Client-Server systems that serve many clients typically bear the majority of the financial cost of the system. When that main cost becomes too large, a P2P architecture can help spread the cost over all the peers. For example, in the file-sharing space, the Napster [OpenNap] system enabled the cost sharing of file storage, and was able to maintain the index required for sharing. In the end, the legal costs of maintaining the index became too large, so more radical P2P systems such as Gnutella [Gnutella01] were able to share the costs even further. Much of the cost sharing is realized by the utilization and aggregation of otherwise unused resource, which results in both net marginal cost reductions and a lower cost for the most costly system components.

(2) Improved scalability. Scalability is limited by factors such as the amount of centralized operations that need to be performed. P2P system achieves improved

system scalability by avoiding dependency on centralized points. Early P2P systems such as Napster [OpenNap] attacked the scalability problem by having the peer directly download data files from the peers that possess the requested document. As a result, Napster was able to scale up to over 6 million users at the peak of its service. In the future, as the bandwidth and computation power continue to grow, P2P system will enable more automated scaling.

(3) Resource aggregation: A decentralized approach lends itself naturally to aggregation of resources. Each node in the P2P system brings with it certain resources such as compute power or storage space. Applications that benefit from huge amounts of these resources, such as compute-intensive simulations or distributed file systems, naturally lean toward a P2P structure to aggregate these resources to solve the larger problem.

(4) Increased autonomy. In many cases, users of a distributed system are unwilling to rely on any centralized service provider. Instead, they prefer that all data and work on their behalf be performed locally. P2P systems support a high level of autonomy simply because they require that the local node do work on behalf of its user. The principle example of this is the various file sharing systems such as Napster[OpenNap], Gnutella[Gnutella], and FreeNet [FreeNet]. In each case, users are able to share file resource while maintaining autonomy and being a final authority over their local system.

(5) Dynamism. In P2P systems, the computing environment is highly dynamic. That is, resources, such as compute nodes, will be entering and leaving the system continuously, either intentionally (e.g., because a user turns off his computer) or unintentionally (e.g., due to a network link failing). Peer groups frequently change.

(6) Performance/Reliability. P2P systems aim to improve performance and reliability by aggregating distributed storage capacity (e.g., Napster, Gnutella) and computing

cycles (e.g. SETI@Home) of devices spread across a network. By aggregating resources through low-cost interoperability, the whole is made greater than the sum of its parts. Because of the dynamism and autonomy of peers, however, issues of scale and redundancy become much more important in achieving performance and reliability than in traditional centralized or distributed systems.

Because of these characteristics, we import P2P technology into the CodeNet project to overcome the shortcomings that exist in previous code resource sharing applications.

2.1.5 Peer-to-Peer Applications

Until now there already existed many types and variants of applications that employ P2P technology, those applications can be classified into three categories: distributed computing, content sharing and collaboration [Barkai01]:

Distributed computing P2P applications split a large task into smaller sub-pieces that can execute in parallel over a number of independent peer nodes. One example implementation is SETI@Home [Ander02] (searching for extraterrestrial life project). Most implementations of this type have focused on compute-intensive applications.

Content sharing P2P applications focus on storing information on and retrieving information from various peers in the network; an example application is Napster [OpenNap]. This type of application allows peers to search for and download files that other peers have made available. For the most part, current implementations have not focused much on providing reliability and rely on the user to make intelligent choices about the location from which to fetch files and to retry when downloads fail. They focus on using otherwise unused storage space as a server for users. These applications must ensure reliability by using more traditional database techniques such as replication. Many research projects have explored the foundations of P2P file system [Ratnasamy01] [Bolosky00] [Kubiatowicz00] [Gribble01]. Filtering and mining applications, such as OpenCOLA [Opencola] and JXTA search [Waterhouse02], are beginning to emerge. Instead of focusing on information sharing, these applications focus on collaborative filtering techniques that build searchable indices over a peer network.

Collaborative P2P applications allow users to collaborate, in real time, without relying on a central server to collect and relay information. Example applications include Yahoo!, AOL, and Jabber instant messaging [Strom01], which enable computer users to send messages to each other through a messaging server. Similarly, shared applications allow people to interact while viewing and editing the same information simultaneously; for instance, P2P games are hosted on all peer computers and updates are distributed to all peers without requiring a central server.

The prototype system designed and implemented in this thesis is in the category of content sharing P2P application.

From the description of its characteristics and applications we can see P2P is a more general resource sharing modality that has much in common with Grid technologies. But, in practice, the technical focus of work in these domains has not overlapped much until recently, one reason is that P2P developers have so far focused entirely on vertically integrated solutions rather than seeking to define common protocols that would allow for shared infrastructure and interoperability. As the P2P applications become more sophisticated and the need for interoperability becomes clearer there will be a strong convergence of interests between P2P and Grid computing. In this thesis, we research this aspect by importing P2P technology into Grid code resource sharing to take advantage of P2P benefits.

2.1.6 JXTA Project

The JXTA Project [JXTA] [Wilson02} was started as a research project by Sun Microsystems, Inc. on April 25,2001. It is intended to be a platform on which to develop a wide range of distributed P2P applications. Technically, JXTA technology is a set of open, generalized peer-to-peer protocols that allow any connected device (cell phone, to PDA, PC to server) on the network to communicate and collaborate. It is not an actual piece of application software.

The goal of JXTA is to provide a general-purpose network programming and computing platform. The objectives of JXTA include:

- Interoperability: by enabling inter-connected peers to easily locate each other, JXTA-based system nodes participate in community-based activities and offer services to each other seamlessly across different P2P systems and different communities.
- Platform independence: JXTA is designed to be independent from programming languages, system platforms, and network platforms.
- Ubiquity: JXTA is designed to be implementable on every device with a digital heartbeat, including applications, desktop computers, and storage systems.

JXTA project approaches the P2P space from the lower level by providing a set of protocols based on XML messages, as shown in Figure 2-6.

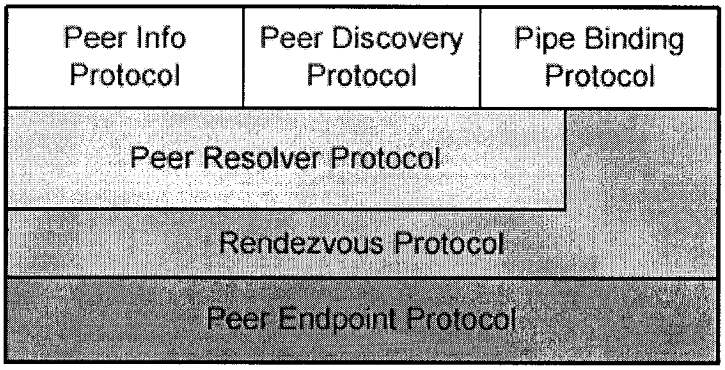


Figure 2-6: The JXTA Protocol Suite

Peer Endpoint protocol provides a set of messages used to enable message routing from a source peer to a destination peer. Rendezvous protocol handles the details of propagating messages between peers. Peer Resolver protocol allows peers to send and process generic requests. Peer Information protocol provides peers with a way to obtain status information from other peers on the network. Peer Discovery protocol enables peers to discover peer services on the network. Pipe Binding protocol provides a mechanism to bind a virtual communication channel to a peer endpoint.

The JXTA project has a layered architecture, which consists of three building layers: a core layer, a middle services layer, and an application layer. The core layer includes protocols and building blocks to enable key mechanisms for peer to peer networking, including discovery, transport (including firewall handling and limited security), and the creation of peers and peer groups. The services layer provides hooks for supporting generic services (such as searching, sharing and added security) that are used in many P2P applications. And finally, the application layer supports implementation of integrated applications, such as distributed search & indexing, shared resources (CPU, storage, etc.), instant messaging, collaborative work. The entire system is designed to be modular, and to allow developers to pick and choose a collection of these services and applications that suits their needs.

JXTA provides an abstract language for peer communication, enabling a wider variety of services, devices, and network transports to be used in P2P networks. It takes the complexity out of the network and operating environments so developers can quickly build highly distributed applications. However, JXTA might not be suited to specific standalone P2P applications. In an individual application, the network overhead of XML messaging might be more trouble than it is worth, especially if the application developer has no intention of taking advantage of JXTA's capabilities to incorporate other P2P services into the application.

2.2 Overview of Software Reuse

Software Reuse is a very important concept in software engineering. It is defined as "the process of creating software systems from existing software rather than building software systems from scratch" [Krueger92]. "The idea of software reuse is as old as software engineering itself " [Grinter01], as a means for overcoming the software crisis--the problem of building large, reliable software systems in a controlled, cost-effective way, the concept of building shared libraries of software components that could be used among different programs was proposed by Doug Mcllroy at the first NATO software engineering conference [Jacobson 97].

The motive of software reuse is simple: by reusing components developers would not spend time writing code that already existed, and the existing code would already have been tested for accuracy and completeness. Today software reuse is "an approach to developing systems where artifacts that already exist are used again" [Tracz95]. The types of artifacts that could be reused have evolved from the initial software reuse proposal. In addition to generic source code, today's software reuse programs reuse requirements, analysis models, design structure, and specification and document. In this thesis work, we mainly focused on reusable generic source code retrieval mechanism.

"Software reuse has been recognized as a realistic and promising way to improve both the productivity and the quality of new software projects" [Frakes94a]. It improves software productivity and shortens software developing time by avoiding creating existed software components, increases software reliability and lowers maintenance cost by adopting tested software components.

Software reuse has four steps: abstraction, storage, retrieval and recontextualization of the components to be reused [Grinter01]. Abstraction focuses on designing a reusable artifact, abstracting components from tested and documented software. Storage means storing components into repositories, usually a rational database. Retrieval is making those components available to those who need to use. Recontextualization is making components understandable to those who will incorporate it into their new systems, to ensure components are technically compatible. However, this thesis focuses mainly on retrieval mechanism for software reuse.

There exists great diversity in the software reuse technologies, for example: software reuse libraries, application generators, source code compilers, and generic software templates. Among those technologies, Software Reuse Libraries (SRLs) has captured much attention [Frakes94b] [Basili97] [Atsumi02]. SRLs are based on composition technologies, in which the components are self-contained entities such as data structures, programs, objects, and the like, reusing these components is accomplished by developer finding, understanding and adapting the components into a new application. In definition, SRLs are a set of repositories that use specialized methods for reusable software components classification, storage, retrieval, and delivering.

In [Zhong00], the process of the reuse-oriented software development with software reuse library is presented as Figure 2-6. It involves development-for-reuse in which reusable components are abstracted and brought together to store into a software reuse library; and development-with-reuse in which reusable components are retrieved from the library on the basis of specific requests and reused in the construction of a new software system. More specifically, the whole reuse-oriented software development process with SRL contains three distinct phases: the construction/storage phase, the location/retrieval phase and the adaptation/generation phase. The first phase focuses on the construction (either from the scratch or from existing software) and storage (usually in a rational DBMS) of reusable software components. The second phase deals with applying queries to find the desired components and retrieve those components from SRL. The last phase focuses on adaptation and transformation of the retrieved components to be integrated into the new software system.

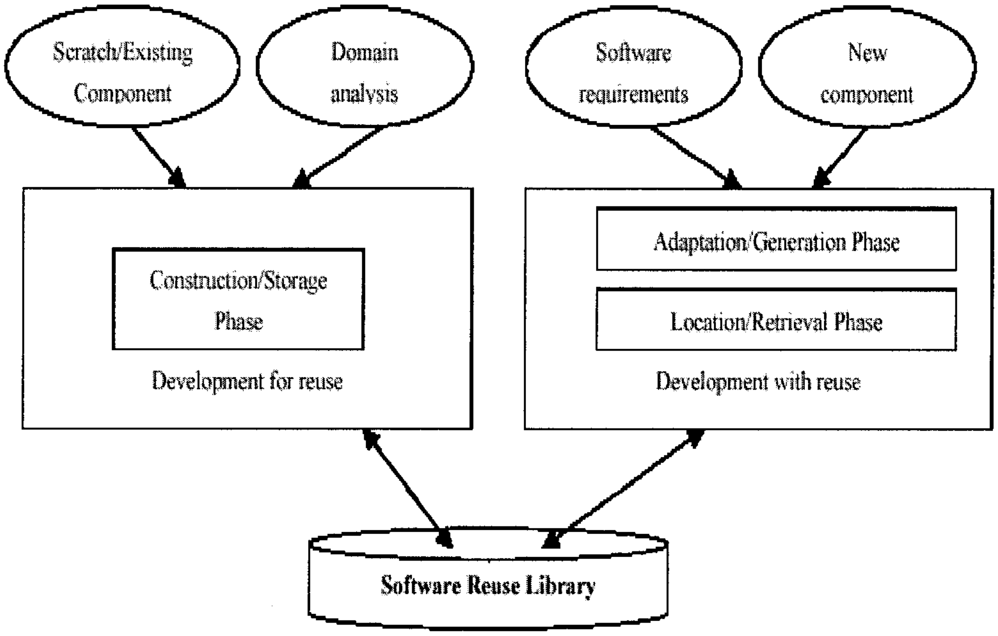


Figure 2-7: Reuse-oriented software development process with Software Reuse

In [Zhong00] and [Zhang00], the prototype system design adopted a centralized SRL server infrastructure, which make the prototype DORLM (Distributed Object-based Software Reuse Library Module) face an obvious dilemma [Henninger97]: in order for the approach to be useful, the repository must contain enough components to support developers, but when many components are available, it becomes troublesome for finding and choosing appropriate reusable software components, which is the bottleneck of the prototype system. In this thesis, we suggest a solution to this dilemma by replacing the centralized SRL structure with a decentralized model in which a large number of distributed software component repositories exist.

For current software reuse research, a bottleneck is the classification schema and retrieval method of components [Damiani96]. Particularly, when large repositories of components are available, classification and retrieval for reuse should be flexible to allow the selection of components adaptable with a limited effort. In this thesis, we suggest a new model that mainly focuses on software reuse retrieval mechanism.

2.3 Chapter Summary

In this chapter, we introduce the basic background knowledge related to our proposed model. Some key concepts and technologies from both Peer-to-Peer computing and soft reuse are discussed in this part. Conceptually, P2P is an alternative to the centralized and client-server models of computing, it has many advantages over client-server models for resource sharing. Software reuse is a realistic and promising way to improve both the productivity and the quality of new software projects by creating software systems from existing software components, rather than building software systems from scratch. Software reuse is a foundational characteristic of the CodeNet project.

3. RELATED WORK

The rising popularity of network-based applications among end users has stimulated research interests in resource sharing in large-scale networks, which led to the creation of a lot of resource sharing models for distributed environment. Among those models, the most successful ones are: Jini, Globus, Napster and Gnutella. In this chapter we will discuss these 4 models. By studying the mechanisms underlying these models, we can learn some important principles that can be adapted and used in our model design and development.

3.1 Jini

3.1.1 Overview

Jini [Jini] is an infrastructure developed by Sun Microsystems for building scalable, robust, distributed systems and parallel applications using Java [Baker01]. It provides a set of application programming interfaces (APIs), runtime conventions and network protocols that can facilitate building and deploying of distributed computing applications. Jini enables dynamic deployment and configuration of distributed resource, provides simple mechanism to enable components to plug together to form a resource community with minimal planning, installation and human intervention. Resource components in Jini may be physical devices or software objects that provide services over published interface. In Jini terminology such devices or software objects are called services, Jini community is the collection of services and clients communicating with the Jini protocols.

Jini technology assumes a changing network environment, in terms of both the components that make up the network and the way these components interact. Jini-based software gives Jini services self-configuration and self-management capabilities, which enable services and clients to communicate immediately on a network without human intervention. The Jini community is also self-healing in that services that leave community for intentional reasons or unintentional reasons do not affect the remaining services' operation. A Jini client that loses contact with a server can recover and continue processing.

Jini allows any devices with a processor, some memory and a network connection to offer services to other components on the Jini network. This class of devices includes not only all the things we traditionally think of as computers, but also most of the things we think of as peripherals, such as storage devices, printers and specialized scientific instruments. Increasingly, the definition will also encompass a host of other devices, such as cell phones, personal digital assistants (PDA) and microprocessor controlled devices [Waldo99].

Jini architecture is defined in terms of the Java programming language, so the type of system used for service descriptions is universal. That means in Jini networking, any Java Virtual Machine (JVM) can provide a single execution environment, no matter which platform hosts the virtual machines [Arnold99]. Jini communication is built on top of object serialization and Remote Method Invocation (RMI) [Wollrath97], the Java equivalent of a remote procedure call, which enables clients to handle the desired remote objects. Jini can also use RMI to pass objects as arguments and to return values, making it easy to move objects as well as data across the network between JVMs since Jini programming approach is object-oriented programming.

3.1.2 Jini Resource Sharing Mechanism

The basic Jini architecture is shown in the following Figure 3-1. As illustrated in the figure, the workflow of Jini resource sharing mechanism is composed of 6 steps:

1. Discover:

First of all, a network service (hardware or software) that wants to join a Jini community sends out a packet, which is multicast over the LAN for available Lookup Service (LUS). The packet contains the necessary information for the LUS to respond to the requester. If the LUS gets the request, it will immediately give a response to the requester providing necessary information for network service to locate the LUS.

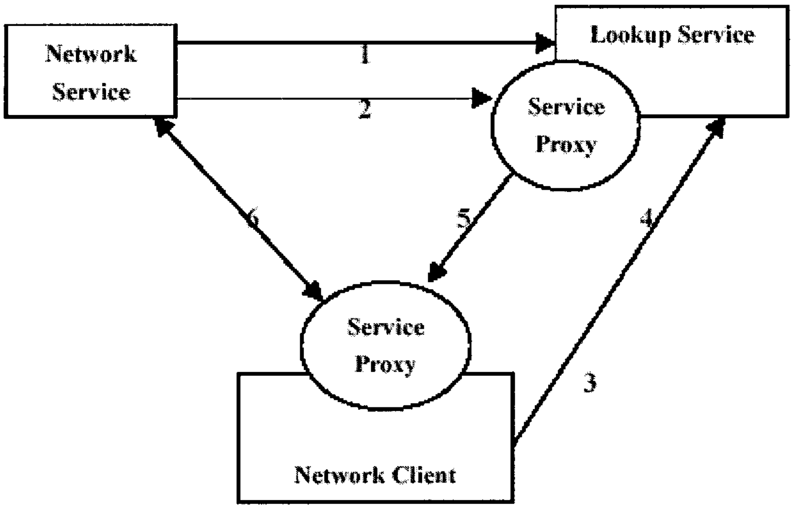


Figure 3-1: Jini Resource Sharing Architecture and Workflow

2. Join:

Upon receiving a response from the LUS, a network service can register to join the Jini community by exporting a proxy object of this service into the LUS.

3. Discover:

If network clients want to get access to services in Jini community, first they must look for the LUS in the same way as network services do-- multicast request over the LAN asking for an available LUS.

4. Lookup:

After the network client gets response from the LUS, it will send to the LUS a message requesting the required service. Such a request takes the form of asking for an object implementing a particular Java language type. For example, a client could ask for something—either hardware or software—that implements a Java printing interface, rather than asking for something called a printer.

5. Response:

The LUS responds to the network client by sending back a copy of the required registered service's proxy object.

6. Use:

The network client interacts with network service via service proxy object.

From the above workflow description, we can find three main components of the Jini technology that enable Jini services and clients to spontaneously communicate with limited need for administration. Those components are:

- Lookup Service – It is essentially a meta-service or naming service that keeps track of all existing Jini services on the network. It is similar in function to the RMI registry, the CORBA naming service etc. It accepts registrations from services, each of which is called a service item, composed of service's proxy object, service ID, and a set of attributes associated with the proxy. A lookup service guarantees the uniqueness of the service among multiple lookup services with an assignment of a service ID, which is a random 128 bit number generated from time at instigation together with the network host address. The lookup service supports a template search based on any combination of the three criteria: service ID, type (Java interfaces or classes) that a service supports, and associated attributes. To search for a service, the client fills out a template using only the fields it is interested in.

- Discovery Protocol– In order for services to be able to register themselves with a lookup service, they initially send multicast messages out on the network searching for one LUS. To discover available services, clients must also do the same. This process is known as discovery. It has the advantage that neither services nor clients need to be aware of the location of a lookup service in advance. Most Jini systems use multicast protocol for discovery, but some Jini applications also use unicast protocol, which requires that the Jini services or clients must know the IP address and the port number to locate and communicate with an available LUS. Comparably, multicast is more suitable for spontaneous networking.

- Proxy Object – Jini Clients use services through proxy objects that they download from the lookup service server. These proxy objects provide the code for interface that needed to invoke a particular remote service. The proxy objects are what the Jini services register with the lookup service.

With Jini network services and clients, those core components work together to form a dynamic Jini resource sharing community.

3.1.3 Advantages and Disadvantages

The biggest advantage of Jini is dynamism. Jini technology makes a network more dynamic by allowing the plug-and--play of resource as services. Because of this feature of dynamic management, Jini technology is widely used in distributed computing and cluster computing. For example, J. Batheja used Jini and JavaSpace to support parallel/distributed computing over a networked cluster [Bathja01]. Another important advantage of Jini technology is autonomy. Because a Jini network is spontaneous and self-managing, any resource can join Jini community as services, thus resulting flexible aggregation of resource without much human administration and intervention.

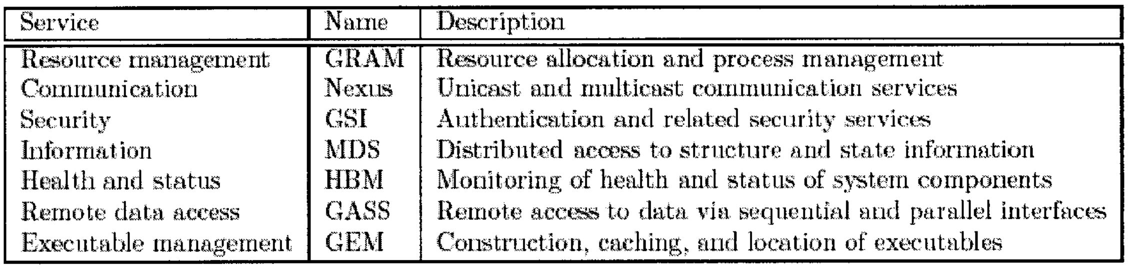
However, since the Jini lookup service is intended to be used only in a LAN setting that can support multicast protocol, by itself it is not appropriate for managing a large number of Jini services spread over a Grid environment. Though there is a strong interest for integrating Jini and Grid technology [Baker01] [Suzumura01] [Furmento02], the current researches take advantage of Jini technology only within Grid domain scope. Jini application is the LAN-based system. Jini infrastructure can't support the whole Grid networks that contain multiple Grid domains. Besides, in Jini community, the LUS server is the key central component. If the LUS malfunctions, neither the new services can register to join the Jini community nor Jini clients can get suitable proxy object to communicate with Jini services, thus the whole system will fail over. So the Jini infrastructure has too much dependency on central LUS server, Jini model is not completely decentralized for resource sharing mechanism.

