

2000

Software library for reuse-oriented program development.

Sheng, Zhong
University of Windsor

Follow this and additional works at: <http://scholar.uwindsor.ca/etd>

Recommended Citation

Zhong, Sheng., "Software library for reuse-oriented program development." (2000). *Electronic Theses and Dissertations*. Paper 3509.

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

INFORMATION TO USERS

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

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

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

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

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

**ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

**SOFTWARE LIBRARY FOR REUSE-ORIENTED PROGRAM
DEVELOPMENT**

By

Sheng Zhong

**A Thesis
Submitted to the College 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**



**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-62310-6

Canada

Sheng Zhong, 2000

© All Rights Reserved

Abstract

Distributed system or reuse-oriented program development system may call for software reuse library (SRL, repository) to serve as a resource provider by the usage of reusable software components. We are trying to solve the problems of storing and classifying, locating/retrieving, and delivering the large number of software components through the SRL in an effective way.

In this thesis we report on the design and construction of a prototype software system, DORLM (Distributed Object-based Software Reuse Library Module), used to investigate the integration of DBMS (database management system), IRS (information retrieval system), NLP (natural language process) and CORBA (Common Object Request Broker Architecture) for software reuse and reuse-oriented program development in a distributed computing context. The DORLM provides an effective way to store, retrieve, and deliver reusable software components as an aid of reuse-oriented program development in the distributed environment.

**To my parents,
my brother,
my teachers,
and my wife Lijun,
For their love and support**

Acknowledgements

I would like to acknowledge the support and guidance provided by Dr. R. Kent, whose time, dedication and effort has contributed in guiding me through my study and thesis work. Without his patience and guidance it would have been impossible to complete this thesis in this time frame.

I am deeply appreciative of Dr. Chen for being my internal reader and her invaluable suggestions and comments.

I would specially like to thank for the important comments and advice given by Dr. Schlesinger who is my external reader.

Special thanks to Dr. Walid Saba for being willing to serve as the chair of the committee.

Finally, a big thank-you to my wife Lijun, for her patience, encouragement, support and understanding.

TABLE OF CONTENTS

Abstract	iv
Acknowledgements	vi
List of Figures	x
List of Tables	xiii
1 Introduction	1
1.1 Overview of Software Reuse	1
1.2 Reuse-oriented Software Development Process	2
1.3 Current Problems and Limitations of Software Reuse	4
1.4 Major Component Classification and Retrieval Approaches	6
1.4.1 Descriptive Classification Methods	6
1.4.2 Behavior-based Retrieval Methods	8
1.4.3 Denotational Semantics Methods	8
1.4.4 Formal Specification Methods	9
1.5 Code Generator for Software Reuse	10
1.6 Overview of This Thesis	13
1.6.1 Motivation	13
1.6.2 The Objective of This Thesis	13
1.6.3 Overview of the Proposed Approaches	14
1.7 Organization of This Thesis	17
2. Approaches to Classify, Retrieve, and Deliver Software Components	19
2.1 The Component Specification Acquisition	19
2.1.1 Component Structure	19

2.1.2 Component Classification	21
2.2 Automatic Indexing Process	22
2.2.1 Semantic Case System	22
2.2.2 A Process for Lexical, Syntactic and Semantic Analysis	26
2.2.3 Lexicon	30
2.3 The Retrieval Approach	30
2.4 An Approach to Deliver Reusable Component for Reuse	32
2.4.1 Requirements Phases and Methodology	32
2.4.2 Code Generator Process	33
3. Prototype Design and Implementation: DORLM	35
3.1 System Overview	35
3.2 System Structure	37
3.3 Development Environment	39
3.4 System Design Specification	41
4. Using Prototype System and User Interfaces	53
4.1 System Setup	53
4.2 Server Interface	55
4.2.1 Log on the System	56
4.2.2 Server Operation Interface	57
4.2.3 Insertion of Components	59
4.2.4 Server User Query Components	60
4.2.5 Server User Modify Components	62
4.3 Client Interface	63

4.3.1 Browse the Library	63
4.3.2 Client User Retrieval of Components	64
4.3.3 Code Generation	65
4.4 System Testing and Limitations	68
4.4.1 Overview of Testing	68
4.4.2 Limitations of the DORLM	71
5. Conclusion and Future Work	73
5.1 Conclusion	73
5.1.1 Innovation	73
5.1.2 Achievements	74
5.2 Future Work	74
APPENDIX A: The DORLM Code Definition	76
BIBLIOGRAPHY	84
VITA AUCTORIS	90

List of Figures

Figure 1	Reuse-oriented software development process	3
Figure 2	The semantic case system	23
Figure 3	Partial Natural Language-based Retrieval System Process	31
Figure 4	Forms-based Code Generator Process	34
Figure 5	Architecture of the DORLM	36
Figure 6	DORLM Use Case Diagram	42
Figure 7	Server User Use Case Diagram	42
Figure 8	Client User Use Case Diagram	43
Figure 9	Main DORLM Class Diagram	46
Figure 10	User Register Sequence Diagram	47
Figure 11	Client User Query Library Sequence Diagram	48
Figure 12	Server User Insert the Component Sequence Diagram	49
Figure 13	Server User Update the Component Sequence Diagram	50
Figure 14	Server User Delete the Component Sequence Diagram	51
Figure 15	DORLM setup package in Windows 98/NT	54
Figure 16	The server user interface	55
Figure 17	The server user register interface	56
Figure 18	The server user login dialog	57
Figure 19	The server operation interface	58
Figure 20	Insert class component interface 1	59
Figure 21	Insert class component interface 2	60
Figure 22	The result of a sample class query	61

Figure 23	The result of a sample method query	61
Figure 24	A sample modify process	62
Figure 25	The main client user interface	63
Figure 26	The result of the library browsing	64
Figure 27	A sample query and query result interface	65
Figure 28	A sample code generator process 1	66
Figure 29	A sample code generator process 2	67
Figure 30	The result of a sample generated code	67

List of Tables

Table 1	Main semantic cases describing software components	25
Table 2	Case triggers in the case system	25
Table 3	Parsing grammar of case parsing process	28

CHAPTER 1.

Introduction

In this thesis we report on the design and construction of a prototype software system, DORLM (Distributed Object-based Software Reuse Library Module), used to investigate the integration of DBMS (database management system), IRS (information retrieval system), NLP (natural language process), and CORBA (Common Object Request Broker Architecture) for software reuse and reuse-oriented program development in a distributed computing context.

1.1 Overview of Software Reuse

Software reuse is defined as the “process of using existing software components rather than building them from scratch” [KRUEG92]. Reusable software components include not only the generic source code, but also other aspect of the software lifecycle including design structure, specifications, and documentation. However, this thesis work focuses mainly on developing a software reuse library system to reuse object-oriented source code, so called reuse-oriented software development.

Software reuse library (SRL, software repository, software base, etc.) is a set of software components that use specialized methods for reusable components classification, storage, retrieval, and delivering [LIAO93] [AMILI98] [ARABA98] [LUO99].

Software reuse has been recognized as one of most the realistic and promising ways to improve software productivity, quality and reliability, reduce maintenance costs and shorten the time required to release software for client use.

While software reuse has been under investigation for about two decades, it still remains an active area of research field [AMILI98]. This can be explained by the observation that new technologies, such as distributed computing, World Wide Web, new programming languages (e.g. Java) and paradigms (e.g. distributed objects), keep opening new opportunities for posing new technical challenges.

1.2 Reuse-oriented Software Development Process

Reuse-oriented software development is a kind of software reuse approaches ¹ that makes use of a software reuse library from which reusable components may be extracted. Figure 1 shows this general idea. It involves development-for-reuse when reusable components are created and brought together into a software reuse library; and development-with-reuse when reusable components are selected from the library on the basis of specific requirements and reused in the construction of a new software component.

Also noted in Figure 1, the reuse-oriented software development process is composed of three distinct phases: the construction/storage phase, the location/retrieval phase and the adaptation/generation phase.

The first phase deals with the construction and storage of components in the SRL in a way enabling their later retrieval. Before we can put components into the library, we

¹ Literature often makes a distinction between two general approaches for software reuse: the generative approach [AMILI95], and the building blocks approach [KRUEG92]. The generative approach consists of reusing the process of developing software, as embodied in things such as program generators, fourth generation's tools, executable specification language interpreters, and transformational systems. The building blocks approach consists of reusing the products of software development. The reuse-oriented development process proposed in this thesis work focuses mainly on the building blocks approach.

must first have the components. Such reusable components could either be constructed from the scratch or be found from existing software. Techniques for structuring, storing and classifying the components in an efficient way enabling their later retrieval are required.

The second phase focuses on the location/retrieval aspect for finding the components. The library users try to find the desired component(s) from the SRL by applying their query. If the exact match can't be achieved, a potentially reusable relaxed match may be found. Mechanism for efficient retrieval is required.

The final phase deals with the adaptation of the components and application generation. After possible minor adaptations on the retrieval result, the user can reuse it in his/her own application. Techniques for automating the integration and transformation of reusable components are desired.

This thesis work is to explore practical methods to present a prototype of software reuse library system based on the requirements of this reuse-oriented software development process.

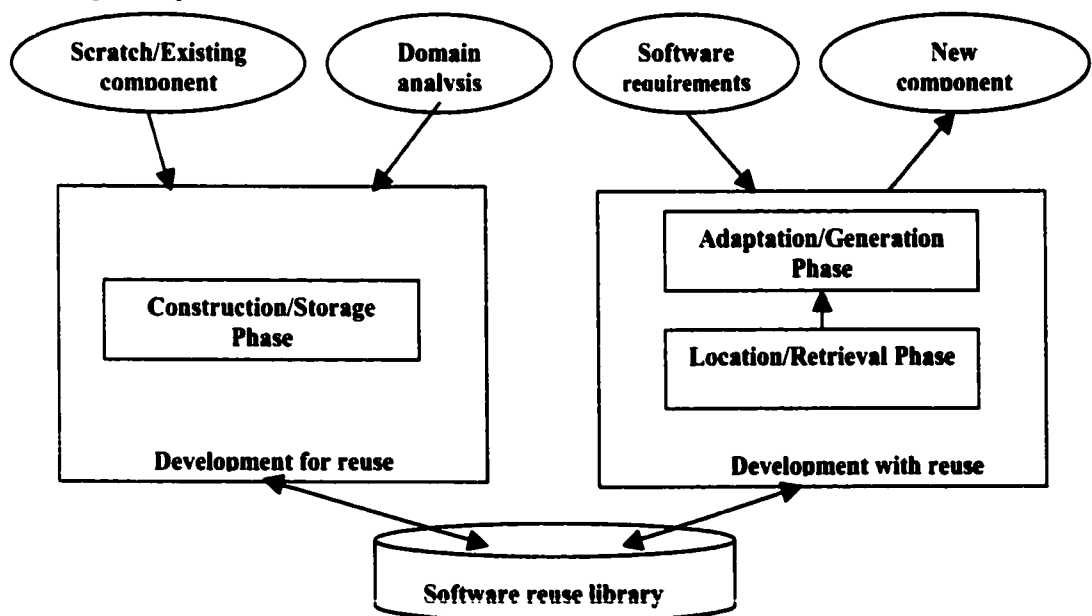


Figure 1: Reuse-oriented software development process

1.3 Current Problems and Limitations of Software Reuse

Two main technical problems currently limit the practice of software reuse: a lack of mechanisms to produce robust, adaptive reusable software components, and a lack of mechanisms to retrieve, adapt and compose software components effectively according to user requirements.

Surveys of software reuse [JBELL92] [AMILI98], conducted to discover user needs and attitudes toward reuse, show that users consider reuse worthwhile; but, most of them (especially those without object-oriented experience) expect more from code generators or from tools for automatic programming than from reuse systems like browsers. This situation is caused by the lack of tools promoting the reuse of software. Often the effort required to create, find, adapt and integrate a generic component into a specific application is greater than the effort needed to create the required software component from scratch [JBELL92].

Cost-effectiveness of Component Classification

Software components must be organized in a software reuse library in a way enabling their later retrieval. Most mechanisms for cataloguing software components are based on a classification scheme. One of most used classifications schemes is based on a set of keywords, but it is difficult (because users have to select manually appropriate terms) and expensive (because usually keywords have to be constructed, describing software components specifically and exhaustively). Mechanisms for automatic indexing of software components are required to make a software retrieval system cost-effective.

Effectiveness of Software Retrieval

Localizing components in small software libraries may be straightforward. Users can learn quickly where the available components are and can select them by name or by browsing the library. Finding reusable components in large distributed libraries is not as simple. Browsing the library can be laborious. Data transaction across the network and response time are also the big issues. Thus, mechanisms allowing faster searching are needed to retrieve a component through the specification of its main features.

From the re-user point of view, it is desired to retrieve software through the use of queries in natural language. The surveys introduced in [JBELL92] [FALOU95] [AMIL98] also show that most of the interviewed users prefer natural language interfaces to retrieval systems over keyword-based interfaces or semantic-based interfaces. It seems more intuitive for users to specify their requirements through a sentence in natural language than to select appropriate keywords, terms for facets in classification schemes or boolean combinations of keywords.

Effectiveness of components adaptation and generation

After component candidate(s) is found, it is desired to have a code generator tool to generate automatically computer code or another program through a component adaptation and integration process based on the user requirements. A code generator is desired in the final phase of reuse-oriented program development process. However, building code generator is great challenge due to the difficulties in recognizing appropriate cases, defining the requirements as well as validating the output. Few reuse

papers resolve this problem, although code generators are highly demanded in software reuse [AMILI95].

1.4 Major Component Classification and Retrieval Approaches

Classification and retrieval methods for the software reuse library play key roles in software reuse. This section describes some major recent approaches for software classification and retrieval with some reviews. We classify those approaches into four catalogues:

- ✓ Descriptive classification methods
- ✓ Behavior-based retrieval methods
- ✓ Denotational semantics methods
- ✓ Formal specification methods

1.4.1 Descriptive Classification Methods

Descriptive classification methods propose a classification scheme depending on a textual description of the software components. The schemes specify some attributes to be used as keywords or descriptors of a software component, mostly focusing on the action that component performs and on the objects manipulated by the component. Both classification and retrieval are performed by specifying (controlled, structured) a list of descriptive keywords for each attribute in the scheme.

Some researchers [PRIET91] [MATSU93] [KARLS95] [ARABA98] propose faceted classification schemes for classifying software components. The approach, which uses facets as descriptors of software, traces its roots to library science. It extends the

simple keyword based approach insofar as a component is no longer described by an unstructured set of keywords; rather, a multi-dimensional search space is defined with each dimension referred to as facet. Examples of such facets include: function; object/item type; medium; and system-type. The classification of the ASSET [ASSET00] software components consists of the facets *component specific data, distribution, component type, collection, domain, function and object*. Due to the relative the simplicity of its concepts and the convenience of putting it in operation, this family of methods is commonly used at the present time.

The primary disadvantage of these methods is that users must have a knowledge base of the reuse sources and keyword set. These approaches are based on a controlled vocabulary, usually derived from domain analysis, which establishes a limited set of terms or phrases that the user or indexer has to select manually for classification or retrieval activities. Retrieval under a controlled vocabulary normally requires a significant effort from librarian and untrained users of the library.

Most recent research efforts aim to overcome the limitations of the traditional descriptive methods. Free-text indexing approaches [MAHAJ99] [KULYU99] [FELIC99] [EIKHO99] classify the components by a free vocabulary index; Automatic indexing approaches [LEELI97] [CHEN98] build indices by extracting automatically the document describing the component functionality; Some knowledge-based approaches were proposed through the natural language analysis [GIRAR93] [AMBRO94] [LANGE97] [FELIC99] [LIHUI00], the frame-based classification [SCOTT96] [HECKE98], the neural network [MERKL98], and the formal concept analysis [LIND00].

1.4.2 Behavior-based Retrieval Methods

Behavior-based retrieval methods, [PODGU93] [HALL93] [PBAI95] [QIANG99] recognize that in the context of information retrieval, software components have a discriminating feature that sets them apart from other retrievable components, namely, they are executable. These methods use the executability of software components as a basis for the selection of candidate components from a software library. Retrieval is performed by executing software components in the library according to the input provided by the user and comparing the resulting output with the one supplied by the user. These approaches can be effective if users have a pre-established set of test cases when submitting their queries, which, unfortunately, is not always the case.

Also these methods are difficult to use. In order to use a behavior-based library, a re-user must produce a query under the form of a sample of inputs and their corresponding outputs, or a set of inputs and an assertion that outputs must satisfy. They are expensive retrieval methods due to the real execution on every related component. However, with the advent of new methods of behavioral modeling and understanding to offset the cost factors of code test execution, these approaches still have appeal [AMILI98].

1.4.3 Denotational Semantics Methods

Denotational semantics methods depend on the denotational partial semantic definition of software components. These methods proceed by checking a semantic relation between the user query and a surrogate of the candidate component. The surrogate of the software component may be a partial functional description, or a

signature of the component. The type-based approach [FISCH98] [QIUAN98] and semantics-properties [ZAREM96] [KFTAM97] [LUQ99] approach both belong to this catalog.

Type-based retrieval uses the component type as a search key to query the library. There may be many components in the library sharing the same type, so the retrieval accuracy is very low. This approach may just give a big cut. Also this approach is limited within the functional-based component (e.g. Miranda programs, C programs) retrieval.

The semantics-properties methods are based on using a variety of semantic properties of functional based components, such as signature, demand property of arguments, transmission property of arguments and length property on list arguments. Similar to type-based retrieval, it provides limited accuracy for retrieval.

1.4.4 Formal Specification Methods

Some approaches for using formal specifications in software retrieval have been developed [CHENG92][WINGJ95] [ATKIN96] [PENIX99]. In these approaches, queries are formal requirement specifications and the system retrieves relevant software from a library of formally specified components by invoking a theorem prover (e.g. first-order predicate logic, formal specification language, etc.) to determine if component specifications satisfy the requirements. These approaches are free from ambiguity and provide better precision than informal methods. They can also be applied to non-executable components.

The disadvantage of these methods is that they are difficult to implement. The processing time for the search algorithms may be excessive depending on the approach

taken. The current state of theorem-proving technology suggests most of these approaches are impractical [PODGU93]. Also the methods are difficult to use because the re-user must be familiar with the appropriate representation of the query (signature, formal functional specification, partially elaborated specification, etc). It is easier and cheaper to use textual descriptions for queries than formal specifications.

Research on more intuitive and effective software retrieval system is ongoing. The goal of exploring the highly specialized methods of software classification and retrieval is to build software retrieval systems based on use of user friendly interfaces and retrieval effectiveness to support software reuse or to improve the productivity of software development and maintenance.

1.5 Code Generator for Software Reuse

A code generator is defined as “a tool or a set of integrated tools that inputs a set of specifications and generates the codes of an application” [AMILI95]. It generates computer code or another program from one object description and information acquired from a software library. Viewed as a translator, a code generator may be applied in the final phase of reuse-oriented program development process.

