2001

Design and construction of a library-based software reuse model to support distributed and grid computing.

Michael Hui. Zhang
University of Windsor

DESIGN AND CONSTRUCTION OF A LIBRARY-BASED SOFTWARE REUSE MODEL TO SUPPORT DISTRIBUTED AND GRID COMPUTING

by

Michael Hui Zhang

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
2000

0-612-62307-6

[illegible]

Abstract

In this thesis, we report on the design and construction of a distributed, collaborative, object-oriented, library-based modeling system (CodeNet) to support reusing and adapting components. By understanding and resolving the issues involved in the design and implementation process of CodeNet, this work contributes to reuse-oriented software design and development in the context of distributed computing.

Dedication

This paper is dedicated to my parents, my brother and his wife, my girlfriend, my teachers, and my friends, for their love, patience and support.

Acknowledgements

I would like to acknowledge the support and guidance provided by Dr. Robert D. Kent, whose time, dedication and effort has contributed in guiding me through this thesis. He also significantly contributed to the quality and consistency of this thesis by thorough reviews and by providing excellent feedback.

Special thanks to my parents and my brother's family for their encouragement and support and my girlfriend for her love and helping.

Last but not least, I would like to thank all my colleagues in the grad lab for providing a friendly atmosphere through my Master program.

Table of Contents

List of Figures

CHAPTER 1. Introduction

The power of new generation computers and, the same time their cheapness has produced an increasing demand for software systems of higher and higher complexity. While the field of programming had made tremendous progress, the development of a large software system involves a number of difficulties in both technical and non-technical aspects [SOM89], such as: the problem being solved were not well understood by all the people involved in the project; people had to spend a lot of time communicating with each other rather than writing the code; people leaving the project affectedly the work of other people, replacing an individual required an extensive amount of training about the project requirements and system design. Many solutions were proposed and tried, but the common consensus was to view the final software system as a complex product and the building of it as an engineering job.

A way to improve the understandability and maintainability of a legacy system is modularization. Modularizing an existing system consists of replacing a simple large program or module with a functionally equivalent collection of smaller modules [CAN95]. Modularization is also useful for downsizing large applications from mainframes to distributed client/server platforms [SNE94]. Indeed, a changing hardware platform is becoming a vital importance question for the economy and the competitiveness of many companies. Therefore, software systems developed for the old platform have to be available on the new one. In many cases reusing the existing systems and adapting them to the new platform is cost-effective and can be preferable to new development [SNE95]. Another reason for modularizing a system is the possibility to reuse its single modules in the development of new software systems.

Software reuse has long been recognizes as a way to improve both the productivity and the quality of new software projects [BIG89, FRAKES94, LIM94]. The reuse of software components that have been already tested may reduce the costs of software development and increase software productivity. Moreover, reuse-oriented software development can reduce the maintenance cost [BAS90], because maintenance operations on a modularized system can be better localized. On the other side, reuse is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation in the new form.

The concept of "software reuse" is not new. Software should be massed produced and stored into repositories in order to be used to compose more complex components and systems during software development. One of the most promising ways to make the population of a repository of reusable assets cost effective and to obtain useful results in a short time is by extracting and reusing them from the existing software [ARN92, CAL91, CAN94, DUNN93, HALL, PRI91]. Existing systems record in various forms (requirements or design documents, code, test cases, and user manuals) a large amount of knowledge and expertise. Therefore, they can become the main source of reusable components.

In our system (model), the term reuse will refer to the process of obtaining reusable assets from the class-based library. Design and reuse processes involve the following activities:

1. Identify the system components and their relationships;
2. Understanding their meaning and producing the related specifications;
3. Accessing the class library to identify reuse candidate components;
4. Inserting, updating and packaging the components (classes) in the library independently;
5. Retrieval from the library by the natural language and generate the object-oriented application code for the end-users.

1.1. Motivation

Software reuse has been recognized as one of the realistic and promising ways to improve software productivity, quality and reliability, reduce maintenance costs and shorten time-to-market [MCC99, JAC97]. Two of the challenging problems in engineering an application from reusable parts are: (a) software component retrieval and (b) reuse by composition and generation. A third challenge is adaptation of reusable components to fit the needs of a new application system. However, at present much of the software reuse activity has only been oriented toward the reuse of parts -- especially function -- stored in reuse libraries, and most of the reuse systems were built in stand-alone environments.

The latest technologies are bringing a remarkable impact to software reuse. The most significant change is raised by the object and network technology.

Object technology offers significant benefits in the construction of individual programs, but the real power of object-based information systems is generally easier to modify and maintain which can rapidly adjust to changing circumstances [MCC96]. By inheritance and encapsulating methods with data structures, an object-based library can execute complex analysis and data manipulation operations to search and transform multimedia and other complex objects. Object technology makes software reuse a lot easier-- it represents an orientation shift where the emphasis is to build cleaner, more standardized objects in the reuse library which cut down on interdependencies which in turn helps make objects reusable. Especially, the Object-Oriented language enforces what you want developers to do and not to do [MEYER94].

At the same time, the fast growth of the Internet makes the research and collaboration on software reuse using the World Wide Web especially encouraging. Internet has created a potential market that is much bigger than any one before. The distributed software reuse has quickly gained favour due to their intuitive interfaces and powerful yet simple features.

These raise the question of how to make objects more reusable in the general network and how can distributed objects be mixed and matched to form complete systems?

The prototype we decided that the way to build this portable prototype was to create a reusable library architecture. It should be based on the class objects and can make objects more reusable everywhere and distribute objects to form a complete system. Meanwhile, it should provide a foundation for code reuse, scalability and the flexibility in the distributed environment. Not only would this save time, but also it would result in better quality systems for our end-users.

1.2. Criteria for Success

This thesis deals with the identification and extraction from the existing class library to generate the programming source codes in accordance with the end-user requirements. In particular, a new approach to construct reuse systems by a distributed library system model and a natural language-based application generator is investigated. The criteria for success, to be judged in the following chapter, are as follows:

- Review of the existing reuse technologies;
- Description and evaluation of the existing database and distributed approaches;
- Design and development of a distributed library-based model for software reuse in the general network;
- Description and evaluation of modules used in this system;
- Evaluation the contribution and creation of this new approach;
- Prototype implementation of this system to show that it is dynamic;
- Evaluation of this new model by demonstration and execution.

1.3. Outline of the Thesis

The remainder of the thesis is organized as follows:

Chapter 2 describes the current major technologies of software reuse, which include state of the practice, research direction, gaps and challenges.

Chapter 3 explains library-based reuse concepts, how to build reuse class library in the distributed environment, the critical technologies applied in our model.

Chapter 4 describes the detailed design, features and implementation of the prototype system called CodeNet [1], and gives demonstrations on how to use CodeNet.

Chapter 5 highlights the evaluation along with the future work.

1 CodeNet is a new-released name of our system. It has an original name called DORLM (Distributed Object-based Software Reuse Library Module) [ANDY00].

CHAPTER 2. Literature Review

2.1. A Simple, Sensible Idea

There is virtually no software project in which reuse should not be practised. Software reuse is the process of creating software systems from predefined software components. The basic idea underlying software reuse is simple. Rather than build each software system from scratch as has been the normal software practice, with software reuse we capitalize on the similarity of software systems by building them from reusable architectures and building blocks [MCC98]. Because there is such a high degree of similarity among software systems, this idea makes good sense.

Reuse makes sense because the similarity found across software systems is enormous and undeniable. When we compare software systems, we usually find 60% to 70 % commonality from one software application to another. This includes code, design, functional and architectural similarities. At all levels of development from requirements specifications to code, there are components that by the nature of implementing tasks and representing information on a computer must appear over and over again in software applications [ADL95]. New technologies such as software automation, object-orientation and client-server do not change this; however, they do make it easier to take advantage of software similarities. Some software similarities can be predefined and built into software tools (such as reusable code patterns in generators); others can be created as reusable components, which are stored in software reuse libraries.

Although the idea of software reuse is simple and obvious, its implementation is not. The practice of software reuse often requires a change in the corporate culture, software process, software tool set and software skill set; as well as, something to reuse. Reuse is more difficult to implement than other software technologies because it works best when applied above the single system level where there is more opportunity to reuse components and to get the pay-back from the investment in reuse [SAR99]. The broader the base on which to practise reuse the better. Ideally, reuse programs should extend across multiple systems, project teams and even organizational boundaries [MCC98].

Also, without the availability of the enabling technologies discussed in this section, reuse-based system would be far less feasible and effective.

2.2. Representation

Typically, the representation is a description of the reusable asset at some level of abstraction. The types of reusable assets that may be represented include processes and parts of processes, objects, products, and relationships among processes and products [DOD95]. The assets may be physical (e.g., a hardware device) or logical (e.g., a concept). Representation is a critical technology for the interdependent issues of process modelling, product modelling, architecture implementation, and their interrelationships. Common model representations are needed to facilitate the recognition and exploitation of reuse opportunities within and across projects and domains [ASP93]. Such representations must be sufficiently expressive to capture structure, functionality, behaviour, and constraints [DOD95]. At the same time, such representations must be feasible to implement in terms of storage size and processing time for the models.

2.3. Process Modelling

The term software process refers to processes that are intrinsic to developing and evolving software systems [SIT96]. Models of reusable processes will be some of the most important reusable assets within a domain. Such reusable processes must be repeatable, defined, measured and optimising for specific application and product lines within a domain [DOD95]. Rigorous models increase the probability of achieving these goals.

There are two faces or sides of reuse that must be incorporated into the development process to support the practice of software reuse:

- Activities for creating or acquiring reusable components.
- Activities for using reusable components as building blocks in the creation of new systems.

A reusable component may be code, but the bigger benefits of reuse come from a broader and higher-level view of what can be reused. Software specifications, designs, test cases, data, prototypes, plans, documentation, frameworks, and templates are all candidates for reuse [MCC94].

2.4. Composition and Generation

Composition and generation are two major software reuse technologies for the development and maintenance of software systems.

2.4.1. Composition

Composition refers to the development of new programs from reusable modules (and more recently architectural frameworks) which largely retain their form and identity in the new programs [BIRI87].

The first approach to composition relies upon "mining" or "scavenging" existing designs and code for reusable assets. The second approach to composition reflects the position that reusable assets should be designed, developed and supported for reuse. At a higher level of abstraction, the third approach to composition, software schemas, emphasises reusable algorithms and data structures rather than source code components per se [DOD95]. This approach, which is still in the research stage, uses formal semantic descriptions of the algorithms and data structures represented in the schemas [GOG89].

2.4.2. Generation

In contrast to composition, generation technologies rely on reusable patterns rather than reusable modules. Such patterns may take the form of code or transformation rules. For example, an application generator contains embedded knowledge of the semantic patterns that occur frequently in a given application domain [GRI94]. Executable code is generated directly from a specified need for instances of such patterns. Examples of this technology include parser generators and compiler compilers used to create new language translators and application generators.

There are the four approaches to generation:

- Employ the primitive pattern generations that are a part of high-level programming languages [DOD93].
- Based upon the semantics of a very narrow and well-understood application domain.
- Generate executable code directly from software design specifications.

- The transformational approach, uses a formal specification language to capture the intended semantic behaviour [DOD93].

The most promising approach may be a hybrid of both generative and compositional technologies. This technique is based on the notion that several kinds of assets (code, tools, languages, framework) should be designed and packaged as a compatible set [DOD95], so together they make the task of the application engineer much easier.

2.5. Object Technology and Reuse

Object-based programming supports reuse through class definitions, which provide modularity and information hiding, and polymorphism, which allows procedures to be used with a wider range of arguments. The object-oriented philosophy is compatible with software reuse in that it seems to encourage people to identify and create reusable abstractions rather than inventing new code [JF98]. Object-oriented toolkits and frameworks foster the reuse of designs.

To succeed with object technology we must succeed with reuse. We must recognize that reuse means fundamentally changing the software development process, including the OO development process. An OO/reuse discipline inherits more than creating and using class libraries. It requires formalizing the practice of reuse by including support for reuse in OO development methods, tools, training, and measurements.

To employ software reuse combined with OO development effectively, reuse-driven methodologies are an absolute requirement to get any reuse at all for large applications [MCC95]. A reuse-driven methodology for objects consisting of six facets:

- Methodology
- Tools
- Run Time Environment
- Class Library Management
- Testing
- Continuing Operations.

Work is underway to define all six facets, then OO technology enable better support for management and training for reusable components.

2.6. Language Mechanisms

Language mechanisms are the means by which software engineers express the implementations they derive in software development [DOD93]. The reuse of implementations is totally dependent upon the ability of a programmer to make it work in a programming language. The relevant research areas include:

- program modularity (the expression of a program as a collection of parts),
- polymorphism and its particularly important specialisation in inheritance
- environmental issues, including the concurrent execution of, the distribution across multiple processors of, and the handling of exceptional conditions by program artifacts.

Languages supporting these facilities allow programmers to develop programs possessing the characteristics necessary for reusability, portability and maintainability.

The major gaps in language mechanisms from the perspective of reusability include how to reuse software designed for computer systems with thousands of processors, local and wide area networks with thousands of systems, and how exception handling migrates into reusable components from the surrounding environment.

2.7. Software Reuse Library and Repositories

The most immediate problem in reuse is the building up of a repository of reusable software components [GIR98]. Although reusable software components can be designed and implemented during the development of new software projects, existing software is widely considered to be the main source for the extraction of reusable assets [MAA91, KIM90].

A reuse repository is a database for storing reusable assets. A reuse library is a repository plus a search interface, an indexing scheme for the assets in the repository, and facilities for change management and quality assessment of the assets. The library concept has expanded to include providing support for helping the user to understand the assets, since this is necessary for software reuse [DOD95]. In some cases, the concept of reuse library includes additional capabilities such as composition and application instantiation.

There are several issues involved in building a repository. The first is a platform for the repository. The primary alternatives are database management system (DBMS), information storage and retrieval (IR) systems, AI-based systems, and hypertext. There are lots of tools for all of these. The second issue is the indexing of the components. Many techniques for indexing reusable components have been proposed. They fall into four categories: library science, artificial intelligence, hypertext, and formal specification

[DOD95]. Some operational libraries that are based on knowledge representation, employs a semantic network/rule-based hybrid scheme, captures domain requirements and (some) elements of software architecture as a context model for components, which are then linked to the requirements and the architecture [WAL92]. An additional problem is the difficulty involved in actually obtaining assets (especially large ones) that have been identified and requested, due to network overload and inadequate performance.

There has been much work on reuse libraries, such as interoperability, interface design, distributed heterogeneous databases, database security, asset quality assurance, change management, and automated support for controlled vocabulary indexing [HEL91]. Better representations of library collections are needed to help users find and understand the parts they need.

2.8. Reuse in the General Network

The popularity of network has provided a new solution to cheaper and more efficient Software Reuse Libraries (or SRLs). Network tools such as Netscape and Mosaic provide simple and user-friendly methods to search and extract reusable assets from the remote SRLs [POU95].

Distributed SRLs have quickly gained favour due to their intuitive interfaces and powerful yet simple features. But the task implied by network-based software reuse involves making SRLs accessible to users in machines from the mainframe-based to workstation-based or other kind of environments. We need a way to not only locate reusable software but also a way to locate other items of interest, such as key personnel information about on-going programs, the latest developments in various technologies, and trade studies previously conducted etc. The desired information retrieval tool has to handle all these types of information and environments.

Another a major consideration of network implementation of SRL concerns security problem. System must actually authorize users through the use of subnet masks based on Internet Protocol (IP) address; this allows the acceptable level of access control to users.

2.9. Software Engineering Environment

A Software Engineering Environment (SEE) consists of platforms, framework services, repository, and Computer-Aided Software Engineering (CASE) tools [DOD95]. Major problems exist, especially for employing reusable assets, in that implementations of open systems, framework interface standards, and distributed repositories for objects are not yet available easily.

CHAPTER 3. Library-Based Reuse in the General Network

3.1. General Concepts of Library-Based Reuse

A single or virtual software reuse library (a library of interoperable libraries) is developed to provide users with quick and easy access to reusable software assets of interest. Collections of assets (requirements, design, code, test, documentation, etc.) are acquired, classified, certified and made available to users either directly from the library system or from a server in a seamless manner using client processes [PER92].

Many organizations focus their reuse initiatives on a reuse library where members of the organization can both store reusable assets and retrieve these assets when they need them. Software reuse libraries use specialised methods for component classification and retrieval. A recent survey of application programmers [BELL92], which was conducted to discover user needs and attitudes toward reuse, shows that users consider reuse worthwhile, but most of them (especially those without object-oriented experience) expect more from application generators or from tools for automatic programming than from reuse systems like browsers. A browser's usefulness actually depends on the features of software libraries as well as on the expertise of users. Such tools are useful in the case of small libraries; in the case of users that have good knowledge of the library content and extensive experience in its utilisation or as complementary mechanisms of faster retrieval systems, like keyword-based reuse systems [PRI90].

3.2. Object-Orientation Technique in Library-Based Reuse

In order to deal with all the difficulties arising in the library-based reuse area, especially the technical ones, a relatively new design and coding methodology-- object orientation --

has been applied widely for this purpose. "Object-oriented programs are made up of interacting components called objects" [BRAD86]. These objects may correspond to real world entities, to computer hardware and software component, or to data structures. Software construction using object-oriented language is the "assembly of objects to form a system" [JOE00]. An object is defined via its class, which determines everything about an object. Objects are individual instances of a class. A class is a structured collection of fields (variables) and methods (constructors and functions). These two components along with classes make up the central components of an object-oriented application. To use a class within a program, an object of that class must be created.

Object-oriented languages and methods have two features that are of special benefit to library-based reuse: Information hiding and inheritance.

Information hiding is a concept that was known long before object-orientation came to the scene [JOE00]; however, many researchers today feel that the concept came closed to being concrete by using object-oriented methods, for object oriented methods focus on showing "what" is to be achieved by each object, while hiding "how" this is achieved internally. This feature is very helpful in library-based reuse, since it lets the developers interact with components in the basis of their known functionality and not their internal representation. Information hiding is achieved in object-oriented languages through "encapsulation", which "places a wall of code around each piece of data" [BRAD86].
Inheritance, on the other hand, is the feature proper of object-oriented languages and methods. Inheritance is the concept where objects acquire (inherit) some or all of the existing properties of other, already-defined, objects. The objects whose features are

reused in other objects are known as "classes", and the whole set of classes and objects that inherit from them is known as a "class hierarchy". Thus, "programmers no longer have to start each module with a blank page, but instead write a single statement that references some class that is already in a library" [BRAD86]. Each subsequent statement defines how the fact that certain structures will not be defined from scratch, but will inherit from already defined one [HEN95]. Objects and classes can be seen as reusable building blocks: "A minimum of coupling and a maximum of cohesion in class design helps in generating such building blocks" [GALL95].

Object-oriented techniques as an "evolution", not a "revolution" [BRAD86], for object-orientation supplements the existing components of existing software with libraries and techniques to extract those components and make use of them, as opposed to building new components from scratch and this constituting a "revolution" every time a new software is to be developed.

3.3. Build Reuse Class Library in the Distributed Environment

With the growing use of individual workstations, personal computers, legacy systems and increased rates of adoption of the Internet and other networking platforms, reuse of software components available in open distributed systems is a promising approach for efficient development of complex distributed applications.

A complete distributed software reuse library is a way to unify the broad range of existing and new systems in an organization in a consistent and coherent framework that allows for the sharing of information and other resources across the enterprise. Distributed applications within this framework must be simple, flexible, and manageable.