3.2 Globus

3.2.1 Overview

The Globus project [Globus] is a multi-institutional research effort that seeks to enable the construction of computational grids that integrate geographically distributed computational and information resource [Foster02d]. It provides a software infrastructure for building applications to handle large-scale distributed, heterogeneous computing resource as a single virtual machine. This project is led by Ian Foster and Carl Kesselman based primarily at Argonne National Laboratory and the University of Southern California since 1997 [Foster97].

The central element of the Globus project is the Globus Metacomputing Toolkit (GMT), which defines a set of low-level tools, protocols and basic services required to construct a computational Grid. Here we list the basic services in Table 3-1; those core Globus services have become a de facto standard for building basic Grid computing architecture introduced in Chapter 1.



Service	Name	Description
Resource management	GRAM	Resource allocation and process management
Communication	Nexus	Unicast and multicast communication services
Security	GSI	Authentication and related security services
Information	MDS	Distributed access to structure and state information
Health and status	HBM	Monitoring of health and status of system components
Remote data access	GASS	Remote access to data via sequential and parallel interfaces
Executable management	GEM	Construction, caching, and location of executables

Table 3-1: Core Globus Services [Foster98]

Those basic services provide solutions for particular problems, for example, Nexus defines a low-level communication API to support a set of application-level communication protocols; GSI (Globus Security Infrastructure) focuses on security issues such as authentication & authorization. The Globus services have well-defined interfaces so they can be incorporated into Grid applications in an incremental way.

The layered architecture of the Globus Metacomputing Toolkit is analogous to an hourglass. At the neck of the hourglass there is a small set of core abstractions and services from which many different high-level behaviors can be mapped onto many different underlying technologies. These abstractions and services provide uniform access to diverse implementations of local services, and building blocks upon which global services can be built. A local domain need only provide these local services. Global services can be built without knowledge of local implementation.

3.2.2 Globus Resource Sharing Mechanism

Among all Globus core services, the Globus Resource Allocation Manager (GRAM) and the Globus Metacomputing Directory Service (MDS) are used for resource management and sharing.

The Globus Resource Allocation Manager (GRAM) is at the bottom of the Globus layered architecture. It provides the local component for resource management [Czajk97]. Each GRAM manages a set of resource that are controlled under the same site-specific allocation policy, which is often performed by a local resource management system. For example, a single GRAM could provide access to the nodes of a parallel computer, a cluster of workstations, or a set of machines operating within a Condor pool. Thus, many GRAMs, each responsible for a particular “local" set of resource, can be grouped to build a computational Grid.

In Grid built with Globus, computational tools and applications can express resource allocation and process management requests in terms of a standard API since GRAM provides a standard network-enabled interface to local resource management systems. Until now GRAM can work cooperatively with six different local resource management tools: Network Queuing Environment (NQE), EASY-LL, LSF, LoadLeveler, Condor, and a simple "fork" daemon. Within the GRAM API, resource requests are expressed in terms of an extensible resource specification language, which plays an important role in the definition of global services requests.

GRAM services also provide building components based on which a range of global resource management strategies can be constructed. A general resource management architecture is defined by Globus project team, as illustrated in Figure 3-2. In this architecture, a kind of standard called resource specification language (RSL) is used as a common notation for expressing resource requirements. Resource requests are expressed by a grid-enabled application in form of a high-level RSL expression. The domain-specific resource discovery and selection policies are implemented by resource brokers, which transform abstract RSL expressions into progressively more specific domain-specific resource requirements until a specific set of resource is identified. After this process, a so-called ground RSL expression, in which a specific set of GRAMs are identified, is produced.

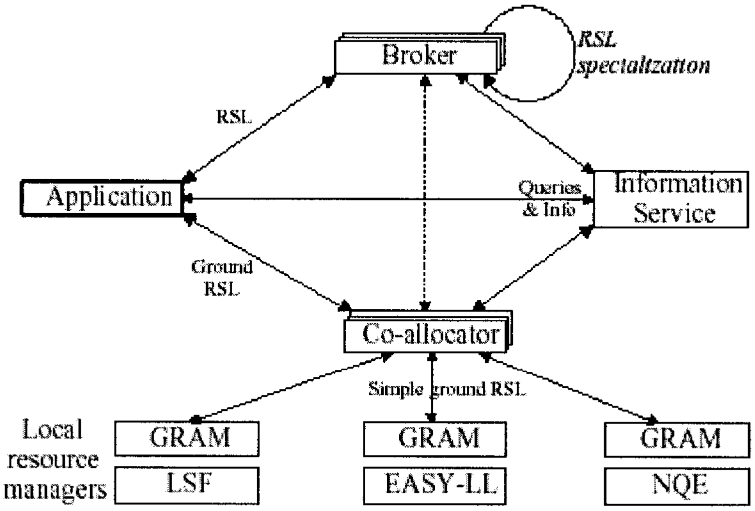


Figure 3-2: The Globus Resource Management Architecture [Foster98]

The final step in the resource allocation process is to decompose the RSL into a set of separate resource allocation requests, which are dispatched to the suitable GRAMs. In high-performance computations, it is important for system to co-allocate network resource to ensure that a given set of resource is available for use simultaneously. Within Globus, a resource co-allocator provides this service by breaking the RSL into pieces, distributing it to different GRAMs, and coordinating the return values. Different co-allocators can be constructed to implement different approaches to the problems of allocating and managing ensembles of resource.

In Grid built with Globus, the resource and computation management are implemented in a hierarchical fashion. An individual GRAM supports the creation and management of a set of processes, or Globus job, on a set of local resource. A computation created by a global service may then consist of one or more jobs, each created by a request to a GRAM and managed via management functions implemented by that GRAM. This resource management mechanism makes it very simple that local services can be used to support a rich set of global functionality.

The Globus Metacomputing Directory Service (MDS) [Fitzg97] provides information service for Grid resource sharing (as illustrated in Figure 3-2). The dynamic nature of Grid environments requires that toolkit components, programming tools, and applications must be able to adapt their behavior in response to changes in system structure and state. In order to support collaboration in this kind of dynamic environment, MDS is designed to provide an information-rich environment in which dynamic information about system components is available. MDS stores and makes accessible information such as the architecture type, operating system version and amount of memory on a computer, network bandwidth and latency, available communication protocols that describe the availability status of the Grid resource.

MDS consists of a set of tools and APIs for applications to discover, publish, and access information about the structure and state of a computational Grid. It keeps information in a standard data representation, which is defined by the Lightweight Directory Access Protocol (LDAP) [Howes95], in a LDAP server. LDAP defines a hierarchical, trees-structured name space called a directory information tree and is designed as a distributed service: arbitrary subtrees can be associated with distinct servers. Thus, the local service required to support MDS is exactly an LDAP server (or a gateway to another LDAP server, if multiple Grid domains share a server), plus the utilities used to populate this server with up-to-date information about the structure and state of the resource within that domain. The global MDS service is simply the ensemble of all these servers.

Globus project is still in development. From the GMT 1.0 of 1998 to the GMT 2.0 release in 2002, the Globus Toolkit has evolved rapidly into a "the de facto standard" for Grid computing. For resource sharing and management services, MDS-2 supersedes MDS-1, which pioneered the use of Grid information service concepts but did not address all requirements. The Globus MDS-2 architecture [Czajk01] provides a configurable information provider component called a Grid Resource Information Service (GRIS) and a configurable directory component called a Grid Index Information Service (GIIS) that aggregates information from multiple GRISs. Currently Globus team releases the latest Globus Toolkit 3.0 version (GT3), which is based on the new Open Grid Service Architecture (OGSA).

3.2.3 Advantages and Disadvantages

In Globus infrastructure, global resource can be well organized, published, discovered with the help of GRAM and MDS services. The design principle of Globus is to support system collaboration, resource sharing in heterogenous multi-institution environments, resulting in a lot of technical advantages and usage benefits including heterogeneity transparency, good scalability and collaboration ability.

However, the roles of resource brokers and LDAP servers make the resource discovery components centralized. In every Grid domain, Grid applications or users have to send requests to those centralized brokers or LDAP servers to transform high-level RSL or to retrieve directory information for further resource access. If those components are offline or at fault, the performance of the Grid domain will be decreased greatly, thus affect the efficiency of the whole Grid. Besides, in practice, it is harder and more complex to implement the Globus resource sharing infrastructure than other resource sharing models.

3.3 Napster

Napster [Napster] was developed for sharing MP3 files over the Internet among a huge set of users. It was among the first P2P content sharing P2P applications to gain widespread recognition and use. Napster achieved great support due to its ease of use, its accuracy, and of course the large demand that had developed for MP3 music files.

The Napster solution enables MP3 file sharing under the control of a centralized directory server that maintains basic addressability and availability information about the user nodes and the meta-information about the shared files. The centralization allows a quick search for the requested file and assists in identifying the most suitable location to download the files. The actual file transfer still happens over a direct TCP connection between the requester and owner nodes. Napster Itself is a closed application, but there is an open protocol known as OpenNap [OpenNap] that is based on Napster. OpenNap extends the Napster protocol to allow sharing of any media type, and the ability to link server-like peers together.

Napster is a high-profile P2P network that gives its members the revolutionary ability to connect directly to other members' computers and search their hard drives for digital music files to share and trade. The operations of Napster are described in Figure 3-3. Napster community members download a software package from Napster and install it in their computers. The Napster central computer maintains directory of music files of members who are currently connected to the network. The directory is automatically updated when a member logs on or off the network. Whenever a member submits a request to search for a file, the central computer provides information to the requesting member who can then establish a connection directly with another member's computer possessing that particular file. The download of the target file takes place directly between the members' computers, bypassing the central computer. After a successful download, the central directory is updated with the client address where a new copy of the file is now located.

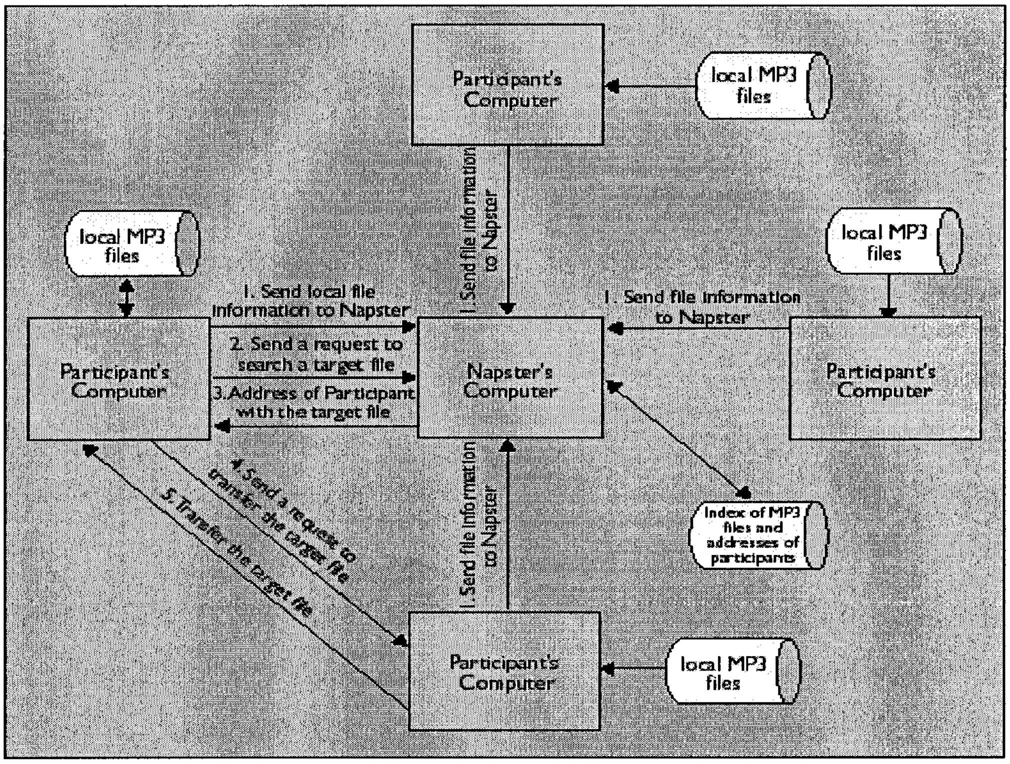

Participant's Computer
local MP3 files
1. Send file information to Napster
local MP3 files
local MP3 files
1. Send local file information to Napster
1. Send file information to Napster
2. Send a request to search a target file
Participant's Computer
Napster's Computer
Participant's Computer
3. Address of Participant with the target file
4. Send a request to transfer the target file
5. Transfer the target file
1. Send file information to Napster
Index of MP3 files and addresses of participants
Participant's Computer
local MP3 files

Figure 3-3: Napster Content Sharing Mechanism

The biggest advantage of Napster and similar applications is they allow the sharing of widely dispersed information stores without the need for a central file resource server, thus reduce the cost by spread file resource over all the community member computers. Community members can join and leave the Napster community freely without affecting the function of other components in the community, thus improving the dynamism.

However, Napster protocol also shows some limitations because of its dependency on the central directory server, which makes the sharing model not decentralized. If the central directory server malfunctions, the whole Napster system will be at fault. Besides, the capability of centralized directory server also constraint the size of the file resource sharing community, thus results in scalability limit.

3.4 Gnutella

Gnutella [Gnutella] is an open protocol that is designed to allow for the sharing of all file types, providing distributed discovery and sharing of resource across the networking. Comparing with Napster, Gnutella is distinguished by its support for autonomy and by its decentralized architecture. A Gnutella network consists of a dynamically changing set of peers connected using TCP/IP. Each peer acts as a client (an originator of queries) and a server (a provider of file information) and hence is called a servent.

A computer wishing to participate in a Gnutella network does so by contacting other Gnutella servents. The acquisition of those other servents' addresses is not actually part of the Gnutella protocol. Once connected, servent seeks out other servents by sending searching query. To search for a file the query is sent to all connected servents. If the servent has a matching file it responds with an answer. Gnutella is a pure P2P model in which no advertisement of shared resource occurs. Instead, each request from a servent is flooded (broadcast) to directly connected servents until the request is answered or a maximum timeout. Thus, the entire network form a dynamic, self-organizing network of independent entities. This virtual, application-level network has Gnutella servents at its nodes and open TCP connections as its links. The Gnutella protocol is illustrated in Figure 3-4 [Kant02].

Gnutella nodes provide client-side interfaces through which users can issue queries and view search results, accept queries from other servents, check for matches against their local data set, and respond with corresponding results. These nodes are also responsible for managing the background traffic that spreads the information used to maintain network integrity.

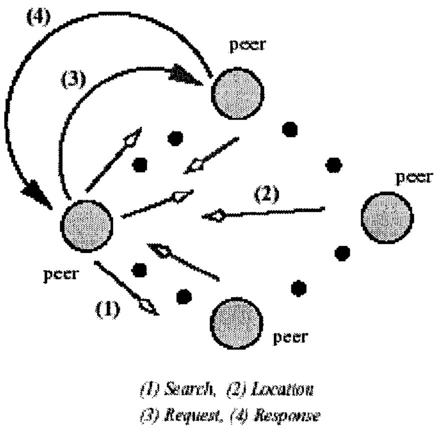


Figure 3-4: Gnutella Content Sharing Protocol

In order to join the system a new node/servent initially connects to one of several known hosts that are almost always available (e.g., gnutellahosts.com). Once attached to the network, nodes send messages to interact with each other. Messages can be broadcasted (i.e., sent to all nodes with which the sender has open TCP connections) or simply back-propagated (i.e., sent on a specific connection on the reverse of the path taken by an initial message). Several features of the protocol facilitate this broadcast/back-propagation mechanism. First, each message has a randomly generated identifier. Second, each node keeps a short memory of the recently routed messages, used to prevent re-broadcasting and to implement back-propagation. Third, messages are flagged with a time-to-live (TTL) parameter.

In Gnutella model, the messages allowed in the network are:

- Group Membership (PING and PONG) Messages. A node joining the network initiates a broadcasted PING message to announce its presence. When a node receives a PING message it forwards it to its neighbors and initiates a back-propagated PONG message. The PONG message contains information about the node such as its IP address.

- Search (QUERY and QUERY RESPONSE) Messages. QUERY messages contain a user specified search string that each receiving node matches against locally stored file names. QUERY messages are broadcasted. QUERY RESPONSES are back-propagated replies to QUERY messages and include information necessary to download a file.

- File Transfer (GET and PUSH) Messages. File downloads are done directly between two peers using GET/PUSH messages.

The advantages of Gnutella protocol are its decentralized structure and self-management ability. However, since Gnutella uses broadcast protocols for resource discovery, it requires a lot of network bandwidth, and hence does not prove to be very scalable. Anyway, Gnutella is very efficient in limited communities such as a company or a research institute network.

3.5 Chapter Summary

In this chapter, we discuss some existing resource sharing models: Jini, Globus, Napster and Gnutella. Those models have many reasonable features that can be used for designing a new improved mechanism. We list these qualities below in a feature, or attribute, matrix, Table 3-2, in order to more easily compare those different resource-sharing models with C/S model. As shown in Table 3-2, Jini is excellent for its self-management and dynamism; Globus does well in interoperability among heterogeneous networks with high scalability; Napster is famous for its popularity and cost sharing; Gnutella does quite well in establishing decentralized infrastructure and providing good autonomy.

Feature	Resource Sharing Model				C/S Model
	JINI	Globus	Napster	Gnutella	
Decentralization	Medium	Medium	Medium	High	Low
Scalability	Medium	High	Medium	Low	Medium
Interoperability	Medium	High	Low	Low	Medium
Dynamism	High	Medium	Medium	High	Medium
Cost of Ownership	Medium	High	Low	Low	High
Autonomy	High	Low	Medium	High	Medium
Targeted Environment	LAN	Grid	LAN&WAN	LAN	LAN&WAN

Table 3-2: Comparison of the Existing Models with C/S Model

4. PROTOTYPE DESIGN & IMPLEMENTATION

4.1 Motivation

Compared with other Grid computational resources, code resource has its own character. First, it has a wide range of distributed providers. Unlike computational resources or scientific facilities that are provided by a limited number of centralized expensive supercomputers or special devices in some Grid domains, code resources can be provided by a huge number of software developers from thousands of computers distributed in all the Grid domains. Second, code resource sharing is required by a large number of distributed users among geographically dispersed groups, such as support for distributive collaborative programming. Every software developer is a potential user who needs to search and retrieve proper reusable code components from code repositories. So, as the networks become more pervasive and large-scaled, code resources are increasingly prone to being more widely distributed and decentralized than any other computing resource.

As we stated in Chapter 1, however, previous systems such as CodeNet adopted a client/server infrastructure that does not support dispersed code repositories. With the rapid development of network technology, traditional code resource sharing mechanisms are not entirely suitable for large Grid network environments. We require a new sharing mechanism that should be decentralized, heterogeneous, scalable and dynamic enough to support both local and remote access to code resource with low cost and high autonomy.

From the description in Chapter 3, we considered some successful resource sharing mechanisms that are better than the traditional C/S model in some aspects. None of those models can meet all of the requirements completely of distributed code resource sharing. Jini does not fully support wide area networking resource sharing. Globus is not completely decentralized. Napster's performance depends on central directory server and Gnutella is not very scalable. Therefore, we need to design a new code resource sharing model for Grid environment to replace the conventional centralized C/S code resource sharing mechanism.

4.2 System Design Goals

Based on the characteristics of code resource and Grid networks, the new code resource sharing prototype system must be designed to achieve the following goals:

- The system must help the user to search and retrieve the suitable reused code components from all the available code repositories in the local domain.

- The system can send requests to code resource in a decentralized manner, that is, the system doesn't need to get access to a centralized code repository server or directory server, rather, multicast the requests to all the available local domain code repositories without knowing their locations in advance.

- The system can get access to code resource in both local domain and also other remote Grid domains. Thus, new users or resources can join the resource community from several possible domains, thus improving the scalability of the underlying systems that serve the community.

- The system can help users to control the scale of the code searching.

- The system must address interoperability issues. Users are able to get access to code repositories established on different platforms or database systems.

- The code resource within code repositories can join or leave from the community dynamically; this process will not affect other community resources.

- The system will adopt a loose-coupled architecture to support flexibility.

- The system can enable computers as nodes to plug together to form a code resource community with minimal planning, installation, administration and human intervention, thus improving the autonomy of the community.

- Low-end computers (e.g. personal computers) can also be supported by the system to run as a resource node in the code resource community.

4.3 Prototype Model Overview

After defining the goals of the target system, we design a prototype software system that provides a more advantageous mechanism for code resource sharing within Grid environment than the previous C/S model.

When running our prototype software, every participating Grid node (computer or supercomputer) can send and receive query messages in a way that makes the node both server and client. To search reusable software components from local domain code repositories, the user node multicasts query messages to all the available code repositories (other system nodes) in the local domain; if no suitable code components can be found in local domain, the user node will send the query message to some special system nodes that enable communication between remote domains. Thus, system users can get access to widely distributed code repositories built on different platforms within all Grid domains and, also, the reusable code component providers can share their code resource to all users among Grid domains. In this way, an aggregation code resource community with diverse code repositories is naturally formed within Grid networks. Figure 4-1 shows the overall architecture of this proposed code resource sharing model.