In code generators, algorithms and data structures are automatically selected so the software developer can concentrate on what the reuse system should do rather than how it is done. That is, code generators clearly separate the component specification from its implementation. It is possible for even non-programmers familiar with concepts in an application domain to create software applications from software reuse library.

Code generators are appropriate in application domains where

- ✓ Many similar software programs are written,
 - ✓ One software component is modified or rewritten many times from a software library, or
 - ✓ Many prototypes of a system are necessary to converge on a usable product.
- [KRUEG92].

In all of these cases, significant duplication and overlap results if the software systems are built from scratch. Code generators generalize and embody the commonalities, so they are implemented once when the code generator is built and then reused each time a software system is built using the generator.

The abstractions presented to the user of a code generator typically come directly from the corresponding application domain. Different applications need different models of data and computation in the generated programs, and these different models are amenable to different exposition techniques. Examples include,

- ✓ Textual specification languages, Application-oriented languages, fourth-generation languages, etc.
- ✓ Templates
- ✓ Graphical diagrams
- ✓ Interactive menu-driven dialogs
- ✓ Structure-oriented interfaces [CLEAV88] [KRUEG92] [AMILI95]

Martin [MARTI85] enumerated a number of mostly behavioral properties that code generator should exhibit, including

- ✓ User-friendliness,
- ✓ Usable by nonprofessional programmers,
- ✓ Support for fast-prototyping,
- ✓ Applications take an order of magnitude less time to develop than with traditional development, etc.

There are many different kinds of code generators today, including stack based code generator, accumulator based code generator, rule based code generator (allowing the integration of legacy or user-specific code by specification rule), web based code generator, pattern-based code generator (approach to generate domain specific application code using patterns, e.g.), forms-based code generator, template-driven code generator to name but a few.

It is impossible to give a more precise operational definition of what constitutes a code generator without excluding known classes of code generators. This is due to the fact that the specification language used, and hence the generation technique depends very heavily on the application domain. For the same reasons, it is difficult to design a development methodology for code generators that is appropriate for all code generators. Although the development of code generators in general has received little attention in the literature, by contrast, developing with code generators has received a fair amount of attention [AMILI95].

1.6 Overview of This Thesis

1.6.1 Motivation

Distributed SRL & software reuse are required as one primary conceptual domain of Virtual Prototyping, Modeling & Simulation (VPMS). VPMS is a research project conducted by Dr. R. Kent at the University of Windsor.

Distributed system like VPMS or reuse-oriented program development system may call for software reuse library (SRL, repository) to serve as a resource provider by the usage of reusable software components. We are trying to solve the problems of building such SRL in an effective way to store, classify, retrieve, and deliver the large number of components in order to share and reuse resources across the network.

So far most SRLs have been designed and built to reuse pure functional-based software components, such as Miranda programs or imperative software, such as C programs [PBAI95] [ZAREM96] [KFTAM97] [FISCH98] [QIANG99] [EIKHO99] [LUQI99]. Little work has been done to effectively reuse object-based software components, such as object-oriented programs [ARABA98].

1.6.2 The Objective of This Thesis

This thesis presents the design and implementation of a distributed object-based software reuse library system called DORLM. The DORLM serves as an object code class resource provider to provide an effective way of storing, retrieving, and delivering reusable software components, as an aid of reuse-oriented program development in the distributed environment. The goals of the DORLM aims to:

- ✓ Present the approaches to implement all three stages of reuse-oriented program development.
- ✓ Build mechanisms for the automatic indexing of the object-based software component in order to provide an effective approach to component classification.
- ✓ Create a friendlier user interface for component retrieval through the use of queries in natural language.
- ✓ Build a simple forms-based code generator to deliver an executable code fragment for re-user.
- ✓ Build a distributed object-oriented system under CORBA-based architecture.

The basic approaches proposed in this thesis, are described in section 1.6.3.

1.6.3 Overview of the Proposed Approaches

Software Classification and Retrieval:

Through the analysis of the four current major approaches of software classification and retrieval in section 1.4, we think that the behavior-based approaches and the denotational semantics approaches are not suitable for classifying and retrieving object-based component, they are more appropriate for functional-based component retrieval; the formal specification-based methods are difficult to implement; the advantage techniques of the descriptive classification methods are suitable for object-based component retrieval.

This thesis proposes an approach combining main advantages of descriptive classification methods for component classification and retrieval (facet classification, automatic indexing and knowledge-based system) in order to improve retrieval effectiveness and provide a friendlier user interface through the use of classification and queries in natural language.

We have already pointed out in section 1.4.1, the major drawbacks of the traditional descriptive classification schemes for software components:

- ✓ They are based on a controlled vocabulary that must be constructed manually for each application domain;
- ✓ Both classification and retrieval require important human effort because users must select appropriate terms for each facet in the classification scheme from usually a long list of terms in the controlled vocabulary;

These drawbacks can be dealt with:

- ✓ Building mechanisms for the automatic indexing of software components in order to provide a more cost-effective approach to software classification;
- ✓ Allowing a free vocabulary in both classification and retrieval activities, instead of having to select appropriate terms from a controlled one;
- ✓ Using semantic relationships between terms from a lexicon in order to compute the similarities between terms to improve retrieval precision.

The central idea of the proposed approach is to allow the automatic classification and retrieval of object-based software in SRL through sentences describing the software

functionality and to take advantage of the lexical, syntactic and semantic information that can be extracted from these sentences to construct queries and indexing units more useful and precise than traditional keyword or facet descriptors.

A classification scheme is proposed to classify object-based components in a software reuse library according to the comment sentences embedded in the component describing the component functionality. A semantic case system is applied to translate the indexing sentences and query sentence into a frame-based internal representation. The components are retrieved according to the similarity between the internal representations of the query and component classification.

Component Adaptation/Generation:

Most current software reuse projects and papers are limited in retrieval exploration [AMILI95]. The DORLM tries to explore a simple forms-based code generator for the adaptation and generation of the candidate components from the software reuse library. Once the re-user finds the candidate component(s) he/she want to reuse for his/her programming development need, the DORLM provides a forms-based code generator to deliver an executable code fragment for re-user. Through a user-friendly graphic interface, what the re-user needs to do is just to input several essential parameters value, which will be used to initialize the specific classes and methods of the components he/she is interested.

Architecture based on CORBA Interoperability Components

The Common Object request Architecture (CORBA) provides the software infrastructure to develop distributed object-oriented system. DORLM uses the CORBA-based architecture enables the distributed library access objects communicate and interoperate with the distributed SRL across the network. Interoperability is supported by Object Request Brokers (ORBs) which communicate through the Internet Inter-ORB Protocol (IIOP). This architecture increases use of and reliance on SRL transaction in the distributed collaboration context.

1.7 Organization of This Thesis

This thesis is organized into six chapters.

Chapter 1 gives a brief overview of background knowledge involved in the thesis work. It includes overview of software reuse, reuse-oriented program development process, current major software classification and retrieval approaches, code generator for software reuse, and an overview of this thesis.

Chapter 2 discusses the details of our approaches to classify, retrieve, and deliver object-based software components through the software reuse library.

Chapter 3 gives the details of the design and implementation of the prototype system. Major modules of the system and functionalities of the system are explained.

Chapter 4 describes the DORLM interfaces and gives some examples on how to use the DORLM. A number of screen shots are made to illustrate the procedure of performing the component insertion, updating, deleting, retrieval and code generation step by step. Also the system testing and the limitations of the DORLM are introduced.

Chapter 5 highlights the conclusions along with some recommendations for future work.

CHAPTER 2

Approaches to Classify, Retrieve, and Deliver Software Components

In this chapter, we describe our approaches to classify and retrieve object-based software components in a software reuse library through the use of classification and queries in natural language and to deliver reusable code fragment through a forms-based code generator. Other supports of our reuse system are discussed elsewhere.

The chapter is organized into four sections:

Section 2.1 discusses the component structure and classification scheme.

Section 2.2 describes an automatic indexing process to interpret sentences in the component classification.

Section 2.3 discusses query in natural language.

Section 2.4 describes our approach to code generation.

2.1. The Component Specification Acquisition

The design of the component structure and classification in a SRL plays the key role in the storage phase of reuse-oriented program development in order to promote the retrieval effectiveness. This section discusses the component structure and classification scheme proposed in this thesis.

2.1.1 Component Structure

Since the object-based components (like Java classes) in the SRL may have arbitrary variables and functions, and the structure of the components have various

patterns, a complete component structure (CS) is defined with a list of the segments. The component structure representing the complete component data model is extracted when a component is inserted. It is used to build the library database.

Definition 1: Component structure

A first order definition of the component structure (CS) is defined as

CS= < CF, IL, CH, IP, VL, CP, FF, FP>, where

- ✓ **CF (class features)**: describe the general features of this class in natural language to be obtained from the class comments.
- ✓ **IL (import list)**: include number of import items and their names.
- ✓ **CH (class head)**: include name of the class, class attribute, its parent class, and class package. **CH= < CN, CA, PC, PN>**, where **CN** is name of the class, **CA** is the class attribute, **PC** is name of the parent class, and **PN** is the name of the class package.
- ✓ **IP (implement part)**: include the list of the interface names implemented by this class.
- ✓ **VL (variable list)**: include number of the variables, their names and attributes.
- ✓ **CP (constructor part)**: include the constructor head, number of arguments passed in with their corresponding types, and the constructor content definition.
- ✓ **FF (function functionalities)**: describe the function's functionalities in natural language to be obtained from each function's comment.
- ✓ **FP (function part)**: include number of the functions. For each function, function head, number of the arguments passed in with their corresponding types, and function content definition are also included.

A component structure encapsulates all component information to be used as a single data record stored into a distributed database constructed using the DBMS (e.g. Oracle, PostgreSQL).

2.1.2 Component Classification

We are interested in some criteria for obtaining the component classification from the component structure to index the component for retrieval purpose. For the simplicity in our approach, the classification scheme proposed in this thesis describes each component in the software reuse library through four attributes: the class name, its parent class, general features of the class, and functionality of each class function.

Definition 2: Component Classification

The component classification (**CC**) is defined as $CC = \langle CN, PC, CF, FF \rangle$, where

- ✓ **CN** is name of the class
- ✓ **PC** is name of the parent class
- ✓ **CF** (class features): describe the general features of this class in natural language
- ✓ **FF** (function functionalities): describe the function's functionalities in natural language

The component classification classifies the components in the SRL to be used for building automatic indices through an automatic indexing processing. The automatic indexing process will automatically translate **CF** and **FF** in natural language into a frame-based representation by applying partial natural language analysis. After the automatic

indexing process, the component classification representing the component index is used to locate the components in SRL effectively since the index capture some semantic information associated with the functionality of the software component.

2.2. Automatic Indexing Process

The automatic indexing process proposed in this thesis is used to automatically translate natural language sentence into a frame-based internal representation to capture the semantic information of the component classification. A similar process is used to create the same internal representation of a user's query in natural language. The main purpose of this process is to find out whether the indexing sentence and the query sentence are semantic equivalent or similar for the future retrieval by comparing the frame-based internal representations of the sentences rather than the whole sentences or a set of simple keywords created manually. The basic constituents of this process include a semantic case system for the interpretation of sentences describing component functionality, a process for lexical, syntactic and semantic analysis of these sentences into a frame-based internal representation, and a lexicon.

2.2.1 Semantic Case System

A partial solution to define two sentences' semantic equivalent or similar, is to find a representation language allowing reduction of the expression to a canonical form. A canonical form requires that every expression equivalent to a given one can be reduced to a single form by means of an effective procedure, so that the test of equivalence between descriptions can be reduced to the test of identity of the canonical forms.

Although some authors consider that it is unlikely that there could be a canonical form for English, there have been some proposals of canonical representation languages for English, like the Conceptual Dependency Theory [MAULD91], which have limited its application to very restricted domains.

The semantic case system proposed in this work provides a partially canonical form to represent an indexing sentence. The representation is not completely canonical because it simplifies the semantic case and allows for some ambiguities, but captures the core knowledge of an indexing sentence and is sufficient for retrieval.

The semantic case system for an English sentence describing software functionality consists of a sequence of one or more semantic cases. It consists of semantic cases, case triggers, and noun phrase cases. The constituents of the case system along with the syntactic elements in a sentence from which they are extracted are illustrated in Figure 2. Constituents appear in *italic*. Constituents between parentheses are optional.

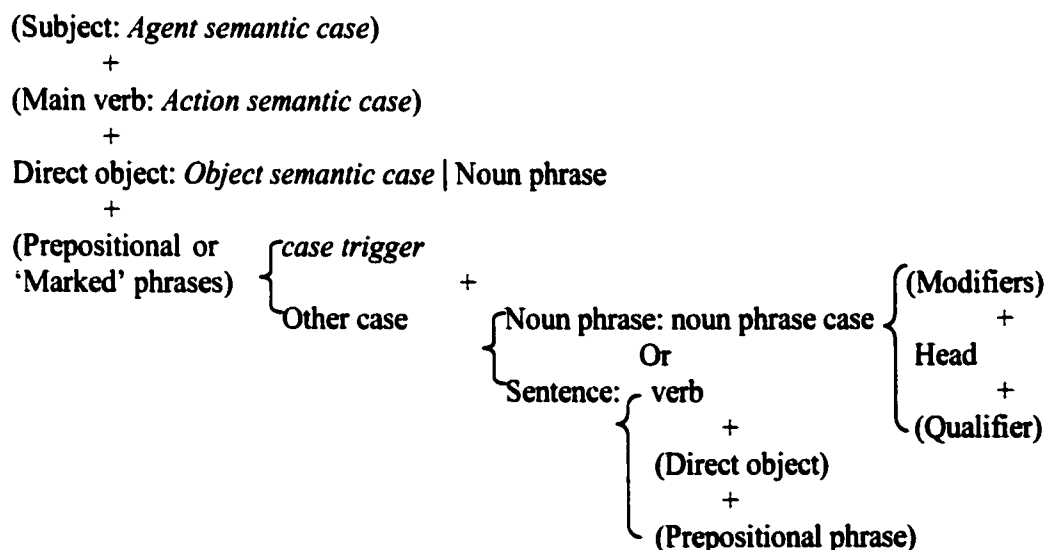


Figure 2: The semantic case system

A sentence consists of a subject (making reference to the component itself and possibly omitted), maybe followed by a verb (representing an action), followed by a noun phrase (representing the direct object of the action) and perhaps some embedded prepositional phrases or phrases marked through particular terms (representing entities or actions related to the main action or object). Note that most adverb terms, most adjective terms, and stop words are not considered in this semantic case system.

Adverb or adjective terms can be embedded in a sentence. Considering that these terms do not contain significant information for retrieval purposes, most of them are discarded by this semantic case system.

Stop words basically refer to prepositions and determiners, for example, articles (the, a, an), demonstrative pronouns (this, that, these, those), and quantifiers (all, half, every, each, some, any, most, many, few, little, no...). They are also discarded by this semantic case system since they also do not contain substantial information for retrieval purposes.

Semantic case: represents either the semantic role that a particular syntactic compound in an indexing sentence plays as a descriptor of the functionality of a component or other descriptive associated with the component.

- ✓ Agent -> component identifier+ (alias)+description
- ✓ Action -> verb (**Action semantic case**)
- ✓ Object -> Direct Object (**Object semantic case**)
- ✓ Noun or verbal phrase preceded by a case trigger ->(case triggers) + Noun phrase | verbal phrase.

The description of the semantic cases currently considered in this work is shown in table1.

Semantic case	Description
Action	A function performed by the component
Agent	The component identifier, alias, description, etc.
Direct Object	The object, target or result of an action.
Other Case	Other objects other than direct object, maybe describing purpose, location, manner, etc.

Table 1: Main semantic cases describing software components

Case trigger: is a term or groups of terms suggesting the presence of a particular semantic case in a sentence. Case triggers are mainly prepositions or verbs followed by a particle. Case triggers determined by analyzing various software comment sentences are listed in table 2.

Semantic case	Semantic case trigger
Direct Object	for, to, at in, into, on, onto, over, within
Other Case	to (destination) during, for (Duration) for, to (Object) by, by-using, through, using, with, via (Instrument/Manner) at, in, in, into, on, onto, over, within (Location/Time) designed_for, designed_to, implemented_for, for, in, that, thus,

	to, which, while (Purpose)
--	----------------------------

Table 2: Case triggers in the case system

Noun phrase case: A noun phrase consists of a sequence of zero, one, or more cases representing modifiers of a head noun and one case representing the head noun in the noun phrase, or these same cases with an additional case that represents a qualifier of the main noun phrase. The constituents of a noun phrase are: (constituents between parenthesis are optional).

- ✓ A noun phrase: (Modifiers) + Head + (Qualifier)
- ✓ Modifier: (adverb Modifier) + (adjective Modifier) + (noun Modifier)
- ✓ Head: head noun
- ✓ Qualifier: Qualifier trigger + noun phrase
- ✓ Qualifier trigger: mainly prepositions or verbs followed by a particle (e.g. of, about, composed_of, consisting_of, etc)

2.2.2 A Process for Lexical, Syntactic and Semantic Analysis

Lexical, syntactic and semantic analysis of both user's queries and indexing sentences of components is a process performed to map the indexing sentence into a frame-based internal representation, according to a semantic case system described in section 2.2.1.

The analysis is performed in two main phrases:

- ✓ Lexical analysis
- ✓ Case parsing

Lexical analysis: Lexical analysis is to identify the grammatical classes of the individual words in a sentence with their categories (verb, noun, adjective, adverb, and stop words) by looking up a lexicon database that we built.

Considering the limitation of the lexicon database in this work, a very simple strategy has been used for processing unknown words in a sentence:

- ✓ A special grammatical category unknown is defined that is not present in the lexicon, and is assigned to this category;
- ✓ Case parsing accepts the presence of a word under the unknown category either as a modifier or head noun.

Case parsing:

After lexical analysis, both syntactic and semantic analyses of queries or indexing sentences of component are performed, using a parsing grammar supporting the semantic case system.

The parsing grammar implements a subset of the grammar rules for English sentences by applying the semantic case system. Table 3 shows this parsing grammar supporting this semantic case system. Terms between parentheses are optional.

The translation mechanism uses the parsing grammar to parse either sentences describing software components or user's queries. It maps indexing sentences into a

frame-based internal representation of sentences. Most adverb, adjective terms, and stop words are removed from the sentences during the parsing process.

Sentence	Declarative_sentence imperative_sentence Nominal_clause
Declarative_sentence	Agent Action (Direct_object_case) (Other_case)
Imperative_sentence	Action (Direct_object_case) (Other_case)
Nominal_sentence	Direct_object_case (Other case)
Direct_object_case	Direct_object
Other_case	Other_case Sentence
Agent	Noun_phrase
Action	verb (adverb)
Direct_object	Noun_phrase Direct_object_trigger Noun_phrase
Other_case	Other_case_trigger Noun_phrase
Direct_object_trigger	FOR TO AT IN INTO ON ONTO OVER WITHIN
Other_case_trigger	TO DURING FOR BY BY_USING THROUGH USING WITH VIA AT IN INTO ON ONTO WITHIN FOR THAT THUS WHICH WHICH DESIGNED_FOR DESIGNED_TO IMPLEMENTED_FOR
Noun_phrase	(Modifiers) Head (Qualifier)
Qualifier	Prepositional_qualifier Other_qualifier
Prepositional_qualifier	qualifier_trigger Noun_phrase
Qualifier_trigger	OF ABOUT CONSISTING OF

Table 3: Parsing grammar of case parsing process

Frame Building: Through the case parsing process, CF and FF of the component classification are translated into a set of facets with their semantic roles. Then a whole component classification is mapped into a frame-based internal representation indexing a component.