Building a software reuse library in the distributed environment have until now remained a fine art among a few developers Internet Service Vendors and corporations. Most of the reuse systems were still built in stand-alone environments. Problems associated with distributed design are difficult. These problems have included resistance to network failures, inability to maintain persistence, numerous communication mechanisms and co-ordination models, and the inability to dynamically react to changed or unknown network states [LEO99].

Today developers of distributed software reuse libraries have had to resort to solutions that involve complex, multi-language, object-oriented, transaction-oriented software that is expensive and hard to maintain. Or, they have had to internally develop solutions that build code on top of bare metal such as TCP, and on top of a patchwork of middleware and other point technologies of messaging, protocols, object models, and more [RON94].

For a distributed software reuse strategy to be successful, there must be common communication interfaces and behaviour specifications between the libraries and objects within the network. To function effectively, and to allow an organization to take full advantage of true distributed library for software reuse, a distributed computing infrastructure must be designed around a central core. In today's market, the simplest means of establishing that central core, and of extending the distributed computing concept across an enterprise seamlessly and securely is Sun Microsystems' Java technology [SUN00]. The Java language and the Java virtual machine provide an unprecedented opportunity to solve age-old distributed application problems.

3.4. Java: an Example of Internet Object-Oriented Programming for Software Reuse

Java is an object-oriented language for Internet applications developed by Sun Microsystems [SUN00]. Its distinguished features can be summarized as simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language.

Java makes it possible to write component software in the Java programming language. Components are self-contained, reusable software units that can be visually composite components, applets, applications, and serverlets using visual application builder tools. Software reuse in Java across three dimensions.

- Reuse Across Projects

Reuse across projects means that application source code, objects, or components from one project can be productively leveraged in the next [JOE00]. Java is taking this a step farther by not only empower reuse across projects within firms, but also support reuse across projects at different companies by establishing collaborative repositories of reusable Java components that member companies agree to share.

- Reuse Across Tools

Reuse across tools highlights the ability of Java development tools of modifying, combining, or rewriting Java applications created with any other Java tools.

- Reuse Across Architectural Layers

Reuse across architectural layers points to the ability to move application modules or business logic from one layer of an enterprise application to another [ROY95]. Java's portability makes this dramatically easier for platform independence. Migrating Java components across architectural layers is being made far easier by the adoption of Java in products such as Oracle and Unify Modelling Language (UML).

3.5. Critical Technologies for Library Operations

Technologies that support reuse are the tools used to develop, find, understand, improve, integrate, and test reusable components. Research on more friendly and effective reuse systems is ongoing. A considerable number of tools and mechanisms for supporting reuse activities in software development have been proposed. They provide assistance either to application developers for retrieving, understanding, customising and composing components in a software library, or to library managers to create, organize and reorganize reusable development information in the software library [WEI94, URBAN96, WEI92].

The subsections that follow describe the current status of the critical technologies and the necessary improvements to them for widespread adoption of mature software reuse practices.

3.5.1. Retrieval Methodology Based on Natural Language Specification

Most software retrieval systems usually retrieve a set of reusable candidates ranked by similarity with user requirements. However, users don't want to invest a great effort in selecting a component. So, in most cases, the list of retrieved candidates is not completely analyzed even if the highly ranked components are discarded. The user assumes that the best-suited components are the ones on the top of the list of candidates (e.g. the first to the third). Thus, to select a component from the list, the user examines only the associated information for those first components. If none of them satisfies his requirements, maybe he will try to refine or rewrite the original query but, in most cases he will abandon the search. Therefore, retrieval systems should exhibit more precision in their answers, by discarding some obviously unwanted components from the set of candidates and by retrieving only the ones that satisfy more precisely the requirements of the user.

Most software retrieval systems retrieve components through a set of keywords provided by a user employing either a controlled or a free vocabulary. These systems are simple and effective for experienced users. The effectiveness breaks down for users not familiar with the proper terminology. Such users may not know a proper keyword, and therefore may use a synonym, a related term, or a more general or specialised term. In such cases, keyword-based systems fail because they don't provide an answer or they retrieve a great number of irrelevant software components.

The current survey also shows that most of the interviewed users prefer natural language interfaces to retrieval systems than keyword-based interfaces. It seems more friendly for a user to say, "I want to get a component that can do calculation operation." than to think in proper keywords, corresponding classification schemes and boolean combinations of keywords. This is yet more critical for users not familiar with common terminology or even worse with keywords systems based on a controlled vocabulary.

Queries in natural language allow for improving the precision of retrieval systems. Precision can be improved by allowing queries and indexing software components with multiple words or phrases extracted from the natural language descriptions. In a particular domain, software libraries operate with a great number of common concepts (e.g. file in UNIX system) and retrieval through single words reduces the precision of the system by retrieving a great number of components that are mostly irrelevant to the final requirements of the user [GIR93]. To achieve high precision is more crucial in software domains than in typical domains of information retrieval systems. In software retrieval systems the main purpose is not to retrieve all the material in which a particular pattern exists but rather those that best fit the desired functionality.

Natural language processing techniques have been applied in information retrieval systems mainly at the lexical, syntactic and semantic level [GIR93]. Some improvements on retrieval effectiveness have been obtained with multi-keyword systems, using statistical methods to index documents based on the multiple occurrence of keywords in a text. These improvements are not enough and current research [PRI90] recognizes that some form of natural language processing should be necessary to increase the effectiveness in text retrieval and other text processing operations. Thus, most recent information retrieval systems incorporate natural language processing techniques and this experience seems to be useful for the design of software retrieval systems.

3.5.2. Lexical Analysis and Parsing

When insert reuse assets, such as classes, to a software reuse library, one of the more common things we will be required to produce is lexical analyzer and parser. They range from simple to complex and are used for everything from looking at command-line options to interpreting source asset.

The purpose of lexical analyzers is to take a stream of input characters and decode them into higher level tokens that a parser can understand. Parsers consume the output of the lexical analyzer and operate by analyzing the sequence of tokens returned. The parser matches these sequences to an end state, which may be one of possibly many end states [MCM00]. The end states define the goals of the parser. When an end state is reached, the program using the parser does some action -- either setting up data structures or executing some action-specific code. Additionally, parsers can detect -- from the sequence of tokens that have been processed -- when no legal end state can be reached; at that point the parser identifies the current state as an error state. It is up to the application to decide what action to take when the parser identifies either an end state or an error state.

The lexical analyzer and parser actually perform their processing in three phases [MCM00]: parse the source code to determine definitions and references, resolve references in the contents of the symbol table and generate the cross-reference report. In more detail:

1. Determine Definitions and Note References

The first phase of the cross-reference tool goes through all the source code in the specified directories (and their subdirectories.) The lexical analyzer and parser collects information on the following constructs:

- Package Definitions
- Class Definitions
- Interface Definitions

- Method Definitions
- References to other symbols

For each of the above definitions found, a new symbol is created. This symbol may reference other symbols such as a superclass that is being extended as part of a class definition. During the first pass, these references might not yet be available; they could be defined later in the same source file or in another file that has yet to be parsed. Because of this, any references to superclasses, implemented interfaces, return types for methods and parameter types are stored as placeholders by their names only [SCOTT00]. These placeholders are held in an instance of the DummyClass class [SCOTT00]. These will also be resolved during phase two.

2. Resolve Definition References

After the source-code parse, we have a symbol table that lists all constructs defined in the source files. Each of these definitions may reference other definitions (for superclasses, variable types and so on). This second phase will walk through all definitions in the symbol table and resolve those references.

At the conclusion of this pass, the symbol table contains all symbols defined in the parsed source files and proper resolutions to defined symbols. The next step is to find references to the defined symbols.

3. Report Generation

This final phase looks at the data contained in the symbol table and generates a cross-reference report.

3.5.3. Application Generator

Application development is a complicated process. Developing and describing the standards and methodology to be used in application development are even more complicated. Because the business needs of today force us to develop applications rapidly and to maintain them with limited resources, common standards must be enforced, code reuse must become a daily activity, and productive tools must be implemented to accelerate the application development process [PAT99].

The application generator provides the functionality to assist in the population of the reuse base with generic code components and all the additional information (e.g. natural language descriptions) required describing and reusing the components. This approach reduces the need for an organization to write its own data capture, transformation and load programs. Meanwhile, automatic application generation is a way to ensure consistency between design and implementation stages of the software life cycle.

But how to design reusable software (from scratch or from existing components) is not an easy task. Good knowledge of the application domain and past experience in the development of applications in the domain are required. Existing components must be properly qualified according to their reusable attributes before their redesign and inclusion in the library base [CAL91, GIR92]. Reuse guidelines are identified and attached to the components in the library base in order to suggest typical uses of the component and design guidelines to adapt them to a new context.

Generic code components will be created from scratch or by an abstraction process from specific ones. Learning by analogy from past development experience and reverse engineering are proposed techniques to deal with this abstraction process [CAL91]. In the techniques for learning by analogy, the idea is to transform solutions of past problems into potential plans to solve new generated problems [MIC86]. Reverse Engineering techniques may provide heuristics to abstract generic specifications from existing software at a particular specification level, and from the implementation level to the design level to the requirements specification level [CHI90]. Some measures of the level of reusability of the specifications extracted from existing applications will be done before their inclusion in the software library. Common functionality, ease of modification, correctness, readability and other attributes of reusability may be measured by applying a set of metrics and models [CAL91].

There are many different kinds of application generators today:

- Stack based application generator: Generate code from source language to target language using stack variable memory.
- Accumulator based application generator: Use an accumulator to compute values. The accumulator holds one of the operands of a computation, the other being given as a parameter of an instruction. The result is placed into the accumulator.
- Rule based application generator: Allowing the integration of legacy or use-specification rule. It potentially is for supporting any language code and any architecture.
- Web based application generator: Generate application in the distributed environment including Internet.
- Pattern based application generator: Approach to generate domain specific application code using patterns.

- Form based application generator: The easy-to-use visual design interface versatile application generator.

3.5.4. Security of Software Reuse Library

As the things of other distributed software agent, a major consideration of Internet implementation of software reuse library concerns security problem.

Most library web servers include a login screen, which indicates that all activity is monitored and that anyone who doesn't like this should logout. Many web servers even don't include any such warning. Intangible Assets Manufacturing stated in its own web policy document [LEO99] that, "most Web servers log all accesses. In theory, we could compile a lot of information about you this way, and some web servers probably do. Heck, they may even sell that information to other companies."

There are several reuse systems, such as the Andrew File System (AFS) [LEO99], Access Control Lists (ACLs) and standard UNIX file permissions, allow reuse system to grant and deny access to the host environment. System may actually authorize users through the use of subnet masks based on Internet Protocol (IP) address; this allows the acceptable level of access control to users.

3.6. Our Approach

Distributed library-based reuse is that area of software engineering that enables developers of software to use already existing, off-the-shelf, components, instead of having to "re-invent the wheel" at the start of every new development project [JOE00]. Distributed library-based reuse model provides the means to integrate the data and logic of multiple information systems into a coherent data management infrastructure [GEP00]. It constitutes a powerful, innovative model for generalized distributed computing. In particular, it is very attractive because it adds object capabilities with distributed computing context that already offer a number of other valuable advanced features [GEP00].

Although the basic concept of reusability is well understood, the software community is still debating on the proper definition of a standard for software reuse. Questions such as which products of the software life-cycle should be reused, how to organize reuse components, and how to make objects (classes) more reusable in the general network and how can distributed objects be mixed and matched to form complete systems are still being asked. Faced to these issues, we try to present a distributed library-based reuse model (CodeNet) for creating, updating and querying a class-based library and generating the specific applications in the distributed environment while considering the viability and applicability of reuse fully.

CHAPTER 4. Design and Implementation A Prototype System: CodeNet

4.1. System Overview

In order to ensure that a model is of practical utility, it is necessary to construct an implementation of it designed to test various aspects of that model. To this end, a prototype system -- CodeNet -- presented in this chapter was created, designed and then implemented.

CodeNet shall provide two kinds of major services: server side service and client side service. In the server side, the users, who should be skilled programmers and experts, can perform operations such as insert, update, delete and query classes and functions (reuse assets) of various object-oriented languages with the library. In this side, we try to provide a lexical analyzer to verify the user's input content (classes and functions). Meanwhile, a dynamic index of classes and functions in the library and a user-friendly GUI will be provided for convenient use of various operations.

In the client side, through the general network, the end-users can design and program the object-oriented applications (such as C++, Java) by this model in accordance with their requirements. What they need to do, for example, input several simple arguments of the application classes, and then this system will generate the suitable object-oriented applications. Actually the end-users don't even require understanding the detailed structure of classes in the library.

4.2. The Selection of Appropriate Implementation Tools

The selection of a suitable run-time environment and software tools to implement CodeNet was not an easy process. Since the CodeNet is a distributed object-based software reuse model, object-oriented and distributed technology should be selected to do precisely that. This meant that the CodeNet implementation would be written as computer software. In the end, four major technologies: UML, Java, CORBA as well as Oracle were used to design and implement the concepts presented in this thesis.

The Unified Modelling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modelling and other non-software systems [UML97]. It is one of the best choices for building distributed object-oriented and reuse-based systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The primary goals in the design by UML were as follow:

- Provide a ready-to-use, expressive visual modelling language in order to develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modelling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns, and components [UML97].
- Integrate best practices [UML97].

Java, as a language and platform, has a number of desirable features that would serve to facilitate an implementation of the CodeNet to prefer it over other general purpose programming languages such as C++, C, Smalltalk etc. and their environment. As mentioned in section 3.4, some of Java features like: (i) its compiled form can be executed on virtually any existing computer; (ii) it is able to interoperate with web browsers within a secure, easy to download environment for the user, and (iii) it is object-oriented, support concrete and abstract classes, multiple inheritance of interfaces, and class serialisation. These make Java satisfy code reuse in each platform and perform tasks such as network socket I/O and concurrency.

OMG's Common Object Request Broker Architecture (CORBA), is increasingly accepted as a standard, cross-platform, cross-language distributed object computing framework. CORBA allows clients to invoke operations on distributed objects without concern for object location, programming language, OS platform, communication protocols and interconnects, or hardware. The CORBA object model provides a standard middleware framework that enables CORBA objects to interoperate with each other. CORBA defines a software bus that allows clients to invoke operations on objects that can reside locally or remotely [CORBA99]. Moreover, to provide higher-level reusable components, the OMG also specifies a set of CORBA Object Services that define the standard interfaces to access common services, such as naming, trading, and event notification. By using CORBA and its Object Services, programmers can integrate and assemble large, complex distributed applications and systems by using features and services from different providers. Thus, CORBA was selected as one of the major implementation techniques for CodeNet.

Oracle is the world's most popular database for Internet computing. Oracle 8 has brought the relational database world into the distributed object area [ORACLE99]. The new features of the product make it possible to combine the newer distributed object-oriented structures with the traditional relational constructs. With Oracle 8, building and maintaining systems will be faster, more easily and for lower cost. Meanwhile, Oracle 8 includes significant enhancement to keep pace with the technological requirements of demanding Internet applications. Through integrated Java Virtual Machine, Oracle 8 has improved performance to support for Java2, and allow applications to efficiently implement and manage robust security policies [ORACLE99].

4.3. Features of CodeNet

CodeNet contains many features to help developers to build object-oriented applications in the general network.

1. User-friendly GUI design

In CodeNet, the newest JFC API (Java2: Java Swing) was used for the GUI design.

- Integrated with the JSplitPane and JTabbedPane, the interface frame looks more concise and flexible. The end-users can highlight one part of the whole interface to input data or do operation. This kind of design offers a visual and comfortable operation platform to users.
- Provides dynamic "Class View" updating through the JTree and JList. When the server side users insert or delete a component with the "class" library, the updating

effect will be displayed in the interface at once. This kind of design will avoid the unnecessary operation mistake and make the model more real-time.

- Dynamic button action, which was designed through the status flags, will make the operations more vivid and precise. That is, one button can execute the different actions in accordant with its dynamic label content.

2. Stored procedure and package

A set of related classes, procedures, functions, and variables are bundled together into a package. This approach modularizes the whole system, the benefit is, it simplifies the system architecture and reduces the development time and cost largely.

In the server side, because the reuse components (classes) in the library have various representative patterns, the information of a complete component is divided into the following segments: (i) key word part; (ii) import part; (iii) class head part; (iv) variable part; (v) constructor part and (vi) function part. The constructor and function part also consist of the three sub-parts: (i) head part; (ii) variable part; (iii) content part. Vector technology was selected by this model to store the component information. Information of each part would be stored into the different vectors. Through the CORBA and Java Socket, these vectors will be delivered to the library and composed the complete components for inserting into the library. The advantage of vector is that all kinds of components may be stored and accessed flexibly without considering the limitation of their amount and patterns. This point is exactly the deficit of the previous reuse system. To combine GUI with information store process fully, GUI must provide the corresponding scheme for browsing, searching and modifying all items of components.

For example, add four buttons to manage the first, last, previous and next item of each component.

3. Error handling and warning box

Since components of the library will be reused by the client users. The component information must pass the lexical analyze before being inserted into the library. In all interfaces, the user-input and user-selected data will be validated against by some rules, such as a class constructor can't have "*return type*", variable name can't be the same etc. On the other side, the run-time error will be warned through the warning box such as server shut down, password isn't correct etc. Moreover, in CodeNet, an automatic message box, which can open and close automatically, was implemented to prompt the users about the information, warnings and error messages.

4. Flexible search engine by the natural language

In the client side, a flexible search engine that was designed by Java Applet and Java Database Connectivity (JDBC) will handle whatever the end-users input by the natural language, and then find the key words of the sentences through checking the grammar and vocabulary. The components that fit for the search requirements will be attained from the remote library server and display in the client-side interface. This kind of design will be suitable for all sorts of users, especially the users who don't have any computer experience. It makes great advances in the reuse area. Most of the old reuse systems just aimed at the users like programmers or developers, not the general users.

5. Faster and more efficient generate various object-oriented applications

In the client side, a powerful application generator is available. It can be used to generate source code about various object-oriented languages (Java, C++ etc.), based on the parameters input by the users. The users don't even require understanding the detailed structure of components in the library. In addition, the generated application can be made up with multi-components of the library. Furthermore, the application generator reserves the program space for the end-users, that is, allowed to add the new program content as users prefer.

6. Multi-thread and distributed control

CodeNet provides a robust environment for reusing remote library components. It automatically handles all communication among clients, servers, and middle tiers, and allows user to quickly and reliably deliver information to web browsers, laptops, servers, and workstations. Also it supported multi-thread real time operations, which permits multiple users to access the library server concurrently.

4.4. System Design and Implementation

The general purpose of our research is to contribute with concrete solutions to controlling software development by turning easier the practice of software reuse. Particularly, the system must be cost-effective, precise in its answers and domain-independent (i.e. independent of the application domain of the software libraries).

This system design and implementation process strictly obeyed the principle of software engineering, comprised with five phases: (1) requirement analysis and specification; (2) system and software design; (3) implementation and unit testing; (4) validation, integration and system testing and (5) delivery and maintenance.