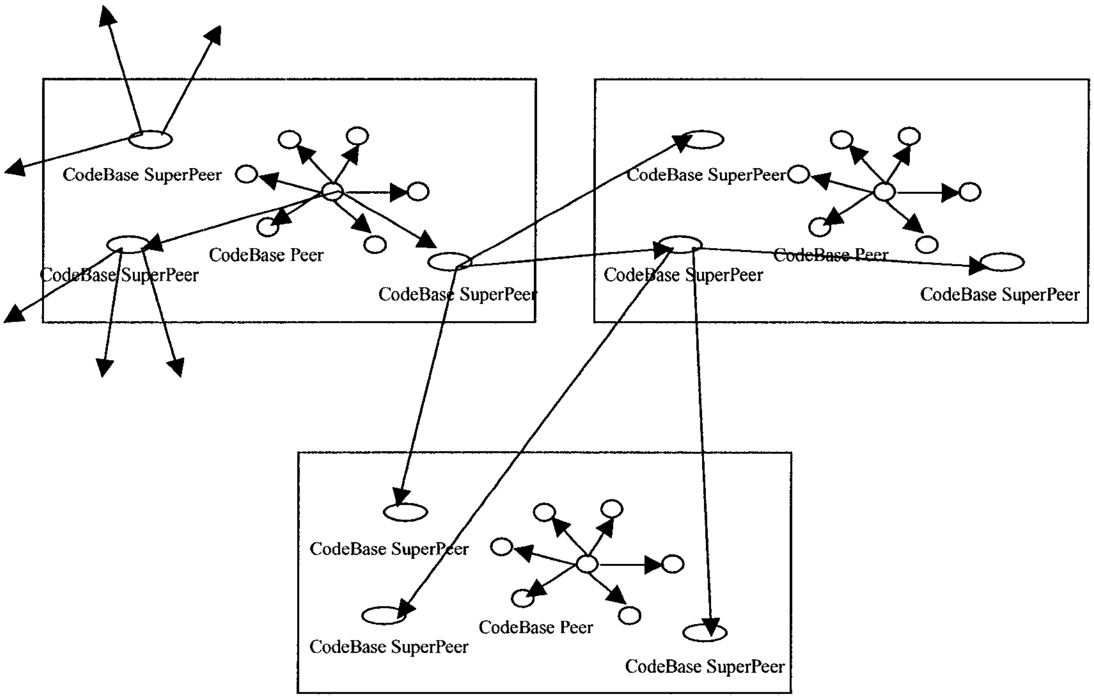


Figure 4-1: Overall Architecture of the Proposed Model

4.4 Prototype Model Design

4.4.1 Roles in Model

In our prototype model, there exist 3 types of roles. Those roles work cooperatively to provide services that integrate all the system nodes together to form a code resource sharing community. Those roles include:

Code Request Peer (CRP): it acts as the code service requestor, which takes parameters set or input by user, analyzes those parameters, forms a well-formatted search request and sends this search request to code repositories. According to user's choice, the CRP can multicast the search request to all the available code repositories in local domain, or route the search request to some gateway-like nodes in local domain (those gateway-like nodes will forward the search request to other repositories in remote domains). After sending out the search request, the CRP waits for searching responses from code repositories, analyzes those response messages, and lists all the searching results on Graphic User Interface (GUI) for user to choose. If a user selects one result from the list, the CRP will send out the retrieval request to the specific code repository that stores the needed reusable software component, wait for the retrieval response from the repository, analyze the response message and finally generate the reusable component source code into a file or display it on the GUI.

Code Service Peer (CSP): it acts as the code service provider in local domain. In essence, CSP is the interface of a core repository. Usually CSP is in listening status waiting for code searching or retrieval requests sent from CRPs. While receiving request, it will change to working status and make transactions. After this process, it will return to listening status again.

There are two types of code requests received by CSP: code search request (defined as A type request in programming) and code retrieval request (defined as B type request in programming). The CSP takes different actions after receiving different requests.

After receiving a search request, the CSP will analyze the request message, transform the message into the proper SQL query statement and execute SQL statement searching in its code repository, which is under the management of a type of DataBase Management System (DBMS). If proper record is found, the CSP will form a response message (defined as A type response in programming) and send back this response to the request CRP; if no result is found, the CSP will change back to listening status without sending any response to request CRP.

Upon receiving a retrieval request, the CSP will analyze the request message, transform the message into the proper SQL query statement and execute SQL statement retrieving source code from code repository. After getting the result, CSP will form a response message (defined as B type response in programming), and send back this response to request CRP.

Code Service SuperPeer (CSS): it acts as the code service provider across Grid domains. CSS functions the same as CSP. CSS is not only the interface of a code repository, but also can send or receive code request other CSSs in remote domains. In a simple view, CSS can be regarded as a "gateway" to connect code resource among different Grid domains.

When a CRP can not find suitable reusable soft component in its local domain, it will send the search request (defined as A type request in programming) to the CSS in the local domain. After receiving the search request from local CRP, first the CSS takes the same actions as CSP does. But if CSS can not find a suitable component in its own repository, the CSS will form a search request message (defined as C type request in programming) and send out this request to other remote domains' CSSs according to its routing list.

After a CSS receives a search request (C type request) from a remote CSS, this CSS will analyze the request message, transform the message into the proper SQL query statement and execute SQL statement searching in code repository. If result is found, the CSS will form a response message (defined as D type response in programming) and send the response back to the original request CSS, the one that receives search request from CRP. If result is not found, the CSS will form a search request message (defined as C type request) and send out this request to other CSSs according to its routing list.

When a CSS receives a response (D type response) from a remote CSS, this CSS will analyze the response message, transform the message into the proper SQL update statement and execute the SQL statement inserting the new reusable component source code into its repository. This process can help the code resource to achieve a distributed storage balance. If next time some local CRPs request for the same reusable component, they can get it within the scope of local domain instead of requesting remote domains through CSS. After this process, the CSS forms a response (defined as

E type response) and send it back to the local CSR that sent out the original search request at the very beginning.

In summary, there exist 4 types of communication messages received by CSS: code search request from CRP (A type request); code retrieval request from CRP (B type request); code search request from other CSS (C type request); code resource response from other CSS (D type response). There exist 5 types of communication messages sent from CSS: code search response to local CRP (A type response), code retrieval response to local CRP (B type response); code search request to other CSS (C type request); code response to local CSS from remote CSS (D type response); code search response to local CRP after getting response from remote CSS (E type response).

Any participating computer as a system node in the model may not run as just one role, two or three roles can be run at the same system node. When a system node run at least as a role of CSP, it is also called CodeBase Peer; when a system node run at least as a role of CSS, it is also called CodeBase SuperPeer.

4.4.2 Working Modes in Model

In our prototype model, there exist two types of working modes: the multicast mode in which reusable code components are searched and retrieved within local domain; the routing mode in which reusable code components are searched and retrieved from remote domains.

Multicasting Mode: In this mode, the CRP multicasts the request within its local domain. Every available local CSP receives the request and makes transaction. After receiving responses from CSPs, the CRP can list on GUI all the qualified reused code components in local domain, and let user select the suitable one from the list to retrieve and generate source code. (The mechanism is illustrated in Figure 4-2).

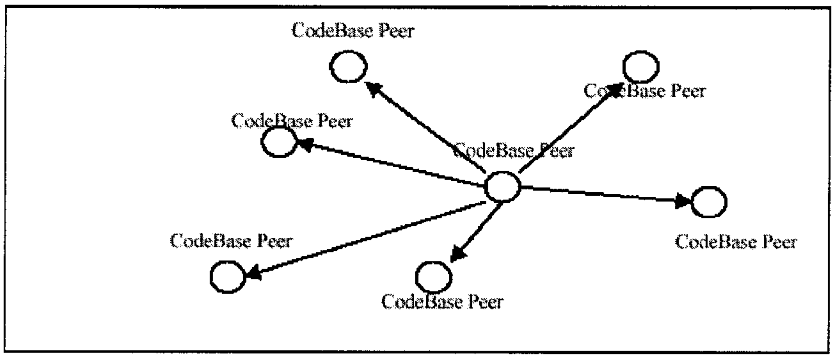


Figure 4-2: Multicasting Mode for Code Resource Searching

The detailed system workflow in multicasting mode is illustrated in Figure 4-3.

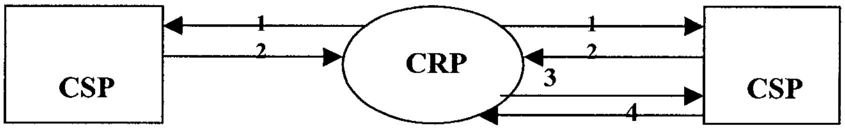


Figure 4-3: System Workflow in Multicasting Mode

As showed in Figure 4-3, system workflow in multicasting mode is composed of the following 4 steps:

1. CRP multicasts search request (A type request) to all the available CSPs in local domain.
2. If qualified code component is found in a CSP's code repository, this CSP will send back a response (A type response) to CRP.
3. CRP sends request (B type request) to CSP asking for retrieving code component.
4. CSP sends back the response (B type response) to CRP.

Routing Mode: in this mode, the CRP sends the request to several local CSSs that not only have big code repository, but also can communicate with other CSSs in remote domains. If the required code component is not found in its repository, the CSS will route the request to several remote CSSs in other domains according to its routing list.

If the qualified code component is found in remote Grid domain, the remote CSS will send the code component to the local CSS, and the local CSS will import the new code component into its code repository and notify the CRP to fetch the code component. (The mechanism is illustrated in Figure 4-1)

The detailed system workflow in routing mode is illustrated in Figure 4-4.

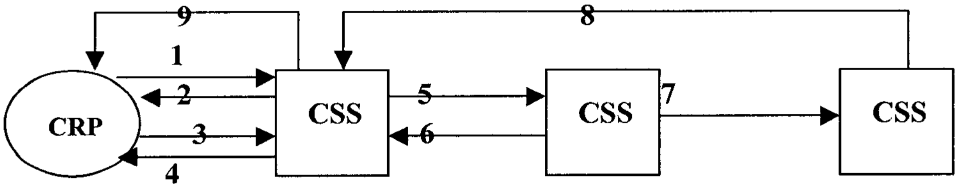


Figure 4-4: System Workflow in Routing Mode

As showed in Figure 4-4, system workflow in routing mode is composed of the following 9 steps:

1. CRP sends code search request (A type request) to CSS in local domain.
2. If required code component is found in CSS's repository, local CSS will give a response (A type response) to CRP.
3. CRP sends query (B type request) to CSS asking for code component.
4. CSS sends back the response (B type response) with required code component to CRP.
5. If required code component is not found, CSS will forward request (C type request) to other remote domains' CSSs.
6. If required code component is found, the remote CSS will return response (D type response) containing required code component to the local CSS.
7. If required code component is not found, CSS will forwards query (C type request) to other domains' CSSs.
8. If required code component is found, the remote CSS will return response (D type response) containing required code component to the local CSS.

9. CSS gives notification response (E type response) to CRP.

4.4.3 Communication Messages in Model

As mentioned in the previous section, there are a lot of communications between system nodes in code source sharing community. Those communications are completed with the help of several types of messages, which are pre-defined by the system. The formats of those types of messages are described as follows:

A type request:

Type	Code Keyword	TimeStamp

A type response:

Type	CSP or CSS IP	Code Keyword	TimeStamp	Code Name	Code Description

B type request:

Type	Code Keyword	TimeStamp

B type response:

Type	CSP or CSS IP	TimeStamp	Code Component	Code Description

C type request:

Type	Code Keyword	TimeStamp	Original CRP IP	Middle CSSs' IPs	Hop Number

D type response:

Type	Code Keyword	TimeStamp	Code Component	Code Description

Middle CSSs' IPs	Original CRP IP

E type response:

Type	CSS IP	Remote CSSs IPs	TimeStamp	Code Name	Code Description

Every message contains several information items: the “Type” item identifies the type of this message, there are 5 message types represented by big case letters of A,B,C,D and E; the “IP” item contains the address information of the system node, it can be used for locating; ”Code Keyword” is the searching criteria, with which the system searches and retrieves suitable reusable code components needed by user; ”Code Name” is the name of the required code component stored in a code repository; ”Code Description” is the detailed statement of the required code component, it can be used to judge whether the code component is qualified for user’s need or not; “Code Component” is source code body of a specific code component; “TimeStamp” item represents the valid period of this message; “Hop Number’ item represents the number of CSSs that still can route this message, if the value is zero, the CSS that receives this message will stop to route the message to the next CSS even if the qualified code component is not found. “TimeStamp” and “Hope Number” items are used for searching scale control that will discussed in section 4.4.6.

For CRP, it communicates with code resource community by sending out 2 types of request messages and receiving 3 types of response messages as showed in Figure 4-5.

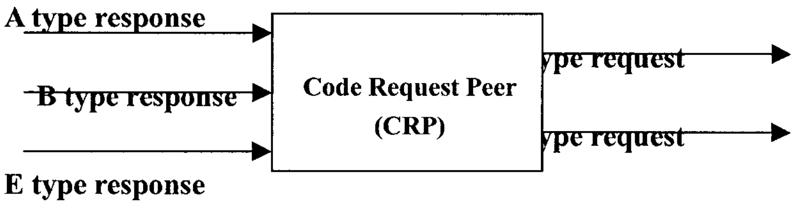


Figure 4-5: Communication Messages for CRP

For CSP, it provides code service by sending out 2 types of response messages and receiving 2 types of response messages as showed in Figure 4-6.

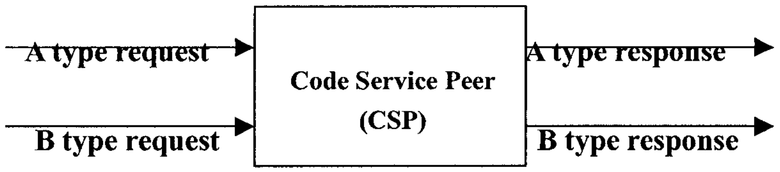


Figure 4-6: Communication Messages for CSP

For CSS, since it acts both as a code service provider and as a gateway to remote CSSs, it functions to provide service by sending out 5 types of messages and receiving 4 types of messages as showed in Figure 4-7.

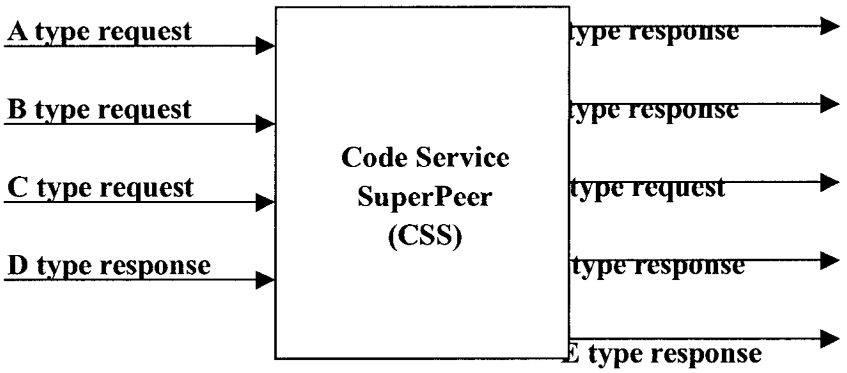


Figure 4-7: Communication Messages for CSS

4.4.4 Structures of System Roles

In our model, the source community is composed of a lot of system nodes. As we described in the previous chapter, every system node plays one or several roles. In implementation, every role is constructed by some functional parts.

Code Request Peer is composed of the following 4 parts:

- Code Request Peer Interface: the GUI interface between user and system.
- Code Request Peer Receiver: analyze the message received and make corresponding transactions. If the message is retrieval result, then call Code Generator.
- Code Request Peer Sender: integrate all the necessary information to form a

structured request message and send out the message.

- Code Generator: generate the well-formed source code for end user.

Figure 4-8 illustrates the Code Request Peer Structure

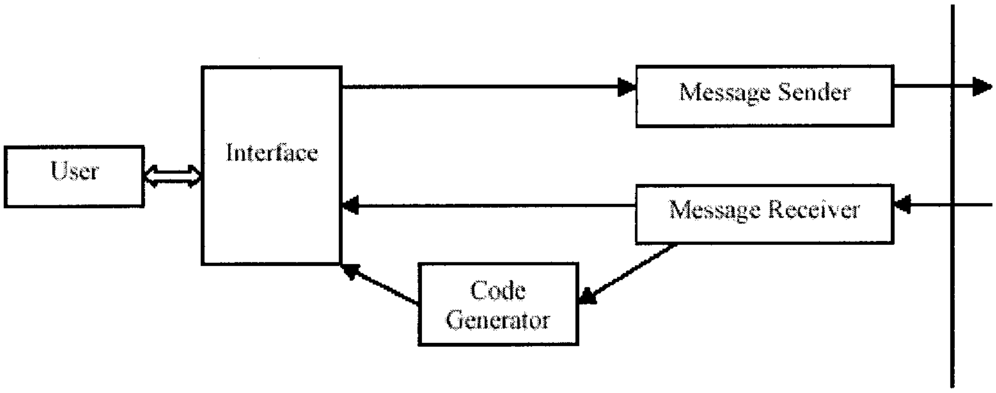


Figure 4-8: Code Request Peer Structure

Code Service Peer is composed of the following 5 parts:

- Code Service Peer Monitor: the GUI that displays CSP working status.
- Code Service Peer Receiver: analyze the message received and activate Transaction Center to make corresponding transactions.
- Code Service Peer Sender: integrate all the necessary information to form a structured message and send the message out.
- Code Service Peer Transaction Center: The transaction component that executes SQL statement in code repository.
- Code Service Peer Code Base: Database for storing code resource as a code repository.

Figure 4-9 illustrates the Code Request Peer Structure

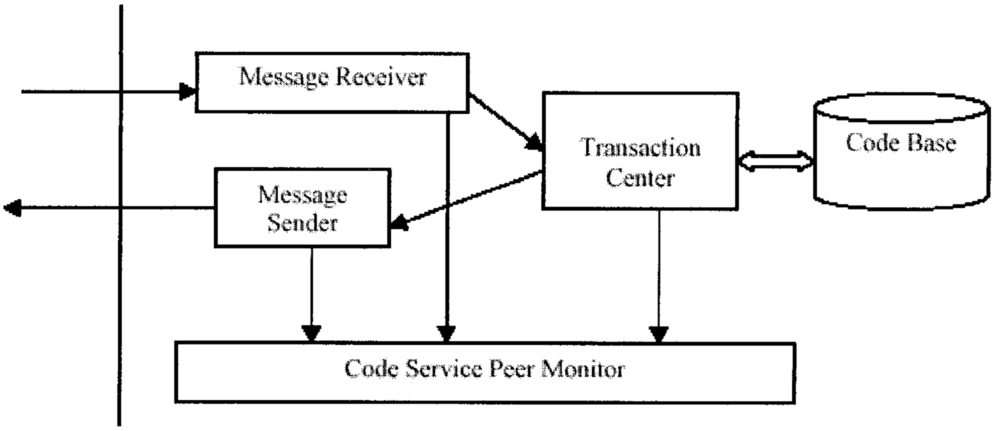


Figure 4-9: Code Service Peer Structure

Code Service SuperPeer is composed of the following 6 parts:

- Code Service SuperPeer Monitor: the GUI that displays CSS working status.
- Code Service SuperPeer Receiver: analyze the message received and activate Transaction Center to make corresponding transactions.
- Code Service SuperPeer Sender: integrate all the necessary information to form a structured request message and send the message out.
- Code Service SuperPeer Transaction Center: the transaction component that executes SQL statement in code repository.
- Code Service SuperPeer Routing List: the file that stores routing information--remote CSS's address.
- Code Service SuperPeer Code Base: Database for storing code resource as a code repository.

Figure 4-10 illustrates the Code Request SuperPeer Structure

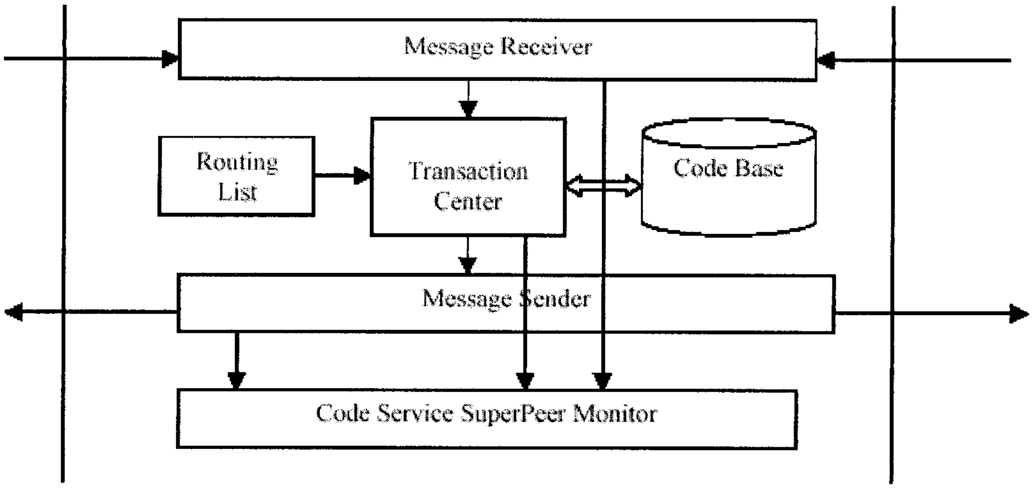


Figure 4-10: Code Service SuperPeer Structure

4.4.5 Scale Control in Model

To maintain efficiency of the model system, three mechanisms are utilized to control code searching scale in our model. Those mechanisms are TimeStamp, HopNumber Control and RoutingHistroy Check

TimeStamp is used in both two working modes. Before CRP multicasts or routers the code search request, a structured data called TimeStamp, which is the calculated result of the request time and the timeout set by user, is added into the request message. As the CSP or CSS receives a request message, it will first check the TimeStamp to make sure that this request is still valid: if yes, the Peer or SuperPeer will make the following transactions; if no, this request will be ignored.

HopNumber Control is used in routing mode. When CSS plans to forward the code request to next CSS, it will check the HopNumber information in the request message. If the HopNumber is zero, that means this CSS is already the last one the request supposed to reach, so it will no longer forward the request; if the HopNumber is not zero, this CSS will change the HopNumber information (minus 1) and forward the request to other remote domain's CSSs according to its routing list. In this way, the user can control the size of the query chain in routing mode for code searching.