An example of this frame-based internal representation of a component classification is shown in the following:

Consider the component of the Cal class, expressed in Java:

```

/* This class is used to perform some simple mathematics calculation */
Class Cal extends Cal_Parent {

    /* The method calculates the sum of two integers */
    int add ( int op1, int op2);

    /* Return the difference of two integers */
    int minus (int op1, int op2);

}

```

A frame-based index of this component is obtained through the following steps:

The classification CC is obtained when the component is inserted:

```

CC = < CN, PC, CF, FF >
CN = "Cal "
PC = "Cal_parent "
CF = "This class is used to perform simple mathematics calculation"
FF = "The method calculates the sum of two integers" +
     "Return the difference of two integers "

```

After an automatic indexing process for interpretation of CF and FF, a frame-based representation of this component is obtained and displayed in the following:

PC:	<i>Cal_parent</i>	
CN:	<i>Cal</i>	
CF:		
	<u>Case</u>	<u>Value</u>
	<i>Agent</i>	<i>Cal</i>
	<i>Action</i>	<i>perform</i>
	<i>Direct Object</i>	<i>mathematics calculation</i>
FF1:		
	<u>Case</u>	<u>Value</u>
	<i>Agent</i>	<i>Cal: add</i>

	<i>Action</i>	<i>calculate</i>
	<i>Direct Object</i>	<i>sum</i>
	<i>Other case</i>	<i>integer</i>
FF2:		
	<u>Case</u>	<u>Value</u>
	<i>Agent</i>	<i>Cal: minus</i>
	<i>Action</i>	<i>return</i>
	<i>Direct Object</i>	<i>difference</i>
	<i>Other Case</i>	<i>integer</i>

2.2.3 Lexicon

The lexicon is a dictionary database used for natural language process. The lexicon has the terms database of five categories (nouns, verbs, adjective, and adverbs, Stop words), and a thesaurus containing synonyms of partial terms.

For reasons of development time and code complexity the author decided to adopt a “keep it simple” philosophy with regard to conceptual proof of this classification and retrieval approach. The lexicon database built in this system is simple and contains partial terms only.

2.3 The Retrieval Approach

An analysis mechanism similar to the one applied to component indexing sentences is used in the DORLM to map a query in natural language to a frame-based internal representation. The internal representation is then compared with the ones associated with component classifications in the SRL and components having a partial or complete match are selected.

Figure 3 shows the structure of this retrieval approach. The lexical, syntactic and semantic analysis is performed on the query to translate it into a frame-based internal representation based on the semantic case system and case parsing process described in

section 2.2. The matching and retrieval analysis is applied to match the internal representation of the query against the one associated to the component classification. The similarity analysis is based on the lexical, syntactic and semantic information available on the internal representations of queries and component classification and synonyms between terms in these structures.

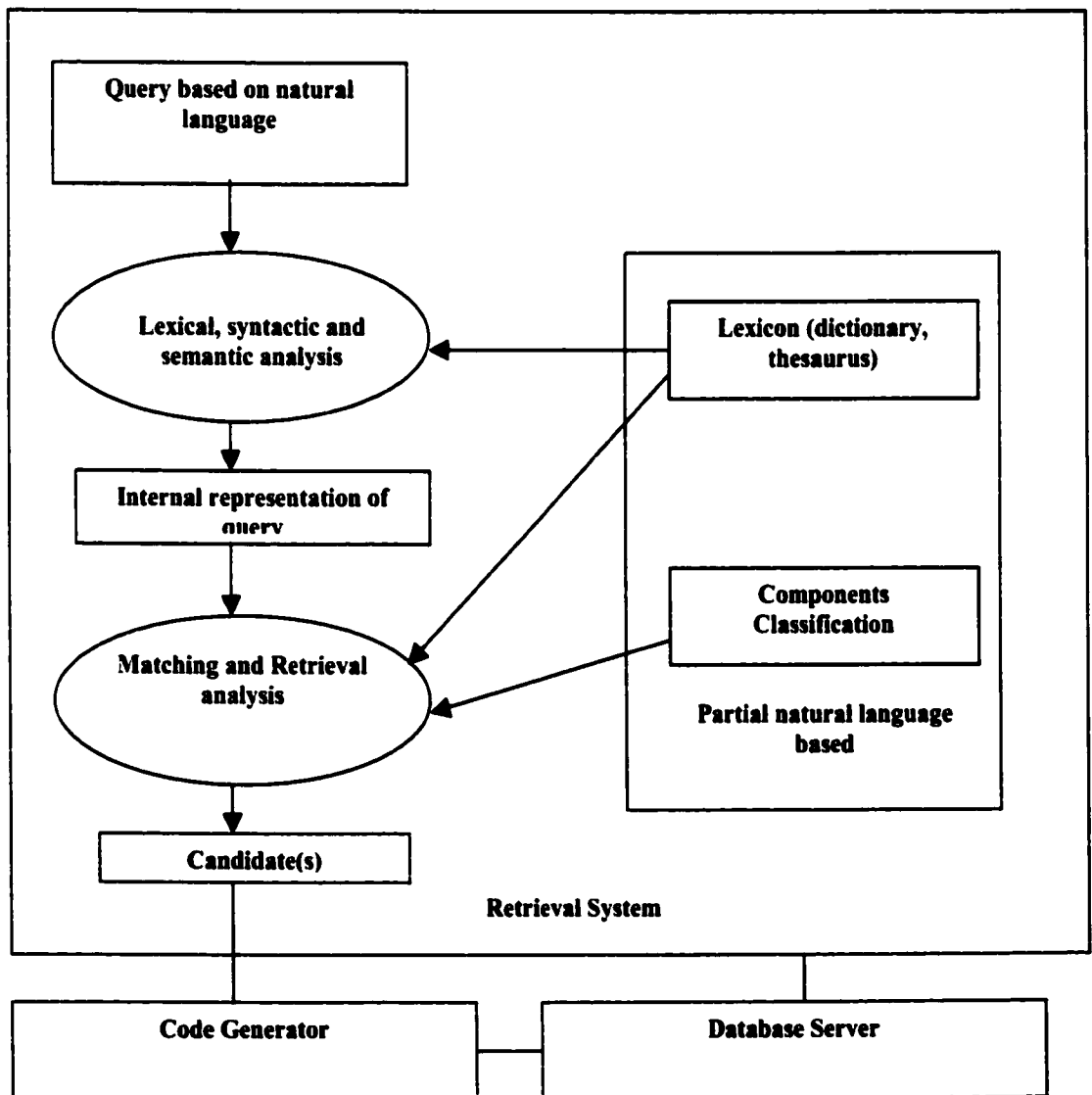


Figure 3: Partial Natural Language-based Retrieval System Process

2.4 An Approach to Deliver Reusable Component for Reuse

In this section, we will focus on another functionality of the DORLM for building new reusable components, namely, the code generator. Once the user finds the candidate class (s) he/she want to reuse for their programming need, the code generator generates an executable code file from the component structure acquired from the SRL. An approach for building such a code generator is discussed herein.

2.4.1 Requirements Phases and Methodology

Based on the component structure acquired from the specification acquisition process in the DORLM and the distributed environment in which the DORLM run, a forms-based code generator is built. The forms-based code generator combines the retrieval approach that is discussed in section 2.3 with the component structure to generate kinds of object-oriented programming code fragments, based on the information of the component stored in the SRL. The general requirements phases and methodology are described in the following.

To build the forms-based code generator, we have to define the **variant** and **invariant** parts of a component, which can be acquired from the component structure, define the specification input form, and define the products.

The **invariant** part of an application consists of the implementation details of the application and all of the defaults assumed by the generator. The **invariant** part of this application includes **IL (Import List)**, **CH (Class Head)**, **IP (Implement part)**, **VL (Variable list)** except their values, the main body of **CP (Constructor Part)**, and the main body of **FP (Function Part)**.

The **variant** part consists of those aspects that the user has to specify including input specifications. The **variant** part of this code includes the value of the **Variable list**, the arguments value of the **Constructor part**, and the arguments value of the **Function part**.

Based on the **variant** part we define, the interactive specification input forms, which are represented as the graphical user interface, are generated for user to specify the values of the **variant part**.

The product generated is a kind of object-oriented programming code fragment based on the component stored in the SRL, such as Java programming code.

2.4.2 Code Generator Process

Figure 4 illustrates the component relationships of the forms-based code generator process.

The user specifies the component candidate(s) upon which the code generator will operate.

The variant and the invariant specification analyses are used to analyze the component specification, define the variant specification and the invariant specification and finally define the specification input forms.

The user specifies the specification input forms via an interactive user interface. The user may be required to specify the arguments value of the component constructor or the arguments value of the component function.

Code integration is a process applied to assemble the specification input and the invariant specification to generate a new programming code.

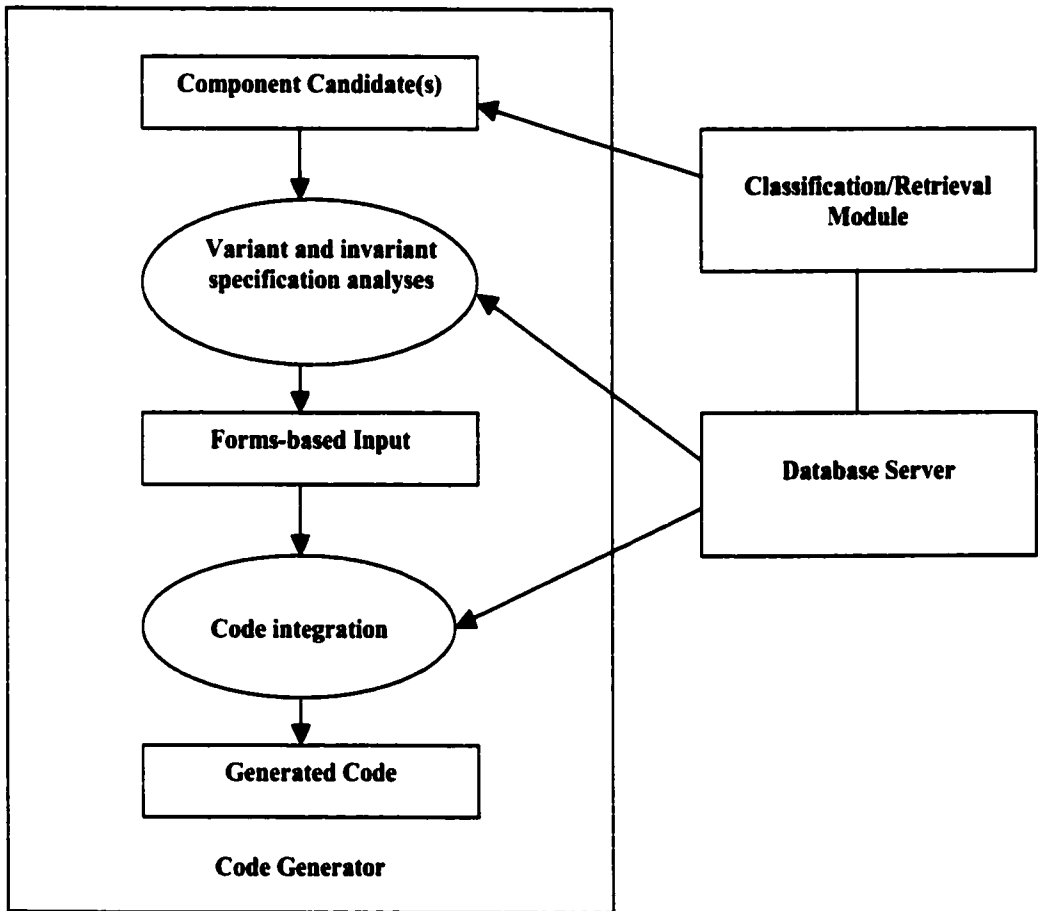


Figure 4: Forms-based Code Generator Process

CHAPTER 3

Prototype Design and Implementation: DORLM

In keeping with the concept of the reuse-oriented software development process, a prototype reusable software base system called *Distributed Object-based Reusable Library Model (DORLM)* was designed and developed. It is a distributed software reuse system that follows the three phases of reuse-oriented software development described in Chapter 1.

3.1 System Overview

The DORLM provides two kinds of major services: server-side service and client-side service across the network. Server-side service allows server-side users, who should be skilled programmers and experts, to insert, update, delete, and query the components (e.g. classes and functions) in the SRL. Client-side service includes components browse, retrieval, and code generation based on client-side users requirement. Each category of users who tries to access our service must register initially. Figure 5 shows the architecture of the DORLM. The system is composed of three main parts:

- ✓ The middle part is the system server. It consists of a Database Server, a User Register Module, and a Classification/Retrieval Module.
- ✓ The left part is designed for the client-side users. Users can access the system server to browse and retrieve the components of the SRL. Through the Code Generator, users may attain the application code fragment, as they prefer.
- ✓ The right side is designed for the server-side users to insert, update, query, and delete the components of the SRL. A Lexical Analyzer is offered to check user-input and user-selected data.

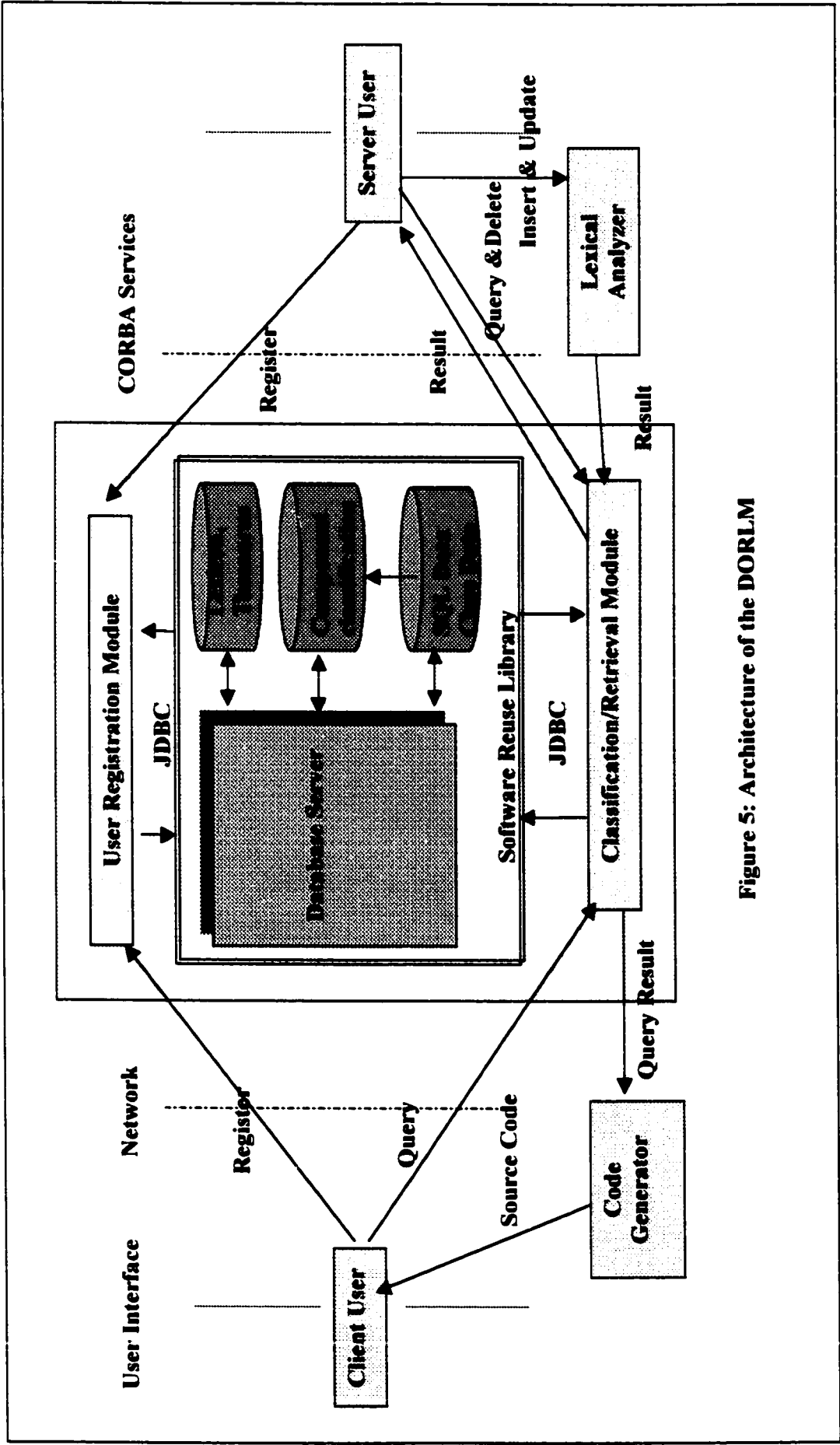


Figure 5: Architecture of the DORLM

3.2 System Structure

The overall system structure and functionalities are discussed below.

- ✓ **Server Interface:** Through server interface, server side users can insert, update, delete and query the library.

- ✓ **Client Interface:** Through client interface, client side users can browse, query and get the generated code through Network (Internet). Also the DORLM offers a dynamic component index of “CLASS” and “FUNCTIONS”.

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

- ✓ **Database Server:** All of the component data, the component classification, and the lexicon including dictionary and thesaurus, and user information are stored in a distributed DBMS. In order to promote the automatic retrieval, the component databases in the SRL were 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 object

identifier. The system will generate the object identifiers automatically for each component when the server-side users insert it to the library.

- ✓ ***Lexical Analyzer:*** It is used to perform the grammar and validation check for the server-side user's operation.

- ✓ ***Classification and Retrieval Module:*** This module is built based on the component classification and retrieval methods discussed in Chapter 2. The Classification/Retrieval Module can allow querying component in natural language, and can offer online search and browsing mechanism.

- ✓ ***Code Generator:*** A forms-based application generator discussed in the chapter 2 is built. Based on the user input and retrieval result from SRL.

- ✓ ***Naming Service:*** Serves as a directory for distributed CORBA objects in the system. With naming services, either client or server modules across the network are treated as the distributed objects, which can be bound and resolved in the DORLM. The naming service allows associating a URL with this Naming Service object. Once the URL is associated with this 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.

3.3 Development Environment

The DORLM is developed using the current advanced object-oriented and distributed technologies. UML, Java, CORBA, Oracle, and PostgreSQL are used to design and implement the concepts presented in this thesis.