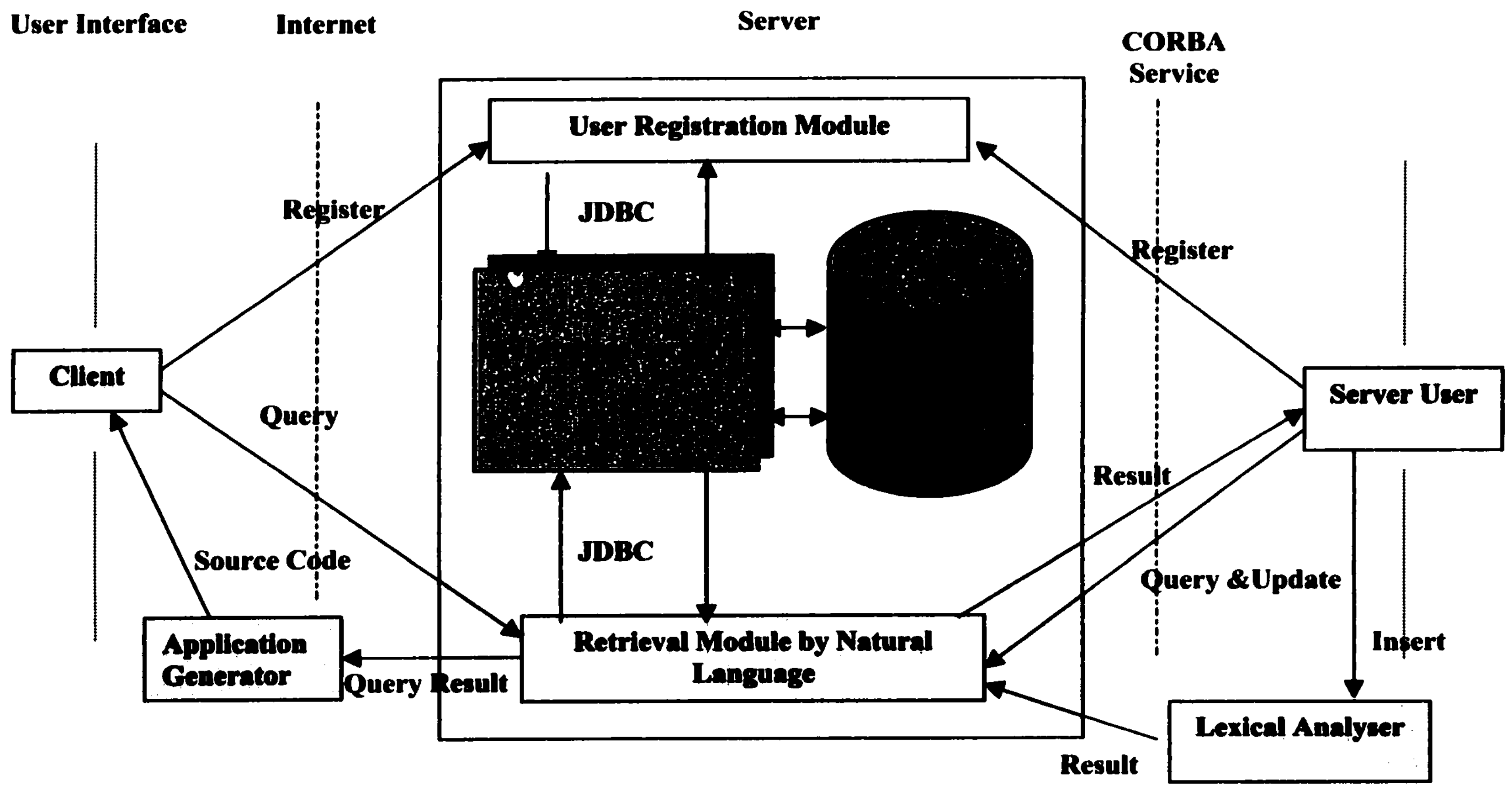
Following sections will highlight the important stages and documents of this system.

4.4.1. System Structure

The architecture of the prototype system is shown in *Figure 1*. This system composed of three main sub-systems:

- The middle part of this model is the library server. It consists of a library server, a user register module, a class library and a retrieval module. It is the system kernel. Most operations and actions will be invoked and controlled here.
- The left part of this model was designed for the client-side users. Users can access the library server after the network security check. Through the application generator, users may attain the application source code, as they prefer.
- The right side of this model was designed for the server-side users who usually are computer experts. The special privilege will be given to the server-side users for accessing the library server. They allow to perform the operations such as insert, update and delete, which the client-side users weren't permitted to do. A lexical analyzer was offered to verify user-input and user-selected data and compose those data to components for storing into the remote library.

Figure 1: Architecture of CodeNet



4.4.2. System Modules

Based on the system architecture, the following nine important modules should be created, designed and integrated into a complete model:

1. Server Interface:

Controls the whole process in the server side. It includes:

- main interface: connects to the different interfaces through four major buttons -- user register; enter system; help and exit.
- user registration form interface: provides the server-side users a way to input their individual information.
- system server-side console: according to the language types, consists of the different tabs such as Java tab, C++ tab. Four parts were designed in each tab: (1) dynamic list of class (component in the library) names; (2) dynamic class index tree includes class names and function names; (3) panel for inputting the class information and displaying the query result, consists of key word part, import part, class head part, variable part, constructor and function part. It is the core of the server-side interface; (4) control panel, by means of invoking the buttons (*insert*, *delete*, *query class*, *query function*, *modify*, *restore*, *help* and *return* buttons) to control the operations of the server side.
- help frame: provides help information and user manual.

2. Client Interface:

Controls the whole process -- search the suitable classes and generate applications for the Internet end-users -- in the client side. It includes:

- search engine: is divided into two sub parts, (1) information part, lets the client-side user input the generated class name, language type and the search standard; (2) control part with buttons -- view all (view all classes in the library), search (query library by the search standard) and clear.
- class index: is divided into two sub parts, (1) information part shows query result about class names and function names by a dynamic tree style; (2) control part with buttons -- submit (to pop up the parameter window) and generate application (to pop up the application window).
- parameter window: based on classes and functions selected by the client-side user, provides the different size frames to input the class parameters.
- application window: displays the generated application source code.

3. User Registration Module:

Provides registration form of the server-side user. Users don't allow to use this system without registration. The registration information includes name, password, address, phone and experience in the computer area. System administrator will give each server-side user a different privilege to access this system. It is an important security part in the model. The server-side users can only access the system through their password.

4. "Class" Library Database:

In order to promote the automatic retrieval, a reuse class library was constructed by both the intra-library relation and inter-library relation.

Every component (class) in the library is handled as an object. All data of a class is treated as a whole one. They connected each other through the identifier. System will generate the identifiers automatically for each component when the server-side users insert it to the library. Besides, a user information table, a language table and a key word table were created for storing the corresponding information.

5. Lexical Analyzer and Error Handling Module:

Perform the grammar and validation check. In this system, a simple lexical analyzer was designed and implemented. It provides a way handle some simple program lexical analysis according to some object-oriented language rules. The work is still in its preliminary stages and will be continued to develop in the future work.

6. Retrieval Module:

A robust library search engine looks for the most relevant and desired components for reuse, based on the statements of the natural language. The retrieval algorithm is based on the detection of similarities between the user query and software components in the knowledge base.

Figure 2 shows structure of the retrieval system. A lexical, syntactic and semantic analysis will be performed on the query to translate it into a frame-like internal representation. The translation process uses lexical and semantic information in a thesaurus and a semantic grammar with constraints. By this means, the Internet users can get the candidate components easily and they even needn't know the library structure.

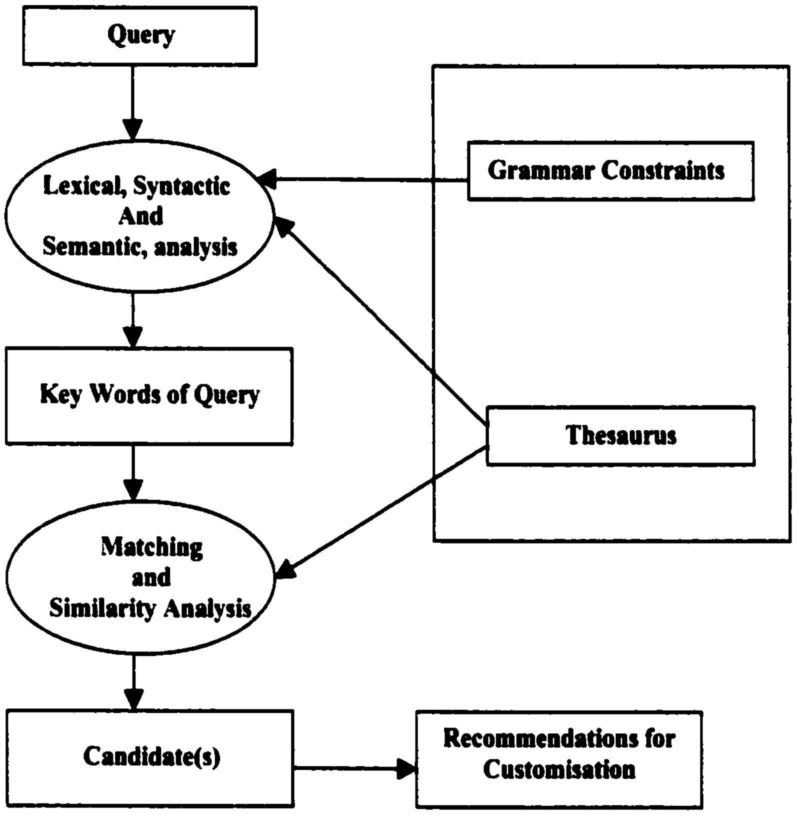

Query
Lexical, Syntactic And Semantic, analysis
Grammar Constraints
Key Words of Query
Thesaurus
Matching and Similarity Analysis
Candidate(s)
Recommendations for Customisation

Figure 2: Structure of the Retrieval System

7. Application Generator:

Provides assistance to generate the source code based with generic code components and all the additional information (e.g. language type and class name). What the client users need to do for application generation is just to input several essential parameters, which must be used in the generated application.

In order to generate the multi-component based application, many sets of component information need to be extracted and stored before combining and forming the application code. The vector technology was employed again for this purpose. After that, all items in these vectors will be united to form the compilable application in accordant with the definite sequence and the object-oriented language rule.

8. Naming Service:

Serves as a directory for CORBA objects in the system. With naming services, either client or server components on the object bus to bind and resolve other components on the object bus by a user definable name. The naming services allow to associate a URL with an object. Once the URL is associated with the object, any client of the web server can access the object reference through the URL. It is a critical module for the distributed system design and implementation.

9. Protocol between Client and Server:

Provides the communication protocol when transferring data package between client-side user interfaces and the library server. The protocols consist of the following data operation process:

- View all components in the library.

  Input: "View All" → Output: all component names and function names
- Query by the natural language.

  Input: "Search" & the search standard defined by the natural language → Output: key words of search standard & query result: candidates of components.
- Submit and get parameter form.

  Input: "Submit" & selected class names and/or function names → Output: class and/or function parameter type and amount.

- Submit the filled parameter form.

  Input: parameter values → Output: "OK".

- Generate application.

  Input: "Generate Application" & application class name & language type→ Output: application source code.

4.4.3. Use Case, Class Diagram and Sequence Diagram

As the most important phase of software engineering -- design and specification, UML (Unified Modelling Language) was selected to specify system actor behaviour (use case), system class architecture(class diagram) and internal module design(sequence diagram).

Figure 3 shows the behaviour case of each actor in the system. *Figure 4* describes each event (function) of the system with design-oriented view. *Figure 5* shows the static relationship between classes and what objects and links the system may have at the any time. *Figure 6-11* shows actions and interactions, events and messages involved among the objects, actions include process of insert, modify, delete, server-side query, client-side query and user registration.