Routing History Check is also used in routing mode to avoid routing loop. If the CSS plan to forward the code request to other CSSs, it will add its own identify information (such as IP) into a structured data called Routing History in the request message. So if the next CSS get the code request, it can check whether itself is already in the Routing History: if yes, that means this request run into a routing loop, and the CSS will ignore the request; if no, that means this request never reach the CSS before, so it will take the corresponding transactions. In this way, the system can keep efficient and avoid wasting computer resource for redundant work.

4.4.6 Domain Split and Mergence in Model

A logical Grid domain is usually represented by a physical LAN; so, a Grid containing multiple domains is a wide area network interconnecting multiple LANs. Therefore, the challenge of splitting and merging domains is indeed a physical network problem. But since the physical network split and mergence need a lot of reconfiguration on networks, our model use a mechanism that can partially solve the split and mergence problem in a logic way instead of physically re-configurating networks, thus makes our model more dynamic and flexible in a unstable environment.

In our model, the code resource integration in local domain within a network is implemented with the concept of multicast group. A multicast group is a Class D IP address (that is, in the range of 224.0.0.0- 239.255.255.255). Multicast message that is sent to an address in this range is not destined for a single target node. Instead, this message can be received by any host node that has joined this specific group. In our prototype system, initially every CSP in a network joins a default multicast group (IP 224.0.0.1 in our implementation), so any search request sent to this group IP will be received and transacted by all the CSPs in local domain. Since the physical routers connected LANs usually disable multicast traffic, the multicast search request will not run across domain scope even if the same multicast group IP is used in a connected neighbor LAN.

Therefore, in our model, the code resource domain split can be implemented by making some CSPs leave the default multicast group to join a new multicast group. A dialog window will pop up asking which multicast group this node want to join into, every time we start up our prototype software program. The domain split procedure is much easier to execute than the physical network split operation. Similarly, the code resource domain merge is completed by merging CSPs belonging to different multicast groups into one multicast group. Thus, the domain split and merge problems within LANs can be changed to multicast group split and mergen problems in our model. Using the same mechanism, we can also solve domain split and merge problems across LANs on condition that the routers connected LANs are set to support multicast traffic.

4.5 Development Environment

The prototype system is developed using the advanced object-oriented and distributed technologies. Currently, the development and execution environment of the prototype system is listed as follows:

- Implementation language: Java 2 (JDK1.4) for Microsoft Windows 98/NT/2000/XP and LINUX
- DBMS: Microsoft Access^{TM} for Windows 98/NT/2000/XP, MySQL for Microsoft Windows 98/NT/2000/XP, MySQL for Linux
- Platforms: Microsoft Windows 98/NT/2000/XP and LINUX
- Execution environment: Microsoft Windows 98/NT/2000/XP, LINUX, Unix with Java Virtual Machine (JVM)
- Communication Port: the communication between system nodes takes place on computer ports. In implementation, we define port 5000 for CRP communication, port 6000 for CSP communication, port 7000 for CSS communication.

4.6 System Design Specification

Our prototype system is designed using Unified Modeling Language (UML), a language that unifies much of engineering practices for modeling system. Based on the object-oriented paradigm, UML is utilized here for specifying, visualizing, constructing and documenting our software system. This section will describe use cases, class diagrams and sequence diagrams of our prototype system.

4.6.1 System Use Cases

As we state in section 4.4.1, there exist 3 types of roles in our proposed model. Every role is expected to complete some functionalities.

Figure 4-11 elaborates the CRP use cases by detailing the functionality a CRP actor expects of the system. The CRP can broadcast or router searching query. After receiving response messages, it can list all the searching results or generate the well-formed source code.

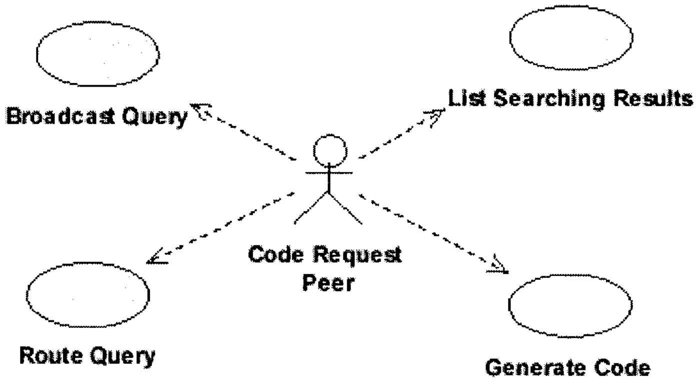


Figure 4-11: Use Case Diagram for Code Request Peer

Figure 4-12 elaborates the CSP use case by detailing the functionality a CSP actor expects of the system. The CSP can listen and receive query, analyze the query to determine whether to search code or retrieve code from code repository, and send back response message.

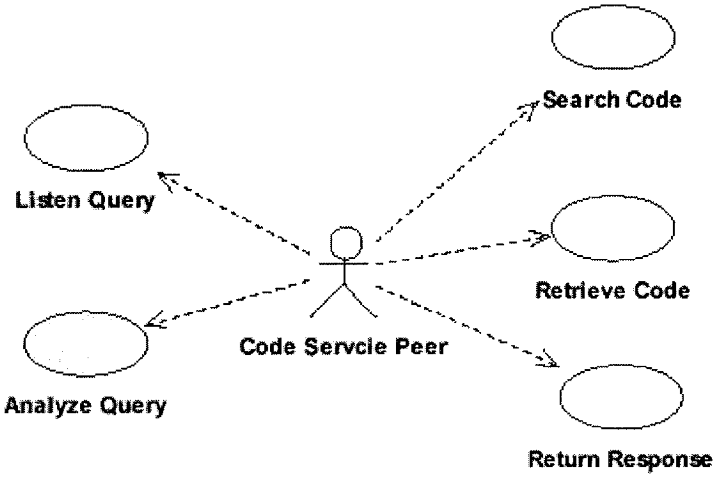


Figure 4-12: Use Case Diagram for Code Service Peer

Figure 4-13 elaborates the CSS use case by detailing the functionality a CSS actor expects of the system. The CSS can listen and receive query, analyze the query to determine whether to search code or retrieve code from code repository or store new code into code repository, then send back response message or forward request to the next CSS.

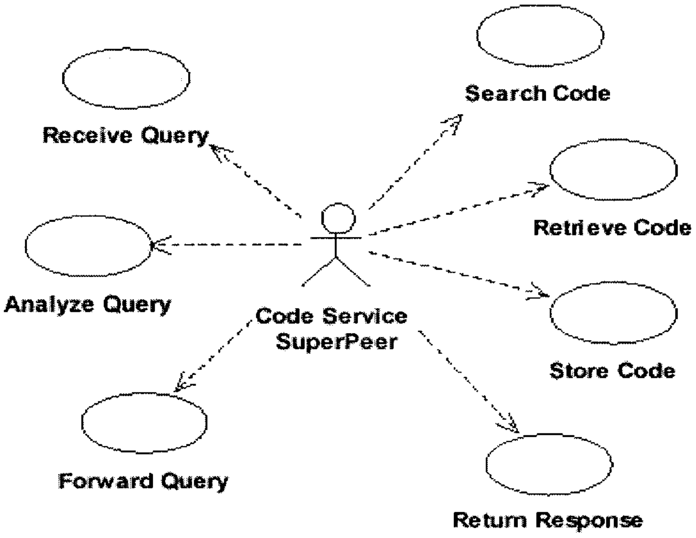


Figure 4-13: Use Case Diagram for Code Service SuperPeer

4.6.2 System Class Diagrams

The major system diagrams are described in following figures of 4-14,4-15 and 4-16. Figure 4-14 shows the static structure of major classes of the CRP. PeerSearchGUI class defines the graphic interface between user and system; PeerSearchTableModel class specifies the methods the Jtable will use to interrogate a tabular data model; PeerSearch class manipulates its components class to work together for communication and transactions; CRPReceiver class is the component class of PeerSearch class, it deals with receiving and analyzing messages received on the port; CRPSender class is the component class of PeerSearch class, it deals with sending out messages; CodeGenerator class is the component class of PeerSearch class, it deals with generate well-formed source code from the message received by CRPReceiver.

Figure 4-15 shows the static structure of major classes of the CSP. PeerServiceGUI class defines the graphic interface to display CSP working status; PeerService class manipulates its components class to work together for communication and transactions to provide code service; CSPReceiver class is the component class of PeerService class, it deals with receiving and analyzing messages received on the port; CSPSender class is the component class of PeerService class, it deals with sending out messages; CSPTransactor class is the component class of PeerService class, it defines methods for transacting with code repositories.

Figure 4-16 shows the static structure of major classes of the CSS. SuperPeerServiceGUI class defines the graphic interface to display CSS working status; SuperPeerService class manipulates its components class to work together for communication and transactions to provide code service; CSSReceiver class is the component class of SuperPeerService class, it deals with receiving and analyzing messages received on the port; CSSSender class is the component class of SuperPeerService class, it deals with sending out message; CSSTransactor class is the component class of SuperPeerService class, it defines methods for transacting with code repositories.

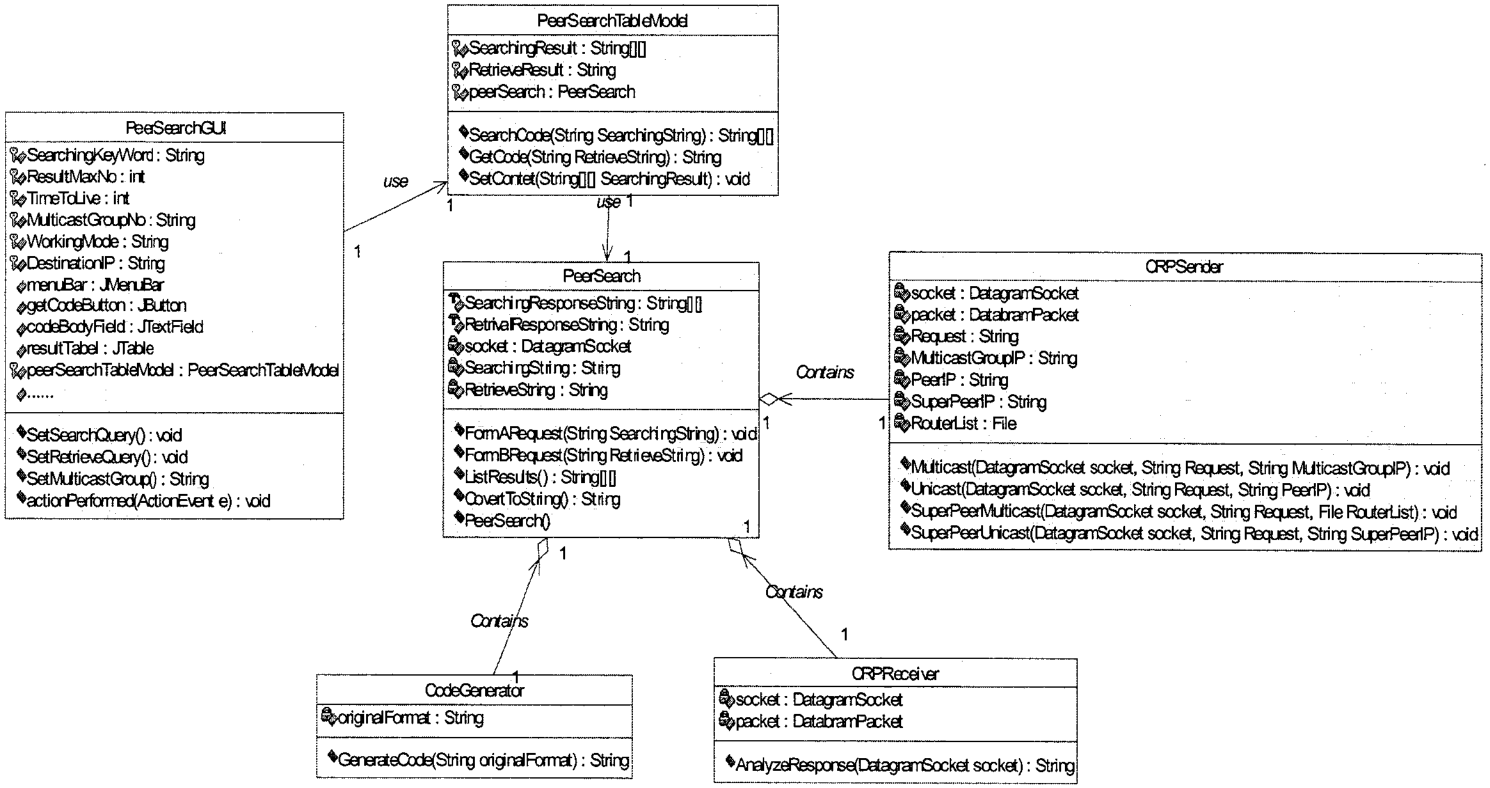

PeerSearchTableModel
SearchingResult : String[][]
RetrieveResult : String
peerSearch : PeerSearch
SearchCode(String SearchingString) : String[][]
GetCode(String RetrieveString) : String
SetContet(String[][] SearchingResult) : void
PeeSearchGUI
SearchingKeyWord : String
ResultMaxNo : int
TimeToLive : int
MulticastGroupNo : String
WorkingMode : String
DestinationIP : String
menuBar : JMenuBar
getCodeButton : JButton
codeBodyField : JTextField
resultTabel : JTable
peerSearchTableModel : PeerSearchTableModel
......
SetSearchQuery() : void
SetRetrieveQuery() : void
SetMulticastGroup() : String
actionPerformed(ActionEvent e) : void
use
1
1
use 1
1
PeerSearch
SearchingResponseString : String[][]
RetrivalResponseString : String
socket : DatagramSocket
SearchingString : Stirng
RetrieveString : String
FormARequest(String SearchingString) : void
FormBRequest(String RetrieveString) : void
ListResults() : String[][]
CovertToString() : String
PeerSearch()
Contains
1
1
CRPSender
socket : DatagramSocket
packet : DatabramPacket
Request : String
MulticastGroupIP : String
PeerIP : String
SuperPeerIP : String
RouterList : File
Multicast(DatagramSocket socket, String Request, String MulticastGroupIP) : void
Unicast(DatagramSocket socket, String Request, String PeerIP) : void
SuperPeerMulticast(DatagramSocket socket, String Request, File RouterList) : void
SuperPeerUnicast(DatagramSocket socket, String Request, String SuperPeerIP) : void
1
1
Contains
Contains
1
1
CodeGenerator
originalFormat : String
GenerateCode(String originalFormat) : String
CRPReceiver
socket : DatagramSocket
packet : DatabramPacket
AnalyzeResponse(DatagramSocket socket) : String

Figure 4-14: Code Request Peer (CRP) Class Diagram

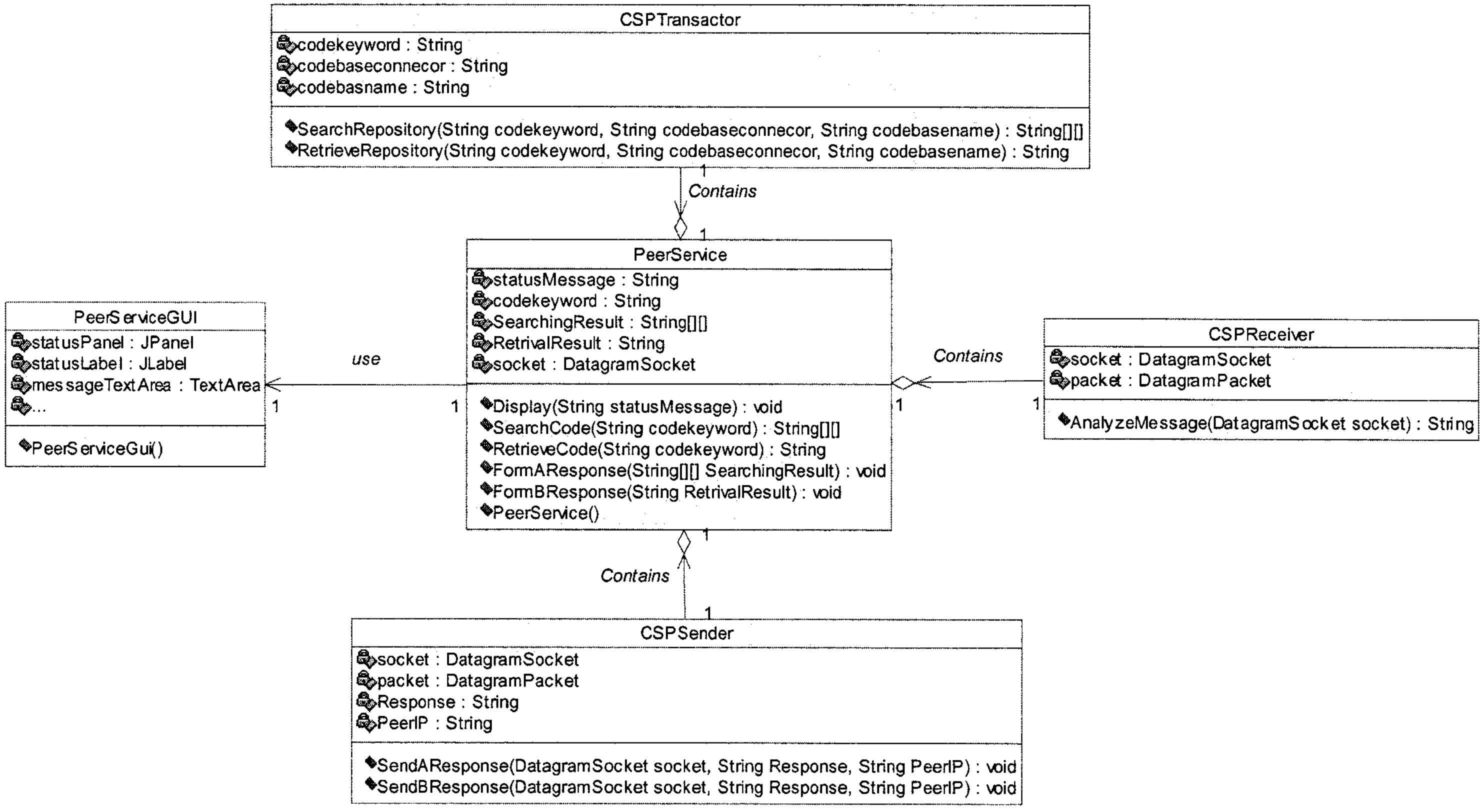

CSPTransactor
codekeyword : String
codebaseconnecor : String
codebasname : String
SearchRepository(String codekeyword, String codebaseconnecor, String codebasename) : String[][]
RetrieveRepository(String codekeyword, String codebaseconnecor, String codebasename) : String
Contains
1
1
PeerService
statusMessage : String
codekeyword : String
SearchingResult : String[][]
RetrivalResult : String
socket : DatagramSocket
Display(String statusMessage) : void
SearchCode(String codekeyword) : String[][]
RetrieveCode(String codekeyword) : String
FormAResponse(String[][] SearchingResult) : void
FormBResponse(String RetrivalResult) : void
PeerService()
PeerServiceGUI
statusPanel : JPanel
statusLabel : JLabel
messageTextArea : TextArea
...
PeerServiceGui()
use
1
1
Contains
1
1
CSPReceiver
socket : DatagramSocket
packet : DatagramPacket
AnalyzeMessage(DatagramSocket socket) : String
Contains
1
1
CSPSender
socket : DatagramSocket
packet : DatagramPacket
Response : String
PeerIP : String
SendAResponse(DatagramSocket socket, String Response, String PeerIP) : void
SendBResponse(DatagramSocket socket, String Response, String PeerIP) : void

Figure 4-15: Code Service Peer (CSP) Class Diagram

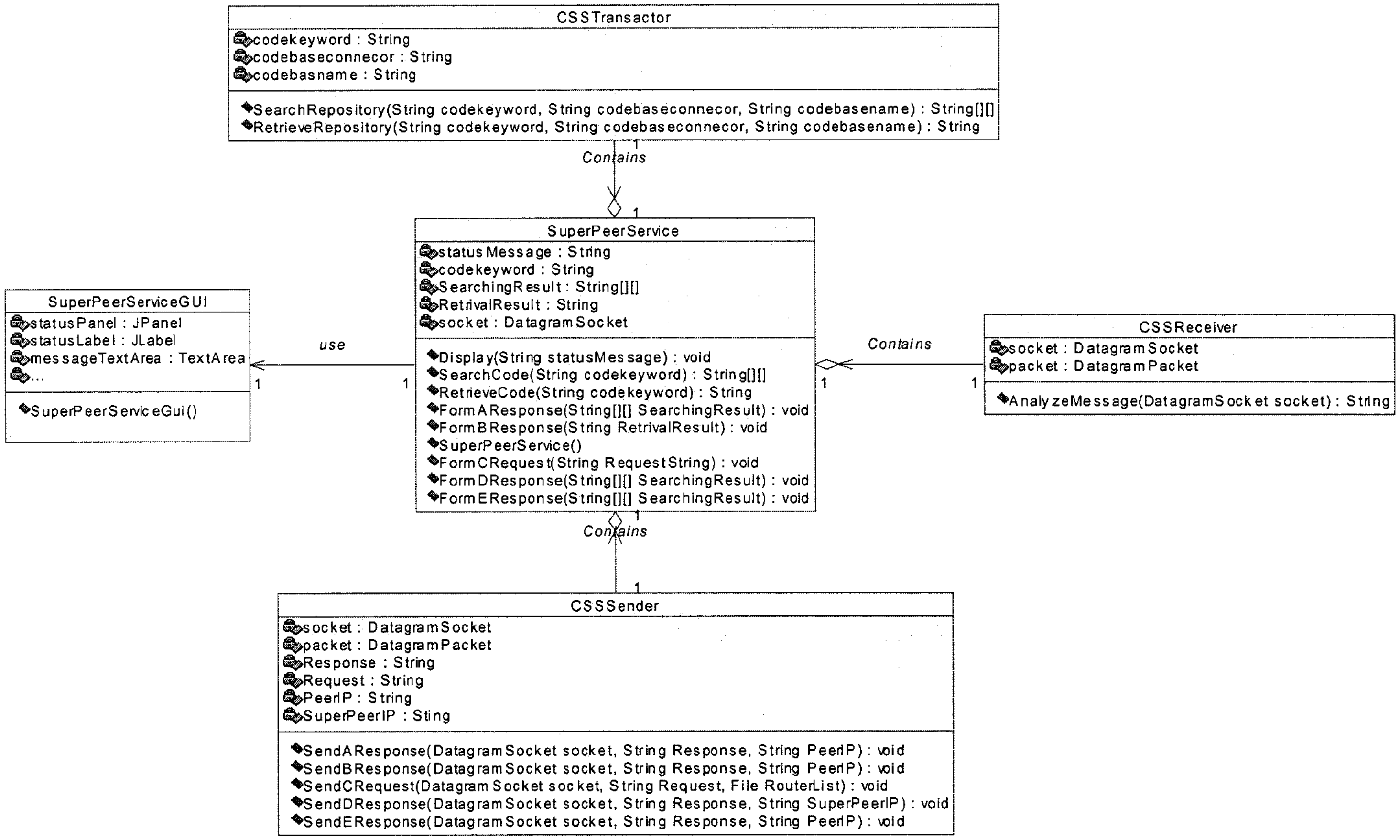

CSSTransactor
codekeyword : String
codebaseconnecor : String
codebasname : String
SearchRepository(String codekeyword, String codebaseconnecor, String codebasename) : String[][]
RetrieveRepository(String codekeyword, String codebaseconnecor, String codebasename) : String
Contains
1
1
SuperPeerService
statusMessage : String
codekeyword : String
SearchingResult : String[][]
RetrivalResult : String
socket : DatagramSocket
Display(String statusMessage) : void
SearchCode(String codekeyword) : String[][]
RetrieveCode(String codekeyword) : String
FormAResponse(String[][] SearchingResult) : void
FormBResponse(String RetrivalResult) : void
SuperPeerService()
FormCRequest(String RequestString) : void
FormDResponse(String[][] SearchingResult) : void
FormEResponse(String[][] SearchingResult) : void
SuperPeerServiceGUI
statusPanel : JPanel
statusLabel : JLabel
messageTextArea : TextArea
...
SuperPeerServiceGui()
use
1
1
CSSReceiver
socket : DatagramSocket
packet : DatagramPacket
AnalyzeMessage(DatagramSocket socket) : String
Contains
1
1
Contains
1
1
CSSSender
socket : DatagramSocket
packet : DatagramPacket
Response : String
Request : String
PeerIP : String
SuperPeerIP : Sting
SendAResponse(DatagramSocket socket, String Response, String PeerIP) : void
SendBResponse(DatagramSocket socket, String Response, String PeerIP) : void
SendCRequest(Datagram Socket socket, String Request, File RouterList) : void
SendDResponse(DatagramSocket socket, String Response, String SuperPeerIP) : void
SendEResponse(DatagramSocket socket, String Response, String PeerIP) : void

Figure 4-16: Code Service SuperPeer (CSS) Class Diagram

4.6.3 System Sequence Diagrams

The eight major sequence diagrams are shown from Figure 4-17 to Figure 4-24, which are used to describe the procedures of communication between and transactions within system roles.

