Component Library Interface for CODE 2.0 (XCodelib)

by

Seema Yeshwant Khedekar, B. E. (Computer Engg)

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Arts

The University of Texas at Austin

May 1995

Component Library Interface for CODE 2.0 (XCodelib)

Approved by
Supervising Committee:

I dedicate this to Aai, Baba and Aaji

Acknowledgments

I extend thanks to Dr. James Browne who gave me a research assistantship and helped me in realizing my goal of writing a Master's thesis. Furthermore, I thank Dr. Browne for the countless hours of guidance throughout the research and writing of this thesis. Also, I appreciate the comments and suggestions from Dr. John Werth during the preparation of this document. I also wish to thank Micheal Kleyn for having cleared up some of my doubts during implementation. I would like to take this oppurtunity to thank our Graduate Secretary, Gloria Ramirez for her invaluable help over the past two years. Lastly I am grateful to all my friends, with special mention to Milind Sohoni, whose constant encouragement and support motivated me throughout my research work.

Seema Yeshwant Khedekar

The University of Texas at Austin

May 1995

Component Library Interface for CODE 2.0 (XCodelib)

Seema Yeshwant Khedekar, M.A.
The University of Texas at Austin, 1995

Supervisor: Dr. James C. Browne

Compositional programming (assembly of a program from components which have already been coded and tested) has three enabling technologies: libraries of components, a means for specification of programs in terms of components and a means of identifying and accessing components. Component libraries ranging from graphical interfaces to computational fluid dynamics are becoming available both from commercial sources and as resources on the Internet.

The CODE 2.0 programming environment implements composition of parallel programs from components which are sequential units of computation. The remaining requirement is support for identification and accessing of components which are appropriate for a given application.

This thesis reports on the design, development and initial use of XCodelib which is a component library interface added to the CODE 2.0 parallel programming environment. XCodelib addresses the requirement for identification and accessing of components which are accessible through the Internet. CODE 2.0 specifies parallel programs as generalized dataflow graphs where the nodes of the graph are units of computation. The Component Library Interface (XCodelib) supports identification

of components and the embedding of those components into CODE 2.0 programs. The location of the libraries and the interface to the libraries is uniform across all libraries whether they are local or remote. Identification and accessing of components is not a trivial task. Each of the libraries on the Internet has a different interface and a different mode of search. The overhead in time and effort to use standard general purpose Internet search mechanisms to locate and access components is excessive in the context of program development.

XCodelib is composed of three parts: the Client Interface, the Component Database and the Internet Interface. The Component Database houses the local library components and caches components from remote libraries. The Client Interface accepts the search queries formulated by the user and translates them into queries against the Component Database. The Internet Interface transparently searches for components across the Internet. The search expressions for library components in the XCodelib interface are as general as possible, and have few arbitrary restrictions. Feedback is provided during the search query processing to guide the user in the formulation of correct search queries. Only relevant information is displayed. There exist multiple modes for displaying the library components. The results returned by a search are displayed by the Client Interface. The *relevance feedback* mechanism is used for aiding in formulation of queries. XCodelib operates in two internal modes, a *Library* mode (which allows browsing of the database) and a *Search* mode where local and remote library components are found using keyword searches.

Contents

List of Figures

Chapter 1

Introduction

> ... one of the primary controllable factors we have for improving software productivity is the number of instructions we choose to develop, either by deferring the development of marginally useful features or by using available software.
>
> *Boehm*(1983)

Large systems in which software is a major component are amongst the most complex products developed by humans. In order to solve a large complex problem, it is often broken down into simpler, more manageable problems that can be solved independently. Each of the parts captures some abstraction of the overall problem. The system is then assembled from the separately developed parts. Compositional programming is an extension of this approach, where a program is assembled, as much as possible from components which have already been coded and tested. Complex systems are thus composed in abstractions from smaller systems which are known to be correct. To enable compositional programming, three questions need to be addressed: how to specify a program in terms of components, how to identify and access components and how to create libraries of components. It is often the case that identification and selection of a component may require an in-depth understanding of the component and/or perhaps modification.