Figure 3: Module Use Case

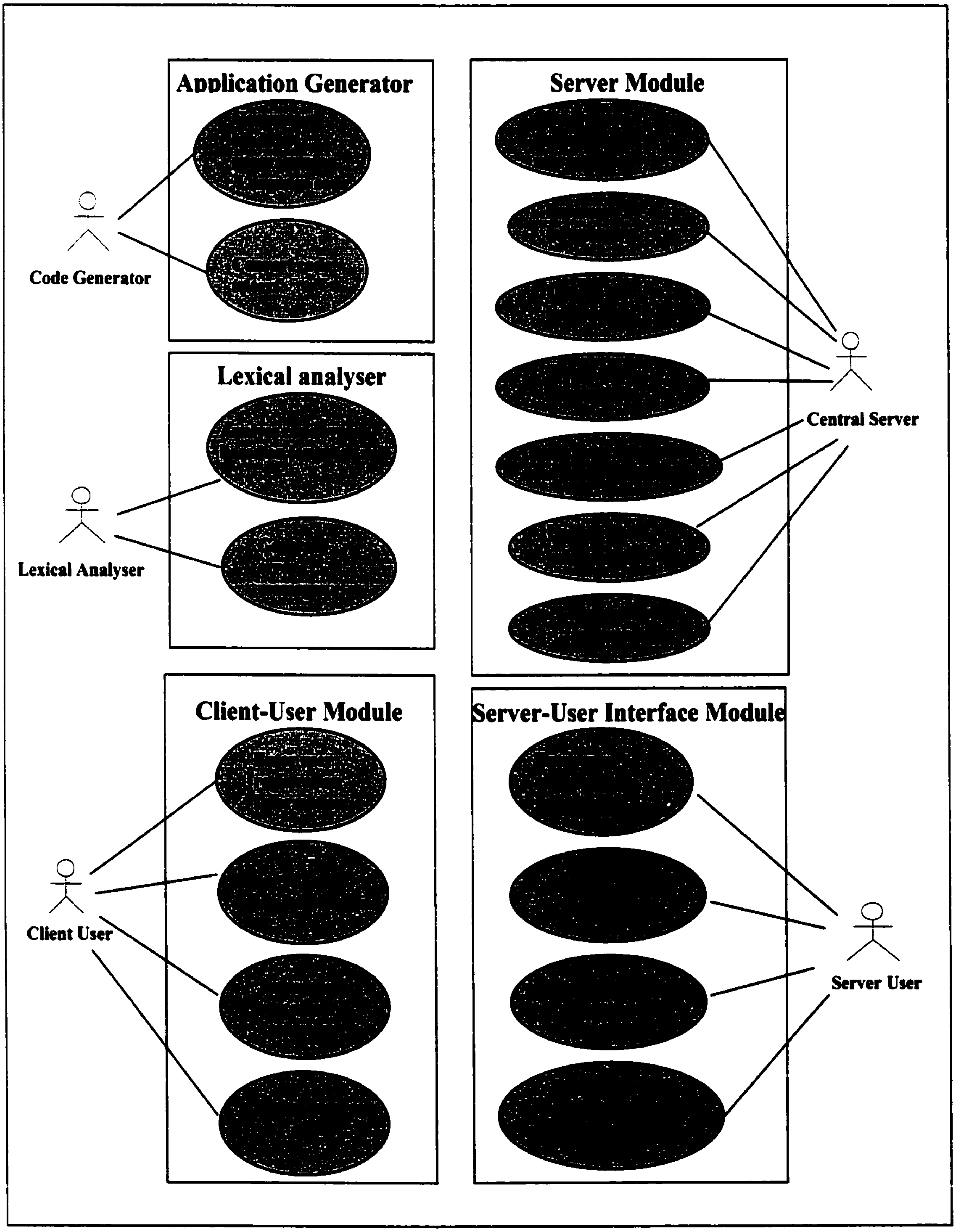
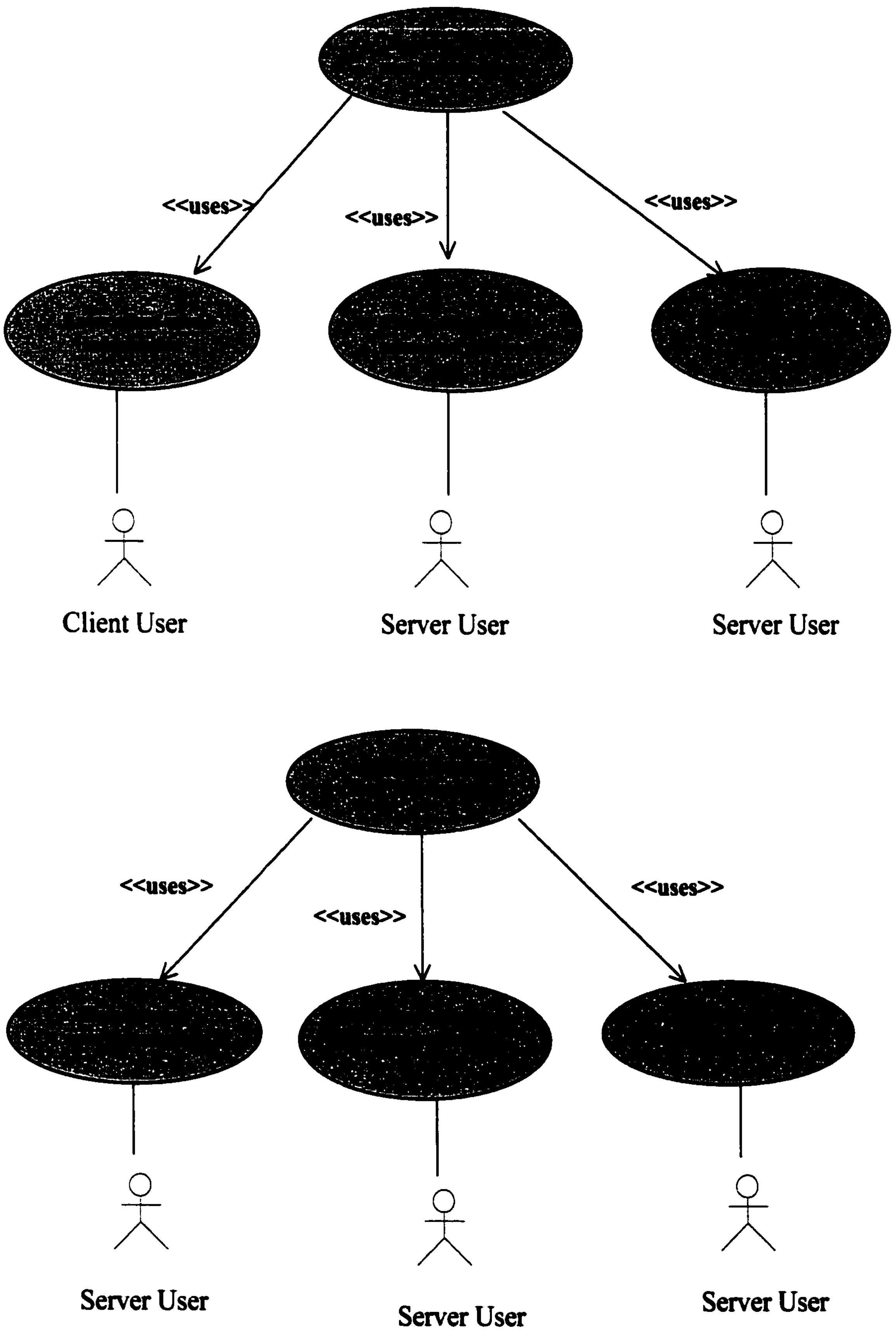
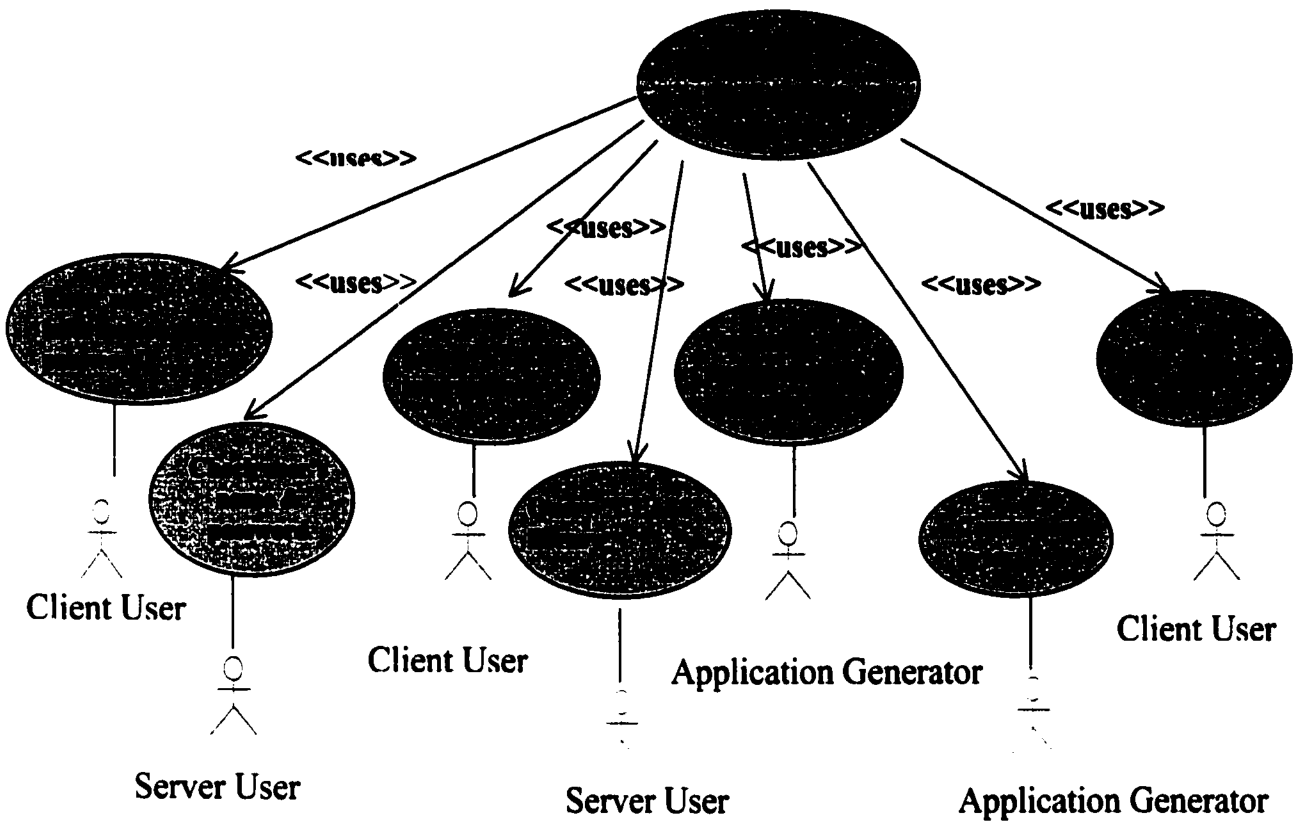
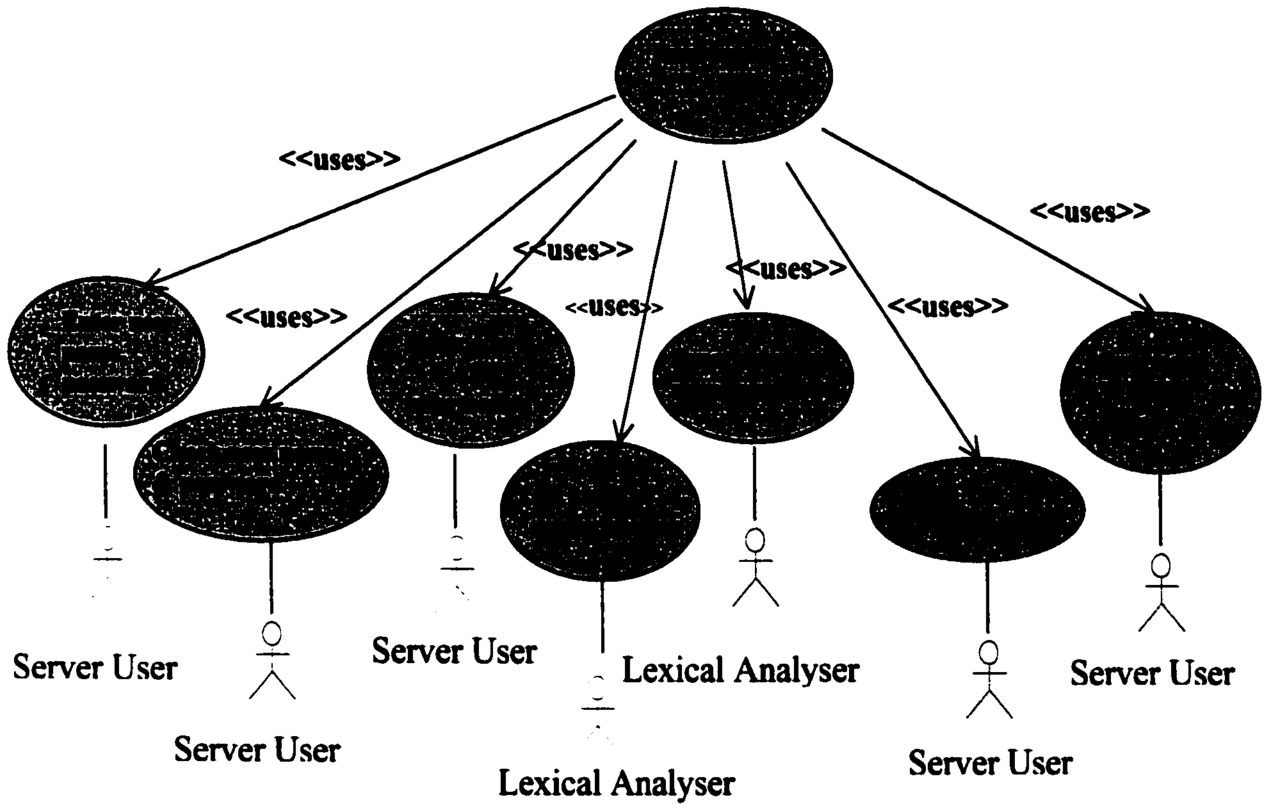
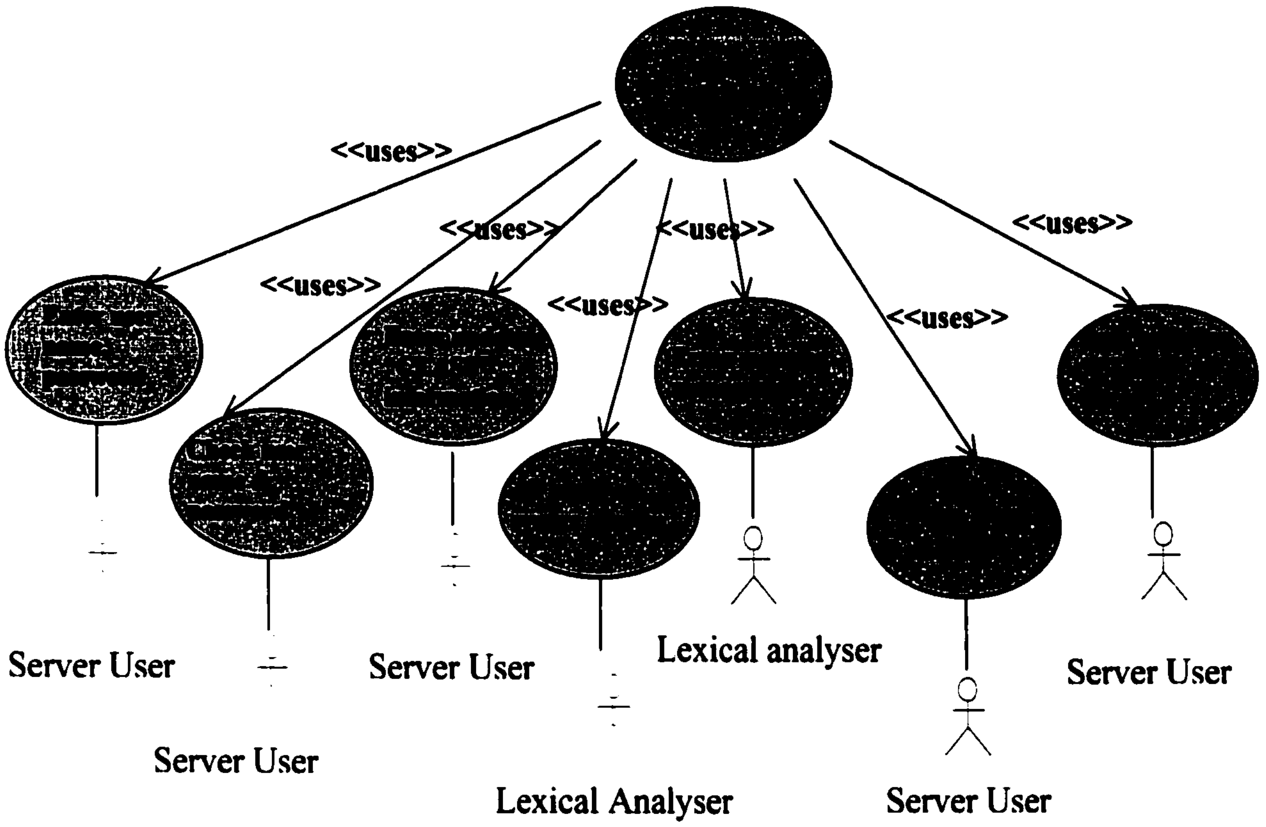


Figure 4: Function Use Case

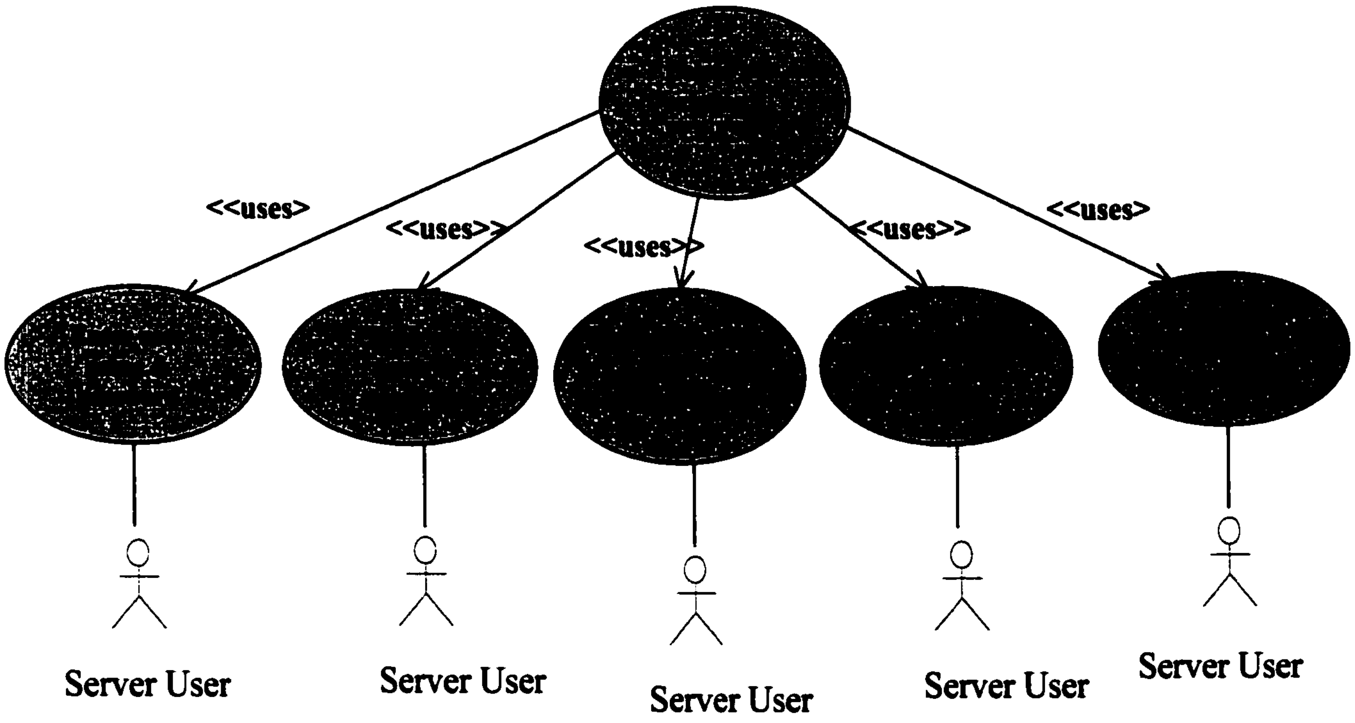



<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
Client User
Server User
Client User
Server User
Application Generator
Application Generator
Client User


<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
Server User
Server User
Server User
Lexical Analyser
Lexical Analyser
Server User
Server User



<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
Server User
Server User
Server User
Lexical Analyser
Lexical analyser
Server User
Server User



<<uses>
<<uses>>
<<uses>>
<<uses>>
<<uses>
Server User
Server User
Server User
Server User
Server User

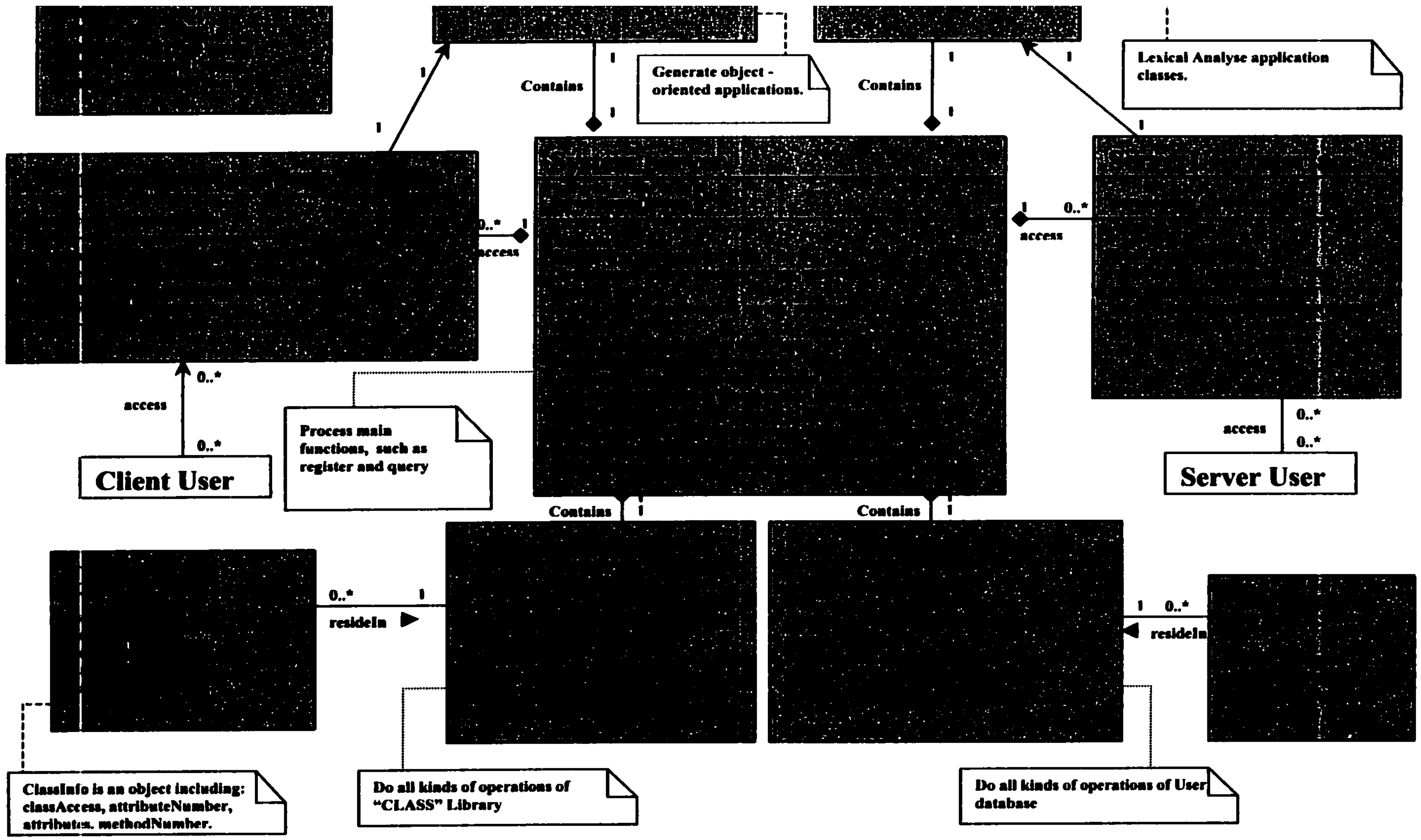

Generate object - oriented applications.
Lexical Analyse application classes.
Contains
Contains
access
access
0..*
1
Client User
Server User
Process main functions, such as register and query
Contains
Contains
residein
residein
ClassInfo is an object including: classAccess, attributeNumber, attributes. methodNumber.
Do all kinds of operations of "CLASS" Library
Do all kinds of operations of User database