Figure 4-17 shows the CRP Broadcast Query Sequence Diagram to describe the procedure of CRP searching and retrieving code component from CSP.

Figure 4-18 shows CRP Route Query Sequence Diagram to describe the procedure of CRP searching and retrieving code component stored in local CSS originally.

Figure 4-19 shows CRP Route Query Sequence Diagram to describe the procedure of CRP searching and retrieving code component not stored in local CSS originally.

Figure 4-20 shows CSP Search and Retrieve Code Sequence Diagram to describe the procedure of CSP transacting with code repositories after receiving request.

Figure 4-21 shows CSS Forward Query Sequence Diagram to describe the procedure of CSS forwarding request from CRP.

Figure 4-22 shows CSS Forward Query Sequence Diagram to describe the procedure of CSS forwarding request from other CSS.

Figure 4-23 shows CSS Search Code Sequence Diagram to describe the procedure of CSS searching and retrieving code component stored in code repository.

Figure 4-24 shows CSS Store Code Sequence Diagram to describe the procedure of CSS importing code component from other CSS.

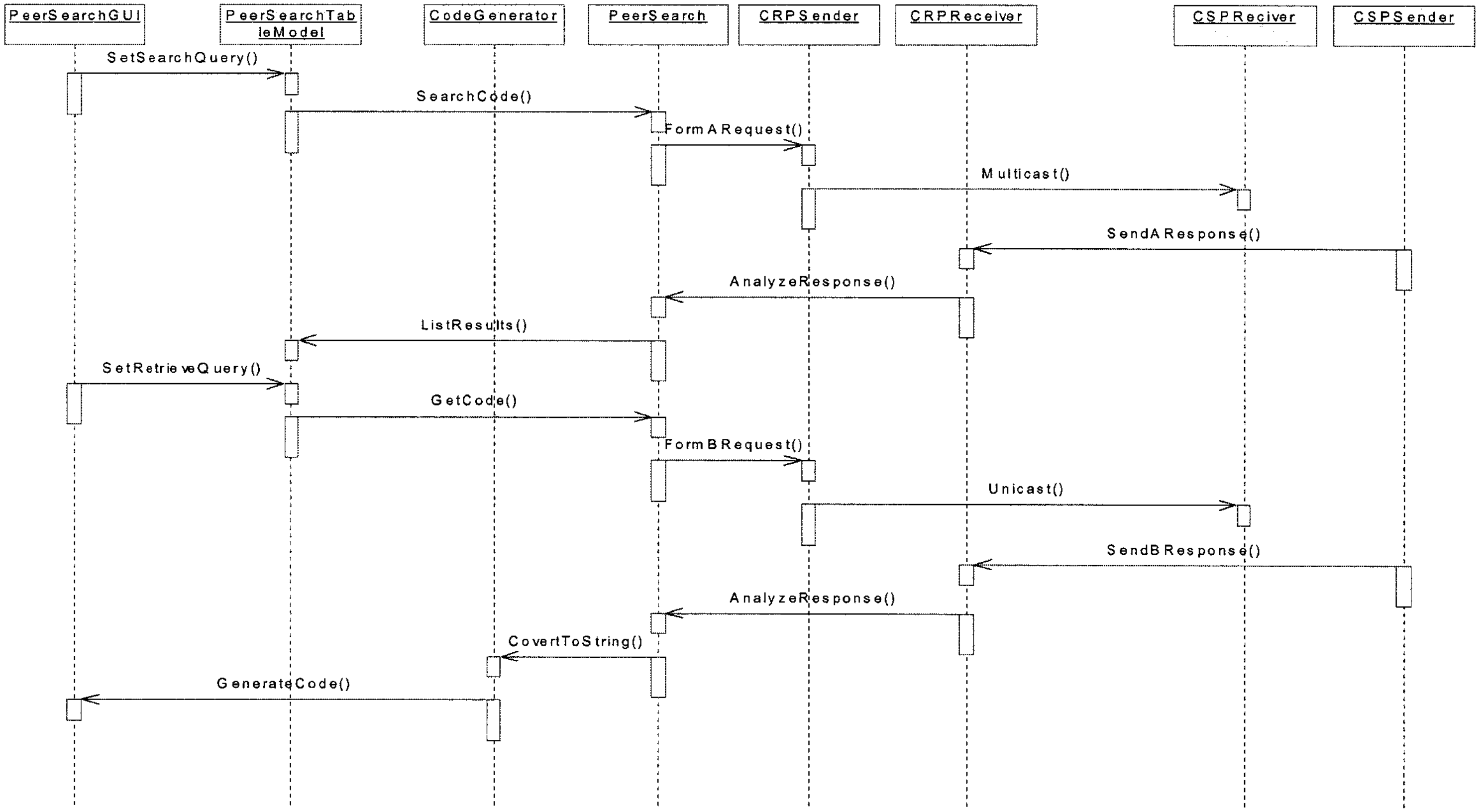


Figure 4-17: CRP Broadcast Query Sequence Diagram

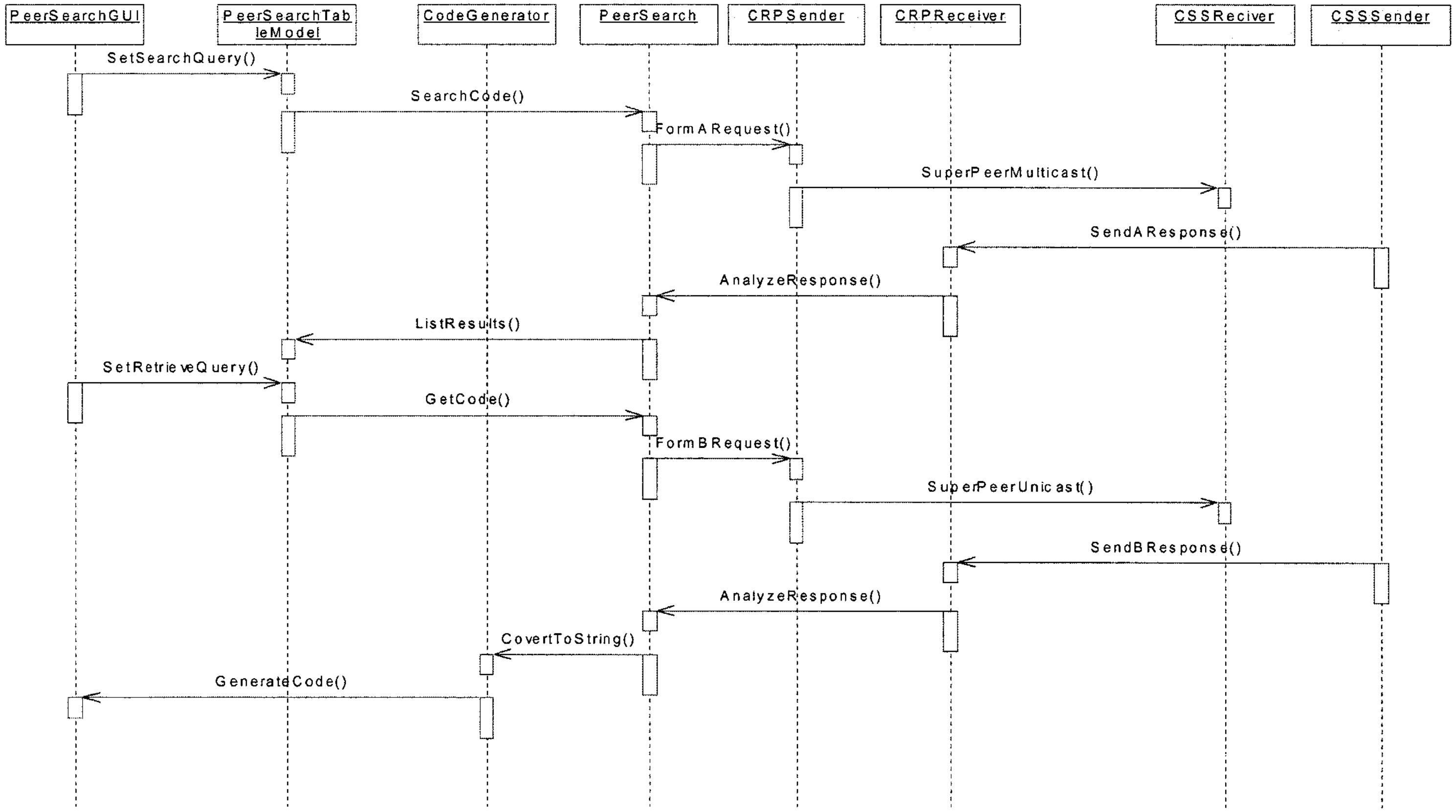


Figure 4-18: CRP Route Query Sequence Diagram [Within Domain]

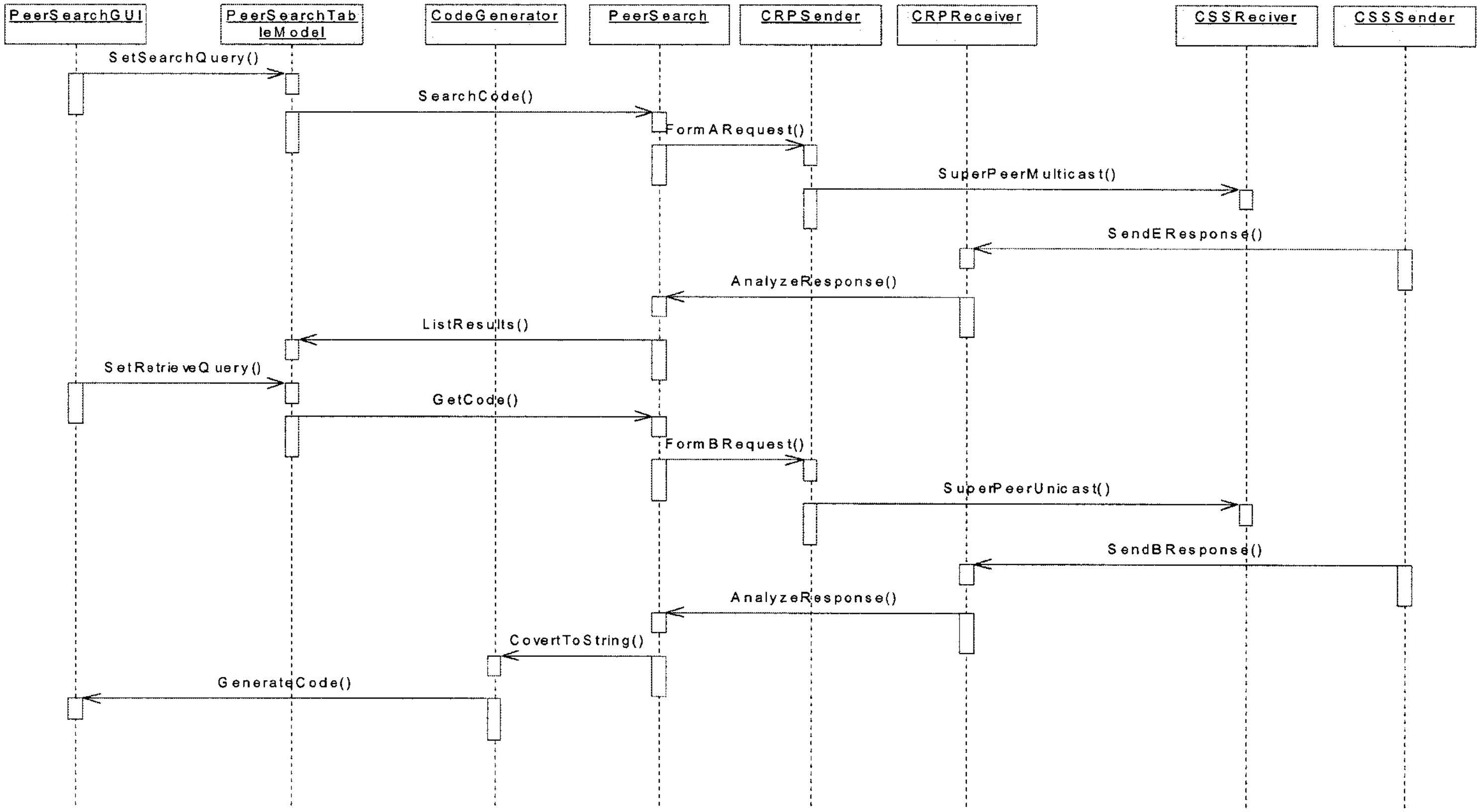


Figure 4-19: CRP Route Query (to CSS) Sequence Diagram [Across Domain]

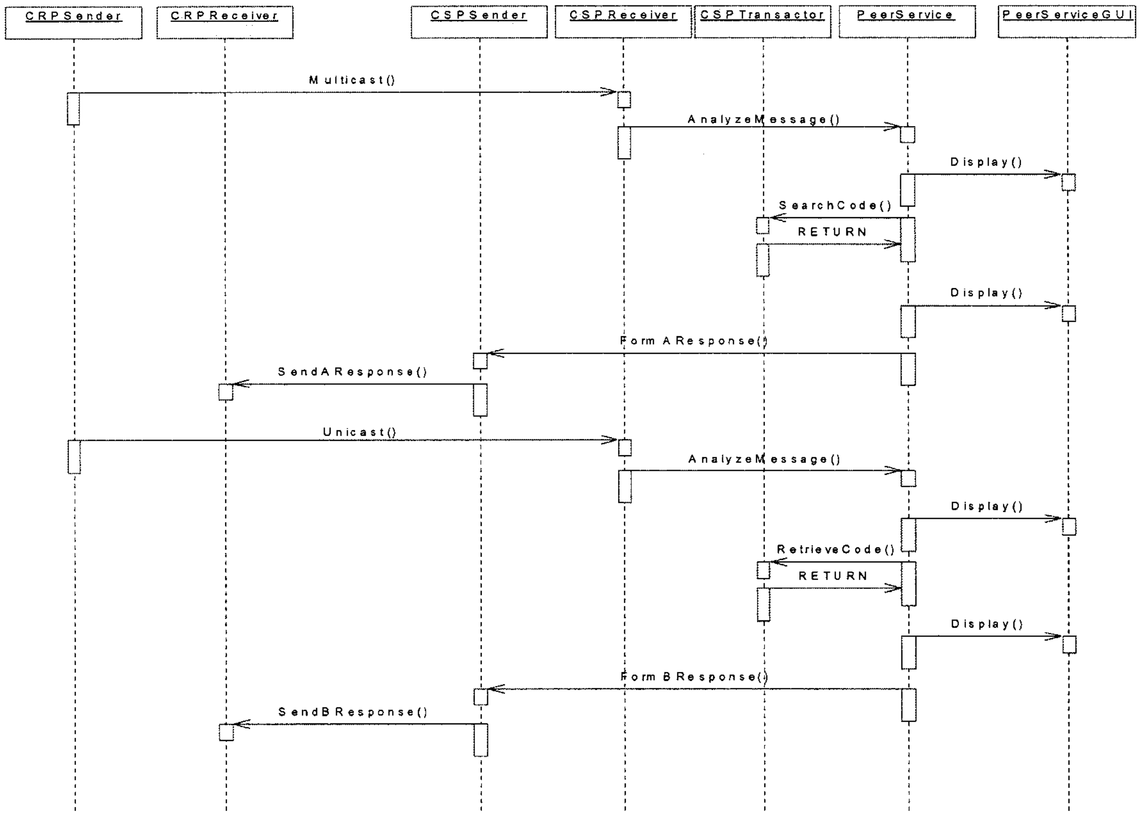


Figure 4-20: CSP Search and Retrieve Code Sequence Diagram

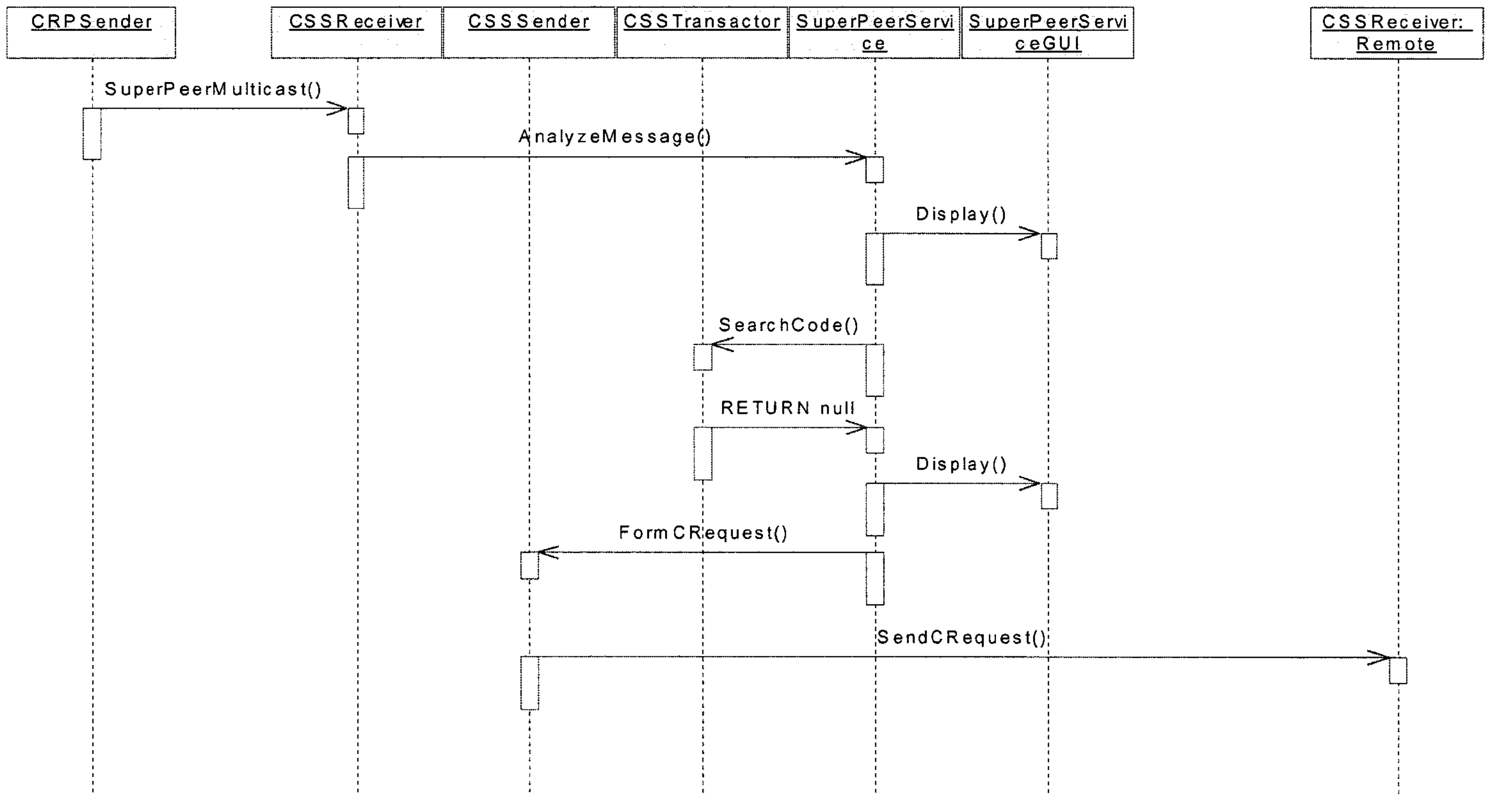


Figure 4-21: CSS Forward Query Sequence Diagram [Query from CRP]