Currently, development and execution environments of the DORLM are listed as following:

- ✓ DBMS: Oracle 8 for Windows 98/NT, PostGreSQL6.5.3 for LINUX
- ✓ Implement language: Java 2 (JDK1.2.2) for Windows 98/NT and LINUX
- ✓ Middleware: ORBacus 3.2.2 (CORBA) for Windows 98/NT and LINUX
- ✓ Platform: Microsoft Windows 98 and LINUX
- ✓ Execution environment: Windows 98/NT, UNIX, LINUX, JDK 1.2.2, ORBacus 3.2.2, and the web browsers (Netscape 4.x) with Java 1.2.2 plugin.

Java, as a language and platform, has a number of desirable features that would serve to facilitate an implementation of the DORLM to prefer it to other general purpose programming languages such as C/C++, Smalltalk etc. Some of Java features are: its compiled form can be executed on virtually any existing computer; it is able to interoperate with web browsers within a secure, easy to download environment for the user; and 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.

Common Object Request Broker Architecture (CORBA) [CORBA99] 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. Moreover, to provide higher-level reusable components, the OMG [CORBA99] 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.

Oracle is the world's most popular database for Internet computing. Oracle 8 has brought the relational database world into the distributed object area. The new features of the product make it possible to combine the newer distributed object-oriented structures with the traditional relational constructs [ORACL99]. 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 Java 2, and allow applications to efficiently implement and manage robust security policies.

At the outside, the choice of support tools had to be made with the respect to the Internet-based access to achieve the goal of distribution. It was decided to embrace the Netscape as oppose to the Internet explorer due to the increased flexibility of the Netscape in term of its multiple platforms support (UNIX, Windows 95/98/NT, and Linux), and support for Java 2.

3.4 System Design Specification

The DORLM is designed using Unified Modeling Language (UML), a language that unifies many of the industries' best engineering practices for modeling system [UML97]. Based on the object-oriented paradigm. UML is used here for specifying, visualizing, constructing and documenting systems. This section will describe the use cases, the class diagrams, and the sequence diagrams of the DORLM.

Use Cases:

There are two categories of users. The first kind of user is the server-side users who can insert, update, delete, and query the SRL. The second kind of user is the client-side users who can query, browse, and generate programming code fragment.

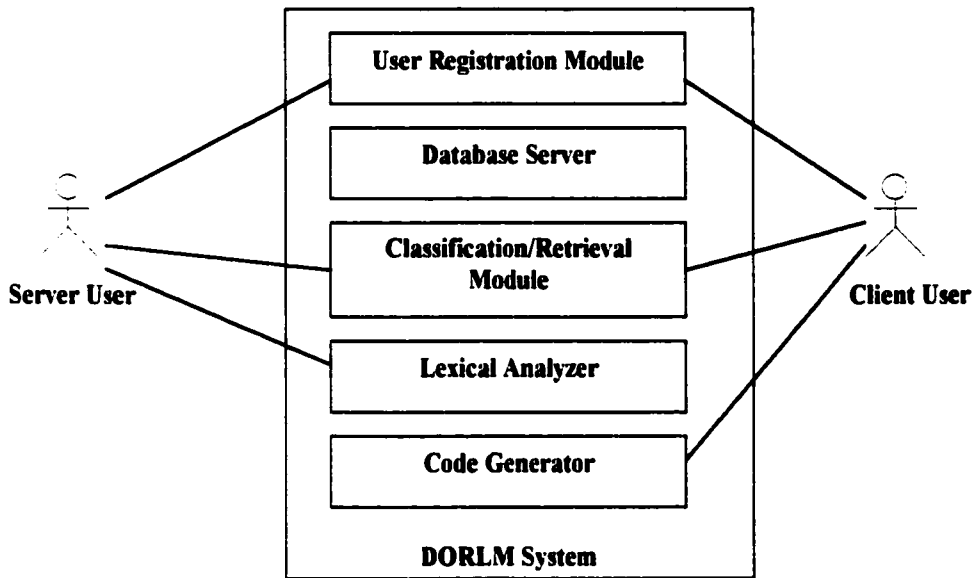


Figure 6: DORLM Use Case Diagram

As shown in Figure 6, the DORLM use case diagram describes the functionality of the system and users of the system. The DORLM has five main subsystems: User Registration Module, Database Server, Classification/Retrieval Module, Lexical Analyzer, and Code Generator.

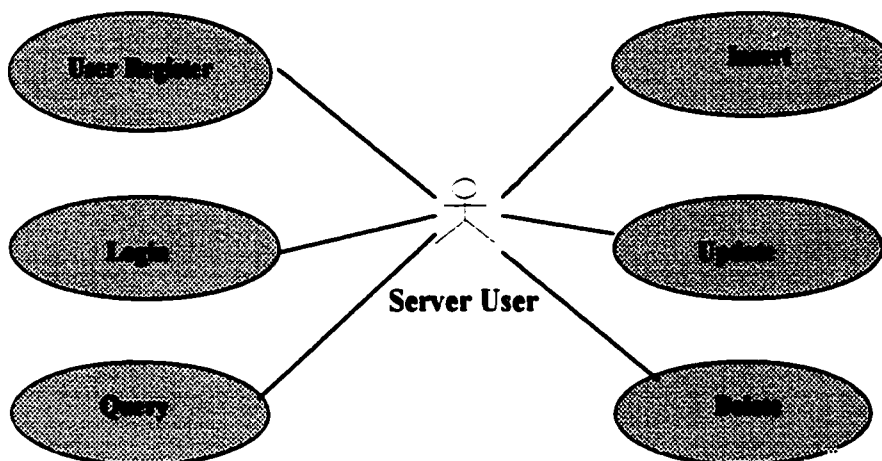


Figure 7: Server User Use Case Diagram

Figure 7 elaborates the server user use case by detailing the functionality a server user actor expects of the system. The server user must register to the DORLM before he/she can access the system. Then he/she can login to the system by specifying his/her user identification and password. After the server user login to the system, he/she can insert, update, query, and delete the components.

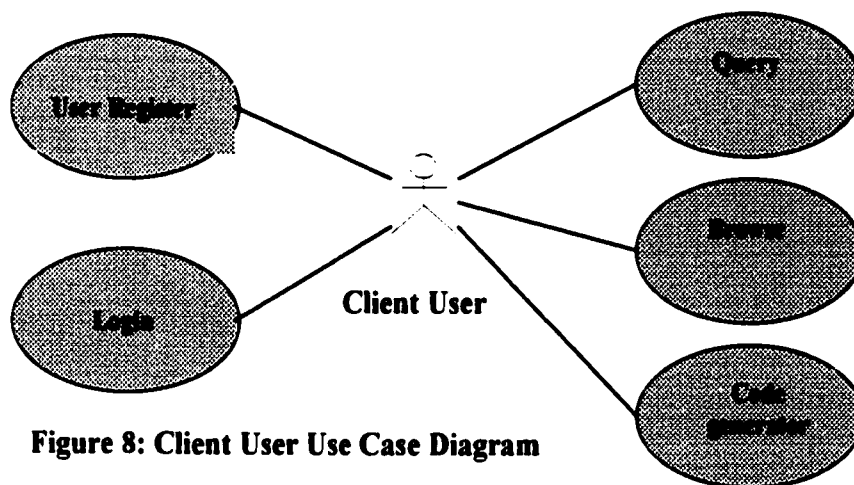


Figure 8: Client User Use Case Diagram

Figure 8 elaborates the client user use case by detailing the functionality a client user actor expects of the system. The client user must register and login to the system when he/she wants to access the DORLM. After the client user login to the system, he/she can browse, query the components of the SRL, and use code generator to generate program code.

Class Diagram:

The major class diagrams of the DORLM are described. Figure 9 shown in the following describes the static structure of major abstract classes of the DORLM.

SystemServer is an abstract class to process the major functions of the DORLM including:

- ✓ Connect to the database servers including the *UserDatabaseServer*, the *ClassDatabaseServer*, and the *LexiconDatabaseServer* (will describe in the following).
- ✓ Login on the DORLM.
- ✓ Perform the various operations received from the server users and the client users, communications with the *CodeGenerator* and *LexicalAnalyzer*, and communications with the various database servers. The operations include the component insertion, retrieval, updating, and deletion from the server user, and the library browse, the component retrieval, and the code generation from the client server.

UserDatabaseServer deals with the functionalities of the storing the new user record and checking if the user record exists.

ClassdatabaseServer deals with the functionalities of the checking if the component exists, storing the component, modifying the component, deleting the component, getting the component record, and getting the component index record.

UserRecord is the database storing the user information.

ClassRecord is the database storing the component records and component index.

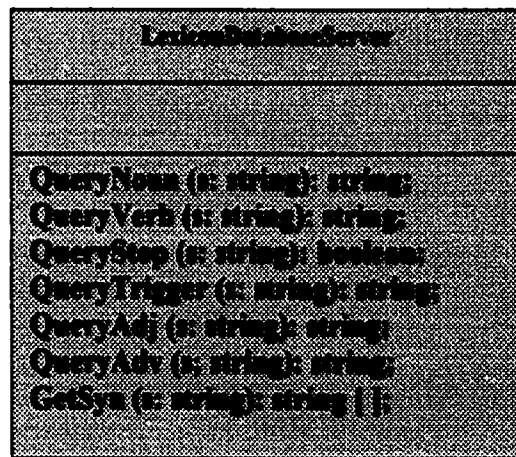
ServerUserInterface is the server user interface class to deal with the various operation requests by the server user. The operation requests include new user register, login, the component insertion, modification, query, and deletion.

ClientUserInterface is the client user interface class to deal with the various operation requests by the client user. The operations include the library browse, query, and code generation.

LexicalAnalyser performs the grammar and validation check.

CodeGenerator performs the code generation process.

LexiconDatabaseServer is an abstract class, which is not shown in Figure 9. It deals with the functionalities of the lexicon query and getting the synonyms of a term.