Figure 6: User Registration Diagram

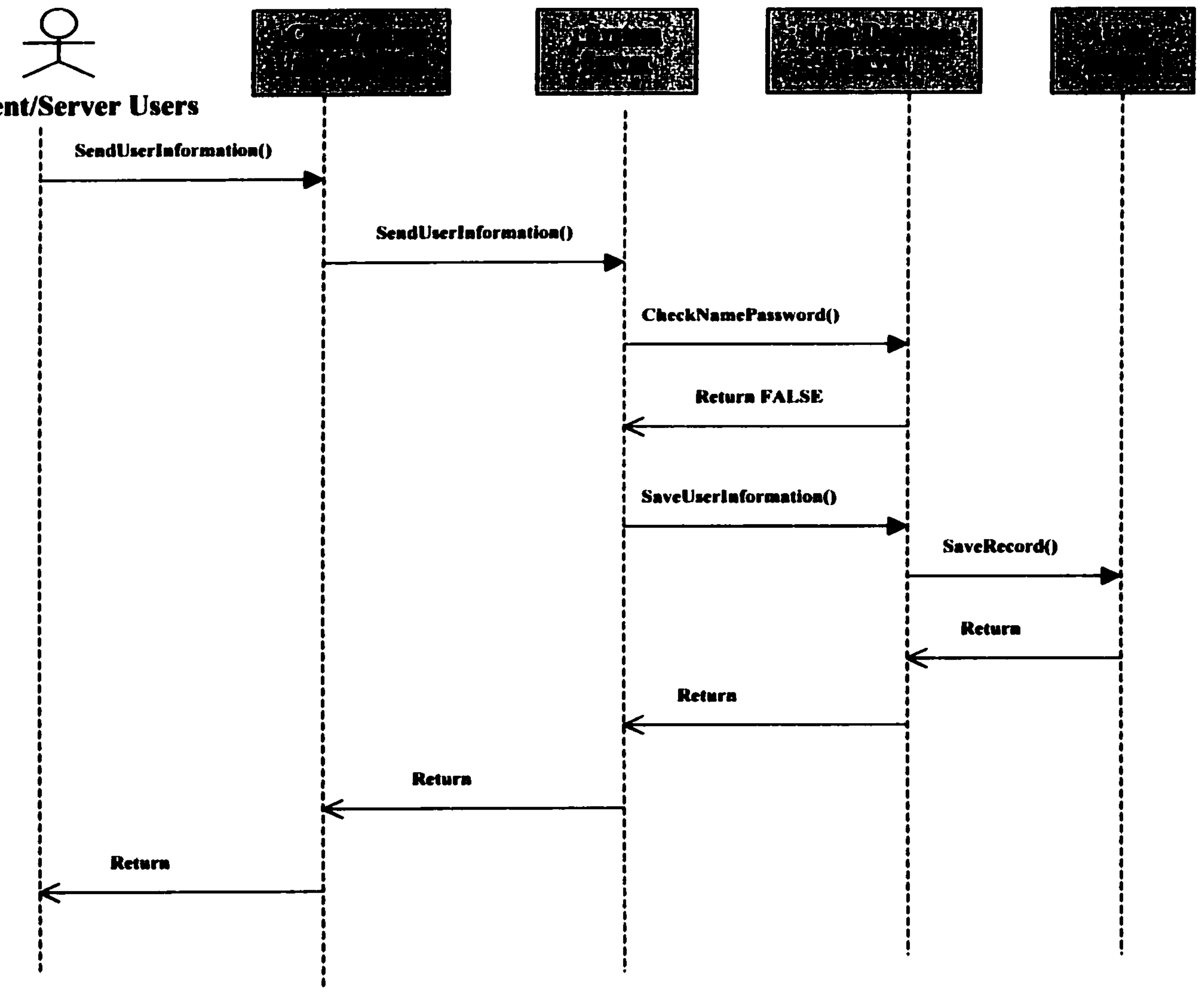
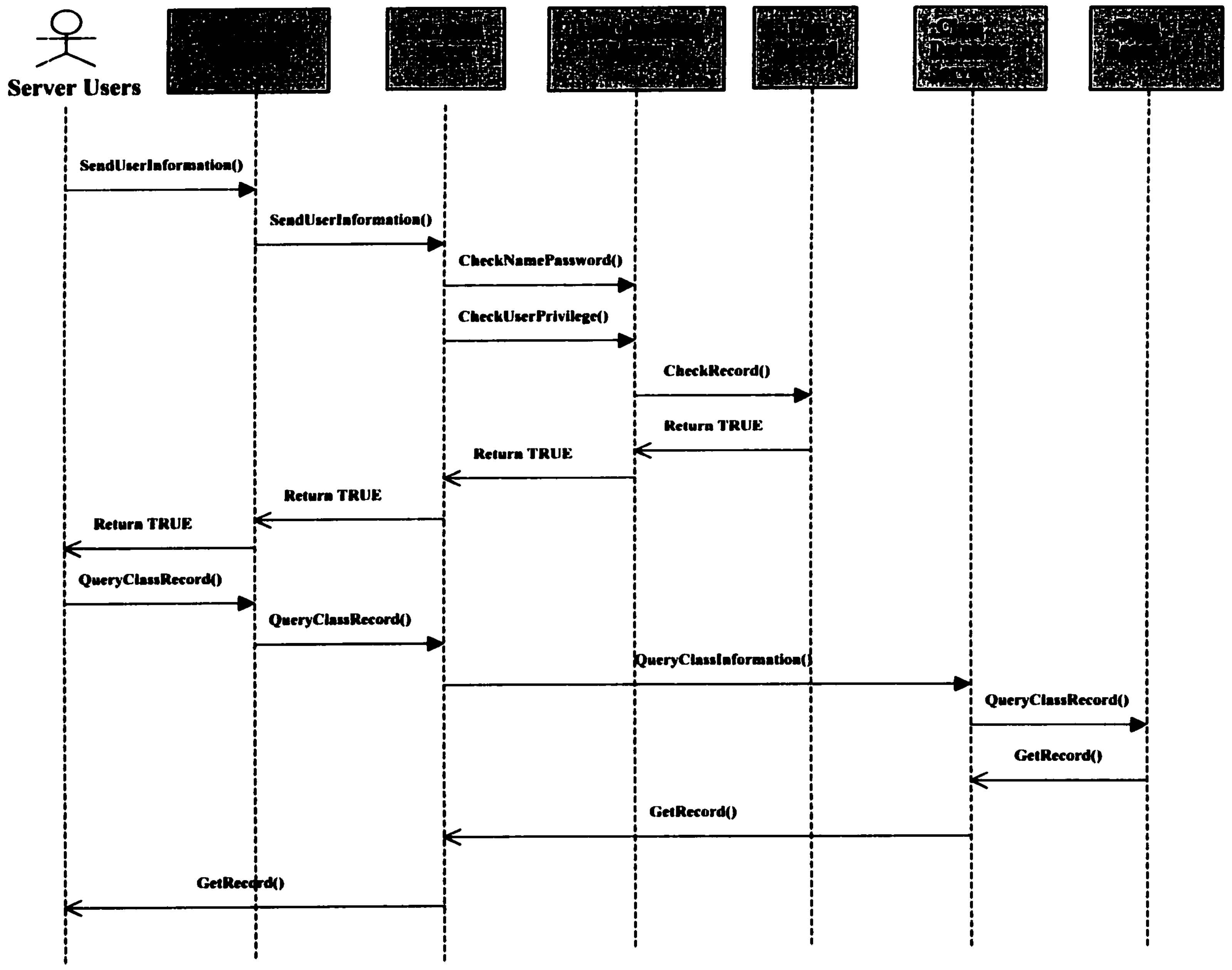
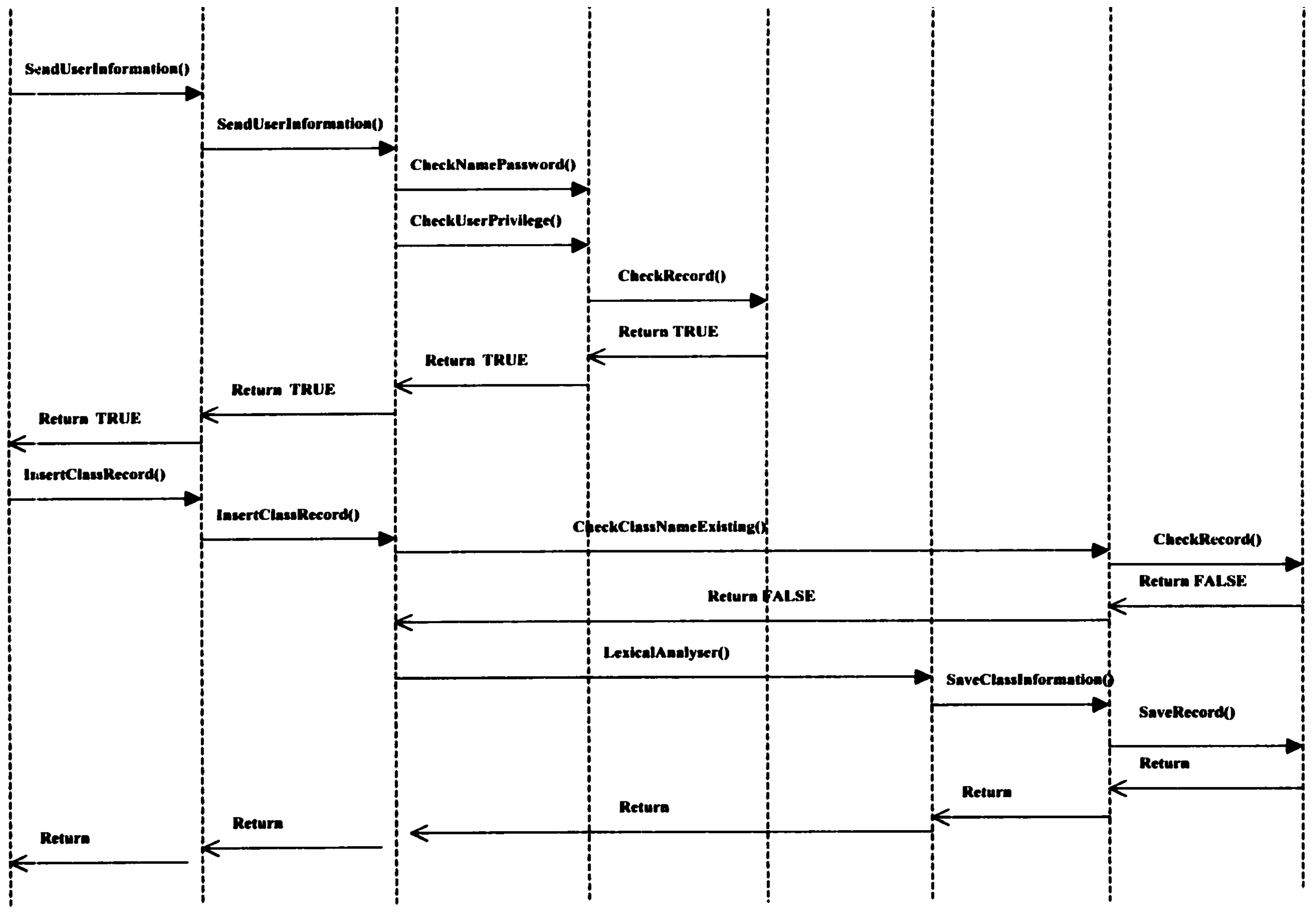


Figure 7: Server-Side Query Class Diagram




SendUserInformation()
SendUserInformation()
CheckNamePassword()
CheckUserPrivilege()
CheckRecord()
Return TRUE
Return TRUE
Return TRUE
Return TRUE
InsertClassRecord()
InsertClassRecord()
CheckClassNameExisting()
CheckRecord()
Return FALSE
Return FALSE
LexicalAnalyser()
SaveClassInformation()
SaveRecord()
Return
Return
Return
Return
Return

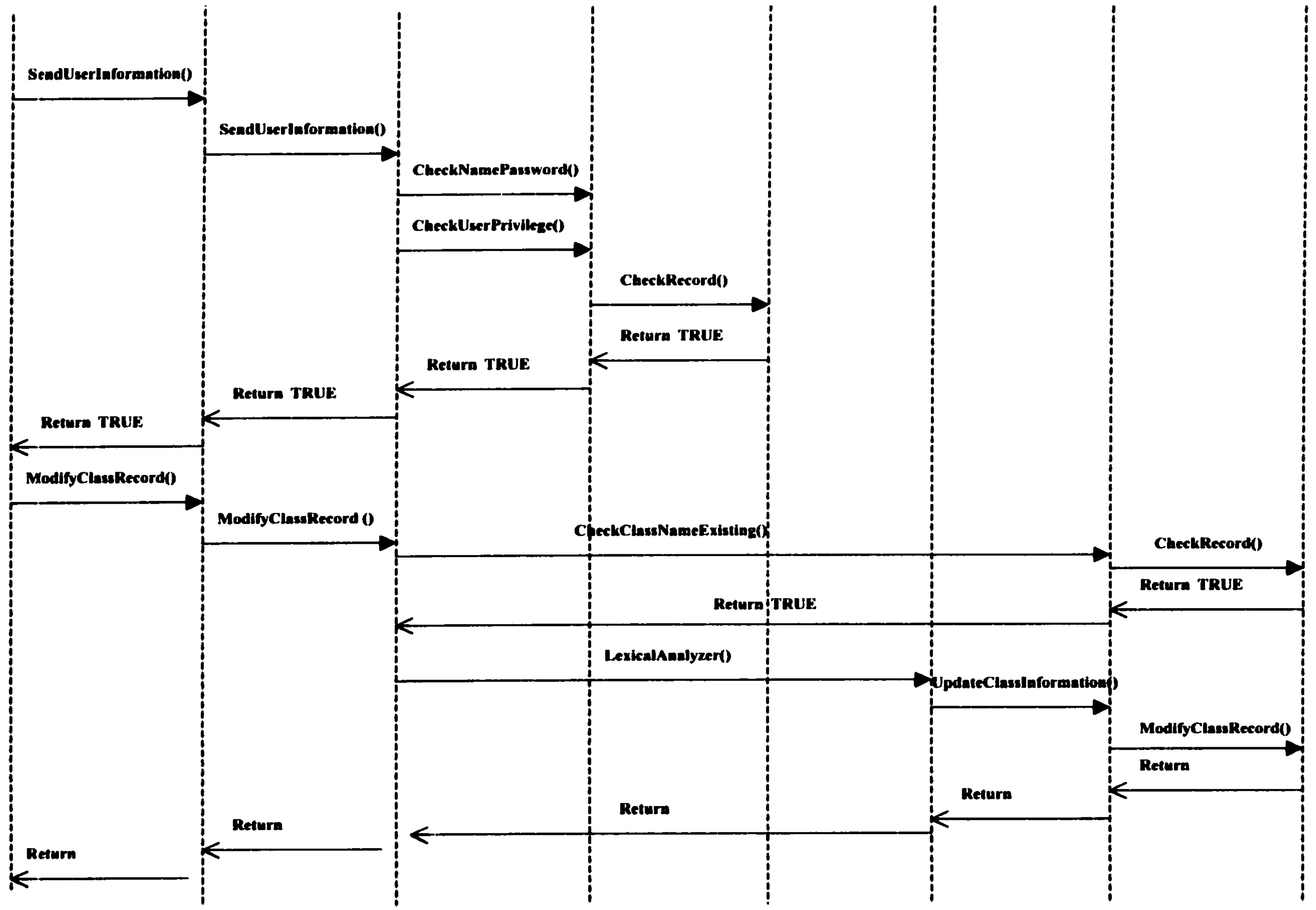

SendUserInformation()
SendUserInformation()
CheckNamePassword()
CheckUserPrivilege()
CheckRecord()
Return TRUE
Return TRUE
Return TRUE
Return TRUE
ModifyClassRecord()
ModifyClassRecord ()
CheckClassNameExisting()
CheckRecord()
Return TRUE
Return TRUE
LexicalAnalyzer()
UpdateClassInformation()
ModifyClassRecord()
Return
Return
Return
Return
Return

Figure 10: Delete Class Diagram

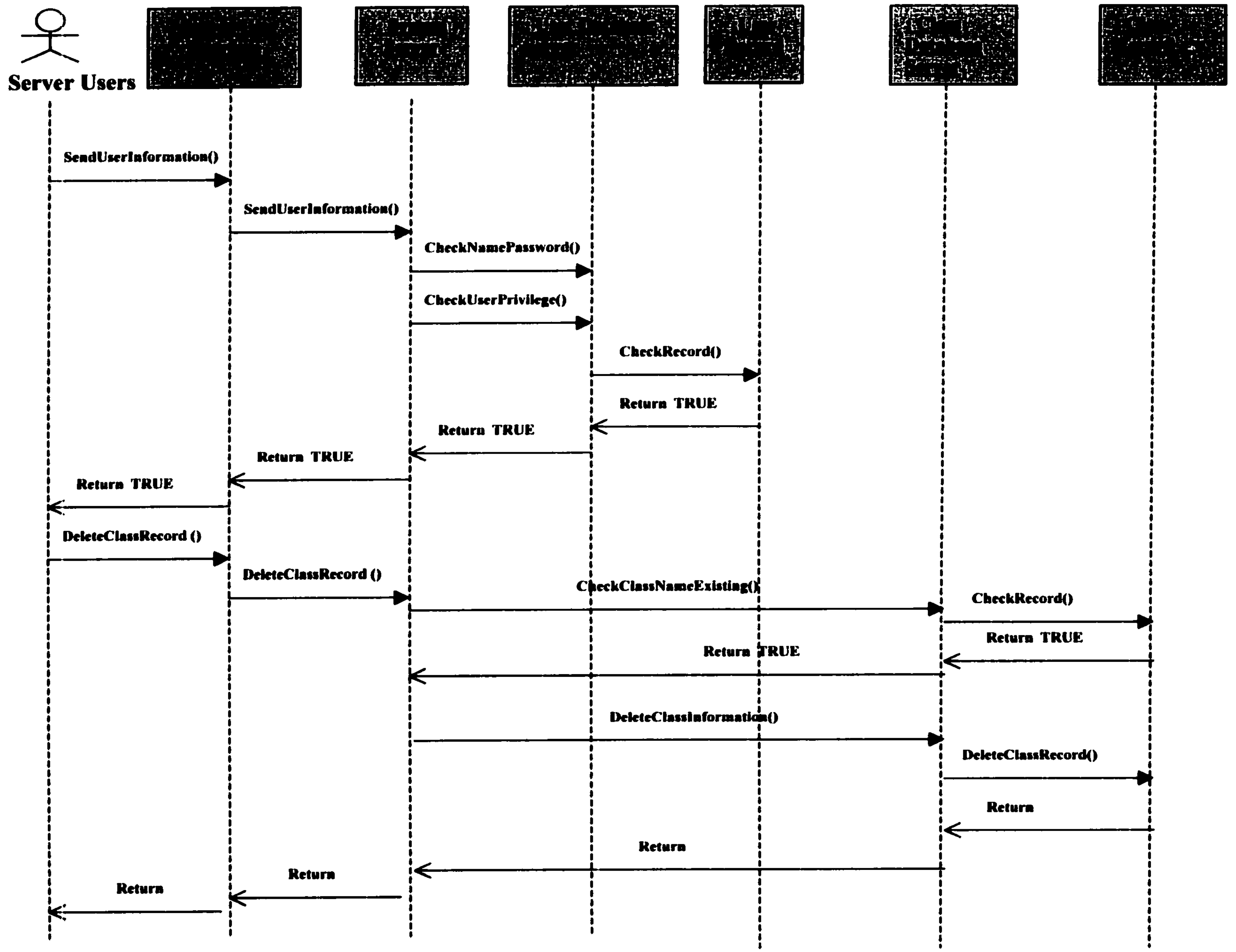
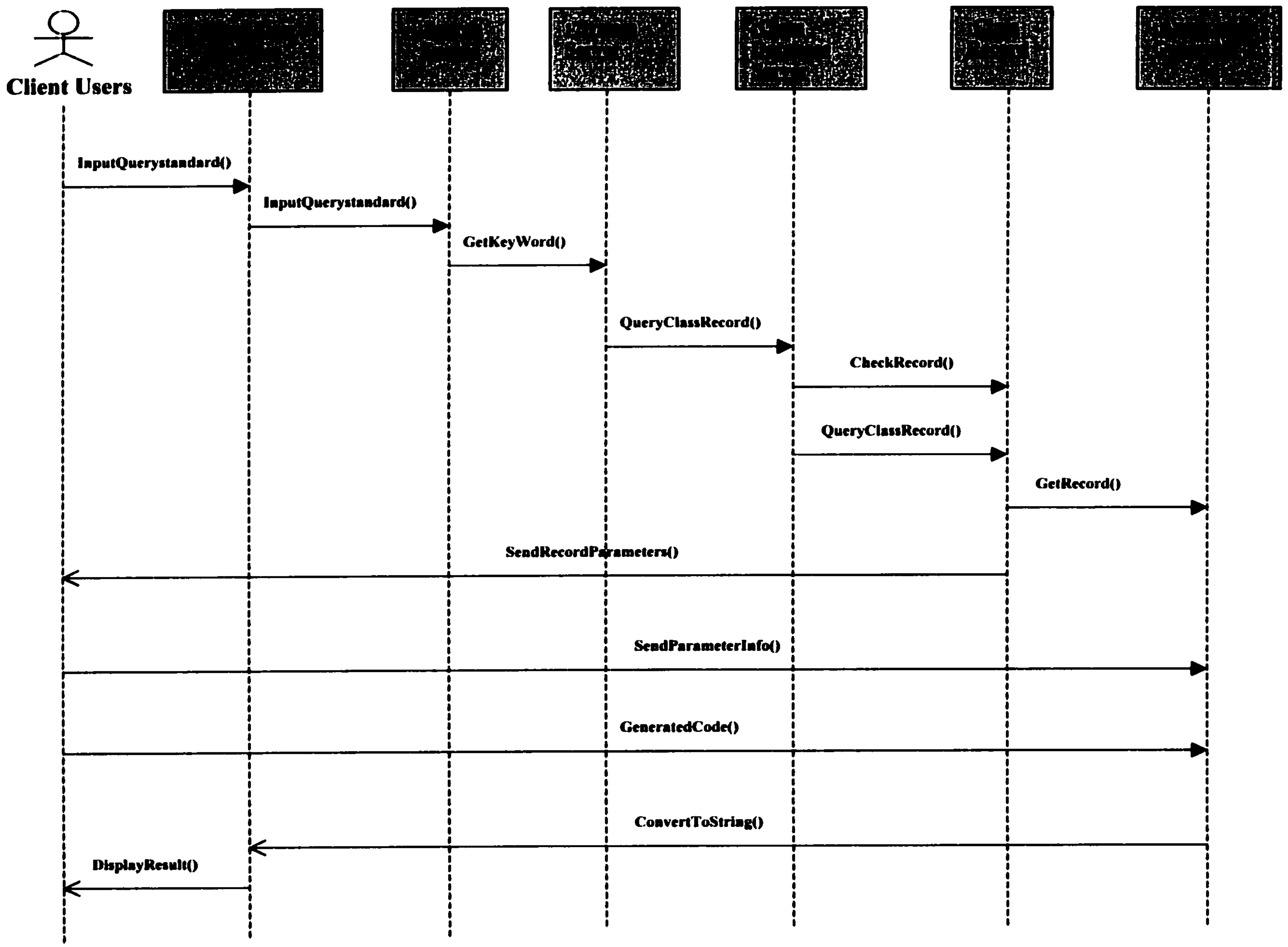


Figure 11: Client-Side Query Class Diagram



4.5. Design Proof and System Testing

CodeNet was developed from a variety of design documents. The documents include the Design Scenario, the Software Requirements Document, and the Design Specification Document. During these activities, system requirements are iteratively defined and recorded as design decisions and implementation descriptions. These include proof of traceability between system requirements and the hardware and software configuration items comprising the design. The levels of system-wide design, system architectural design and detailed design were also documented to provide the CodeNet reasonable design, design refinements, and task implementation.