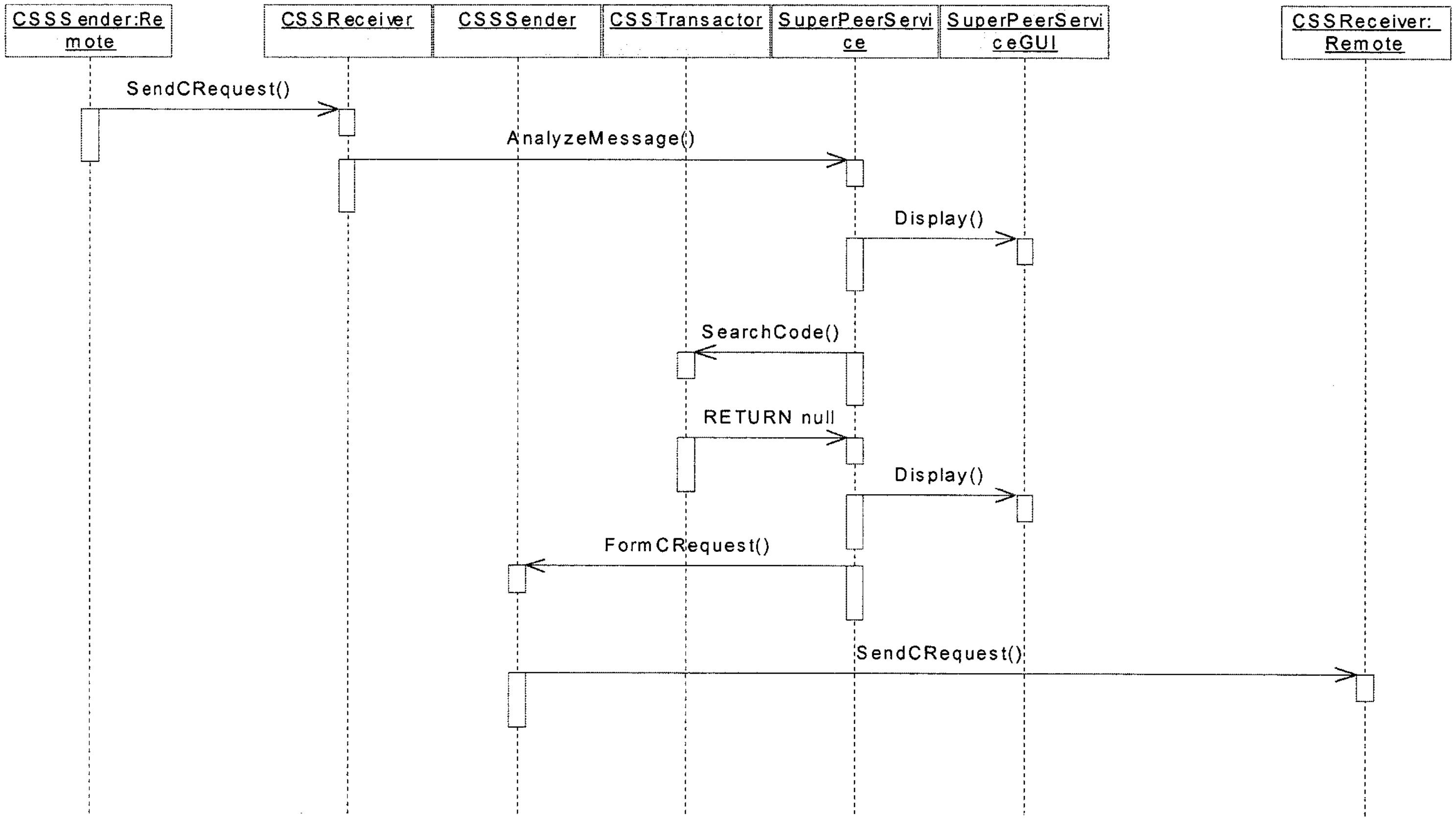


Figure 4-22: CSS Forward Query Sequence Diagram [Query from CSS]

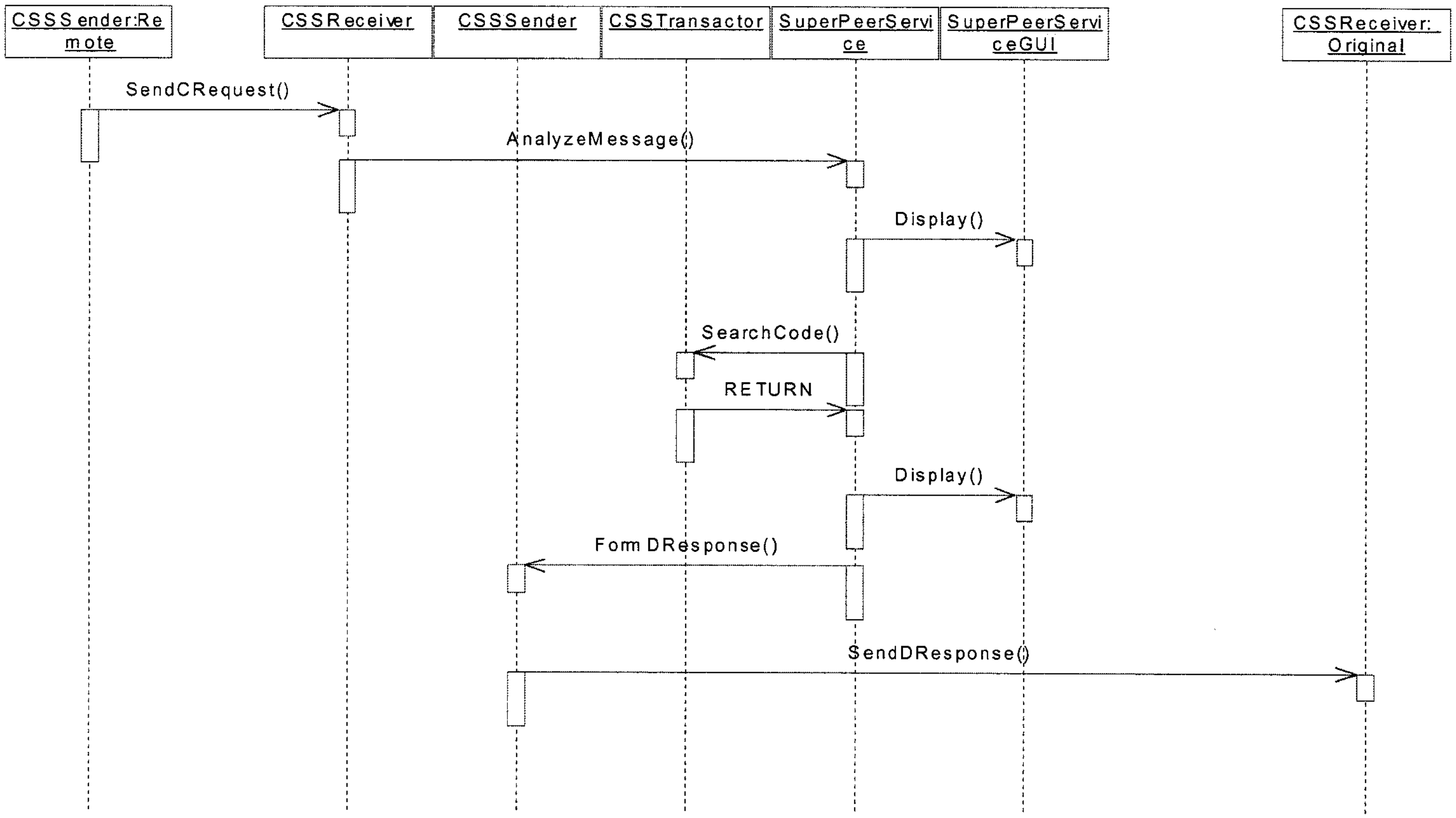


Figure 4-23: CSS Search Code Sequence Diagram

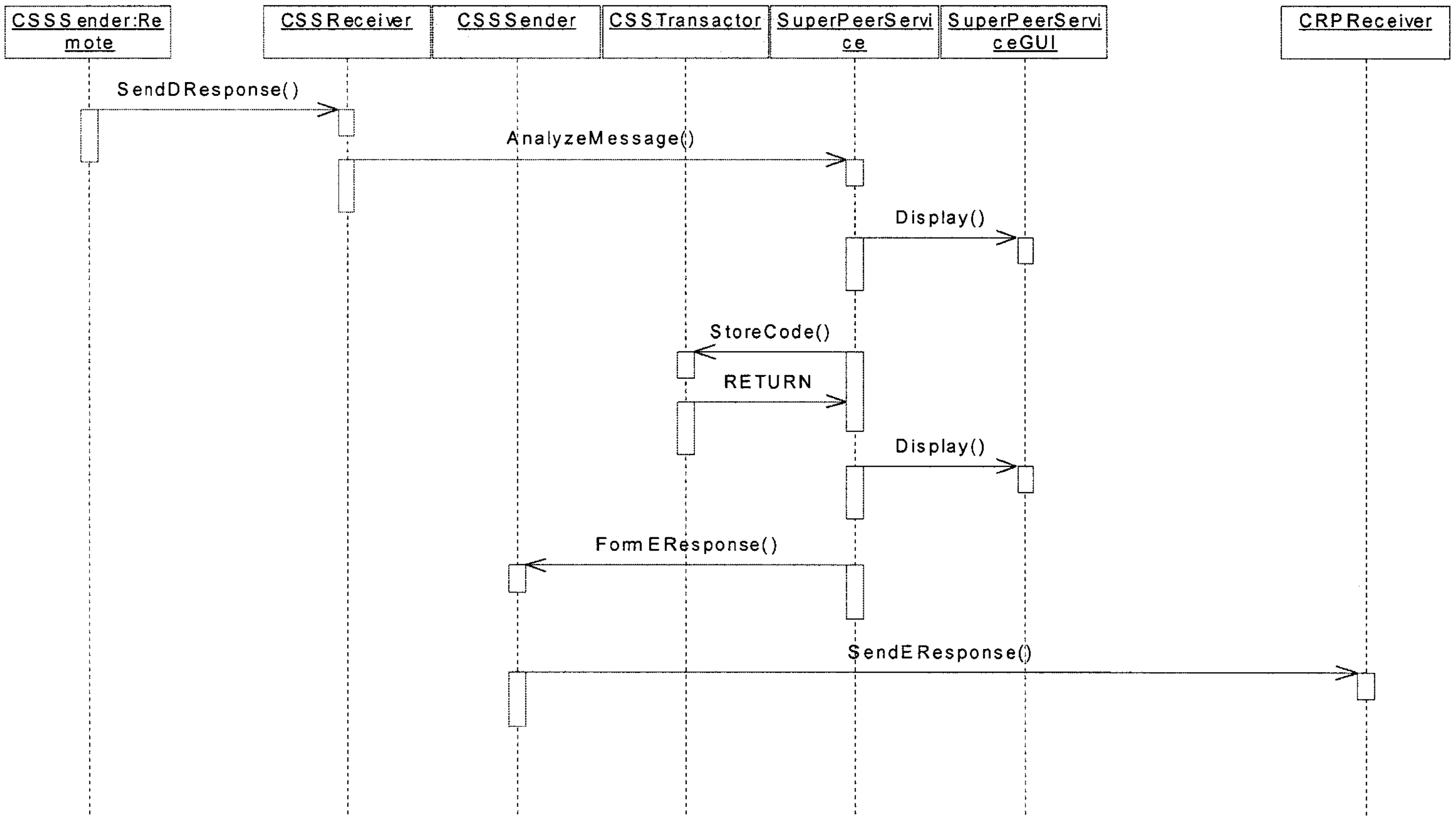


Figure 4-24: CSS Store Code Sequence Diagram

5. SYSTEM DEPLOYMENT AND INTERFACE

This chapter describes how to deploy our system and how to use the system with GUI interfaces. Some captured screens and examples are included to illustrate various operations applied on the system.

5.1 System Setup

Our system was installed and tested on two kinds of platforms, Microsoft Windows98/2000/XP^{TM} and Linux. Java Run Environment (JRE) is required to run all of three system roles: Code Request Peer, Code Service Peer and Code Service SuperPeer

5.1.1 Setup for the role of Code Request Peer

The setup for the role of CRP is straightforward. First get the package of executable program file "CRP.jar" and a text file "routinglist.txt", which stores the addresses of the local CSSs.

To run the program in Windows 98/2000^{TM} environment, simply double-click the CRP.jar file showed in Figure 5-1, it will run the program to make the computer as a CRP. User can search reusable software components from code resource community, browse all the searching result and retrieve the code components through GUI.

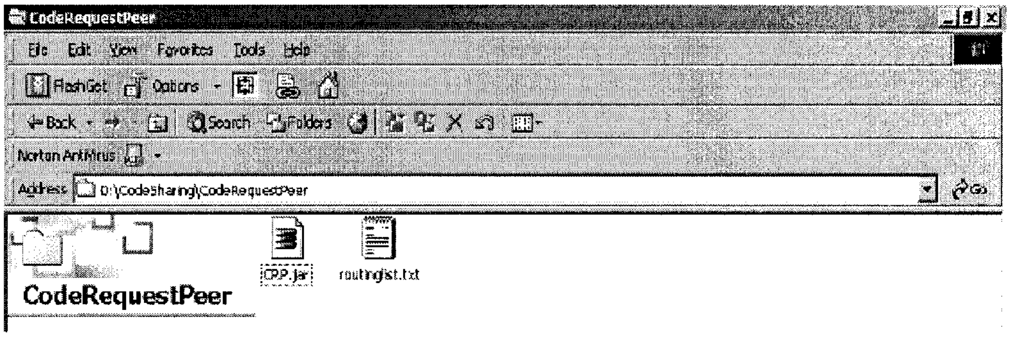


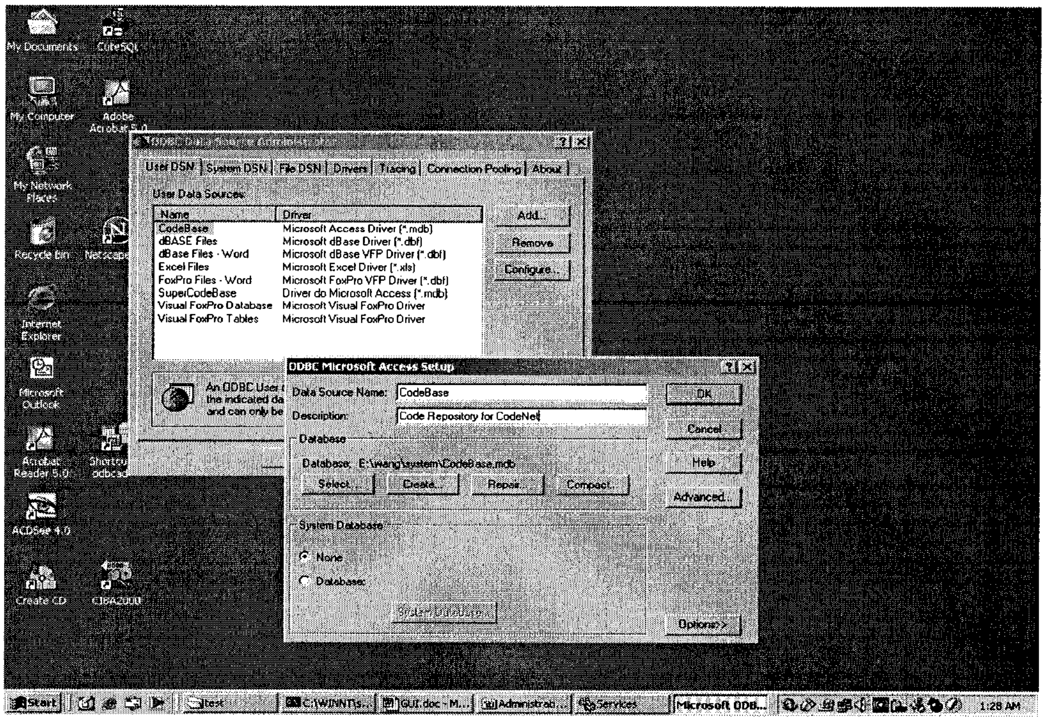
Figure 5-1: CRP Setup package in Windows98/2000

To run the program in Linux environment, simply open the terminal window, and type the following command: java –jar CRP.jar

5.1.2 Setup for the role of Code Service Peer

Unlike CRP, the setup for CSP requires not only JRE, but also Database Management System (DBMS) for code repository management. Two kinds of DBMS are supported by our system till now: Microsoft Access^{TM} and MySQL.

In Microsoft Windows 98/2000 environment with Microsoft Access^{TM}, first setup the database for code repository as showed in Figure 5-2, then double-click the package of executable program file "CSP.jar", it will run the program to make the computer as a CSP.



User Data Sources:

Name	Driver
CodeBase	Microsoft Access Driver (*.mdb)
dBASE Files	Microsoft dBase Driver (*.dbf)
dBase Files - Word	Microsoft dBase VFP Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
FoxPro Files - Word	Microsoft FoxPro VFP Driver (*.dbf)
SuperCodeBase	Driver do Microsoft Access (*.mdb)
Visual FoxPro Database	Microsoft Visual FoxPro Driver
Visual FoxPro Tables	Microsoft Visual FoxPro Driver

ODBC Microsoft Access Setup

Data Source Name: CodeBase

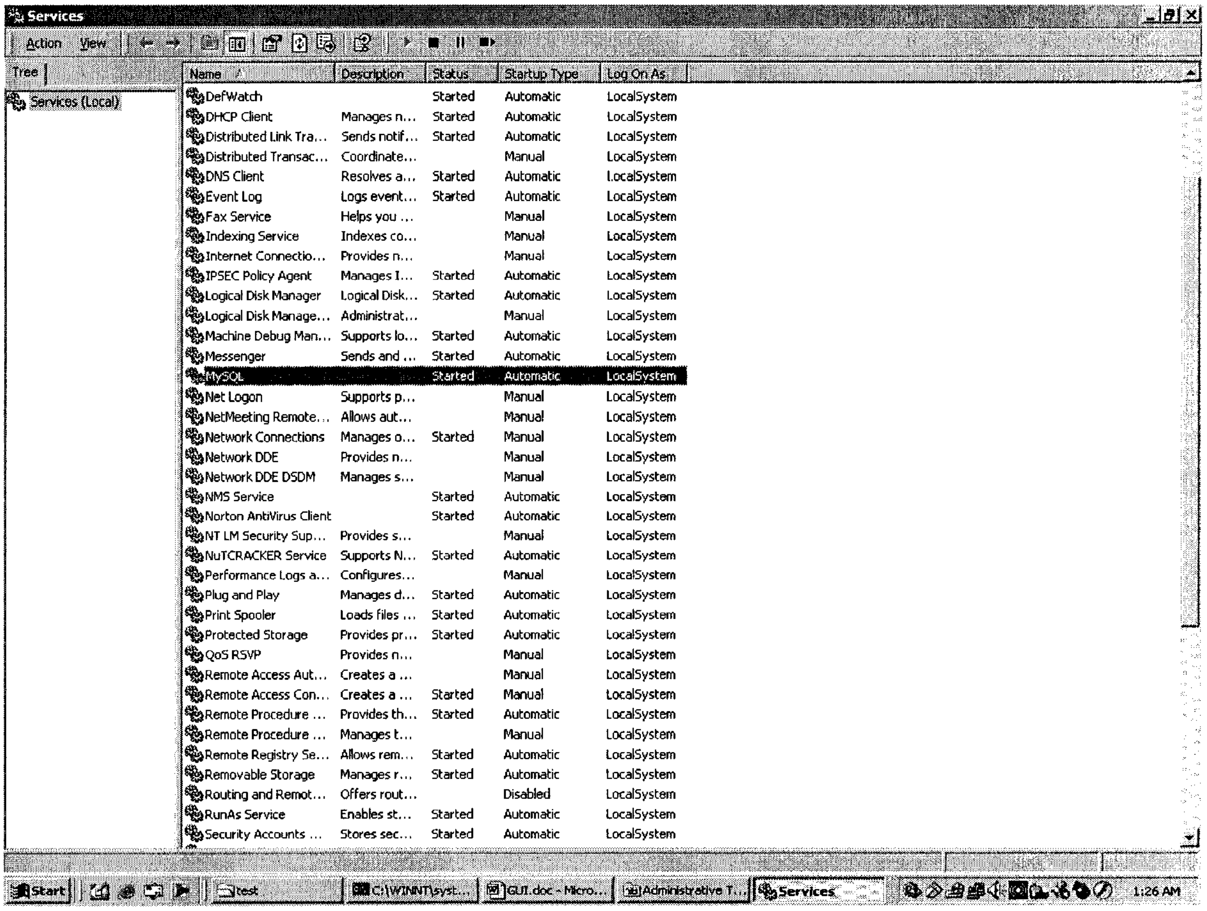
Description: Code Repository for CodeNet

Database: E:\wang\system\CodeBase.mdb

System Database: ● None ○ Database:

Figure 5-2: Microsoft Access^{TM} Database Setup in Windows

In Microsoft Windows 98/2000 environment with MySQL, startup the MySQL database service for code repository as showed in Figure 5-3, then double-click the package of executable program files "CSP.jar", it will run the program to make the computer as a CSP.