Sequence Diagrams: shown in Figure 10 -- 14

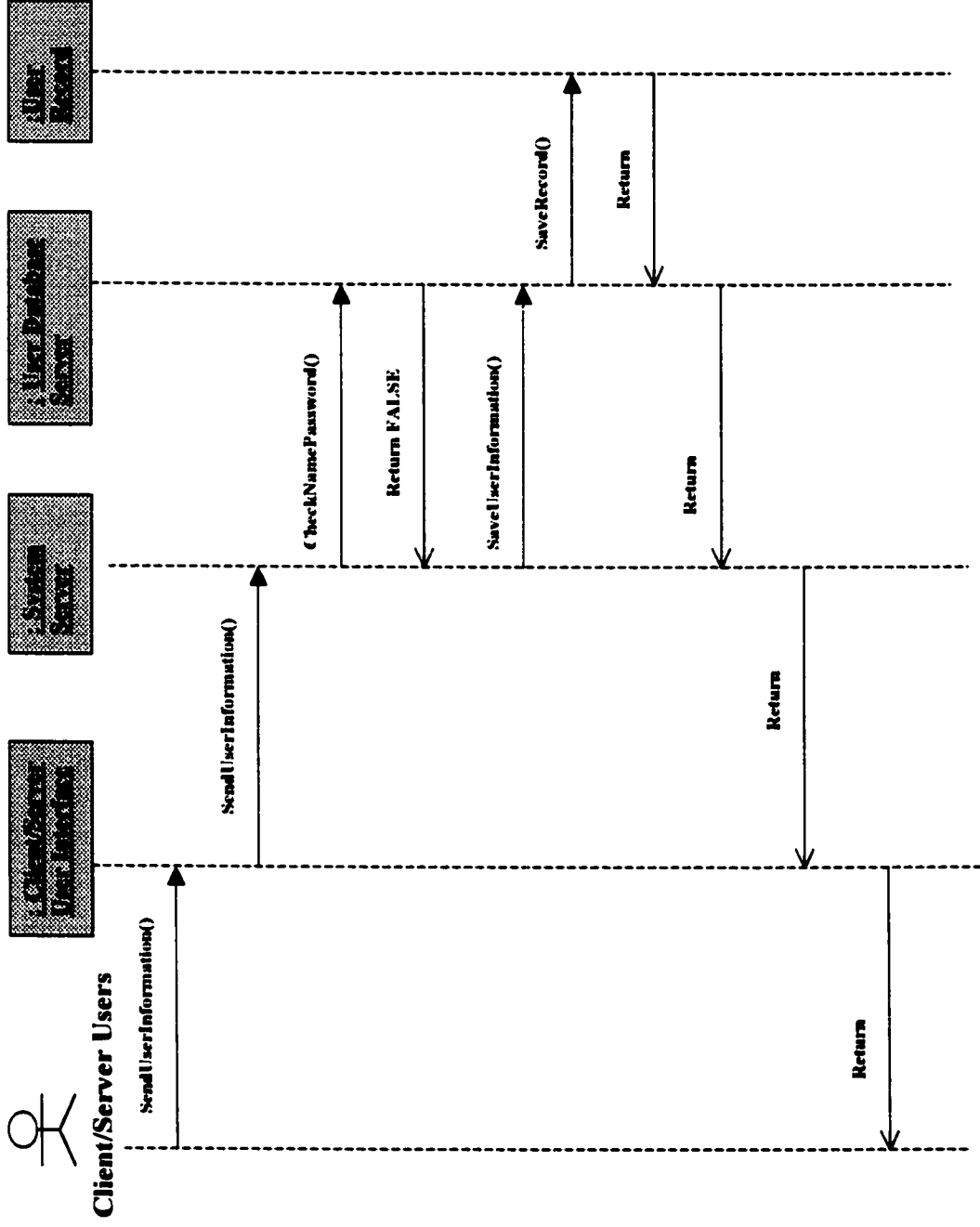


Figure 10: User Register Sequence Diagram

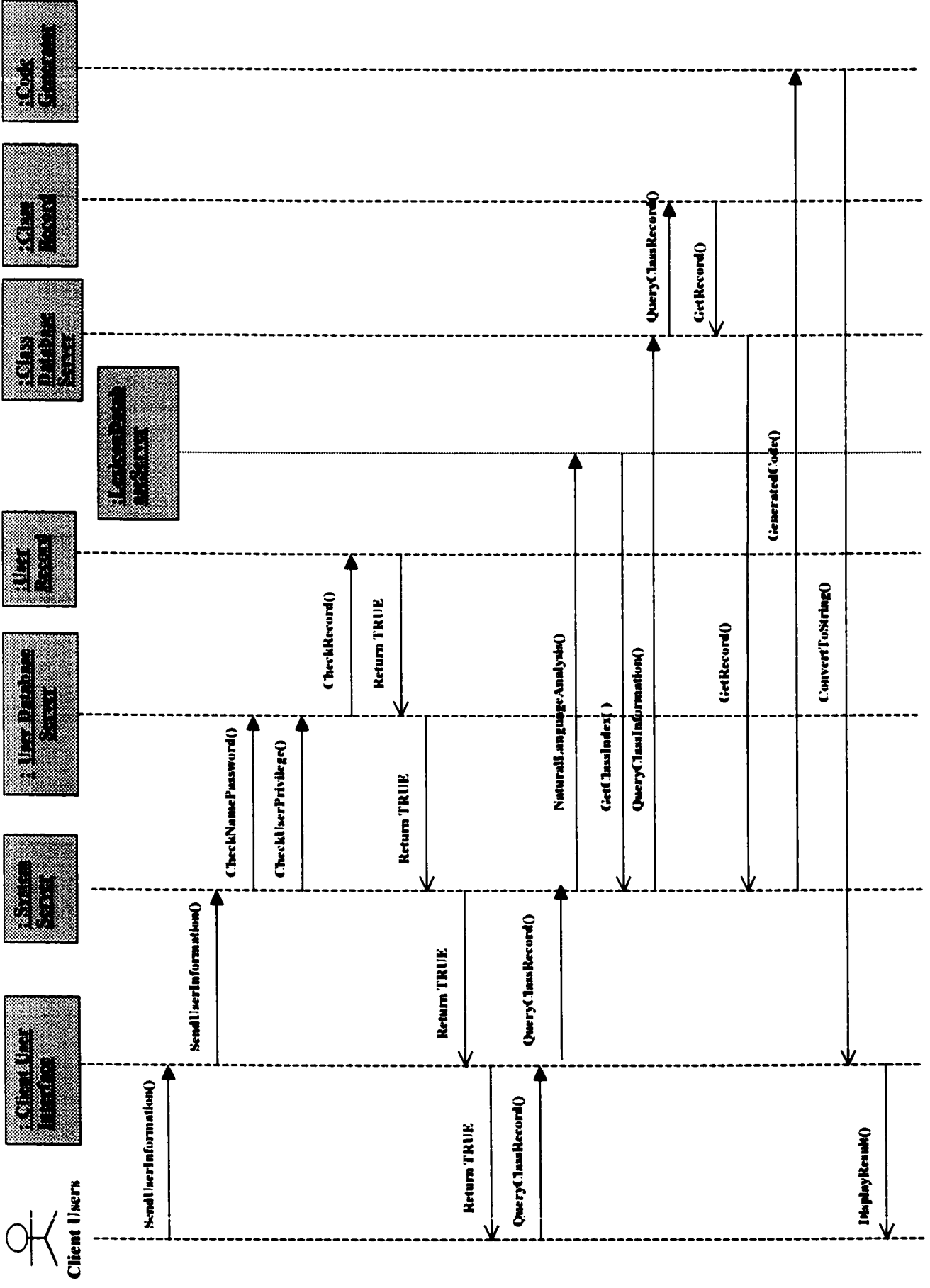


Figure 11: Client User Query Library Sequence Diagram

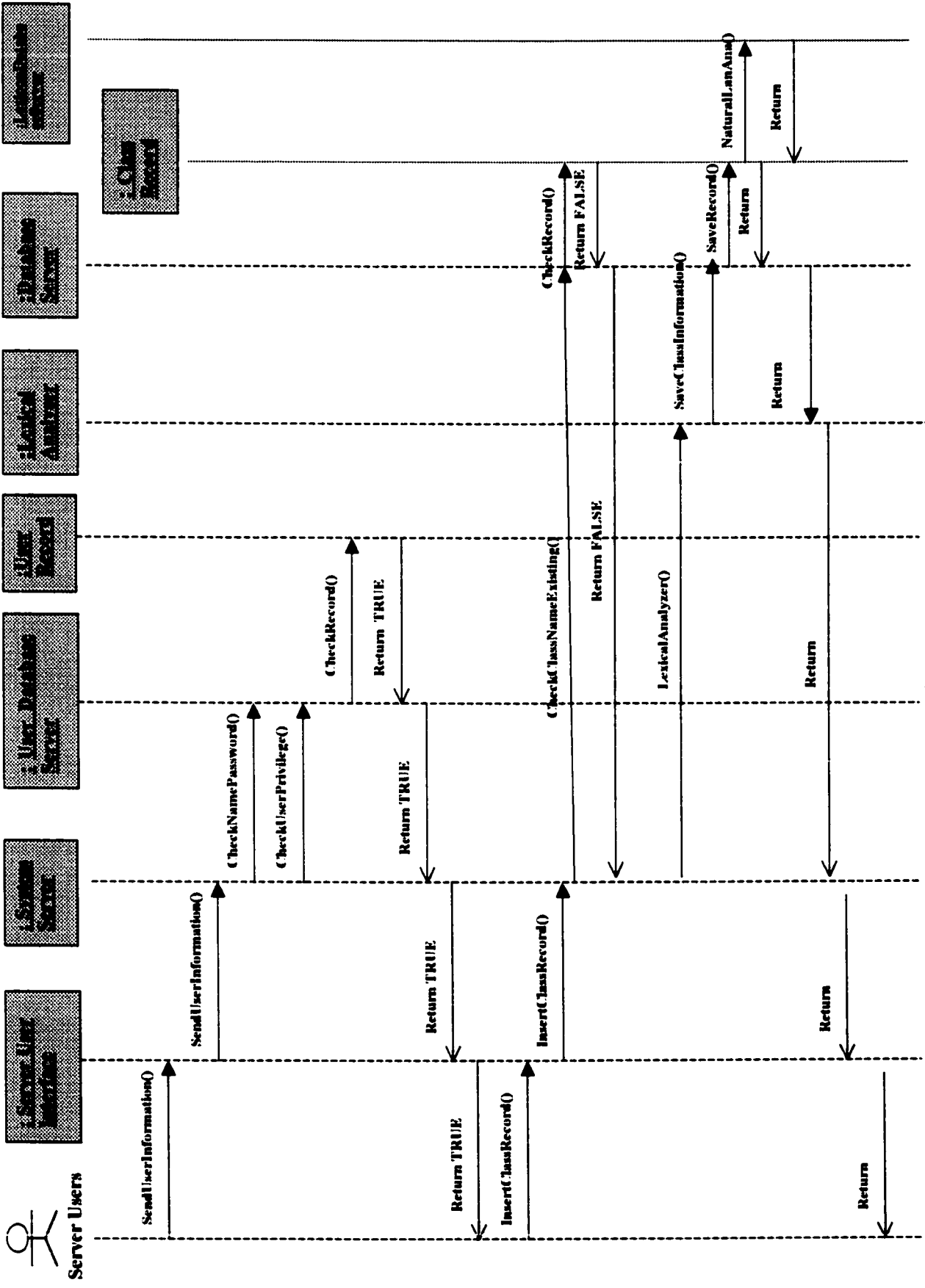


Figure 12: Server User Insert the Component Sequence Diagram

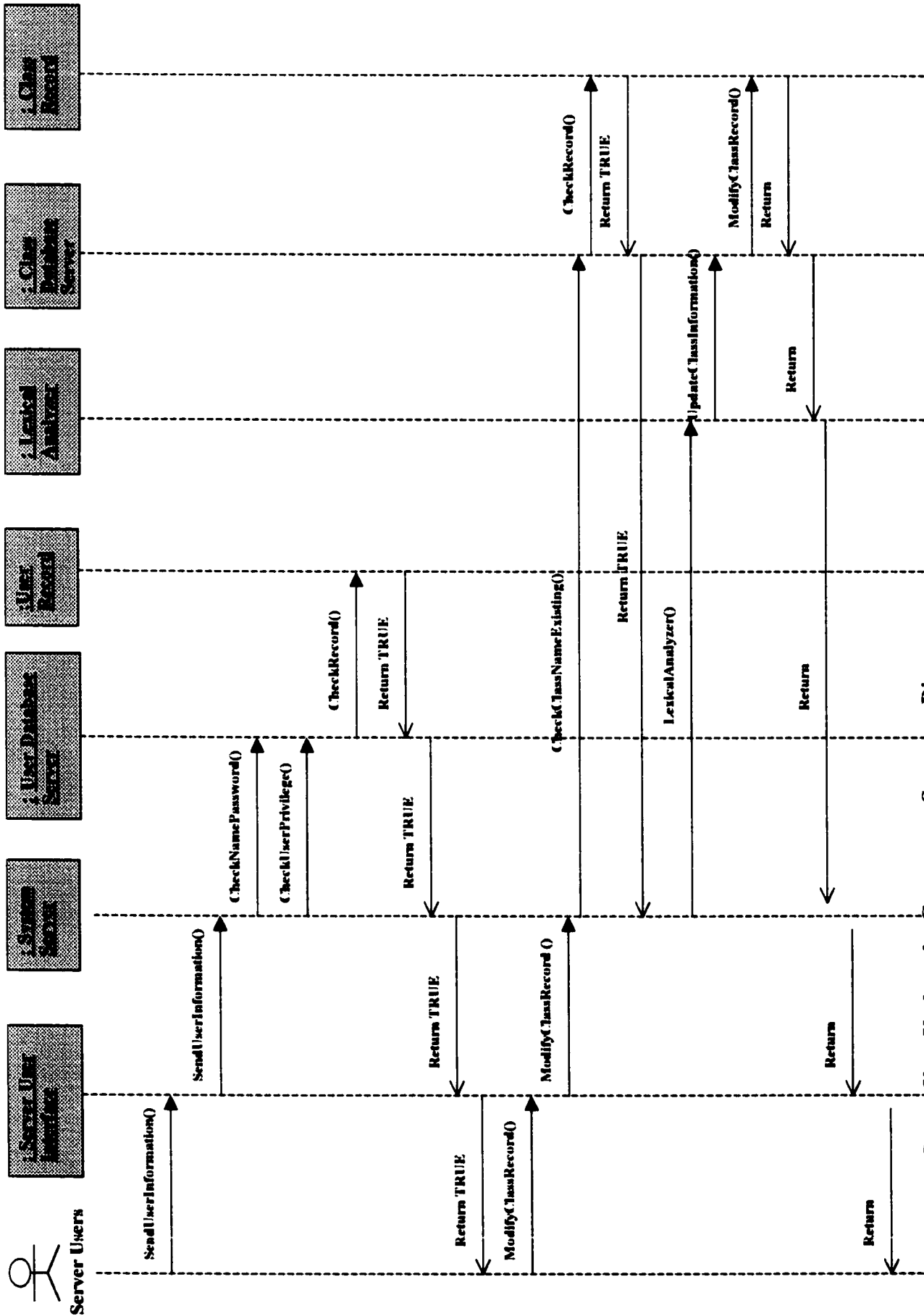


Figure 13: Server User Update the Component Sequence Diagram

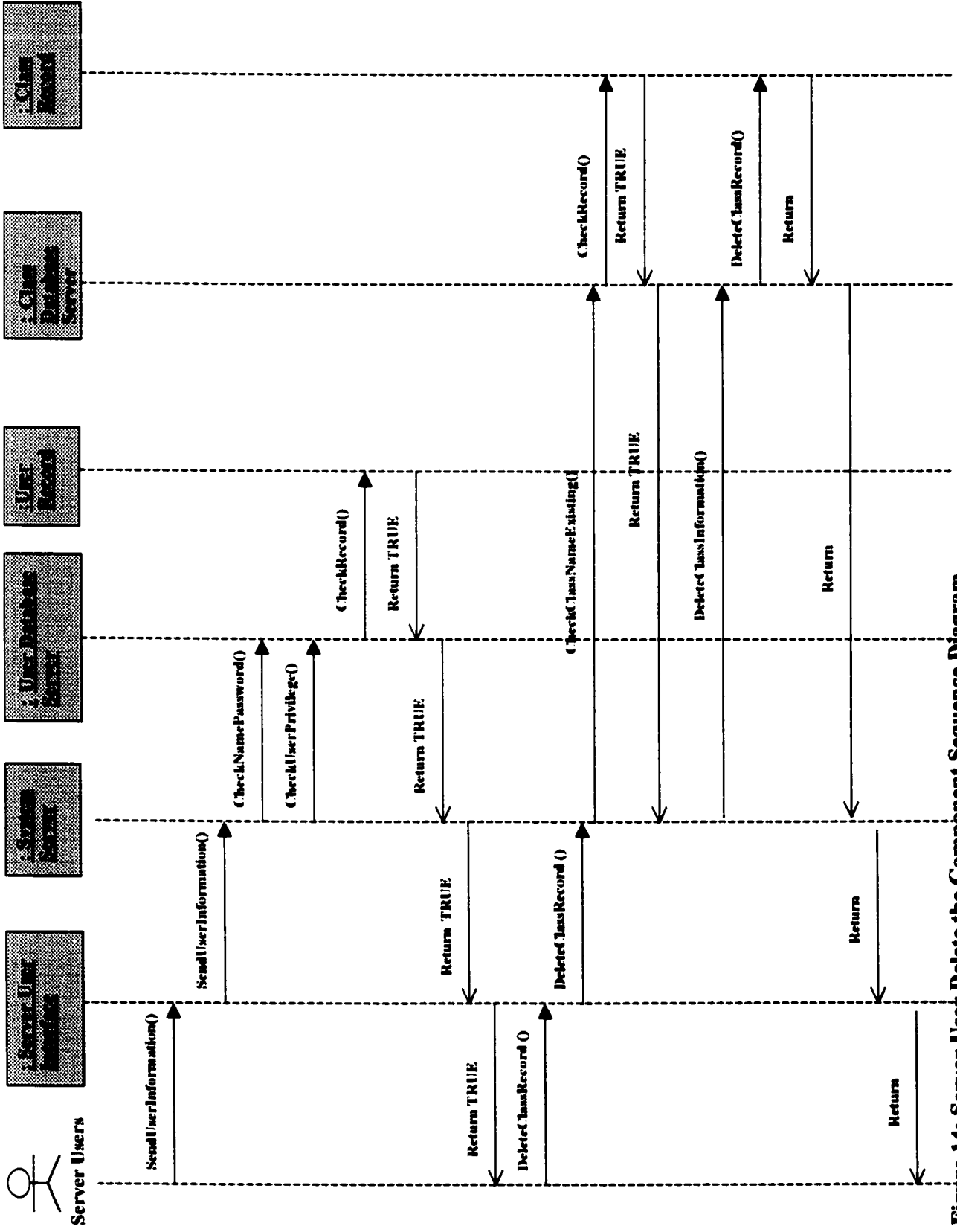


Figure 14: Server User Delete the Component Sequence Diagram

The five major sequence diagrams shown in Figure 10, Figure 11, Figure 12, Figure 13, and Figure 14 are used to describe the procedures of the user register, client user query,

Figure 10 shows the user register sequence diagram to describe the procedure of the user register to the DORLM.

Figure 11 shows the client user query library sequence diagram to describe the procedure of the client user query and code generation. The partial natural language-based retrieval process described in Chapter 2, section 2.3 is performed here.

Figure 12 shows the server user insert a new component to the database. The automatic indexing process, which is described in Chapter 2, section 2.2, including the lexical analysis, case parsing, and the parsing grammar of the case parsing process shown in Table 3 is performed here. The component data is stored in the component database and a frame-based representation of this component is stored in the component classification database.

Figure 13 shows the server user update an existing component.

Figure 14 describes the procedure of the server user delete an existing component.

CHAPTER 4

Using Prototype System and User Interfaces

This chapter describes how to use the DORLM and user interfaces. Some captured software screens and examples are included to illustrate various operations applied to the system. Also the system testing and some limitations of the DORLM are introduced.

4.1 System Setup

Currently, the DORLM was installed and tested on the two kinds of platforms, Windows 98/NT and LINUX.

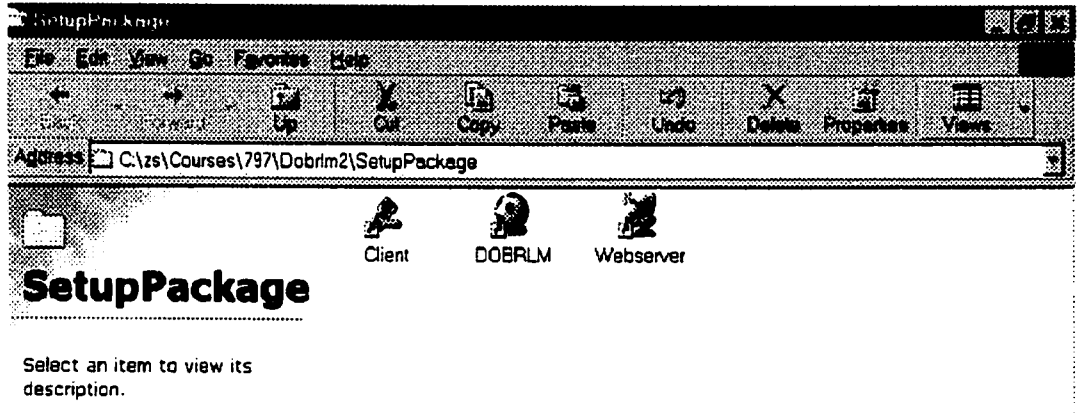
Setup in Windows 98/NT environment:

Figure 15 shows the DORLM setup package with three executable program files in Windows 98/NT. They are **DORLM**, **WebServer**, and **Client**. To run these three programs, simply double-click the appropriate file icons.

- ✓ Run the server-side application: double-click **DORLM** icon, it will set up the DORLM system server, then the server interface shown in Figure 16 will pop up. Server user can register, login the system, insert, update, delete, and query software components through this interface and server operation interface shown in Figure 17.
- ✓ Run the client-side application: double-click **WebServer** icon to set up web server. Then open the client interface by either double-click **Client** icon on the

server machine, or input the URL address through the Web browser to open the DORLM web page across the Internet.

Figure 15: DORLM setup package in Windows 98/NT



Setup in Linux environment:

To run the DORLM in the Linux 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
```

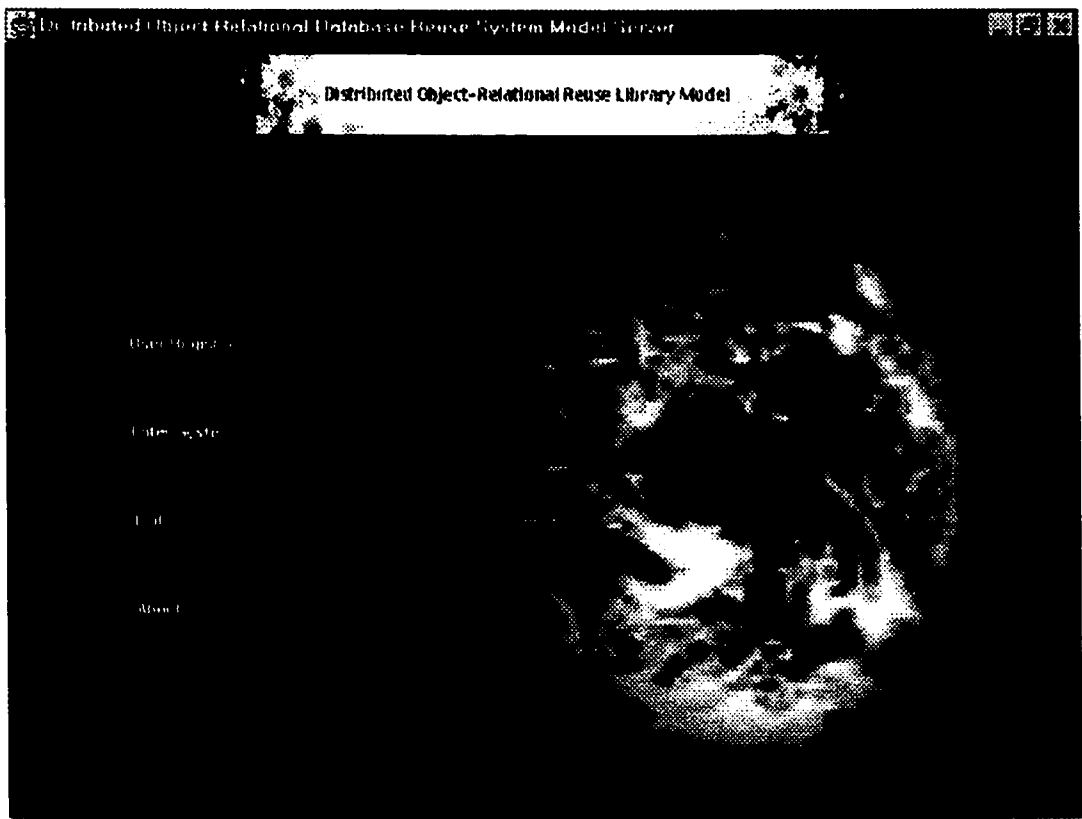
To open the server-side interface, just input **java DORLM_SC -ORBconfig orb.cfg** in the appropriate directory of a new terminal or a machine installed the DORLM, then the server interface shown in the figure 16 will pop up. To open the client-side

interface, just input the URL address through the Web browser to open the DORLM web page across the Internet.

4.2 Server Interface

The server-side user interface shown in Figure 16 has four buttons. If you are new user, you have to click **User Register** Button to fill out register form to get username and password to access the system. If you already have the user name and password, you just click **Enter System** button to login the system. You can click **About** button to get help information, or click **Exit** button to exit program.

Figure 16: The server user interface

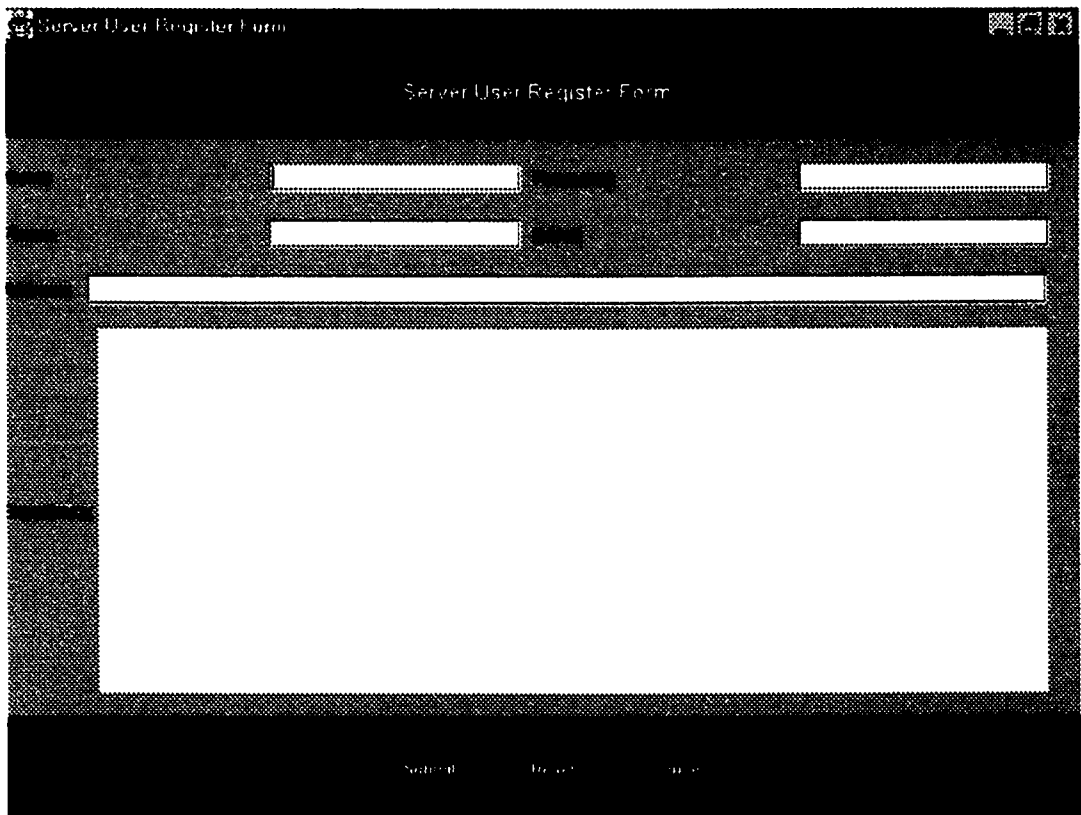


4.2.1 Log on the system

If you are new user of the DORLM, you have to click **User Register** button firstly to fill out the register form to specify the user name and password to access the system.

Figure 17 shows the server-side user register interface.

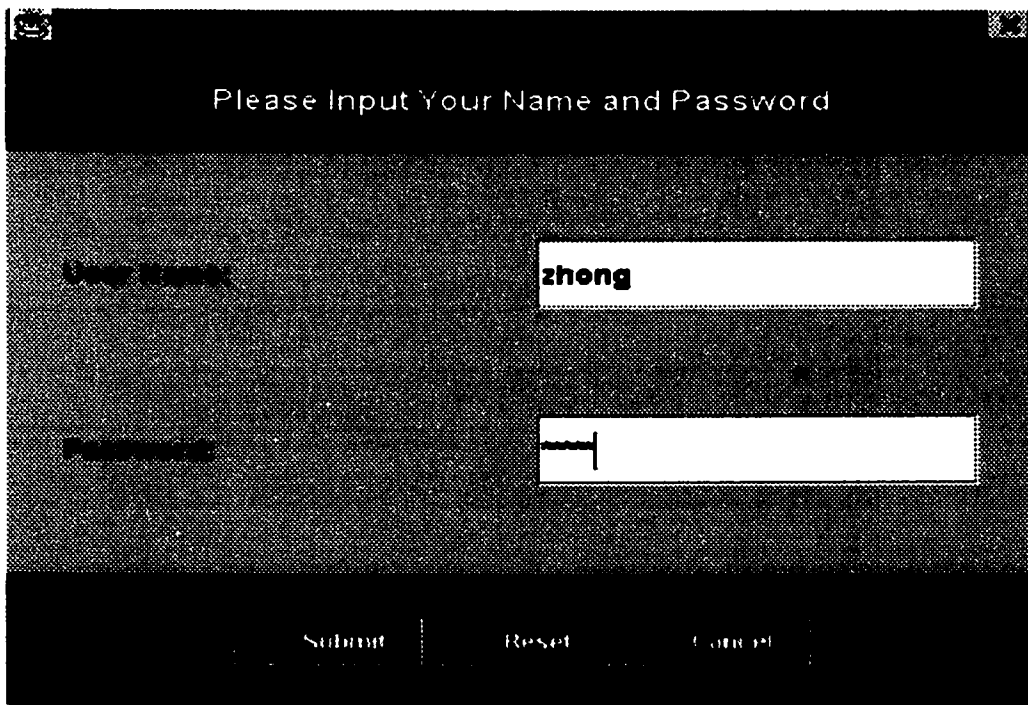
Figure 17: The server user register interface:



The screenshot displays a web browser window titled "Server User Register Form". The page content includes a header with the title "Server User Register Form". Below the header, there are four input fields arranged in two rows. The first row contains two fields, and the second row contains two fields. Below these fields is a large, empty rectangular area. At the bottom of the form, there are three buttons labeled "Submit", "Reset", and "Cancel".

If you click **Enter System** button, a login dialog shown in Figure 18 will pop up. After you enter your user name and password, click **Submit** button. When the user name and password you input are matched with the user information database, the server operation interface shown in Figure 19 will pop up. You can click **Reset** button to re-input, or click **Cancel** button to return back to the server user interface.

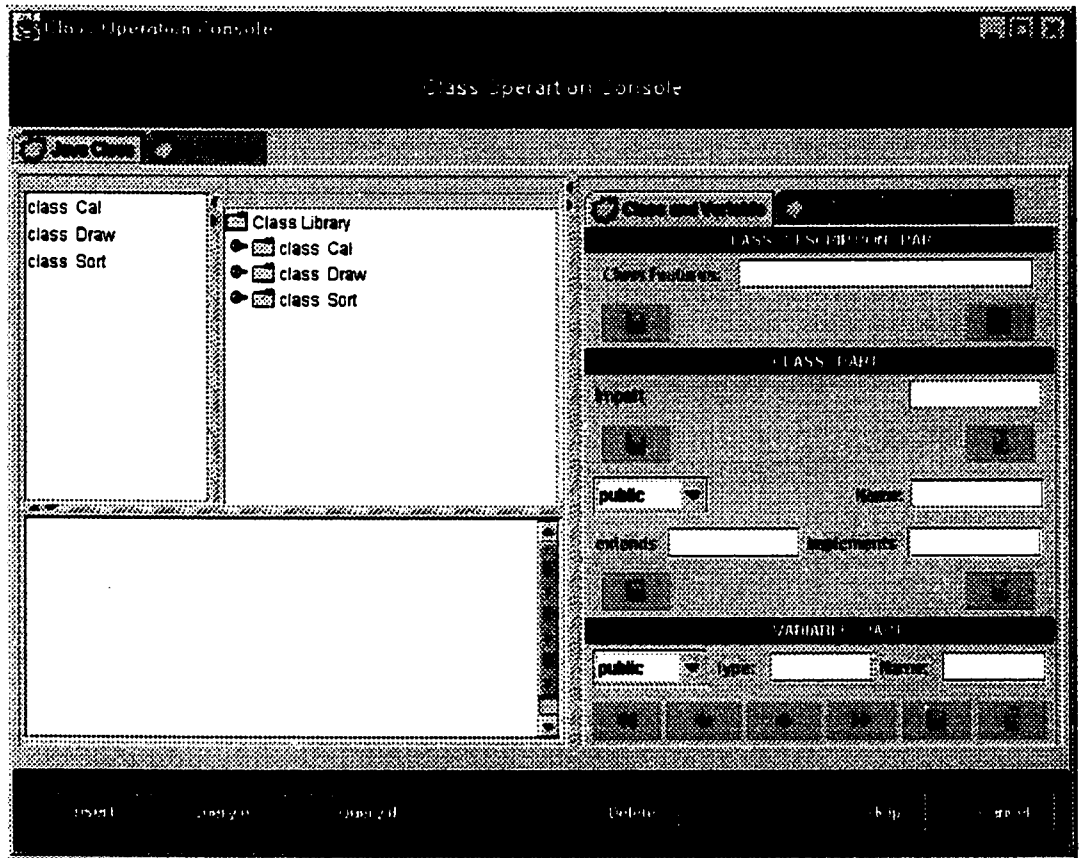
Figure 18: The server user login dialog



4.2.2 Server Operation Interface

Figure 19 shows the server-side operation interface. This interface consists of class index panel, library index tree panel, class specification editor panel, display panel, and operation control panel.

Figure 19: The server operation interface:



- ✓ Class index panel displays the class names stored in the SRL.
- ✓ Library index tree panel displays the classes with their methods displayed using a tree structure. The data of both class index and library index tree are obtained dynamically through the Event Channel of the DORLM.
- ✓ Class specification editor panel is used to insert, update class component to acquire the component specification.
- ✓ Display panel is used to display the component user input.
- ✓ Operation control panel has **Insert, Query (C), Query (F), Modify, Delete, Restore, Help, and Cancel** buttons to execute various operations.

4.2.3 Insertion of Components

If the user clicks **Insert** button on the operation control panel, he/she can insert a class component through the specification editor step by step. Figure 20 and Figure 21 together show a sample insertion process of inserting a Java class.

Figure 20: Insert class component interface 1

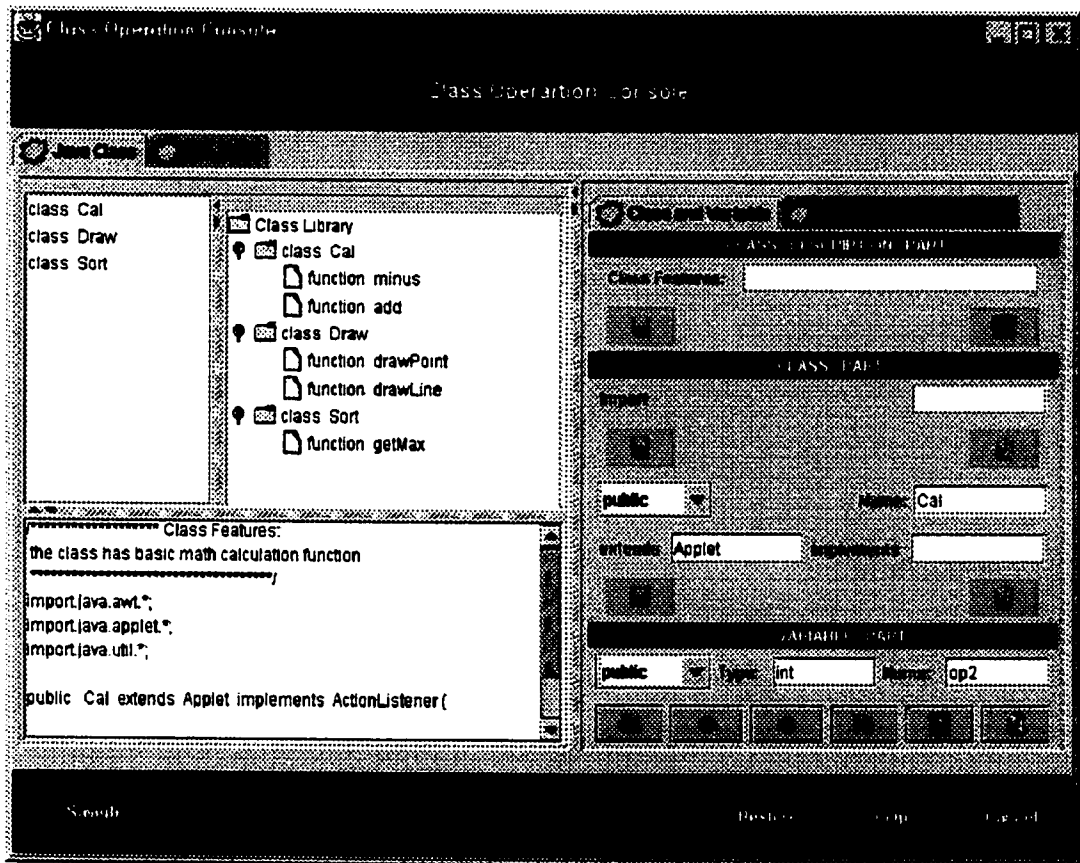
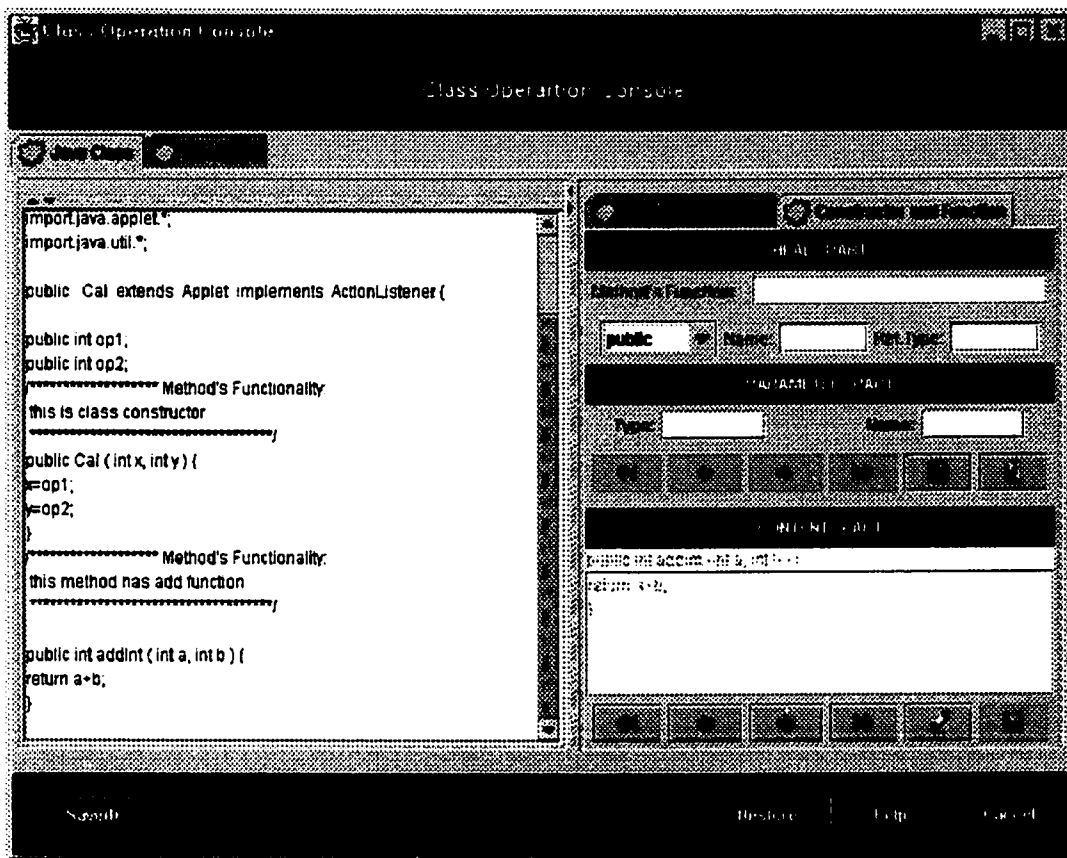


Figure 21: Insert class component interface 2



4.2.4 Server User Query Components

If server-side user click one class name in the class index panel, and click **Query (C)** button, user can get this class information from the SRL. Figure 22 shows the result of a sample class query. If server-side user click one method in the library index tree panel, and click **Query (F)** button, user can get this method information from the SRL. Figure 23 shows the result of a sample method query. The query results may be used to operate update function.

Figure 22: The result of a sample class query

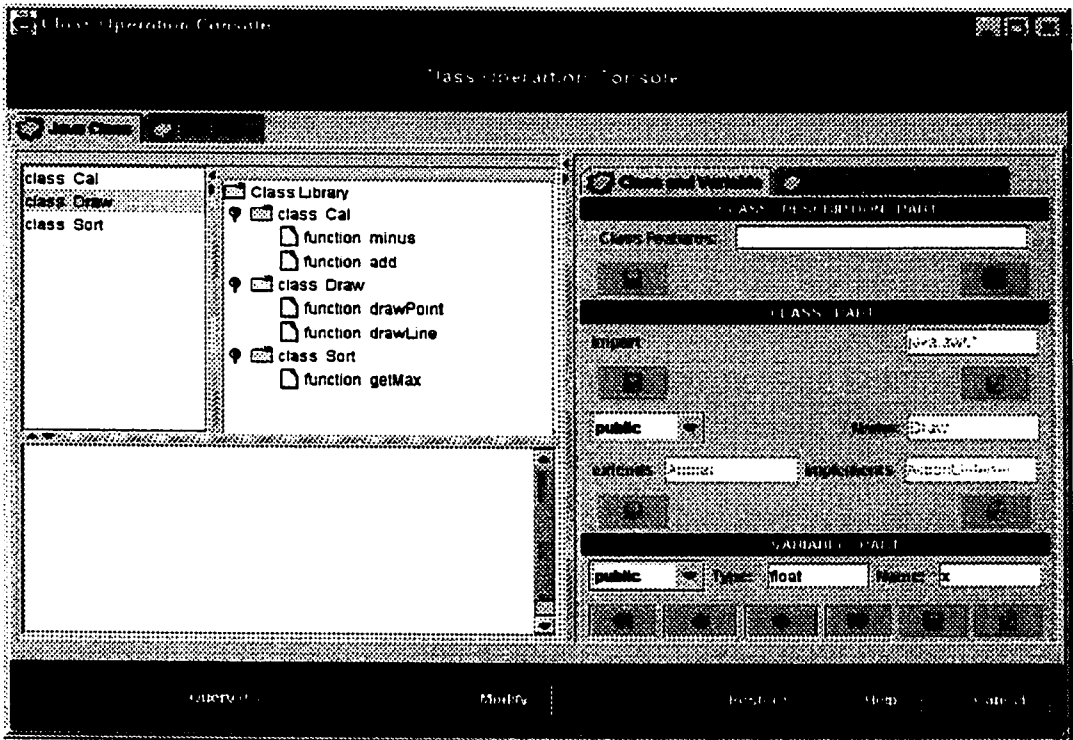
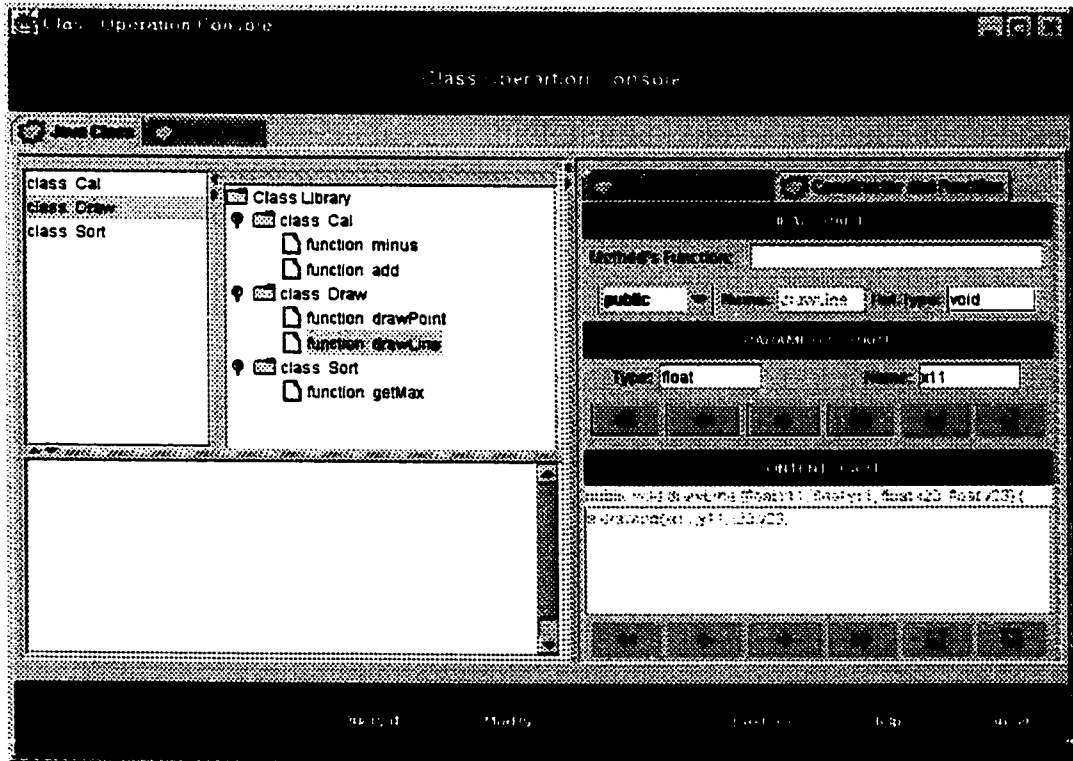


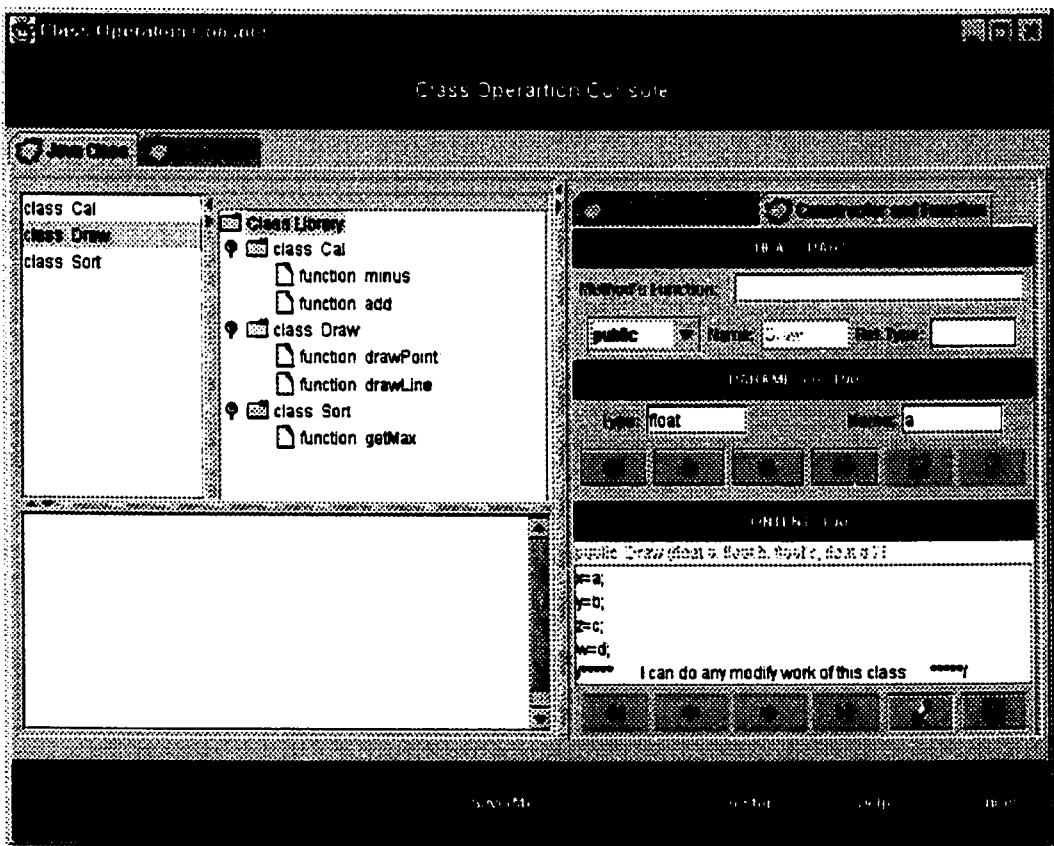
Figure 23: The result of a sample method query



4.2.5 Server User Modify Components

Based on the result of class or method query, user can modify the component information if he/she click **Modify** button. After user finish modifying data through the class specification editor, just click **Save** button to store the modified component data into the SRL. Figure 24 shows a sample modify process:

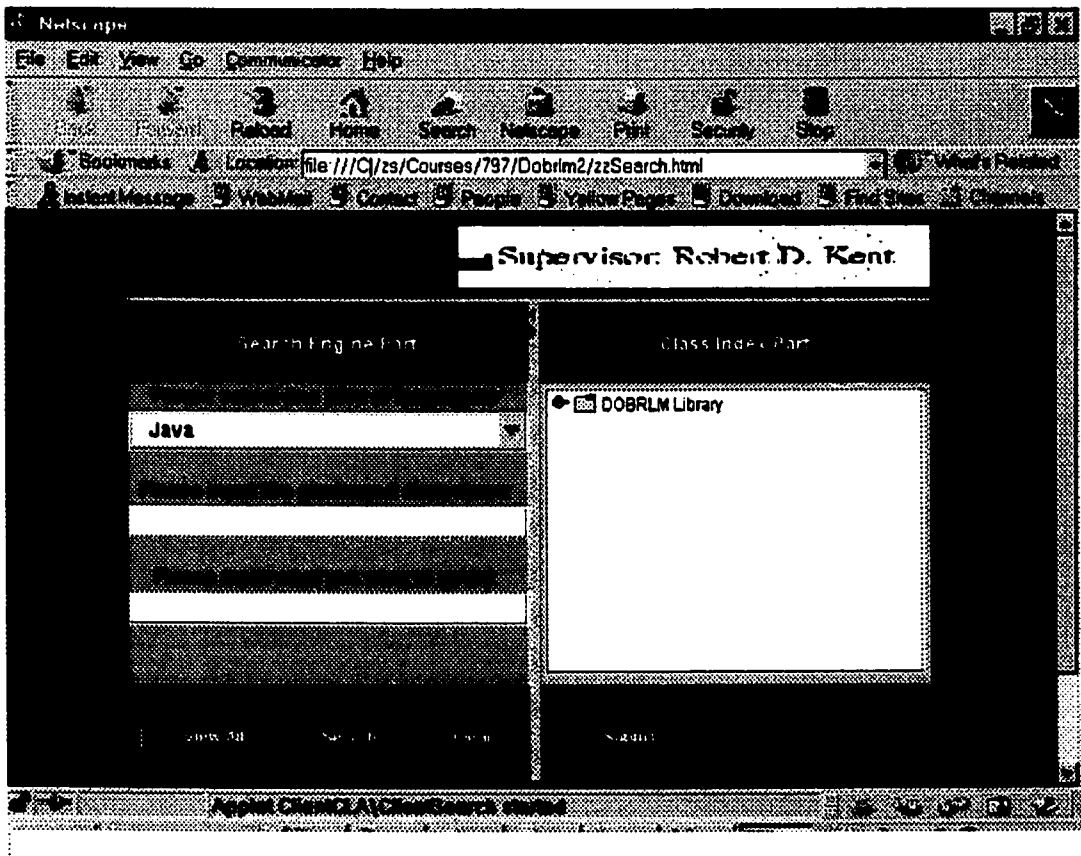
Figure 24: A sample modify process



4.3 Client Interface

After client application starts, a client-side user interface shown in Figure 25 will pop up. The client-side user interface is a Java applet embedded in the DORLM home page. So the client-side user interface can be downloaded via a Java enable web browser. The client-side user interface consists of input panel, display panel, and operation panel.

Figure 25: The main client user interface

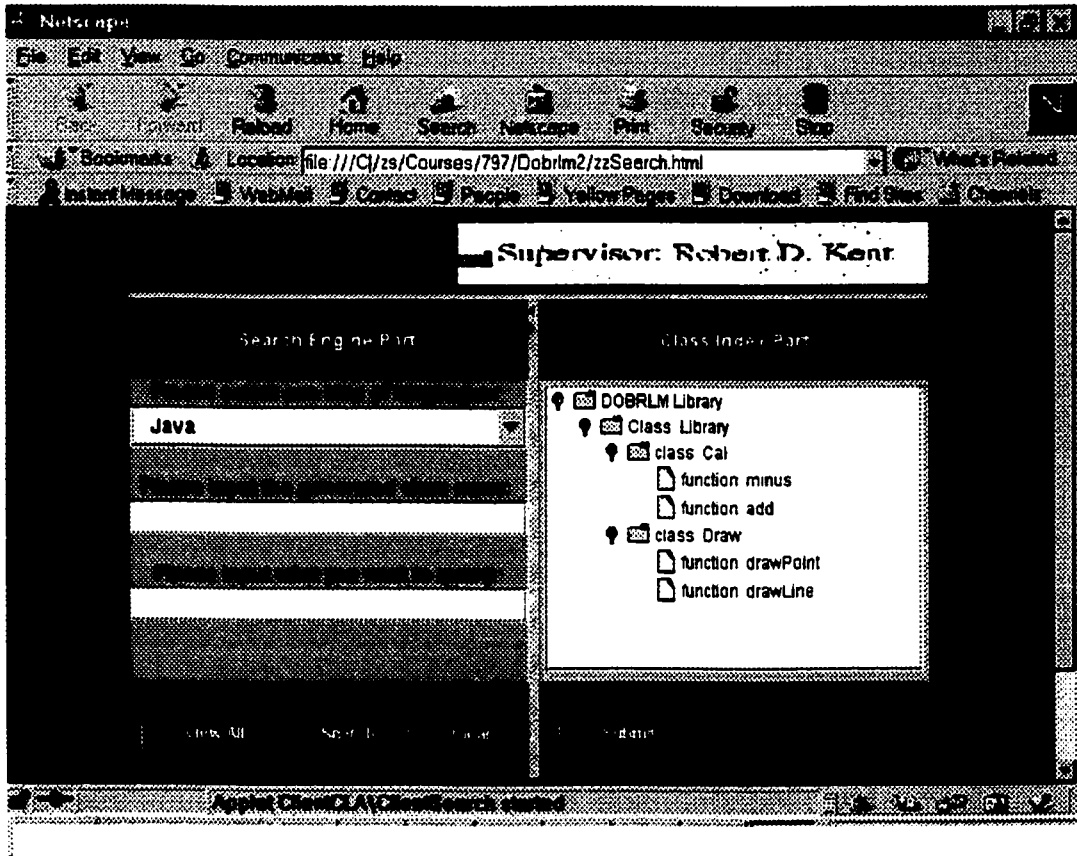


4.3.1 Browse the Library

A browse function is used to obtain the whole component index information from SRL. It is very useful when the SRL is small. A user just need choose a language from

the input panel, and click **View All** button, the query result will be displayed in the display panel. Figure 26 shows the browse result:

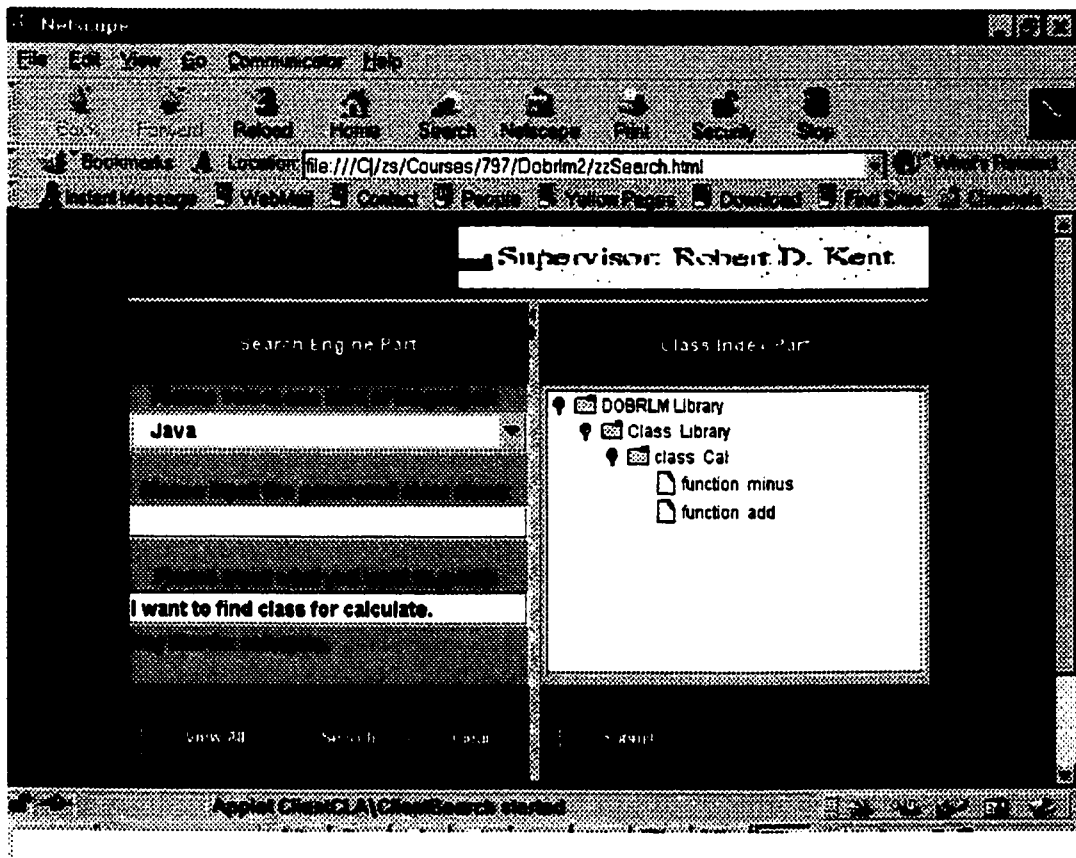
Figure 26: The result of the library browsing



4.3.2 Client User Retrieval of Components

If a user want to obtain a specific component based on his/her requirement, one has to choose a language, and input query in natural language (free text), then click **Search** button, the query result will be displayed in the display panel. Figure 27 shows a sample query (“I want to find class for calculate.”), and the query result.

Figure 27: A sample query and query result interface



4.3.3 Code Generation

After the component candidates index are displayed in the display panel through browsing or retrieval function. If a user find the specific class, method, or a set of classes and methods he/she need, the code generator function can deliver an executable code through these component specification from the SRL. What a user should do is specify a generated class name through the input panel, click the specific classes or methods, and click **Submit** button, follow the generation instruction to input the specific requirement, finally click **Generate Application** button. The generated code will be displayed in a pop-up frame.

An example of these processes is described in the following. Suppose a user find a minus method he/she need from candidates. He/she wants to get this class code. He/she just follows the following steps:

- ✓ Specific the generated class name: Calculation
- ✓ Click function minus in the Class Index Part, and click **Submit** button. The dialogs shown in Figure 28 and Figure 29 will pop up. Input the specific data.
- ✓ Click **Generate Application** button to get the result. Figure 30 shows this result.

Figure 28: A sample code generator process 1

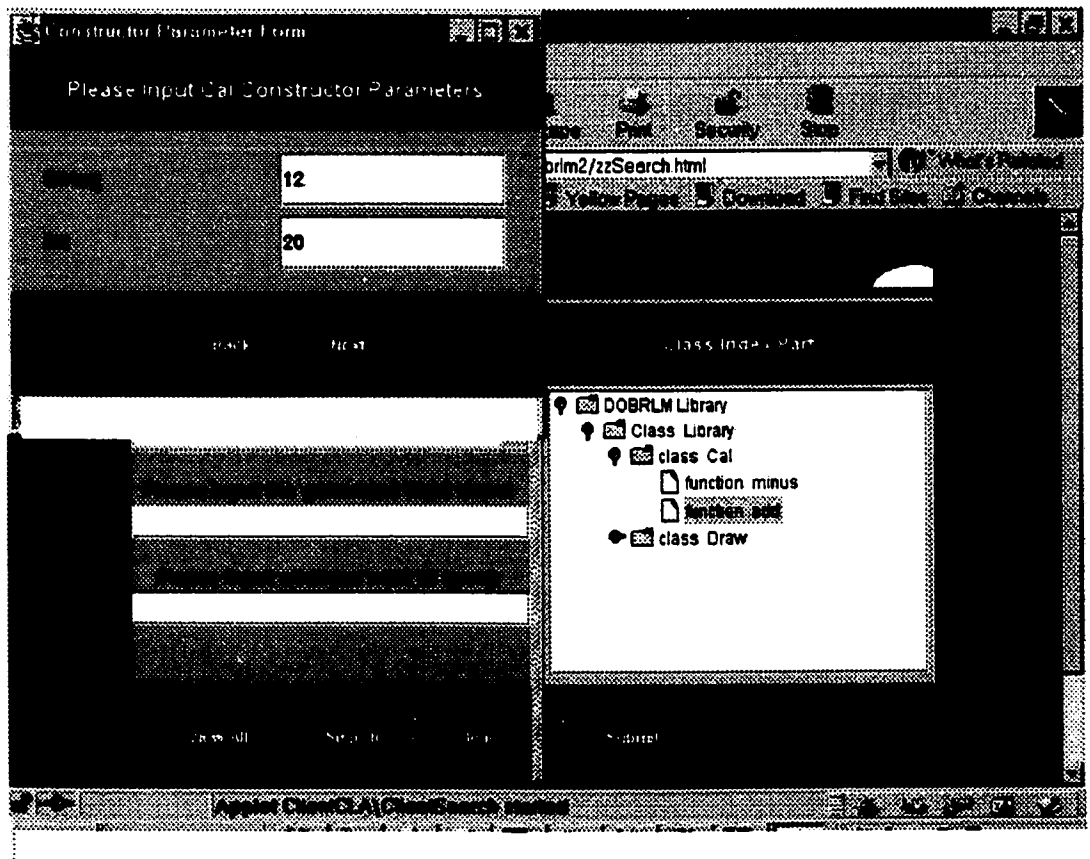


Figure 29: A sample code generator process 2

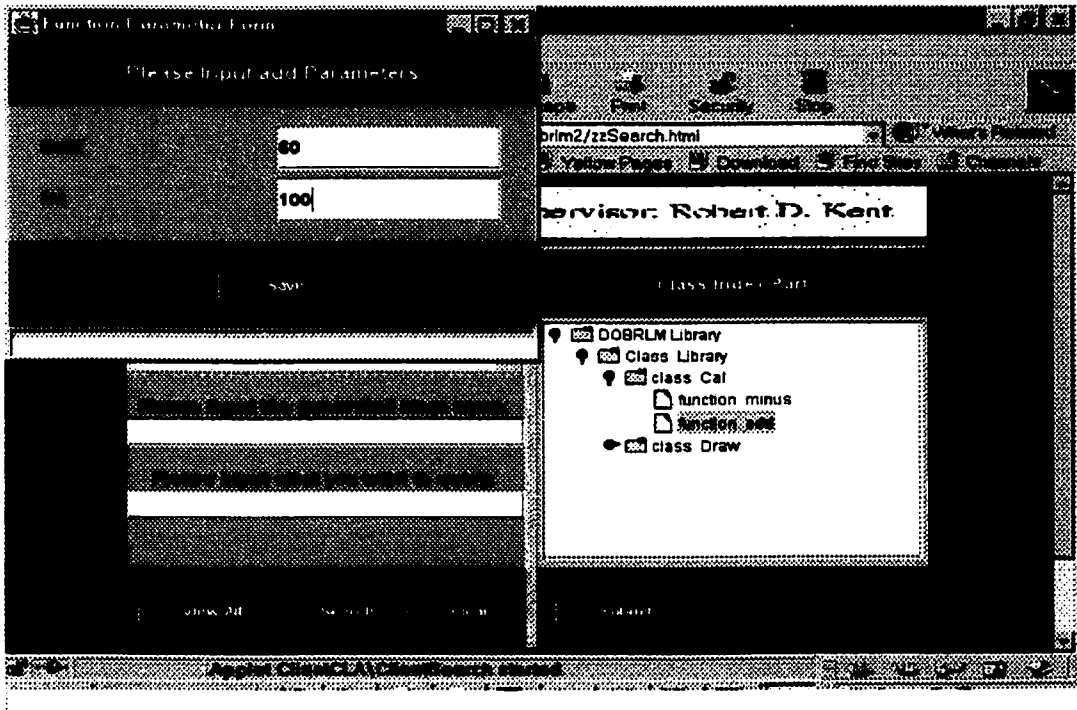
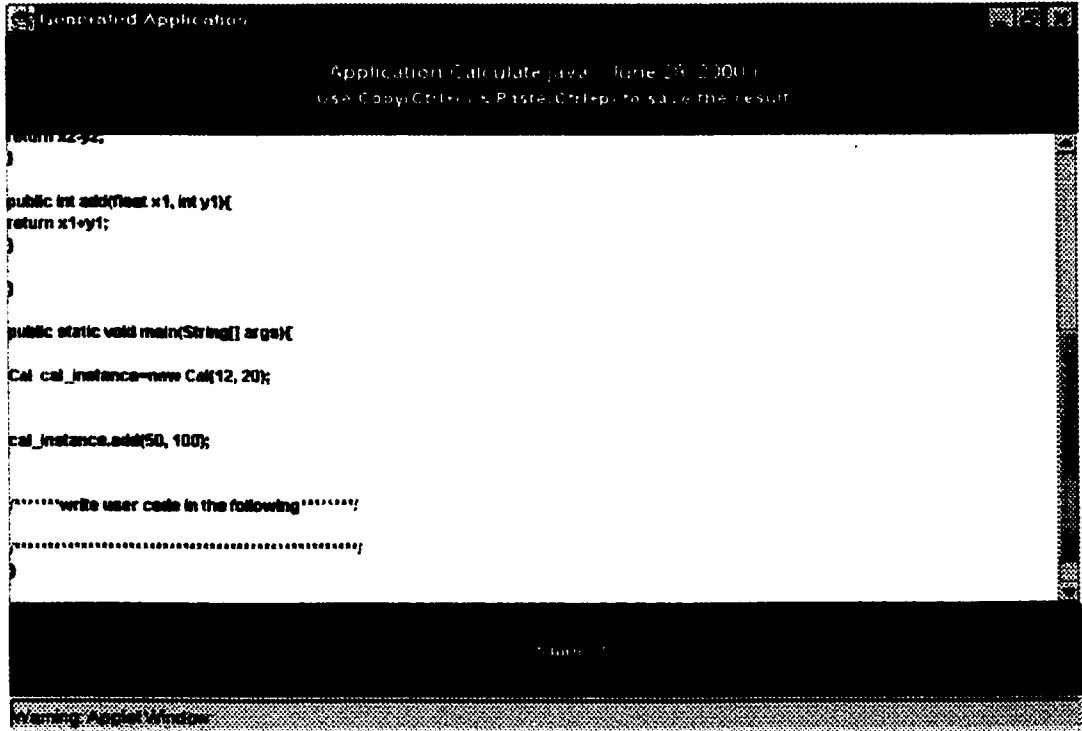


Figure 30: The result of a sample generated code



4.4 System Testing and Limitations

4.4.1 Overview of Testing

This section introduces experiences gained from the DORLM design, implementation, and demonstration aiming to address some limitations of this prototype system. A theoretical testing analysis didn't be applied in detail in this thesis work due to the limitation of time and resources.

The DORLM was developed from a variety of design documents. The documents include the Design Scenario, the Software Requirements Document, and the Design Specification Document. These practices provided the DORLM reasonable design, design refinements, and task implementation.

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 the DORLM in the distributed environment. The results showed the DORLM had reasonable operational behaviors of both the client-side and the server-side, except some bugs, which are discussed in section 4.4.2.

- ✓ **Simulation on the single PC:** At first, The DORLM was developed and stimulated testing on the single PC (the DORLM, Windows 98, ORBacus, and Netscape 4.6 with Java 1.2.2 plug-in are installed). The operations, which include the server-side user register and log on the system, insert, modify, query, and

delete the components, the client-side user browse, query, and generate code, were performed successfully. (Note: TCP/IP in the single PC is 127.0.0.1)

- ✓ **Simulation on the HPC Server (*hpc.uwindsor.ca*) Machine:** The operations, which include the server-side user register and log on the system, insert, modify, query, and delete the components, the client-side user browse, query, and generate code, were performed successfully.

- ✓ **Simulation on the HPC server and the PC:** The test was done on the HPC server (*hpc.uwindsor.ca*) and the PC (the DORLM, Windows 98, ORBacus, and Netscape 4.6 with Java 1.2.2 plug-in are installed). The test was executed in the following steps:

- Connect to the Internet using the PC, and telnet to the HPC server.
- Run the system server of the DORLM on the HPC server: type the following commands in the three terminals of the HPC.

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

- Run the server-side user interface on the PC at home: type the command on the DOS. The various operations, including the server-side user register and log on the system, insert, modify, query, and delete the components, were performed successfully.