4.5.1. Design Proof

The following section will analyze and prove the reliability of system design from the eight aspects:

1. Growth capability. CodeNet has the strong growth capability to deal with the different object-oriented language classes, network environments and multiple clients/servers model. But it is a great challenge to expand its library to store components of the other languages (such as functional languages) and fields (such as mathematics).

2. Interoperability requirements. It is an innovation for CodeNet to combine various technologies and co-ordinate them actions well. Moreover, a set of protocols was built for better communication between the client side and the server side by means of CORBA and Java Socket.

3. Portable capability. CodeNet can transplant between heterogeneous platforms because the communication middleware is CORBA and the implementing language is Java. Moreover, it is convenient for end-users to get the required application code without understanding the detailed structure of classes in the library.

4. Safety, security and privacy issues. A user registration module was build for security consideration. Also, the system was programmed by Java language, which enable to interoperate with web browsers within a secure, easy to download environment for the end-users. However, the security problem of Internet communication between Internet users and software reuse library still needs to be paid attention.

5. Stabilization capability. We considered carefully stabilization issues in the system design stage. CodeNet can run normally and correctly if users perform various system operations according to the user manual. Furthermore, the most exceptions have been caught and system will give the warning information for the illegal operation and input. But some bugs and limitations still exist to affect the system stabilization in a low grade.

6. Identification of system module status. All of system modules can be run properly and co-operatively in the distributed environment. But lexical analyzer, application generator and natural language based search engine also need to be further improved to work perfectly.

7. Easiness to operate and understand. CodeNet is always evident to the user what can be done next and how to exit and not let the users do something stupid without warning them. But due to the time limitation, "help" documents weren't completed fully.

8. All interfaces among system modules as well as external system interfaces. Appropriate interfaces were defined among the different system modules. CodeNet can work normally according to the user's requirements. Through utilization of various GUI design technologies such as dynamic index and dynamic button action, all interfaces among system components work perfectly and are convenient to be operated by end-users.

At a word, CodeNet is a potential and prospective product with a reasonable design and architecture. However, it is just situated in the prototype stage. There are also some limitations of the system model.

4.5.2. System Testing

Due to the limitation of time and resources, A brief theoretical testing analysis was presented to aim at its performance and limitations. The correct and reasonable operational behaviors of the system modules, classes, and methods were obtained during the unit testing and debugging in the implementation stage. Some demonstrations, which are described in the following, were performed to test the client and server connectivity and the various services of CodeNet in the distributed environment. The results showed the CodeNet had reasonable operational behaviors of both the client-side and the server-side.

1. Testing environment.

i) Single PC.

Windows 98, ORBacus, and Netscape 4.6 installed with Java 1.2.2 plug-in. *(Note: TCP/IP in the single PC is 127.0.0.1).*

ii) HPC server (IP: 137.207.154.3: hpc.uwindsor.ca).

iii) HPC server and PC.

HPC server (IP: 137.207.154.3: hpc.uwindsor.ca).

PC (Windows 98, ORBacus, and Netscape 4.6 installed with Java 1.2.2 plug-in).

iv) UNIX workstations.

Two UNIX workstations with different IP address.

2. Testing content

System testing focused on the system function testing, which includes the following testing parts:

i) CodeNet services.

- Server-side services: register and log on the system, insert, modify, query, and delete the components.
- Client-side services: browse, query, and generate code.

ii) Different tiers' connectivity, which includes the client side, the server side and the class library.

3. Testing conclusion

Through the function testing, the various operations based on the client and server services were executed successfully, and the communication between the different tiers was performed normally.

4.5.3. Limitations

Based on the observations of the above system testing, some major premises and limitations of the CodeNet were reported in the following. The refinements of the CodeNet were presented the future work.

- When the system server shut down, the client-side user interface, a Java applet embedded in the HTML, still can be downloaded to the client user machine, but no notification was provided to the client-side user.
- The system server could not run without kill the previous processes after the former system server which has already shut down.
- The system could not insert the component with the same name of the already existing components and the component with more than one constructor.
- The limitation terms built in the lexical analyzer limited the query accuracy. Although the strategy was applied to assign the unknown terms to a noun category during the lexical analysis, the system could not tell whether it is a meaningful query input from the user.
- The classification scheme of the component and retrieval fully depended on the indexing sentences describing the component functions obtained during inserting the

component. The meaningless indexing input could not provide a correct classification enabling its later retrieval.

- Currently, the limitation of the code generator is that the code generator only generates the code to deal with single function call, could not deal with the interactive operations among the different functions. The work leaves to the re-user.

4.6. Demonstration and Execution

For reasons of development time and code complexity, the emphasis of system implementation is the whole system structure and major functions. How to use and execute CodeNet and a variety of screen captures of the demonstration programs will be discussed in the next paragraphs.

Currently, CodeNet was installed and tested on the three kinds of platforms, Windows 98/NT, Linux and Unix.

Figure 12 shows in the Windows98/NT environment, three program packages for running CodeNet: (1) "*CodeNet*" that is used to run the server-side operations; (2) "*zzSearch*" that combines a search engine and an application generator in the client side; (3) "*CodeServer*" that is a server program package in the client side. To execute the operation of this system, you just need double-click the three icons.

Figure 12: Program Packages

- The Server-Side Operations

After double clicking the "*CodeNet*" icon, system will execute three programs: (1) the naming service; (2) the library server in the server side and (3) the client interface in the server side.

A main interface will pop up as *Figure 13*. It consists of four buttons:

- User Register: If clicking this button, a frame will pop up for server-side user registration. Please refer to *Figure 14*. After user fills out this form and clicks "submit" button, his individual information will be delivered to the library server, and a system access privilege will be given to him. At the time, the server-side user can access the system operation console by his password and perform various operations with the library.
- Exit: Exit the System.
- About: Provide help and version information.
- Enter System: After clicking this button, an authorization information form will pop up (see *Figure 15*). By inputting the correct name and password, system operation console will pop up (see *Figure 16 and 17*). Otherwise, warning information box will occur.

In the system operation console, a frame is used as the interface to get the user's information and display the feedback from the library server. Two tabs were designed for management of Java and C++ language components separately.

The left upper portion (display panel) composes three panels. The list and tree are used to display the class and function names in the library. The empty text area is used to display the generated application code when user inserts a class into the library. Through the text area, user can know the corresponding source code with his input information.

The right upper portion (information panel) also contains two tabs (see *Figure 16 and 17*). One is class and variable part; the other is constructor and function part. This portion is used to accept the user's input information and display the query result from the remote library server.

The lower portion (control panel) of user interface composes eight buttons for the library operations.

- Insert: After clicking this button, at first, this button label will become to "*Save(I)*". All text in the information panel will be cleared and wait user to input the class information. As soon as user inputs the class information in the information panel, the corresponding program code will be generated and displayed in the text area of the display panel. If user want to input multiple parameters or functions in the variable, constructor and function part, for operating the insert operation correctly, user must click the "save" button (marked with a *disk*) after finishing inputting each parameter. Meanwhile, when user finishes inputting each part of the information panel, he must click the "finish" button (marked with "*pass*"). Otherwise, system will remind user to do these with a message box. The direction arrow buttons (*First, Last, Pervious, Next*) are used to browse all information sequentially. When finishing a whole class input, click "*Save(I)*" and all information will be delivered to the remote library server. After that, system will tell "*insert success*", and the new class and function names will be added to the list and tree in the display panel dynamically.

- Query Class: User must select a class name from the list in the display panel before clicking this button. Otherwise, system will remind by a warning box. After this, the class and constructor information (not include function information) will be displayed in the information panel. Similarly, The direction arrow buttons were used to browse all query results sequentially.
- Query Function: Similar to the "*Query Class*" except that user must select a function name from the tree indices. After that, the function information will be got from the remote library server and displayed at the information panel.
- Modify: This button can be invoked only after user completed the "*Query Class*" or "*Query Function*" operation. After clicking "*Modify*", at first, this button label will become to "*Save(M)*". Then user can modify all class or function information through selecting the item by the direction buttons. When finishing modifying, click "*Save(M)*" and all updated information will be delivered to the remote library server.
- Delete: Delete the class or function information in the remote library after selecting the class or function name from the display panel. After that, system will tell "*delete success*", and the new class and function names can be deleted to the list and tree in the display panel dynamically.
- Restore: Restore to the original status of the server-side interface.
- Help: Display help information.
- Cancel: Return to main interface.

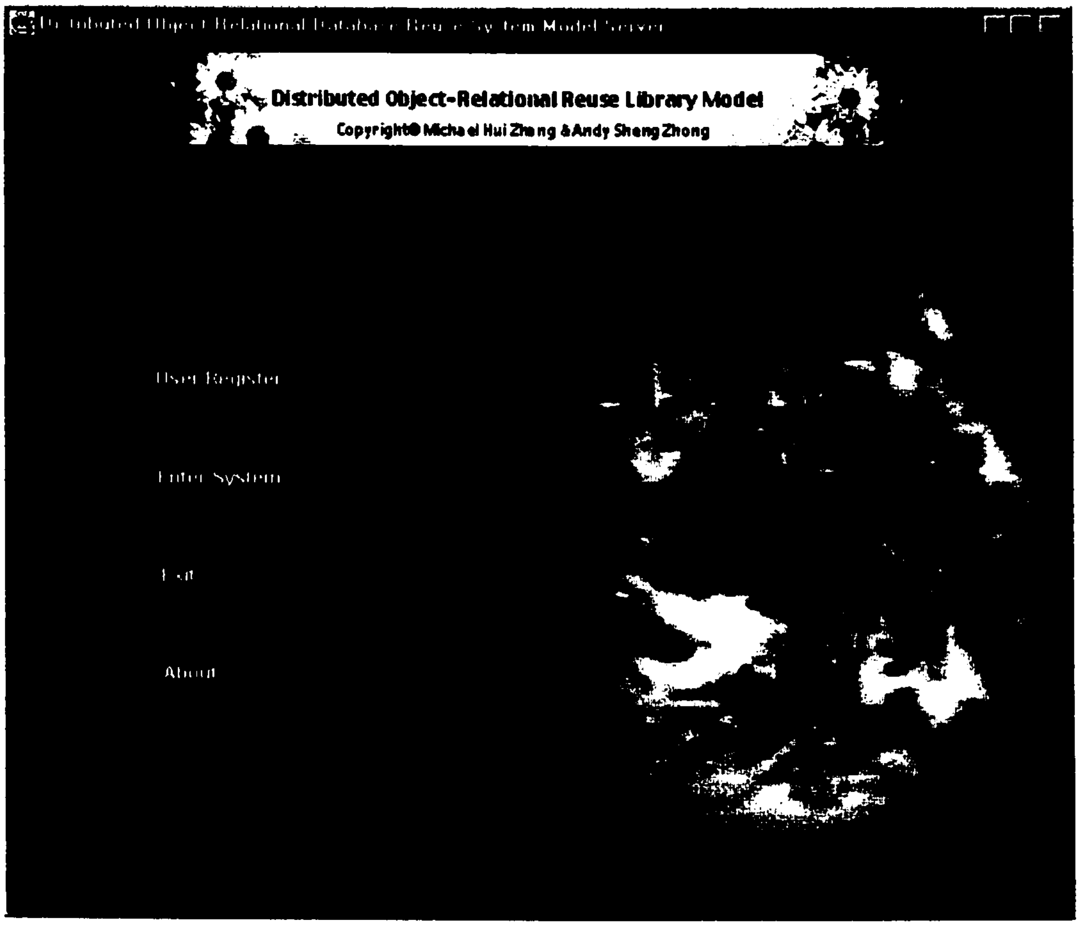

Distributed Object-Relational Database Reuse System Model Server
Distributed Object-Relational Reuse Library Model
Copyright© Michael Hui Zhang &Andy Sheng Zhong
User Register
Enter System
Exit
About