Name	Description	Status	Startup Type	Log On As
DefWatch		Started	Automatic	LocalSystem
DHCP Client	Manages n...	Started	Automatic	LocalSystem
Distributed Link Tra...	Sends notif...	Started	Automatic	LocalSystem
Distributed Transac...	Coordinate...		Manual	LocalSystem
DNS Client	Resolves a...	Started	Automatic	LocalSystem
Event Log	Logs event...	Started	Automatic	LocalSystem
Fax Service	Helps you ...		Manual	LocalSystem
Indexing Service	Indexes co...		Manual	LocalSystem
Internet Connectio...	Provides n...		Manual	LocalSystem
IPSEC Policy Agent	Manages I...	Started	Automatic	LocalSystem
Logical Disk Manager	Logical Disk...	Started	Automatic	LocalSystem
Logical Disk Manage...	Administrat...		Manual	LocalSystem
Machine Debug Man...	Supports lo...	Started	Automatic	LocalSystem
Messenger	Sends and ...	Started	Automatic	LocalSystem
MySQL		Started	Automatic	LocalSystem
Net Logon	Supports p...		Manual	LocalSystem
NetMeeting Remote...	Allows aut...		Manual	LocalSystem
Network Connections	Manages o...	Started	Manual	LocalSystem
Network DDE	Provides n...		Manual	LocalSystem
Network DDE DSDM	Manages s...		Manual	LocalSystem
NMS Service		Started	Automatic	LocalSystem
Norton AntiVirus Client		Started	Automatic	LocalSystem
NT LM Security Sup...	Provides s...		Manual	LocalSystem
NuTCRACKER Service	Supports N...	Started	Automatic	LocalSystem
Performance Logs a...	Configures...		Manual	LocalSystem
Plug and Play	Manages d...	Started	Automatic	LocalSystem
Print Spooler	Loads files ...	Started	Automatic	LocalSystem
Protected Storage	Provides pr...	Started	Automatic	LocalSystem
QoS RSVP	Provides n...		Manual	LocalSystem
Remote Access Aut...	Creates a ...		Manual	LocalSystem
Remote Access Con...	Creates a ...	Started	Manual	LocalSystem
Remote Procedure ...	Provides th...	Started	Automatic	LocalSystem
Remote Procedure ...	Manages t...		Manual	LocalSystem
Remote Registry Se...	Allows rem...	Started	Automatic	LocalSystem
Removable Storage	Manages r...	Started	Automatic	LocalSystem
Routing and Remot...	Offers rout...		Disabled	LocalSystem
RunAs Service	Enables st...	Started	Automatic	LocalSystem
Security Accounts ...	Stores sec...	Started	Automatic	LocalSystem

Figure 5-3: Startup MySQL Database Service in Windows

In Linux with MySQL, startup the service of database for code repository by opening a terminal window and type command “mysqld start”, then open the other terminal window and type the command: “java –jar CSP.jar” to run the program to make this Linux computer as a CSP.

5.1.3 Setup for the role of Code Service SuperPeer

Like CSP, the setup for CSS requires not only JRE, but also Database Management System (DBMS) for code repository management. Two kinds of DBMS are supported by system: Microsoft Access^{TM} and MySQL. Besides, the setup of CSS also needs a text file “routinglist.txt“, which stores the addresses of the remote CSSs.

In Microsoft Windows 98/2000 environment with Microsoft Access^{TM}, set up the database for code repository as showed in Figure 5-2, then double-click the package of executable program files "CSS.jar", it will run the program to establish the computer as a CSS.

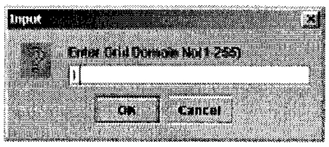
In Microsoft Windows 98/2000 environment with MySQL, start up the mySQL database service for code repository as showed in Figure 5-3, then double-click the package of executable program files "CSS.jar", it will run the program to establish the computer as a CSS.

In Linux with MySQL, initiate the service of database for code repositories by opening a terminal window and type command "mysqld start", then open the other terminal window and type the command: "java –jar CSS.jar" to run the program to establish this Linux computer as a CSS.

5.2 System Interface

5.2.1 Code Request Peer Interface

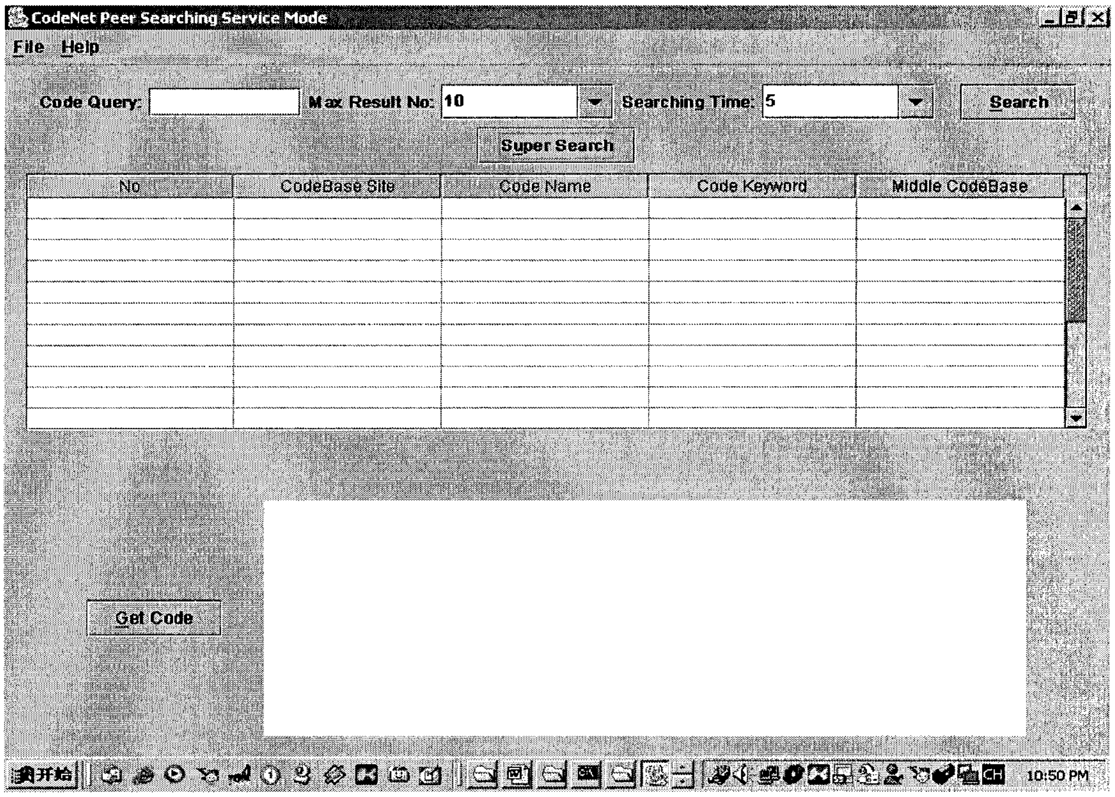
When a computer runs the program to act as a CRP node, first a dialog GUI will pop up asking which domain (multicast group) in this LAN the user wants this computer to join. As shown in Figure 5-4, the default value is group 1. After setting the domain, the system graphic user interface will appear as shown in Figure 5-5:



Enter Grid Domain No(1-255)

1

Figure 5-4: Domain Setting Dialog GUI



No	CodeBase Site	Code Name	Code Keyword	Middle CodeBase

Figure 5-5: GUI for Code Request peer

There are three text input fields for setting searching parameters by user, and two buttons for setting searching modes. These text input fields and control buttons are in the upper part of GUI, their functions are as follows:

1. Code Query: this text field is for setting the searching criteria. In this part the user can input several keywords to describe the required reusable code component.

2. Max Result No: this text field is for setting the number of displayed search results. After CRP sends out a search request, it may receive a lot of responses. The Max Result No parameter will help the program to display only a limited number of the first received results; usually, those results are returned from the fastest or nearest CSPs or CSSs.

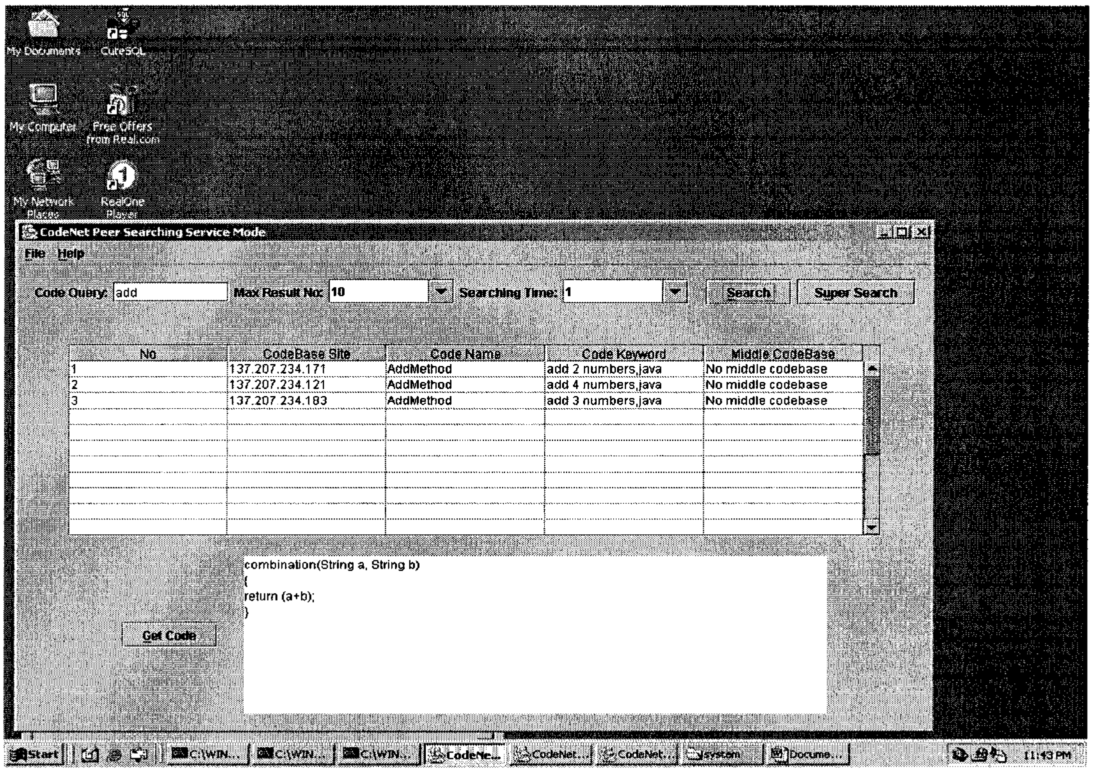
3. Searching Time: this text field is for setting the parameter that restricts the valid time of request.

4. Search Button: it is a trigger to set program working in multicast mode.

5. Super Search Button: it is a trigger to set program working in routing mode.

In the middle of the GUI there is a table for displaying the search results. As shown in an example GUI for multicasting mode in Figure 5-6, three results, which are returned from different code repositories in local domain, are displayed in the table. In this table, "No" column shows the sequence number of the returned results. "CodeBase Site" column shows the IP address of CSP or CSS from which the result is returned. "Code Name" column shows the names of returned results. "Code Keyword" column displays the detailed description on the returned results. "Middle CodeBase" shows the routing path that the search request has taken before getting a result. As shown in an example GUI for routing mode in Figure 5-7, the "Middle CodeBase" of a result is "137.207.234.171-137.207.234.121 -137.207. 234. 183", that means: the search request first reaches the computer whose IP is 137.207.231.171, since no suitable result is found in its code repository, the request is forwarded to the second computer whose IP is 137.207.234.121, and no suitable one is found too. At last the request reaches the third machine with IP of 137.207.234.183, where the result is found and returned.

As the results are displayed in the table, the user can use mouse or keyboard to choose one from those results (highlight a row in the table), and press the "Get Code" button that is laid on the left side bottom of GUI. Then, the detailed well-formed source code will be displayed in the text area, which is beside the "Get Code" button.

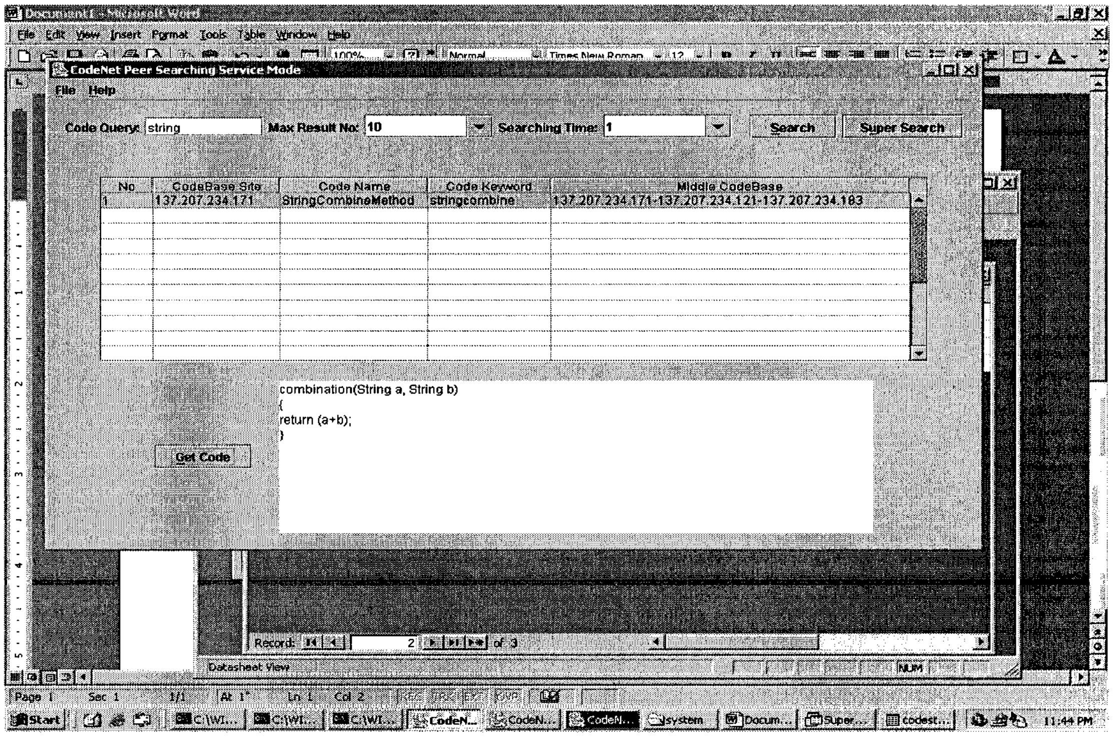


Code Query: add
Max Result No: 10
Searching Time: 1

No	CodeBase Site	Code Name	Code Keyword	Middle CodeBase
1	137.207.234.171	AddMethod	add 2 numbers,java	No middle codebase
2	137.207.234.121	AddMethod	add 4 numbers,java	No middle codebase
3	137.207.234.183	AddMethod	add 3 numbers,java	No middle codebase

```
combination(String a, String b)
{
return (a+b);
}
```

Figure 5-6: GUI Example of Code Request Peer in multicasting mode



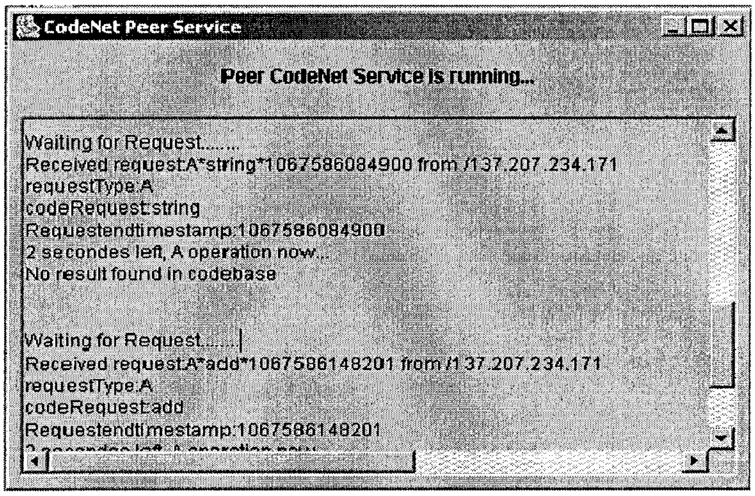
Code Query: string
Max Result No: 10
Searching Time: 1

No	CodeBase Site	Code Name	Code Keyword	Middle CodeBase
1	137.207.234.171	StringCombineMethod	stringcombine	137.207.234.171-137.207.234.121-137.207.234.183

```
combination(String a, String b)
{
return (a+b);
}
```

Figure 5-7: GUI Example of Code Request Peer in routing mode

5.2.2 Code Service Peer Interface



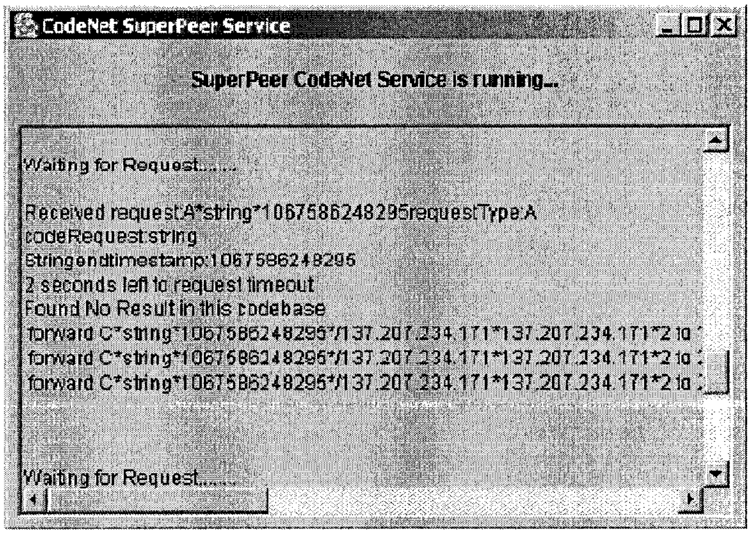
```
Waiting for Request.......
Received request:A*string*1067586084900 from /137.207.234.171
requestType:A
codeRequest:string
Requestendtimestamp:1067586084900
2 secondes left, A operation now...
No result found in codebase


Waiting for Request.......
Received request:A*add*1067586148201 from /137.207.234.171
requestType:A
codeRequest:add
Requestendtimestamp:1067586148201
```

Figure 5-8: GUI of Code Service Peer

The GUI of Code Service Peer is shown in Figure 5-8. Usually, CSP is in listening status. When receiving a request on the port, CSP will display the received message immediately on the GUI. When CSP takes transactions, the transaction message will also be displayed on the GUI window.

5.2.3 Code Service SuperPeer Interface



CodeNet SuperPeer Service
SuperPeer CodeNet Service is running...

```
Waiting for Request.......

Received request:A*string*1067586248295requestType:A
codeRequest:string
Stringendtimestamp:1067586248295
2 seconds left to request timeout
Found No Result in this codebase
 forward C*string*1067586248295*/137.207.234.171*137.207.234.171*2 to
 forward C*string*1067586248295*/137.207.234.171*137.207.234.171*2 to
 forward C*string*1067586248295*/137.207.234.171*137.207.234.171*2 to



Waiting for Request.......
```

Figure 5-9: GUI of Code Service SuperPeer

The GUI of Code Service SuperPeer is shown in Figure 5-9. Usually CSS is in listening status. When receiving request on the port, CSS will display the received message immediately on the GUI. When CSS takes transactions, the transaction message will also be displayed on the GUI window.

6. SYSTEM EVALUATION

In this Chapter we present the evaluation of our prototype system.

6.1 Interoperability

Our system is a java-based application, so it can run on a wide variety of platforms with JRE to guarantee interoperability. Our testing on interoperability is performed using three scenarios: pure Microsoft Window^{TM} platform environment; pure Linux platform environment; and, a hybrid environment.

6.1.1 Simulation in the Pure Windows Environment

To test our prototype system in windows environment, we used 4 computers to deploy system: one computer with Window 2000^{TM} and Microsoft Access^{TM} ran as a CSP, one computer with Window XP^{TM} and MySQL ran as both CSP and CSS, one computer with Window 2000^{TM} ran as a CRP, and one computer with Window 98^{TM} and Microsoft Access^{TM} ran as a remote CSS. The previous three computers were connected with our University LAN, the last one was connected with Internet through cable connection from Cogeco CableTV company. We tested our system on both multicasting mode within our university LAN and routing mode with one local CSS in our LAN & one remote CSS from Internet. On both these two scenarios, the operations including code searching and retrieving were performed successfully.

6.1.2 Simulation in the Pure Linux Environment

To test our prototype system in Linux environment, we use 3 computers to deploy system: one computer with Red Hat Linux (version 7.0) ran as a CRP, one computer with MEPIS Linux (Version 2003.0) and MySQL ran as both CSP and CSS and one computer with Red Hat Linux (version 7.0) and MySQL as a remote CSS. The previous two computers are connected with our University LAN, the last one is connected with Internet through cable connection from Cogeco CableTV company. We tested our system on both multicasting mode within our university LAN and routing mode with one local CSS in our LAN & one remote CSS from Internet. On both these two scenarios, the operations including code searching and retrieving were performed successfully.

6.1.3 Simulation in the Hybrid Environment

To test our prototype system in the hybrid environment, we used 5 computers to deploy system: one computer with Window 2000^{TM} and Microsoft Access^{TM} ran as a CSP, one computer with MEPIS Linux (Version 2003.0) and MySQL ran as a CSP, one computer with Window XP^{TM} and MySQL ran as both CSP and CSS, one Unix workstation ran as CRP and one computer with Window 98^{TM} and Microsoft Access^{TM} ran as a remote CSS. The previous four computers were connected with our University LAN, the last one was connected with Internet through cable connection from Cogeco company. We test our system on both multicasting mode within our university LAN and on routing mode with one local CSS in our LAN & one remote CSS from Internet. On both these two scenarios, the operations including code searching and retrieving were performed successfully.

6.2 Performance

Our system is proposed to replace the traditional C/S application on code resource sharing in CodeNet project, so we tested the performance of our system and compare the experiments results with those of the C/S model system. We simulated the performance testing on our system in LAN environment with 2 nodes, 4 nodes and 6 nodes. The performance is evaluated by the time that it takes for a code requestor to receive response after sending out the request.

We test the system performance with different aggregation capability of code repositories. There are three assumptions about our testing method:

1. All the code components, on average, are stored in the various code repositories and no code redundancy exists.

2. The code components are stored in databases randomly without any index.
3. The evaluation is based on the worst searching case. That is, the required code component is stored as the last record in database.

We test system performance on two scenarios: Microsoft Windows 2000™ platform with Microsoft Access™ and Linux platform with MySQL.

Table 6-1 shows the testing result table of client/server system and our system running on Microsoft Windows 2000™ platform with Microsoft Access™. Figure 6-1 shows a comparison of the performance of our model with C/S model in Microsoft Windows environment. In our testing, the computers that run as nodes have the same hardware specification as: CPU-- Pentium III with 2.8GHz; Memory—512M; Bandwidth of the network-- 100Mbps.