```
java DORLM_SC -ORBconfig orb.cfg
```

- Run the client-side user interface on the PC at home: open the Netscape, to input the http address. The various operations, including the client-side user browse, query, and generate code, were performed successfully.

<http://hpc.uwindsor.ca/~azhong/zzSearch.html>

- ✓ **Simulation on the two UNIX Workstations:** The test was done on the two workstations according to the following steps:

- Log on the HPC server in a workstation and log on the server (137.207.16.120) in another workstation.
- Run the system server of the DORLM on the workstation, which log on the HPC server: type the following commands in the three terminals of the workstation.

*Java com.ooc.CosNaming.Server -ORBconfig orb.cfg
Java ServerCLA.Bserver -ORBconfig orb.cfg
Java WebServer.WebServer*

- Run the client-side user interface on the workstation, which log on the server (137.207.16.120): open the Netscape (Java 1.2 Plug in is required), to input the http address: <http://hpc.uwindsor.ca/~azhong/zzSearch.html>. The various operations, including the client-side user browse, query, and generate code, were performed successfully.

4.4.2 Limitations of the DORLM

Based on the observations of the above system testing, some major bugs and limitations of the DORLM were reported in the following. The refinements of the DORLM are the future work of this thesis.

- ✓ When the system server shut down, the client-side user interface, a Java applet embedded in the HTML, still can be download to the client user machine, but no notification was provided to the user.

- ✓ The system server could not run without kill the previous processes after the system server shut down.

- ✓ The system could not insert the component with the same name of the other components and the component with more than one constructor.

- ✓ The limitation terms built in the Lexicon 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 functionalities obtained during the component insertion. 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, these work leaves behind the re-user.

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

In this thesis we have investigated the problems of structuring the distributed software reuse library to support reuse-oriented program development and developed a prototype software system, DORLM. This system, although crude, is based on the concept of maintaining a structured software reuse library with various tools to serve as a reusable software components builder and provider allowing for effective insertion, modification, deletion, retrieval, and delivery of object-based software components to support reuse-oriented program development in a distributed environment.

5.1.1 Innovation

The most innovative feature presented in the DORLM is the integration of DBMS, IRS, NLP, and CORBA for software reuse and reuse-oriented program development in a distributed computing context.

Different from those SRL system prototypes [PBAI95] [KFTAM97] [QIUAN] [QIANG99] proposed at the University of Windsor to solve the problems of functional-based components classification and retrieval, the DORLM not only tries to solve the problems of object-based components classification and retrieval, but also tries to solve the problems of components adaptation and generation.

5.1.2 Achievements

Based on the work carried out in this thesis we have achieved the following:

- ✓ Provide a foundation for code (i.e. model) reuse, scalability and flexibility of computation in distributed environments.
- ✓ Using the DORLM with its structured object-based software library based on the component structure we proposed provides a practical storage approach for reuse-oriented program developments, especially for reuse in the large.
- ✓ Classification and retrieval based on partial natural language processing techniques provide a user-friendly query interface.

The DORLM is implemented by OOD and OOP. It is easy to modify, maintain, reuse and port to other computing environment. All the interfaces in the DORLM are graphic, user-friendly, and easy-to-use.

5.2 Future Work

Based on the achievement of this thesis, the following future research directions are recommended:

- ✓ Develop specialized database capabilities and functionality to promote sharing and secure access to the repositories for data and process codes in distributed systems, such as the proposed computational grids [CGRID00].

- ✓ Further explore software retrieval based on natural language specification. In our opinion, processing of natural language will contribute significantly to making information more accurately accessible, although advances in that area are limited by the current computer power and on-line availability of world and language knowledge. Also following this general direction, a knowledge repository system should be built for the VPMS system.
- ✓ Using “GRID” [CGRID00] concept to extend the DORLM to support heterogeneous Network protocols rather than only TCP/IP and also to support High Performance Computing (HPC). The goal of the C3-Grid system is to deliver to users a uniform interface to multiple, distributed HPC facilities.
- ✓ Further explore the enhancement of network security. Network security [BROWN99] is a fairly major concern in the current distributed systems; however, the DORLM only deals with the basic security issue. The term *network security* covers an incredible array of services, processes, and requirements for an organization.
- ✓ Apply a theoretical testing analysis and refinements of the DORLM.

APPENDIX A: The DORLM Code Definition

A.1. Overview

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

A.1.1. CORBA IDL: *zz.idl*

This file contains the IDL interfaces of the DORLM.