Figure 13: Server Side Main Interface

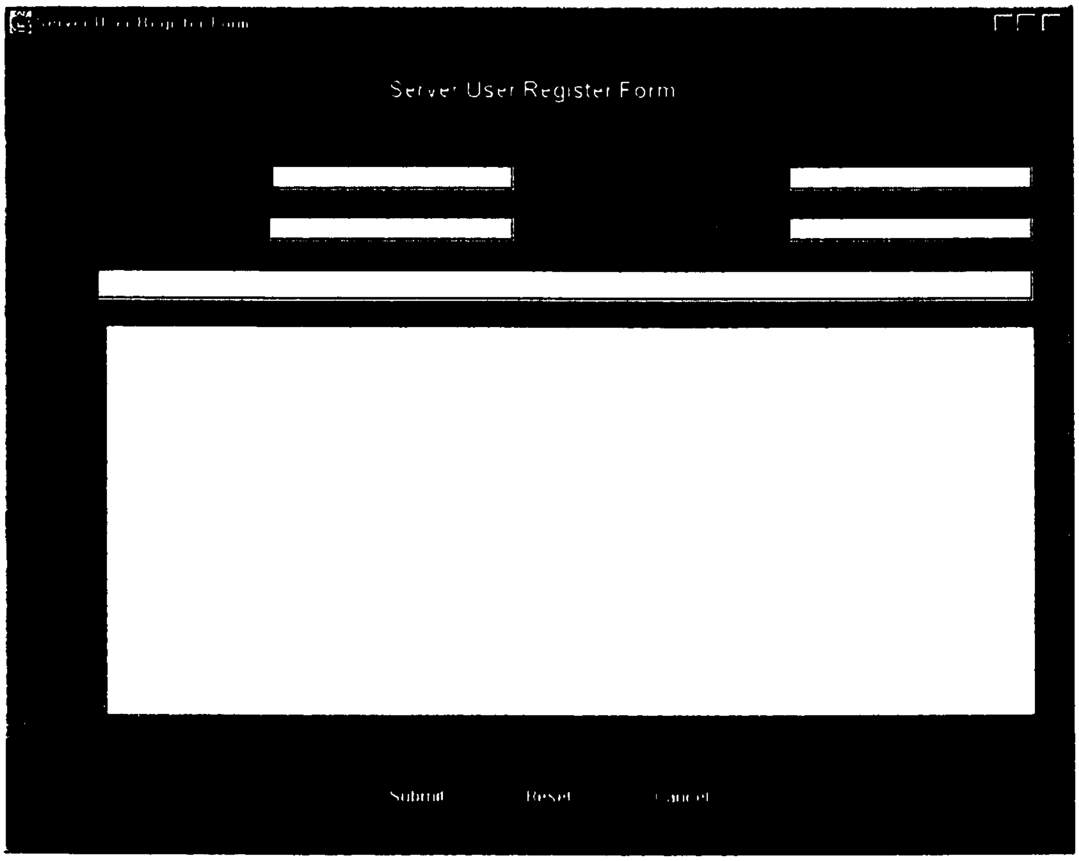


Figure 14: User Registration Form

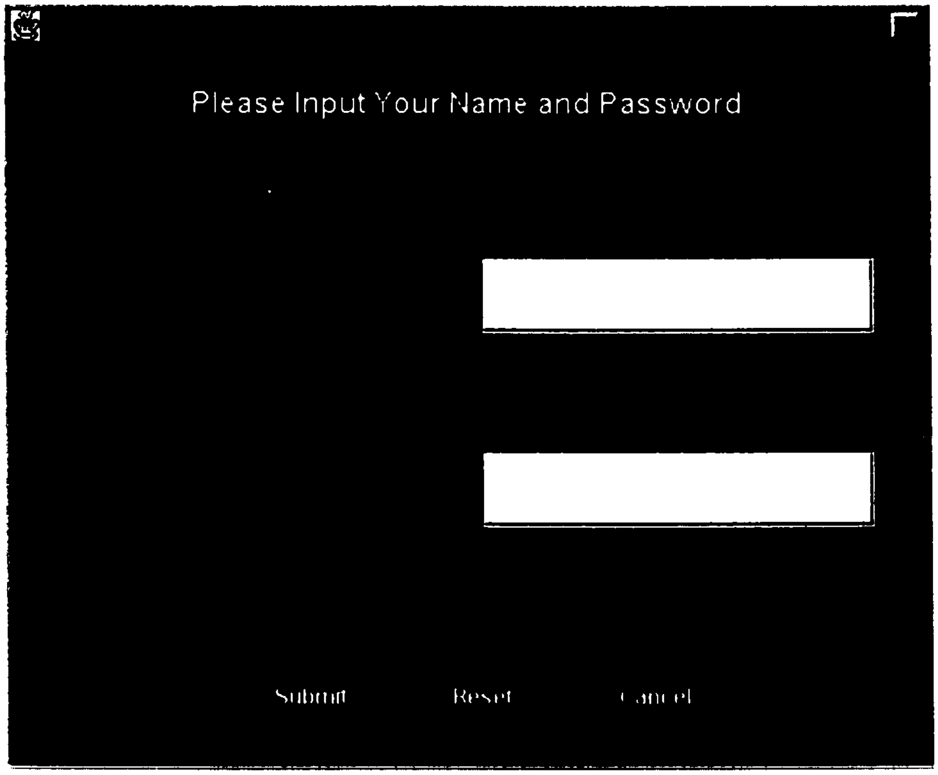


Figure 15: Authorization Form

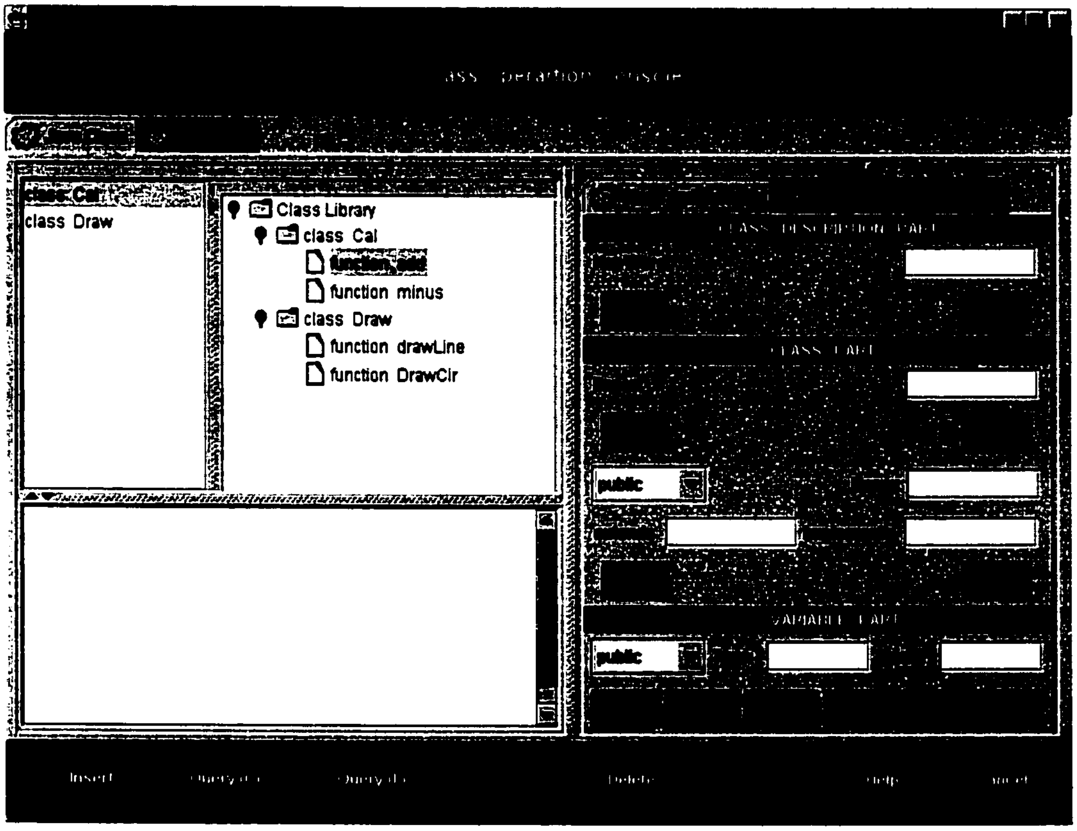


Figure 16: System Operation Console Part 1

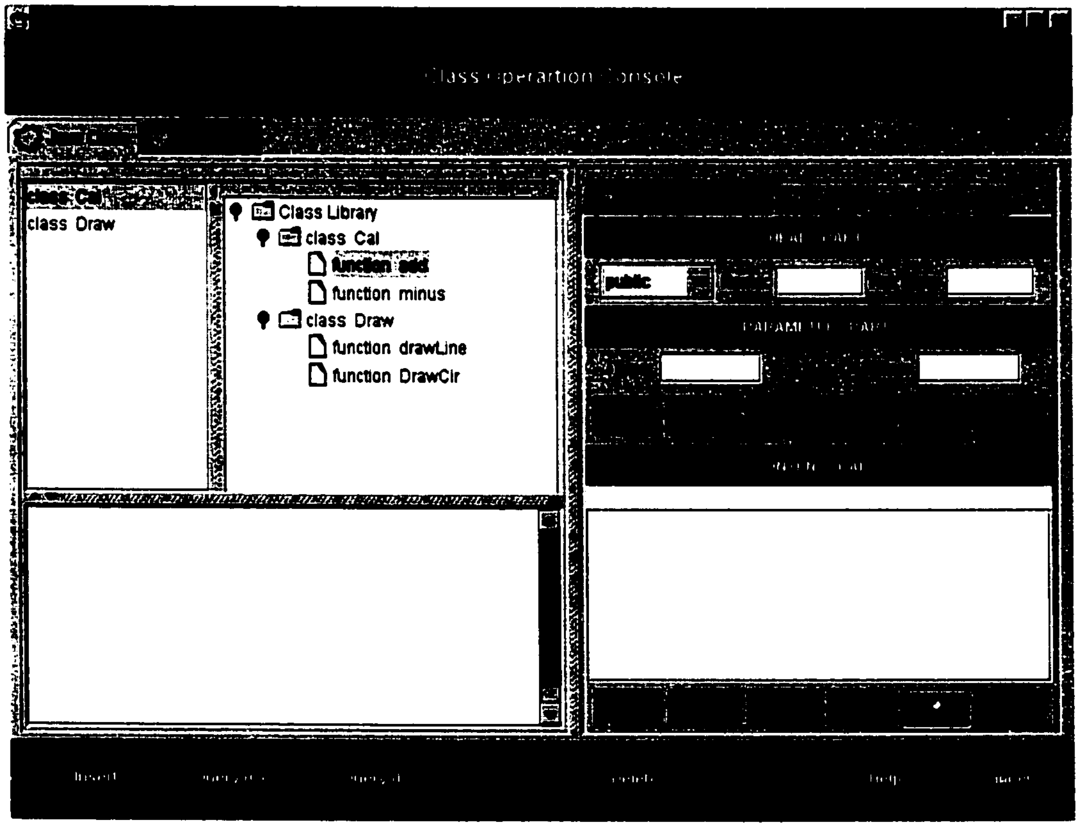


Figure 17: System Operation Console Part 2

- The Client-Side Operations

To run the client-side operations, first of all, double click the "*CodeServer*" icon, system will start the library server program of the client-side. Secondly, double click the "*zzSearch*" icon, system will execute the client applet (see *Figure 17*) in the Netscape.

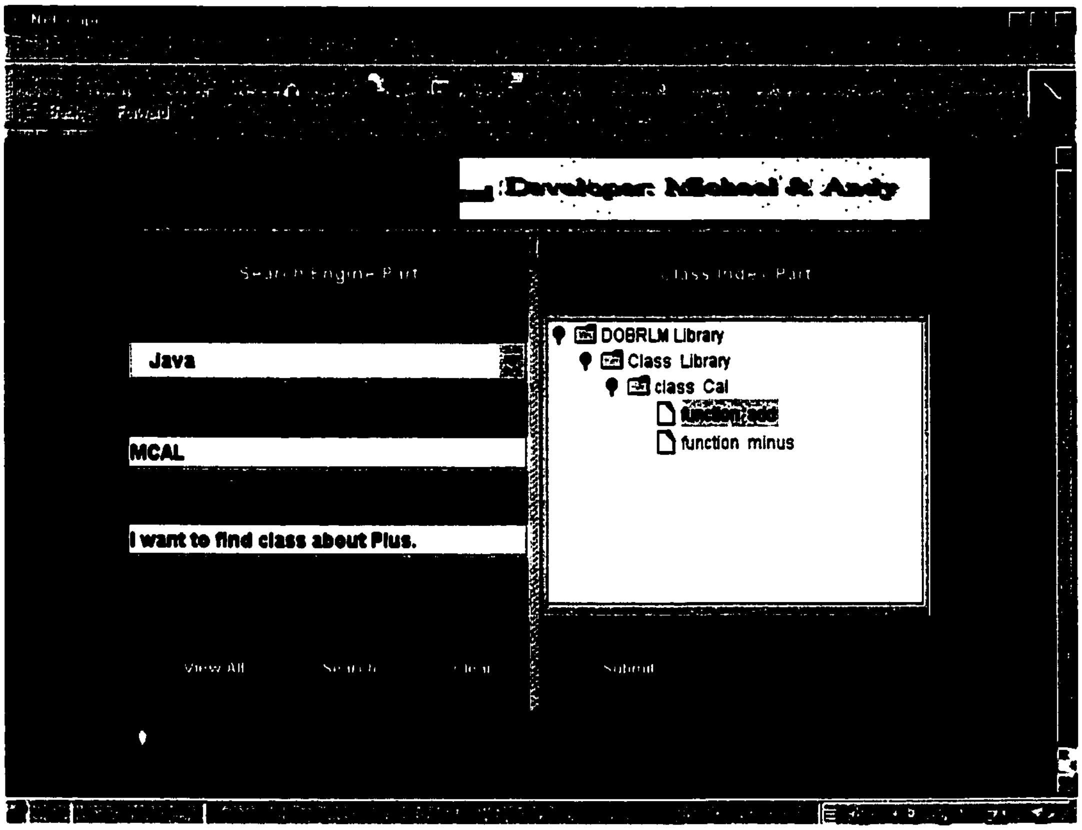
The client applet is further divided into two parts. The left part is the "*search engine*" part that is used to collect user's input and query by the natural language. The right part is the "*class index*" part whose objective is to provide the information for application generator.

In the "*search engine*" part, the first two text fields accept language type and class name of the generated application. The third text field accepts the search standard with the natural language. There were three buttons in the control panel.

- View all: After clicking this button, all class and function names will be got from the remote library server and displayed in the "*class index*" part with the dynamic tree pattern.
- Search: According to the user search standard, and deal with by the search engine described as before, the query results will be got from the remote library server and displayed in the "*class index*" part by a dynamic tree pattern. Meanwhile, the key words of the search standard will be displayed under the query standard text field.
- Clear: Clear the content of text fields and wait for user's input again.

In the "*class index*" part, the display part is a dynamic tree that is used to show the feedback from the remote library server. The control panel includes two buttons:

- Submit: Before clicking this button, user must select class and/or function names from the index tree. Otherwise, a warning box will pop up to remind user. There are three situations after user clicks the "*submit*" button: (1) if user selects a class name, a constructor parameter window will pop up as *Figure 18*; (2) if user selects a function name after finishing inputting the constructor parameters, a function parameter window will pop up as *Figure 19*; (3) if user selects a function name without inputting the constructor parameters, a constructor parameter window will pop up at first, and when finishing inputting the constructor parameters (see *Figure 18*) and clicking the "*Next*" button, a function parameter window (see *Figure 19*) will pop up for user's input. If user want to cancel this action, he can click the "*Back*" button and return to the applet interface.
- Generate Application: After finishing inputting the parameters, this button can be invoked. After clicking this button, an application generation frame (see *Figure 20*) will pop up. The application source code generated by user's requirements will be displayed in the frame.


Developer: Michael & Andy

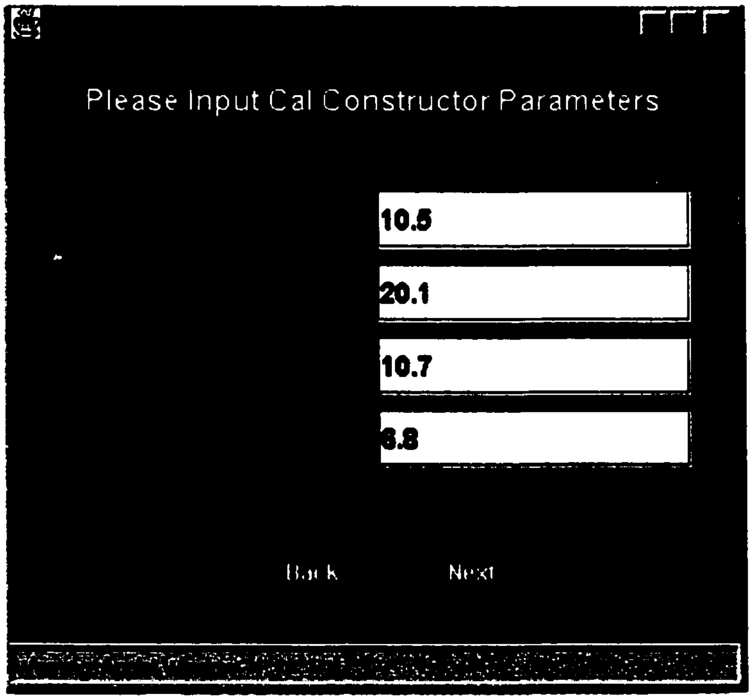
Java

MCAL

I want to find class about Plus.

DOBRLM Library
Class Library
class Cal
function add
function minus

Figure 18: Client Side Main Interface


Please Input Cal Constructor Parameters

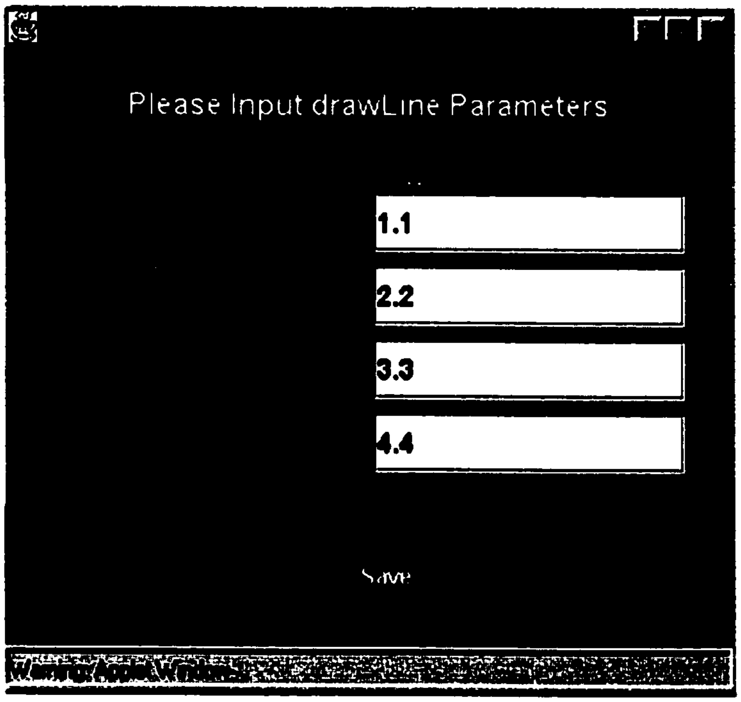
10.5

20.1

10.7

8.8

Figure 19: Constructor Parameter Input Frame



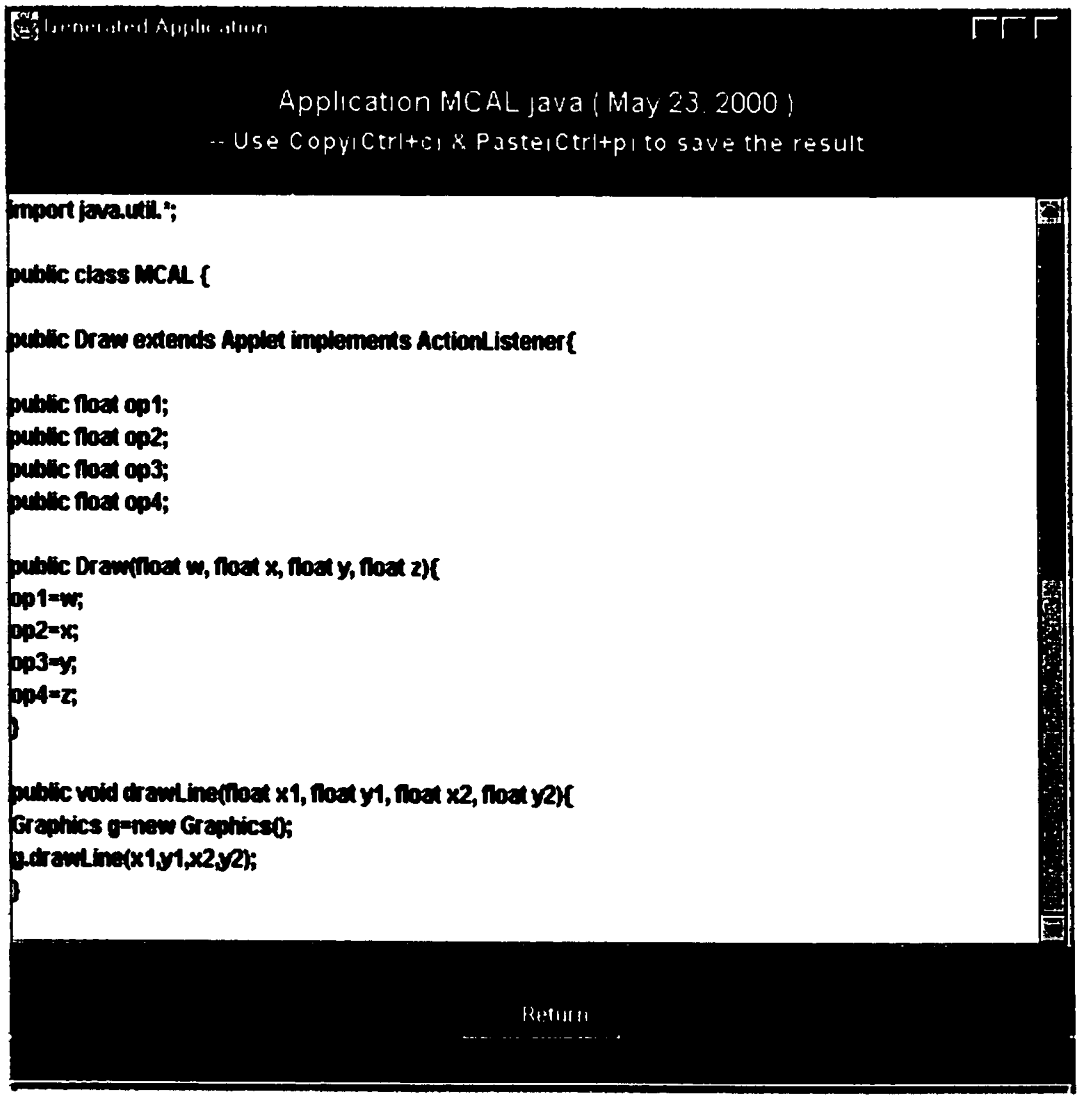
1.1

2.2

3.3

4.4

Figure 20: Function Parameter Input Frame



Application MCAL.java (May 23, 2000)
-- Use Copy(Ctrl+c) & Paste(Ctrl+p) to save the result

```
import java.util.*;

public class MCAL {

public Draw extends Applet implements ActionListener{

public float op1;
public float op2;
public float op3;
public float op4;

public Draw(float w, float x, float y, float z){
op1=w;
op2=x;
op3=y;
op4=z;
}

public void drawLine(float x1, float y1, float x2, float y2){
Graphics g=new Graphics();
g.drawLine(x1,y1,x2,y2);
}
```

Figure 21: Application Generation Frame

To run CodeNet in the Linux and Unix environment, open the three terminals and input the following commands in the appropriate directory of each terminal:

```
Java com.ooc.CosNaming.Server –ORBconfig orb.cfg
Java ServerCLA.Bserver –ORBconfig orb.cfg
Java WebServer.WebServer
```

Note: orb.cfg is an ORB configuration file to specify the IIOP (Internet Inter-ORB Protocol) address of CORBA services.

```
# ORB configuration file
ooc.service.NamService = iioploc://hpc.uwindsor.ca:7721/NameService
ooc.service.EventService = iioploc://hpc.uwindsor.ca:7720/DefaultEventChannel
```

To open the server interface, just input *java CodeNet_SC –ORBconfig orb.cfg* in the appropriate directory of a new terminal or a machine installed the CodeNet, then the server interface shown in the *Figure13* will pop up. To open the client interface, just input the URL address through the Web browser to open the CodeNet web page across the Internet.

CHAPTER 5. Evaluation and Concluding Remarks

5.1. Evaluation of CodeNet

In this thesis, a prototype system was designed and developed in Java language and the feasibility of the distributed library-base software reuse model was investigated by tentative system quality testing. The system is user friendly, and allows users to insert, delete, update and browse the components through various technologies and tools in the server side. By allowing queries in natural language; precisely, by constructing the indexing units of software components with lexical, syntactic and semantic information extracted from the descriptions of software components; and combining a powerful application generator, this model can generate various object-oriented language applications in the general network.

Based on the work carried out in the thesis, a number of conclusions can be formed:

- Using CodeNet with its structure software library provide a better understanding of design and construct a distributed library-based software reuse model.
- CodeNet should be a platform-independent product which integrates lots of technologies such as object-oriented development (OOD), natural language process (NLP), lexical analyzer process (LAP), information retrieval system (IRS) and CORBA, Java database connectivity (JDBC).
- To some degree, the retrieval based on the natural language proposed here will make the reuse model domain-independent and cost-effective.

- The ideal automatic programming generator by retrieving formerly code components that could be used in the generation of the source code increases the translation power for software reuse.
- The system allows user to update library classes. It increases the amount of reuse assets largely and expands this reuse model to a wider field.
- The system is opened to the Internet by using Java language, broker interface and client/server module. All the users can access it from remote area by WWW.