Resource Capability (unit: records)	Client/Server Model (unit: ms)	Our Model (unit: ms)		
		2 Nodes	4 Nodes	6 Nodes
10	95	95	95	95
100	104	105	104	104
1000	115	114	113	113
10000	210	170	150	120
20000	340	210	170	160
30000	465	290	200	170
40000	600	340	210	185
50000	730	420	280	200
60000	851	465	290	210
70000	980	551	310	230
80000	1100	600	340	280
90000	1240	680	390	291
100000	1362	730	420	300

Table 6-1: Testing Result Table for performance on Windows2000 with Access™

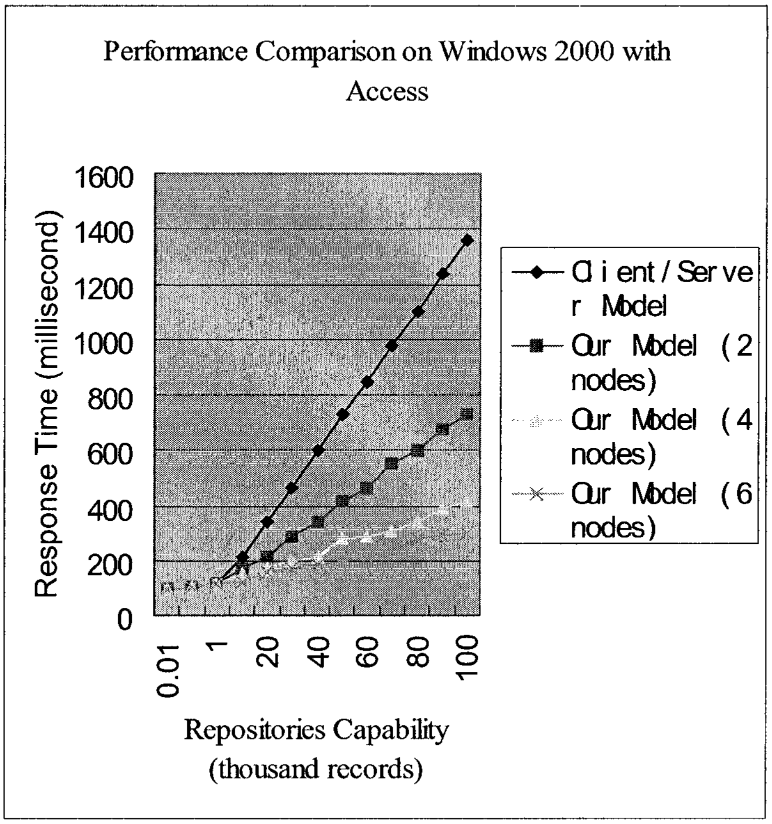


Figure 6-1: Chart for Performance Comparison on Windows2000 with Access^{TM}

Table 6-2 shows the testing result table of client/server system and our system running on MEPIS Linux (Version 2003.0) platform with MySQL(version 4.0). Figure 6-2 shows the chart to compare the performance of our model with C/S model in Linux environment. In our testing, the computers that run as nodes have the same hardware specification as: CPU - Pentium III with 2.8GHz; Memory - 512M; Bandwidth of the network - 100Mbps.

From the result tables and charts we can draw the following conclusion:

If aggregation capability of code repositories is below the level (nearly 1000 records in our condition), there is not obvious performance difference between C/S model and our model. But if aggregation capability is over the level, the larger the aggregation capability is, the higher performance our system achieves over C/S model; and the more nodes there exist in community, the higher performance our system achieves over C/S model.

Resource Capability (unit: records)	Client/Server Model (unit: ms)	Our Model (unit: ms)		
		2 Nodes	4 Nodes	6 Nodes
10	25	25	25	25
100	25	25	25	25
1000	40	32	30	27
10000	130	80	70	66
20000	240	130	80	75
30000	330	170	110	80
40000	430	240	130	95
50000	520	260	150	105
60000	610	330	170	128
70000	701	400	230	135
80000	800	430	240	155
90000	891	490	250	170
100000	992	520	260	200

Table 6-2: Testing Result Table for performance on Linux with MySQL

It is reasonable to anticipate that the time for any single peer node to issue a request and receive a response will have constant time, O(1), complexity using multicast technique within a single domain. However, as the number of request-responses increases to N nodes, issued simultaneously, the time complexity will be O(N). This basic behaviour is verified by inspection of the graphs in Figures 6-1 and 6-2, where the profiles are essentially linear. The deviations from linearity arise, for the most part, from the fact that non-processable incoming requests are currently dropped in our prototype system, and that requests that must be forwarded to a neighbouring domain have non-deterministic response times. Since the number of non-local responses is typically small this effect produces the fluctuations from strict linearity in the figures.

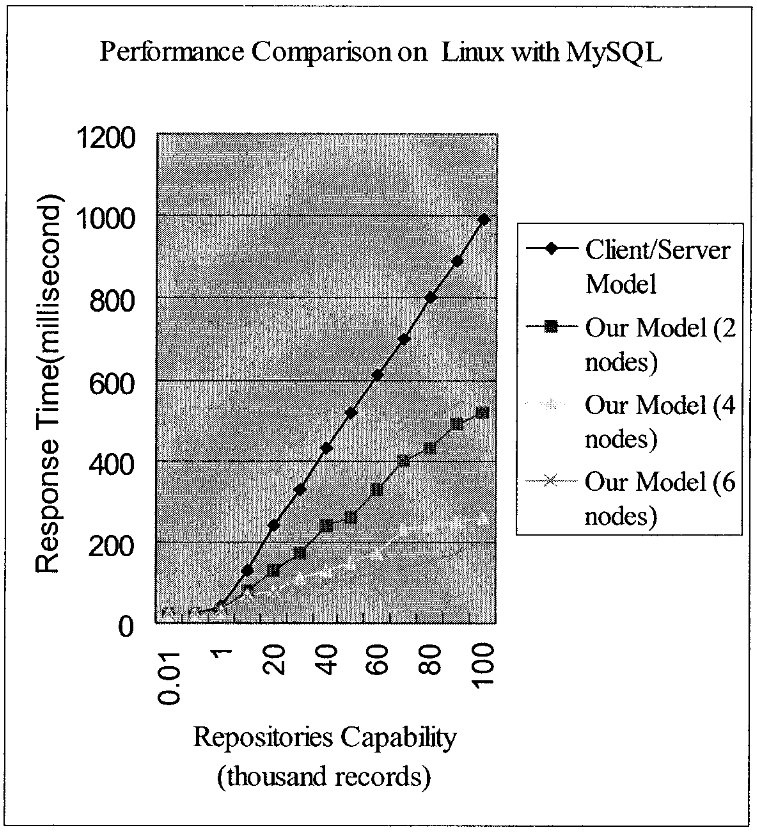


Figure 6-2: Chart for Performance Comparison on Linux with MySQL

6.3 Scalability

Our model uses a decentralized architecture, so the scalability of resource community is not limited by factors such as the amount of operations performed on central components. Besides, our system provides support for low-end computers. This feature makes it possible for a lot of Grid network devices with CPU and some storage space to join resource community as a source code provider. Comparing with high hardware and software requirements for computer as server in C/S model, our solution is more scalable. In our model, the organization of system nodes into groups also improves the source community scalability because new resource nodes can join the code resource community as members in new group without taking too much communication bandwidth in Grid networks.

6.4 Implementability

As stated in Chapter 5, our system is very straightforward to deploy. Any computer with JRE can run our system working as a code requestor. Any computer with JRE and database support can work as a code resource provider. For default system setting, the communication happens on port 5000 for CRP, 6000 for CSP, and 7000 for CSS. If a Grid node does not forbid those ports, it can run our system as any role or all 3 roles at the same time. The administration work needed in our model is to maintain the routing list file on CRP and CSS to make sure the address information in the file is correct so that request can be routed to the right local or remote CSSs. After a system node starts up running the program, it joins the Grid code resource community and cooperates with other node with high autonomy without much administration or human intervention.

6.5 Limitations

Based on the observations of the system running, some limitations of our prototype system are reported in the following.

(1) In our model, when a CSP or a CSS is in process of transacting one request, its communication port is closed. So if a request message is coming at this specific period, it may be ignored.

(2) The communication between system nodes is implemented on port. If the firewall of one Grid domain blocks the specific CSS communication port, the code resources located in this domain will not be accessible by CSSs in other remote domains.

(3) In traditional client/server code resource sharing, all the code resource are stored and managed in a centralized database server, so there does not exist redundancy. In our model, the code resource are dispersed in multiples code repositories

without centralized management, so redundancy may exist. That may potentially take up additional storing spaces in code repositories.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this paper we have proposed a decentralized code (generalized content data) resource sharing model, and investigated the problem of dispersed code repositories sharing. In our prototype system, based on results obtained from an objective suite of evaluative tests, we achieve greater interoperability and higher performance with a decentralized infrastructure than in client/server code sharing applications. We assert that this new model can replace the conventional client/server model for Grid code resource sharing in the future.

7.1.1 Innovation

The most innovative feature presented in our model is importing P2P technology and methodology into code and generalized content sharing mechanisms in Grid environment, thus solving several aspects of the problem of dispersed code repositories sharing, for which the conventional client/server model could not provide efficient support.

Different from other P2P applications that adopt only one discovery approach, our software system uses two discovery approaches on both two scenarios: searching within local domain and searching across remote domains. Thus, we are able to overcome the scalability limits to make our model suitable for Grid environment.

7.1.2 Achievements

We propose a solution for code resource sharing in Grid environment. We design a new model that extends existing source sharing protocols, importing P2P technology into code reuse application area in Grid context. The new model inherits the advantages of P2P systems in dynamism, ease of use, flexibility and decentralization.

Moreover, the new mechanism overcomes the disadvantages of existing protocols and supports large-scale distributed code resource among multiple Grid domains. Besides, the system is platform independent and can be flexible deployed with less administration comparing with client/server model. In Table 7-1, we compare the characteristics of our proposed model with the previous client/server model applying on code sharing area.

Feature	Conventional Client/Server Model	Our Proposed Model
Decentralization	Medium	High
Scalability	Medium	High
Interoperability	Medium	High
Dynamism	Low	High
Cost of ownership	High	Low
Autonomy	Low	Medium
Targeted environment	LAN&WAN	Grid, LAN&WAN
Performance	Medium	Individual Low, Aggregate High

Table 7-1: Comparison of the C/S model and proposed model

7.2 Suggestions For Future Research

During the process of conducting this research program many additional problems and issues arose that provide possible avenues for further investigation. Based on the achievements of this thesis, the following potential future research directions are recommended:

- Further research the enhancement of system security. The open and anonymous nature of P2P networks leads to a lack of security in our prototype system. Some Grid security mechanisms such as authentication and authorization should be imported into our system as future work.

- Design and establish powerful database capabilities and functionalities to improve sharing and secure access to the code repositories.

- Further explore code component searching and retrieval based on natural language specification. In our system, we use keyword as searching criteria, which is not a very efficient way for code components searching and retrieval. The importance of natural language support arises due to the lack of knowledge of actual keywords required to establish database entry matching during query; providing further support through the use of synonyms and ontologies would greatly improve on the query capability, while extending the domain of this research into the area of Semantic Web.

- Apply a theoretical testing analysis and additional refinements of our prototype system.

- Thoroughly test total system performance under heavy loads, in which either the P2P model is deployed on a large Grid test bed or through simulation, varying both the number of peers and the number of requests according to a range of statistical distributions.

- Further explore automated search and retrieval of binary executables and web services to enable "intelligent" deployment of code resources onto the Grid framework, much as DLL's and other utilities are used within workstation and server computing environments. This would extend the notion of compile and link in static workstation environments to its extended, dynamic counterpart within grids.

REFERENCES

1. [Arnold99] Ken Arnold, "The Jini architecture: dynamic services in a flexible network", *Proceedings of the 36th ACM/IEEE Conference on Design Automation*, June 1999

2. [Atsumi02] Noritoshi Atsumi, Shoji Yuen, Kiyoshi Agusa, Shinichirou Yamamoto, "Middleware issues: Library evolution for reliable software", *Proceedings of the international workshop on Principles of software evolution*, May 2002

3. [Ander02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, Dan Werthimer, "SETI@home: an experiment in public-resource computing". *Communications of the ACM, Volume 45, Issue 11*,November 2002

4. [Baker01] Baker, M.; Smith, G., "Jini meets the Grid", *Proceedings of the 2001 IEEE Conference on Parallel Processing*, 2001

5. [Barkai01] Barkai, D. "Technologies for sharing and collaborating on the Net", *Proceedings of the First International Conference on Peer-to-Peer Computing*, Aug 2001

6. [Basili97] Victor R. Basili, Steven E. Condon, Khaled El Emam, Robert B. Hendrick, Walcelio Melo, "Characterizing and modeling the cost of rework in a library of reusable software components", *Proceedings of the 19th International Conference on Software Engineering*, May 1997

7. [Bathja01] J. Batheja and M.Parashar. "Framework for Opportunistic Cluster Computing using JavaSpaces". *The 9th international Conference on High Performance Computing and Networking (HPCN2001), Workshop on Java in High Performance Computing, Amsterdam,* Feb 2001

8. [Bolosky00] Bolosky,W., Douceur, J., Ely, D., and Theimer, M, "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs", *Proceedings of SIGMETRICS, Santa Clara, CA, USA*, Jun 2000.

9. [Casanova02] Henri Casanova, "Distributed computing research issues in grid computing", *ACM SIGACT News, Volume 33 Issue 3*, Sep 2002

10. [Coulou94] George Coulouris, Jean Dollimore, and Tim Kindberg, "Distributed Systems - Concepts and Design", *International Computer Science Series, Addison-Wesley Longman, Inc., 2nd edition,* 1994.

11. [Czajk97] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S.

Tuecke, "A resource management architecture for metacomputing systems", *Technical report of Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill.*, 1997.

12. [Czajk01] Czajkowski, K., et al., " Grid Information Services for Distributed Resource Sharing", *IEEE International Symposium on High Performance Distributed Computing,* 2001.

13. [Damiani96] E. Damiani, M. G. Fugini. "Design and code reuse based on fuzzy classification of components", *ACM SIGAPP Applied Computing Review, Volume 4 Issue 2.* September 1996

14. [Fitzg97] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. "A directory service for configuring high-performance distributed computations". *Proceedings of 6th IEEE Symp on High Performance Distributed Computing, pages 365-375. IEEE Computer Society Press*, 1997.

15. [Foster97] I. Foster, C. Kesselman. "Globus: A Metacomputing Infrastructure Toolkit". *Intl J. Supercomputer Applications, 11(2):115-128,* 1997.

16. [Foster98a] Foster, I., Keselman, C., Tsudik, G. and Tuecke, S, "A Security Architecture for Computational Grids", *Proceedings 5th ACM Conference on Computer and Communication Security, pp 83-91,* Nov.1998

17. [Foster98b] I. Foster, C. Kesselman. "The Globus Project: A Status Report", *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, pp. 4-18,* 1998.

18. [Foster99] Foster, I., Kesselman, C. "The Grid: Blueprint for a New Computing Infrastructure", *Morgan Kaufman Publishers, Inc, San Francisco, California,* 1999.

19. [Foster01] Foster, I., C. Kesselman, S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International J. Supercomputer Applications, 15(3),* 2001.

20. [Foster02a] Foster, I, "What is the Grid? A Three Point Checklist", *GRIDToday,* http://www.gridtoday.com/02/0722/100136.html, July 20, 2002.

21. [Foster02b] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", *Open Grid Service Infrastructure WG, Global Grid Forum,* June 22, 2002.

22. [Foster02c] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "Grid Services for Distributed System Integration". *Computer, 35(6),* 2002.

23. [Foster02d] Ian Foster, “The Grid: A new infrastructure for 21st century science”, *Physics Today*, February 2002

24. [Frakes94a] W.B. Frakes and S. Isoda. “Success Factors of Systematic Reuse”, *IEEE Software, Vol.11, No.5, pp.15-19,* Sept 1994.

25. [Frakes94b] Frakes, W. B. and Pole, T. “An empirical study of representation methods for reusable software components”, *IEEE Transactions on Software Engineering, Vol. 20 , No.8, pp.617-630,* Aug. 1994

26. [Freenet] The FreeNet home page, http://www.freenetproject.org

27. [Furmento02] Nathalie Furmento, William Lee, Anthony Mayer, Steven Newhouse, John Darlington; “ICENI: an open grid service architecture implemented with Jini”, *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, November 2002

28. [GGF] Global Grid Forum. http://www.gridforum.org/.

29. [Globus] Globus Project. http://www.globus.org.

30. [Gnutella] The Gnutella home page, http://www.gnutella.co.uk/

31. [Gribble01] Gribble, S., Halevy, A., Ives, Z., Rodrig, M., and Suciu, D. “What Can Peer-to-Peer Do for Databases and Vice Versa?”, *Proceedings of the WebDB: Workshop on Databases and the Web, Santa Barbara, CA, USA*, 2001

32. [Grinter01] Rebecca E. Grinter, “Session 4: From local to global coordination: lessons from software reuse”, *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work,* September 2001

33. [Groove] The Groove home page, http://www.groove.net/

34. [Henninger97] Scott Henninger, “An evolutionary approach to constructing effective software reuse repositories”, *ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 6 Issue 2,* April 1997

35. [Howes95] T. Howes and M. Smith. “The LDAP application program interface”. *RFC 1823,* August 1995.

36. [Jacobson97] Jacobson, I., Griss, M. and Jonsson, P, “Software Reuse: Architecture, Process and Organization for Business Success”, *ACM Press, New York, NY,* 1997.

37. [Jini] Sun Microsystems, Jini Specification. http://www.sun.com/software/jini/jini_technology.html

38. [JXTA] S. Microsystems. Project JuXTApose home page, http://www.jxta.org,

39. [Krueger92] Charles W. Krueger, "Software reuse", *ACM Computing Surveys (CSUR), Volume 24 Issue 2.* June 1992

40. [Kubiatowicz00] Kubiatowicz, J., Bindel, D., Chen,Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, R.,Weatherspoon, H.,Weimer, W.,Wells, C., and Zhao, B. "OceanStore: An Architecture for Global-Scale Persistent Storage", *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000),* November 2000.

41. [Mewes00] Mewes HW, Frishman D, Gruber C, Geier B, Haase D, Kaps A, Lemcke K, Mannhaupt G, Pfeiffer F, Schuller C, et al., "MIPS: a database for genomes and protein sequences", *Nucleic Acids Res. 28(1), pp.37-40,* January 2000

42. [Milo02] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja1, Jim Pruyne, Bruno Richard, Sami Rollins, Zhichen Xu. "Peer-to-Peer Computing". *HP Laboratories Palo Alto*, March 8th, 2002

43. [Napster] The Napster home page, http://www.napster.com

44. [Opencola] The OPENCOLA home page, http://www.opencola.com

45. [OpenNap] penNap: Open Source Napster Server, http://opennap.sourceforge.net/

46. [P2pwg01] Peer-to-peer working group, "Bidirectional Peer-to-Peer Communication with Interposing Firewalls and NATs*", P2pwg White Paper, Revision 0.091. http://www.p2pwg.org.* May 23, 2001

47. [Ratnasamy01] Ratnasamy, S., Francis, P., Handley, M., Karp R., Shenker, S. "A Scalable Content-Addressable Network", *Proceedings of the SIGCOMM, pp. 161-172,* 2001

48. [Shirky01] C. Shirky, "What is P2P ... and what isn't*", an article published on O'Reilly Network. http://www.openp2p.com/pub/q/all_p2p_articles,* 2001

49. [Strom01] Strom, D. "Businesses Embrace Instant Messaging", *http://enterprise.cnet.com/enterprise/0-9534-7-4403317.html.* January 2001.

50. [Suzumura01] Toyotaro Suzumura, Satoshi Matsuoka, Hidemoto Nakada, "A Jini-based computing portal system", *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, November 2001

51. [Tracz95] Tracz, W., "Confessions of a Used Program Salesman: Institutionalizing Software Reuse", *Addison-Wesley, Reading, MA,* 1995

52. [Tuecke03] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling, "Open Grid Services Infrastructure (OGSI) Version 1.0.", *Global Grid Forum Draft Recommendation, 6/27/2003,* 2003

53. [Vahdat98] Vahdat, A., Belani, E., Eastham, P., Yoshikawa, C., Anderson, T., Culler, D. and Dahlin, M, "WebOS: Operating System Services For Wide Area Applications", *7th Symposium on High Performance Distributed Computing,* July 1998.

54. [Waldo99] Jim Waldo, "The Jini architecture for network-centric computing", *Communications of the ACM, Volume 42 Issue 7,* July 1999

55. [Waterhouse02] Waterhouse, S., Doolin, D.M., Kan G., Faybishenko, Y. "Distributed Search in P2P Networks", *IEEE Internet Computing 6(1), pp.68-72. January-February*, 2002.

56. [Wilson02] B. J. Wilson, "JXTA", *New Riders Publishing* 2002.

57. [Wollrath97] A. Wollrath, J. Waldo, and R. Riggs, "Java-Centric Distributed Computing", *IEEE Micro, Vol. 17, No. 3, pp. 44-53.* May/June 1997

58. [Yu01] XiaoHong Yu. "SpiderNet—A Multi-Server Code Service Model Design for Computational Grid Support", *Master Thesis, University of Windsor*, 2000

59. [Zhang00] Michael Hui Zhang. "Design and construction of a library-based software reuse model to support distrbuted and grid computing", *Master Thesis, University of Windsor,* 2000

60. [Zhao02] Xiaoquan Zhao, "A Centralized Object-Relational Database Based Code Service Retrieval System Tool for Software Reuse", *Master Thesis, University of Windsor,* 2000

61. [Zhong00] Sheng Zhong. "Software Library for Reuse-Oriented Program Development", *Master Thesis, University of Windsor*, 2000

VITA AUCTORIS

Fan WANG was born in P.R.China, 1975. He obtained the Bachelor's degree in Information System from ZhongNan University of Finance & Economics, WuHan, PRC in 1997. He is currently a candidate for Master's degree in Computer Science at the University of Windsor and hopes to graduate in Winter 2004.