```
module DORLM{
    //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);

/***** modeling the component*****/
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 functionality;
    string attribute;
    string name;
    fun_para_list para;
    string body;
};
typedef sequence< function > function_list;

//insert the component
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 fun
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 ();
};

```

A.1.2. 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

```

A.1.3. The server-side user class: *DOBRLM_SC.java*

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

A.1.4. Package: *ServerREG*

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

A.1.4.1. Classes

- ✓ This class is defined to build the graphic user interface of the server-side user main interface.

ServerGUI

- ✓ These classes are defined in hierarchy to build the graphic user interface of the server user register

ServerUserReg

Top

Center

ButtonPanel

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

RegUser

CorbaProcess

A.1.5. Package: *ServerCLA*

This package contains classes that have the major functionalities of the DORLM. The functionalities 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.

A.1.5.1. Classes

- ✓ These classes are defined in hierarchy 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.

BServer

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

ServerPass

- ✓ These classes are defined in hierarchy 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 getting the synonyms.

GetLexicon
GetTerm
Parse
GetSyn

- ✓ These classes are defined in hierarchy 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

A.1.6. Package: *WebServer*

This package contains classes that have the major functionalities of the web server of the DORLM. The functionalities include client-side connection, system server

connection, database server connection, multithread handle, and various operations for the client-side user requests.

A.1.6.1. 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

A.1.7. 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.

A.1.7.1. Classes

- ✓ These classes are defined in hierarchy 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

A.1.8. HTML

- ✓ This is a web page of the DORLM embedded with the Java applet, *ClientSearch*, served as the client-side user interface.

zzSearch.html

BIBLIOGRAPHY

- [AMILI95] Hafdth Mili, Fatma Mili, and Ali Mili, "Reusing Software: Issues and research Directions," IEEE Transactions on Software Engineering, Vol. 21, No. 6, 1995
- [AMILI98] A.Mili, R.Mili, and R.T.Mittermeir, "A survey of software reuse libraries," Annals of Software Engineering 5, 349-414, 1998
- [AMICH00] Amir Michail, "An Exploratory Approach to Software reuse," PHD thesis, University of Washington, 2000
- [AMBRO94] A.P. Ambrosio, "Introducing Semantics in Conceptual Schema Reuse," CIKM'94, Proceeding of the 3rd Int. Conf. On Info. & Know. Management, PP 50-56, 1994
- [ARABA98] S. Araban, "A Two-level Matching Mechanism for Object-Oriented Class Libraries," Ada-Europe 1998: Uppsala, Sweden, PP. 188 – 200, 1998
- [ATKIN96] Steven Atkinson, "A Formal Model for Integrated Retrieval from Software Libraries," Proc. Technology of Object-Oriented Languages and Systems: TOOLS21, Prentice Hall, 1996
- [ASSET00] ASSET – Asset Source for Software Engineering Technology, <http://source.asset.com/information/home.html>
- [BROWN99] Steven Brown, "Implementing Virtual Private Networks," ISBN 0-07-135185-X, McGraw-Hill, 1999
- [COMPO00] COMPONENT-Component Source for Software Engineering Technology, <http://source.component.com/component.html>

- [CHENG92] Cheng, B. and Jeng, J., "Formal methods applied to reuse," Proceedings of the Annual Workshop on Software Reuse, 1992
- [CORBA99] "Overview of the CORBA Component Model," Object Management Group, 1999
- [CHEN98] H. Chen, J. Martinez, A. Kirchhoff, etc, "Alleviating Search Uncertainty through Concept Associations, Automatic Indexing, Co-occurrence Analysis, and Parallel," Computing, Journal of the American Society for Information Science, 49, No. 3, pp. 206-216, 1998
- [CLEAV88] C.T. Cleaveland, "Building application generators," IEEE software, pp. 25-33, July 1988
- [CGRID00] "Building the C3-Grid, An Investment in Canadian Research and Development Infrastructure," C3-Grid Project Draft Document V5.0,
- [EIKHO99] EI-Khouly, M.M; Far, B.H.; Koono, Z., "A new multi-level information retrieval technique for reuse software components," IEEE, Piscataway, NJ, VOL 6, 1999
- [FALOU95] Christos Faloutsos, Douglas W. Oard, "A Survey of Information retrieval and Filtering Methods, " University of Maryland, College Park, Technical Report, CS-TR-3514, August, 1995
- [FELIC99] P. Di Felice, G. Fonzi, "An improved method for indexing of software," Information and Software Technology 41, 413-420, 1999
- [FISCH98] B. Fischer, " A systematic approach to type-based software component retrieval," PhD thesis, TU Braunschweig, 1998

- [GIRAR93] 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
- [HALL93] Hall, R.J., "Generalized Behavior-Based Retrieval," In Proceedings of the 15th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1993
- [HECKE98] David Heckerman and Eric Horvite, "Inferring Informational Goals from Automatic Queries: A Bayesian Approach," Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, July 1998, Morgan Kaufmann Publishers, pp. 230-237, 1998
- [JBELL92] J.L.Bell, "Reuse and Browsing: Survey of Program Developers," Object Frameworks, D.Tsichritzis (Ed.), Centre Universitaire d'Informatique, University of Geneva, July 1992, pp. 197-213.
- [KFTAM97] Kai Fai Tam, "A Semantic-based Approach to retrieving Imperative programs," Master Thesis, University of Windsor, 1997
- [KRUEG92] W. Krueger, "Software Reuse," ACM, June 1992
- [KULYU99] Kulyukin, V. "Application-embedded retrieval from distributed automatic collections," In Proceedings of the Sixteenth National Conference on Artificial Intelligence, 1999
- [LANGE97] Stefan Langer, Marianne Hickey, "Automatic Message Indexing and Full Text Retrieval for a Communication Aid," Proceedings of the ACL/EACL workshop on NLP for Communication Aids, Madrid, 1997

- [LIHUI00] Hui-Feng Li, geunbae, etc., "Lexical Transfer ambiguity resolution Using automatically Extracted Concept Co-occurrence Information," Computer Processing of Oriental Languages, Vol. 3, pp.53-68, 2000
- [LIAO93] Liao, Hsian-chou, Wang Feng-jian, "Software Reuse Based on a Large Object-Oriented Library," Software Engineering Notes, pp74-80 vol18, no1, Jan 1993
- [LUQI99] Luqi, Jiang Guo, "Toward Automated Retrieval for a Software Component Repository," IEEE, 0-7695-0028-5/99, 1999
- [LINDI00] Christian Lindig, "Fast Concept Analysis",
<http://www.gaertner.del-lindig/papers/indexing.html>, January, 2000
- [LEELI97] Lillian Lee, "Similarity-Based Approaches to Natural Language Processing," PhD thesis, Harvard University, 1997
- [MAULD91] M.L. Mauldin, "Conceptual Information retrieval – A Case Study in Adaptive Partial Parsing," Kluwer, Boston, 1991
- [MERKL98] Dieter Merkl, "Self-Organizing Maps and Software reuse," Computational Intelligence in Software Engineering, World Scientific, Singapore, 1998
- [MARTI85] J Martin, "Fourth Generation Languages – Volume I: Principles," Prentice-Hall 1985
- [MAHAJ99] M. Mahajan, D. Bofferman, X.D. Huang, "Improved Topic-Dependent Language Modeling Using Information Retrieval Techniques," in Proc. ICASSP'99, Vol. 1, pp. 541-544, March 15-19, 1999.

- [NIS94] National Institute of Standards and Technology, Glossary of Software Reuse Terms, NIST Special Publication 500-222, Computer Systems Laboratory, Gaithersburg, MD, December 1994
- [ORACL99] "Getting to Know Oracle 8i, " No. A08020-01, Feb. 1999, <http://www.oracle.com>
- [PBAI95] Ping Bai, "Execution Based retrieval of Reusable Software Components," Master Thesis, University of Windsor, 1995
- [PODGU93] Pddgurski, A. and L. Pierce, "Retrieving Reusable Software by Sampling Behavior," ACM Transactions on Software Engineering and Methodology 2, 3, 286-303, 1993
- [PRIET91] Prieto-Diaz, R., "Implementing Faceted Classification for Software Reuse," Communications of the ACM 34, 5, 88-97, 1991
- [PENIX99] John Penix, "Efficient Specification-Based Component Retrieval," Automated Software Engineering, vol 6, PP. 139-170, Kluwer Academic Publishers, 1999
- [QIANG99] June Xuejun Qiang, "Organizing Imperative Programs For Execution-Based Retrieval For Reuse", Master Thesis, University of Windsor, 1999
- [QINXL99] Qin Xiaolin*; Lin Junhai, "An approach to C software reuse based on database techniques," Transactions of Nanjing University of Aeronautics & Astronautics, vol.16, no.2 p.177-81, 1999
- [QIUAN98] Qiuyan An, "Retrieving Function Components From a Reuse Library," Master Thesis, University of Windsor, 1998

- [SALEE99] Sangdon Lee, Hansuk Choi, Youngjong Yang, "Storage and management of object-oriented frameworks," IEEE Piscataway, NJ, Vol 6, Page 1-20, 1999
- [SCOTT96] Scott Henninger, "Supporting the Construction and Evolution of Component Repositories," Proceedings of the International Conference on Software Engineering, Berlin, FRG, 1996
- [UML97] "UML Summary," Rational Software Corporation, 1997
<http://www.rational.com>
- [WINGJ95] Zaramski, A. and Wing, J., "Specification Matching of Software Components," Proceeding of the ACM SIGSOFT Symposium on Foundation of Software Engineering, 1995
- [ZAREM96] Amy Moormann Zaremski, "Signature and Specification Matching," CS-CMU-96-103 School of Computer Science Carnegie Mellon University, January 1996

VITA AUCTORIS

Sheng Zhong was born in 1968 in Beijing, China. He obtained a B.Sc. in Computer Science from the **Northeastern University**, Shenyang, PRC in 1993. He is currently a candidate for a Master's degree in Computer Science at **University of Windsor** and hopes to graduate in the summer of 2000.