5.2. The Future of Software Reuse

Software reuse has been developing bottom-up (starting with code modules, now expanding to designs and requirements, and architectures in domain analysis), which is consistent with how all engineering disciplines have developed. More mature engineering disciplines are indicators of how reuse will evolve in software engineering. Of course, in software engineering this information will likely be on-line in the form of encyclopaedias /repositories with hypertext links and graphical user interfaces, perhaps as a web-based repository on the Internet. It should eventually include domain models, architectures, patterns, rules, mechanisms, generators, standard domain requirements, designs, code, and tests for widely used domains, as well as process elements and models, results of experiments, case studies, and metrics [GIR93]. This information will gradually become more formalized. An interim step in this transfer of technology may be to set up one or more clearinghouses that collect and provide this information.

For this to happen, the following important technologies will need to mature to support software reuse at a later time:

- Configuration Tracking [DOD95]
- Validation
- Product Modelling
- Architecture Implementation
- Application Instantiation

Configuration tracking is a combination of sub-technologies that range from being fairly mature to progressing reasonably well (distributed systems) to being quite immature (identity and conceptual closeness of adapted assets, determination of when adaptation creates a new asset) [DOD95]. The next item, validation that is very important for software reuse, has been the object of research and development for some time, but has experienced difficulty in reaching maturity. The last three items are the kernel technologies of software reuse, which reflects their dependency on other technologies. To a great extent, they consist of their enabling technologies, and it is believed that their maturity will advance as their enabling technologies mature.

5.3. Future Work

Future work includes the integration of this reuse system in the IDEAL environment and the population of the knowledge base with knowledge and software components. This will allow us to evaluate the effectiveness and experiment with the retrieval system in a real environment.

Furthermore, transferring CodeNet to multiple clients/servers model, the improvement of the lexical analyzer and the complement of application generator are responsible for making and testing. Meanwhile, some more complex situations (e.g. a variety of relationships among library components) and more language kinds should be considered and extended eventually. Concrete applications will be developed by using the distributed environment in order to measure the expected reductions in cost and development time through software reuse.

References:

[ADL95] Adler, R., March, "Emerging Standards for Component Software", *IEEE Computer*, Vol. 28, No. 3, 1995, pp.68-77.

[ANDY00] Andy Sheng Zhong, "Software Library for Reuse-oriented Program Development", *Thesis Report*, 2000.

[ARN92] R.S. Arnold and W.B. Frakes, "Software reuse and reengineering", *CASE Trends*, vol.4, no.2, 1992, pp.44-48.

[ASP93] Arango, G., E. Schoen, and R. Pettengill, "Design as Evolution and Reuse", R. Prieto-Diazand W. B. Frakes (eds.), "Advances in Reuse: Selected Papers from the Second International Workshop on Software Reusability", Lucca, Italy, *IEEE Computer Society Press*, Los Alamitos, CA, March 1993, pp. 9-18.

[BAS90] V.R. Basili, "Viewing maintenance as reuse-oriented software development", *IEEE Software*, vol. 7, no.1, Jan.1990, pp. 19-26.

[BELL92] J. L. Bell, "Reuse and Browsing: Survey of Program Developers", *Object Frameworks*, ed. D. Tsichritzis, Centre Universitaire d'Informatique, University of Geneva, July 1992, pp. 197-220.

[BIG89] T.J. Biggerstaff and A.J. Perlis , *Software Reusability*, vol1. I and II, ACM Press and Addison Wesley, 1989.

[BIRI87] Biggerstaff, T., and C. Richter, "Reusability Framework, Assessment, and Directions", *IEEE Software*, Vol. 4, No. 2, March 1987, pp. 41-49.

[BRAD86] Brad Cox, "Object Oriented Programming: An Evolutionary Approach.", Addison-Wesley Publishing Company, 1986.

[CAL91] G.Caldiera and V.R. Basili, "Identifying and qualifying reusable software components", *IEEE Computer*, vol. 24, no. 2, Feb. 1991, pp. 61-70.

[CAN94] G. Canfora, A. Cimitile, and M.Munro, "RE^2: reverse engineering and reuse reengineering", *Journal of Software Maintenance: Research and Practice*, vol. 6, no.2, 1994, pp.53-72.

[CAN95] G. Canfora, A.Cimitile, and G.Vissaggio, "Assessing modularization and code scavenging techniques", *Journal of Software Maintenance: Research and Practice*, vol.7, no.5, 1995, pp.317-331.

[CHI90] E. Chikofsky and J.Cross II, "Reverse Engineering and Design Recovery: A Taxonomy", *IEEE Software*, 7(1), Jan. 1990, pp. 13-17.

[CORBA99] "Overview of the CORBA Component Model", Object Management Group, 1999.

[DOD93] Department of Defense, "Software Reuse Initiative", *Virtual Library Operational Concept Document*, Washington, DC, 1993.

[DOD95] Department of Defense Software Reuse Initiative, *Technology Roadmap*, vol.1, "Technology Assessment", 1995.

[DUNN93] M.F. Dunn and J.C. Knight, "Automating the detecting of reusable parts in existing software", *Proceedings of the 15th International Conference on Software Engineering*, Baltimore, Maryland, U.S.A., IEEE Comp. Soc. Press, 1993, pp. 381-390.

[FISHER] G. Fisher, A. Girgensohn, K. Nakakoji and D. Redmiles, "Supporting Software Designers with Integrated Domain-Oriented Design Environments", *IEEE Transactions on Software Engineering*, 18(6), pp 511-522.

[FRAKES94] W.B. Frakes and S. Isoda, "Success factors of systematic reuse", *IEEE Software*, vol. 11, no.5, Sep. 1994, pp.15-19. Introduction to the special issue on "Systematic Reuse".

[GALL95] Harald Gall, Mehdi Jazayeri, Rene Klosch, "Research Directions in Software Reuse: Where to go from here?", *ACM Press, Proceedings of the Symposium on Software Reusability SSR'95*, pp. 225-228.

[GEP00] Paul G. Brown, "Distributed Component Database Management Systems", Morgan Kaufmann Publishers, 2000.

[GIR92] M.R. Girardi, "Application Engineering: Putting Reuse to Work", *Object Frameworks*, ed. D. Tsichritzis, Centre Universitaire d'Informatique, University of Geneva, July 1992, pp.137-149.

[GIR93] M. R. Girardi and B. Ibrahim, "A Software Reuse System Based on Natural Language Specifications", *Fifth International Conference on Computing and Information*, Sudbury, Ontario, Canada, May 27-29, 1993.

[GIR95] Maria del Rosario Girardi, "Classification and Retrieval of Software through their Description in Natural Language", *Ph.D.Thesis*, Computer Science Department, University of Geneva, Switzerland, 1995.

[GIR98] M. R. Girardi and B. Ibrahim, "New Approaches for Reuse Systems", *2nd International Workshop on Software Reuse,* Lucca, Italy, March 24-26, 1998.

[GOG89] Goguen, J., "Principles of Parameterized Programming," T. Biggerstaff and A. Perlis (eds.), *Frontier Series: Software Reusability, Volume I -- Concepts and Models*, Addison-Wesley, New York, 1989, pp. 159-225.

[GRI94] Griss, M. L., "PANEL: Architecting Kits for Reuse", *Frakes, W. (ed.), Proceedings, Third International Conference on Software Reuse: Advances in Software Reusability, IEEE Computer Society Press*, Los Alamitos, CA, 1994, pp. 216-217.

[HALL] P.A.V. Hall, "Software reuse, reverse engineering and reengineering", *Software Reuse and Reverse Engineering*, P.A.V. Hall, Chapman & Hall, London, pp.3-31.

[HEL91] R. Helm and Y.S. Maarek, "Integrating Information Retrieval and Domain Specific Approaches for Browsing and Retrieval in Object-Oriented Class Libraries", *Proc. OOPSLA/ECOOP '91, ACM SIGPLAN Notices*, vol. 26, no 11, Nov 1991, pp. 47-61.

[HEN95] Henda Hadjami Ben Ghezala, Farouk Kamoun: "A Reuse Approach based on Object Orientation. Its Contributions in the Development of CASE Tools", *ACM Press, Proceedings of the Symposium on Software Reusability SSR'95*, pp.53-61.

[JAC97] Ivar Jacabson, Martin L. Griss and Patrik Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success*, Addison-Wesley, 1997.

[JF98] Johnson, R. E., and B. Foote, "Designing Reusable Classes", *Journal of Object-Oriented Programming*, Vol. 1, No. 2, June-July 1998, pp. 22-35.

[JOE00] Joe Abounader, "Software Reuse and Object Orientation", Queen's University, 2000.

[KIM90] H. Kimoto and T. Iwadera, "Construction of a Dynamic Thesaurus and Its Use for Associated Information Retrieval", *Proc. 13th International Conference on Research and Development in Information Retrieval (SIGIR'90)*, Brussels, 5-7 Sept. 1990, pp 227-239.

[LEO99] Zhengyu Leo Wu, "Software Reuse and World Wide Web", University of Windsor, 1999.

[LIM94] M.C. Lim, "Effects of reuse on quality, productivity, and economics", *IEEE Software*, vol. 11, no.5, Sept.1994, pp.23-30.

[MAA91] Y. Maarek, D. Berry and G. Kaiser, "An Information Retrieval Approach For Automatically Constructing Software Libraries", *IEEE Trans. on Software Engineering*, Vol. 17, no. 8, Aug.1991, pp. 800-813.

[MCC94] Carma McClure, "Reuse: Re-engineering the Software Process", Extended Intelligence, Inc., 1994.
[MCC95] Carma McClure, "Model-Driven Software Reuse", Extended Intelligence, Inc., 1995.

[MCC96] Carma McClure, "Experiences from the OO Playing Field", Extended Intelligence, Inc., 1996.

[MCC98] Carma McClure, "Reuse Engineering: Extending Information Engineering to Enable Software Reuse", Extended Intelligence, Inc., 1998.

[MCC99] Carma McClure, "Value-Added Year 2000: Harvesting Components for Reuse", Extended Intelligence, Inc., 1999.

[MCM00] Chuck McManis, "Lexical analysis and Java", *JavaWorld*, 2000

[MEYER94] Bertand Meyer, *Reusable software: The Base Object-Oriented Component Libraries*, Prentice-Hall, 1994.

[MIC86] R. Michalsky, "Learning Strategies and Automated Knowledge Acquisition: an Overview", *Computational Models of Learning*, Berlin, Springer-Verlag, 1986.

[ORACLE99] "Oracle8 Release 2 New Features Summary", Oracle Corporation, 1999.

[PAT99] Pat McCarthy, *VisualAge Generator Templates: Creating a Well-Tailored and Customized Solution*, IBM Corporation, 1999.

[PER92] D. Perry and A. L. Wolf, "Foundations for the Study of Software Architecture", *ACM Software Engineering Notes*, Vol. 17, #4, 1992.

[POU95] Jeffrey S. Poulin and Keith J. Werkman, "Software Reuse Libraries with Mosaic", *Software Reuse Conference '95*, 1995.

[PRI90] T. Price and M. R. Girardi, "A class retrieval tool for an object-oriented environment", *Proc. 3rd Int. Conf. Technology on Object-Oriented Languages and Systems*, Sydney, 28-30 Nov. 1990, pp 26-36.

[PRI91] R. Pricro-Diaz, "Making software reuse work: an implementation model", *ACM SIGSOFT Software Engineering Notes*, vol.16, no.3, 1991, pp. 61-68.

[RON94] Boisvert, Ronald F., "A Web Gateway to a Virtual Mathematical Software Repository", *2nd International World Wide Web Conference: Mosaic and the Web*, Chicago, Illinois, 17-20 October 1994.

[ROY95] Roy Rada, "Software Reuse", *Intellect Press*, Oxford, England. 1995.

[SAR99] Marjan Sarshar (Ed.), *Systematic Reuse: Issues in Initiating and Improving a Reuse Program*, Springer, 1999.
[SCOTT00] Scott Stanchfield and Terence Parr, "Parsers, Part IV: A Java Cross-Reference Tool", MageLang Institute, 2000.

[SIT96] Murali Sitaraman (Ed.), Fourth International Conference on Software Reuse - April 23-26, 1996, *IEEE Computer Society Press*, 1996.

[SNE94] H.M. Sneed and E. Nyary, "Downsizing large application programs", Journal of software Maintenance: Research and Practice, vol.6, no.5, 1994, pp. 105-116.

[SNE95] H.M. Sneed, "Planning the reengineering of legacy systems", *IEEE Software*, vol. 12, no.1, Jan.1995, pp. 24-34.

[SOM89] I. Sommerville, *Software Engineering*, Addison-Wesley, 1989.

[SUN00] "Sun System Java Homepage", Http:// www.sun.com/sunsoft/hotjava.

[UML97] "UML Summary", Rational Software Corporation, 1997.

[URBAN96] M. L. Urban and I. N. Chang, "Information Architecture as a Framework for Reuse", 1996.

[WAL92] Wallnau, K. C., 1992, "Towards an Extended View of Reuse Libraries", *Proceedings of the 5th Annual Workshop on Software Reuse* (WISR5), Palo Alto, CA.

[WEI92] B.W. Weide, "Scalability of reuse Technology to Large Systems Requires Local Certifability", *Software Reuse Conference '92*, July, 1992.

[WEI94] B.W. Weide and W.D. Heym, "Reverse Engineering of Legacy Code is Intractable", *Ohio State University Technical Report OSU-CISRC-10/94*, 1994.

APPENDIX: CodeNet Source Code Definition

This appendix contains the CORBA IDL and configures classes, the Java packages and classes for the CodeNet Source code. The names of the packages and Java classes are appeared in italic.

1. CORBA IDL: *zz.idl*

This file contains the IDL interfaces of the CodeNet.

```
module CodeNet{

        // user register
        boolean userRegister(in string name,
                             in string pass,in string email,in string phone,
                             in string address,in string exprience);

        // user login
        boolean userLogin(in string userName, in string password);

        // modelling the components
        typedef string classFeature;
        typedef string package;

        // import list
        typedef string import;
        typedef sequence<import> import_list;

        // class head
        struct ClassHead{
                string parent;
                string attribute;
                string name;
         };

        // implement part
        typedef string implement;
        typedef sequence<implement> implement_list;
```

```
// variable list
struct variable{
        string attribute;
        string name;
};
typedef sequence<variable> variable_list;

// constructor part
struct constructor_para{
        string attribute;
        string name;
};
typedef sequence< constructor_para > constructor_para _list;

struct constructor{
        string attribute;
        string name;
        constructor_para _list para;
        string body;
};

// function functionalities and function part
struct fun_para{
        string attribute;
        string name;
};
typedef sequence< fun_para > fun_para _list;

struct function{
        string function;
        string attribute;
        string name;
        fun_para _list para;
        string body;
};
typedef sequence< function > function _list;

// insert the components
boolean insert(in long lanID, in string classFeature, in string classPackage, in
                import_list import, in  Class_Head head, in implement_list
                implement, in variable_list variables, in constructor con, in
                function_list fun );

// get class indices
typedef string class_fun;
typedef sequence<class_fun> class_funList;
```

```
        class_funList getIndex (in long userID, in long lanID);
        // delete the component
        boolean delete(in string className, in string funName);

        // query function
        typedef string function;
        typedef sequence<function> funList;
        funList getFun( in long lanID, in string className, in string funName);

        // query class
        typedef string className;
        typedef sequence<className> classList;
        classList getClassDB(in long lanID, in string className);

        // modify function
        boolean modifyFun (in function, in long lanID);

        // modify class
        boolean modifyClass (in long lanID, in string classFeature, in string classPackage,
                             in import_list import, in Class_Head head, in
                             implement_list implement, in variable_list variables, in
                             constructor con, in function_list fun);

        // retrieval, browse, and generator
        void getRetrieval ( in string text);
        typedef string index;
        typedef sequence<index> index_list;
        index_list query ( in long lanID);
        index_list browse (in long lanID);

        void specify (in string name);
        string generation ();

};
```

2. ORB configuration File: *orb.cfg*

```
# ORB configuration file

ooc.service.NameService=iioploc://hpc.uwindsor.ca:7721/NameService

ooc.service.EventService=iioploc:// hpc.uwindsor.ca :7720/DefaultEventChannel
```

3. CodeNet Server main program: *CODENET_SC.java*

This is the CodeNet server-side user interface. Run this class, can open the server-side main user interface.

4. Package: *ServerREG*

This package contains classes that deal with the user register and log in to the CodeNet.

4.1. Classes

- This is the GUI definition of the server user main interface.

  ServerGUI

- These classes are defined in hierarchy to define the GUI of the server user register

  ServerUserReg
    - ***TopPanel***
    - ***CenterPanel***
    - ***ButtonPanel***

- These classes are defined to perform the new user-registering request and connect to the system server of the DORLM.

  RegUser
    - ***CorbaProcess***

5. Package: *ServerCLA*

This package contains classes that have the major functions of the CodeNet. The functions include implementation of the IDL, getting the indices, inserting the component, querying the class, querying the method, modifying the class, modifying the method, deleting the class, and deleting the method.

5.1 Classes

- These classes are defined to implement the ***zz.idl*** interface to serve as the system server.

  Server_impl
      Connect_DB
      Insert_User
      Login

- This class is defined to bind the system server.

  BindingServer

- This class is defined to deal with the user register and login to the CodeNet in the system server side.

  ServerPass

- These classes are defined to deal with the partial natural language process of translating the component classification sentences and the user query sentence to a frame-based internal representation, the component index storage, and the synonyms operation.

  GetLexicon
      GetTerm
      Parse
      GetSyn

- These classes are defined to build various graphic user interfaces for the server-side user.

  SCClass
  SclientClass
  SCInput
      SCList
          SplitPaneDemo
              SCCTree
  SCControl
  DialogWindsow

DialogWindow2
App_Err_Window

- These classes are defined to perform various operations for the server-side user.

  InsertClass
  BuildTotalData
  LexicalAnalysis
  DeleteClass
  GetListInit
  GetUserID
  ModifyFun
  ModifyCla
  QueryClass
  QueryFun
  WriteIDClass

6. Package: *WebServer*

This package contains classes that have the major functions of the web server of the CodeNet. The functions include client-side connection, system server connection, database server connection, multithread handle, and various operations for the client-side user requests.

6.1. Classes

- These classes are defined to run the web server, handle multithread, and connect to the database server.

  WebServer
  WebServerThread
  Connect_DB

- These classes are defined to perform various operations for the web client user requests in the web server side.

  GetClassIndex
  GetConPar
  GetFunPar

TrimTerm
LexicalAnalysis
AppGen

7. Package: *ClientCLA*

This package contains classes that define the graphics user interfaces for the client-side user and perform various operation requests in the web client side.

7.1 Classes

- These classes are defined to build various graphic user interfaces for the web client user.

```
ClientSearch
    QuerySplit
            CCTree

ClaWindow
    ConWindow
            FunWindow
                    Fun2Window
App_ErrWindow
```

- These classes are defined to perform various operation requests for the web client user to browse the SRL, query the SRL, and code generation.

ConnectWebServer
GetInit
Lexica
AppResult

8. HTML

- This is a web page of the CodeNet embedded with the Java applet, ***clientSearch.class.***

  zzSearch.html

Vita Auctoris

NAME:	Michael Hui Zhang
PLACE OF BIRTH:	Hunan Province, P.R. China
DATE OF BIRTH:	July 1st, 1972

EDUCATION:

- M. Sc., Candidate, Department of Computer Science, University of Windsor, Canada, (2000).
- B. Sc., Department of Computer Science, Changsha Institute of Technology, China, 1994.