The CODE (Computation Oriented Display Environment) 2.0, parallel programming system, addresses the first of these questions [1]. It permits parallel programs to be created by composing, basic units of computation, which are sequential in nature. The remaining requirements are, support for identification and accessing of components which are appropriate for a given application and, building libraries of components. The XCodelib system, which is the subject of this thesis, approaches some of the issues of management of components and component libraries. The major focus is on attainment of uniform methods of selection for the enormously diverse component libraries available through the Internet. In the next section, the CODE 2.0 system is described. System goals are outlined. XCodelib and it's system components are also decribed.

1.1 CODE 2.0

CODE 2.0 is a graphical retargetable parallel programming system [1]. It employs a compositional approach to programming. The programmer expresses parallelism directly and declaratively at a high level of abstraction by composing macro-level units of computation into parallel structures in the form of generalized dependence graphs [2]. The syntax of the "programming language" of CODE 2.0 is essentially a multigraph, with the semantics of the nodes and arcs mostly specified declaratively. Nodes represent computations (and sometimes function/module interfaces), and arcs represent dependencies among them. The CODE 2.0 computational model provides a natural framework for program component reuse. The task of parallel programming is divided into two independent subtasks: creating sequential units of computation using traditional techniques and languages, and composing them into a parallel structure in the form of a graph [1]. Independent specification of nodes promotes reuse of components. The program graph created is used as a template for creating dynamic structures at run-time. These dynamic structures are translated into executable code for the target architecture. CODE 2.0 generates programs for various target architectures from this graphical representation, and so is architecture independent. It uses a graphical/textual user interface for program specification.

1.2 Goals of XCodelib

When the idea for a component library interface system was originally envisioned, two important goals came to mind. The first was to support reuse in the CODE 2.0 domain and, the second was to have a uniform interface which would enable use of components from remote libraries all across the Internet. To support reuse in CODE 2.0, a system local database of reusable program components needs to be maintained. Since the notion of an electronic global community is rapidly maturing, it was deemed essential that the library interface system should not only have capabilities for selecting components from system local libraries, but there must also be capabilities to select from remote libraries. Moreover, there must be a uniform interface for selecting from both system local libraries and remote libraries. The XCodelib system fulfills both these goals. The selections from both, local and remote libraries are done uniformly and transparently. Only necessary and relevant information is presented, so that context-sensitive searches can be efficiently done. The system is designed so that the selected components can easily be embedded in CODE 2.0 programs. The XCodelib system is an exercise in information management.

1.3 XCodelib

XCodelib is a component library interface, which meets the goals outlined in the preceeding section. It addresses the requirement for identification and accessing of components which are available through the Internet. It permits a uniform interface to the diversely formatted libraries accessible through the Internet. It also supports embedding of those components into CODE 2.0 programs. XCodelib guides the programmer through the process of finding software components both hierarchically and by searching. In order to help the programmer find and reuse software components effectively, XCodelib supports the following five steps towards component reusability: finding components, understanding components, downloading components, modifying components, and incorporating components into the program. All of these steps are carried out through a graphical interface. XCodelib caches components identified as interesting in a local database. Since the organization of a large collection of components is critical for solving search problems, a relational database is used for local storage of the components. Components are found easily by translating search queries into queries against the database. Simple, but powerful queries and a graphical interface combine to make XCodelib a user-friendly system.

There are several benefits in reusing existing software. Quality is improved, since a reusable component has already been tested and is therefore reliable. Programmer productivity improves. The programmer also learns new coding techniques from existing components. XCodelib is a good way to attain these benefits.

XCodelib is composed of three parts: the Client Interface, the Component Database and the Internet Interface. Figure 1.1 shows a block diagram of the

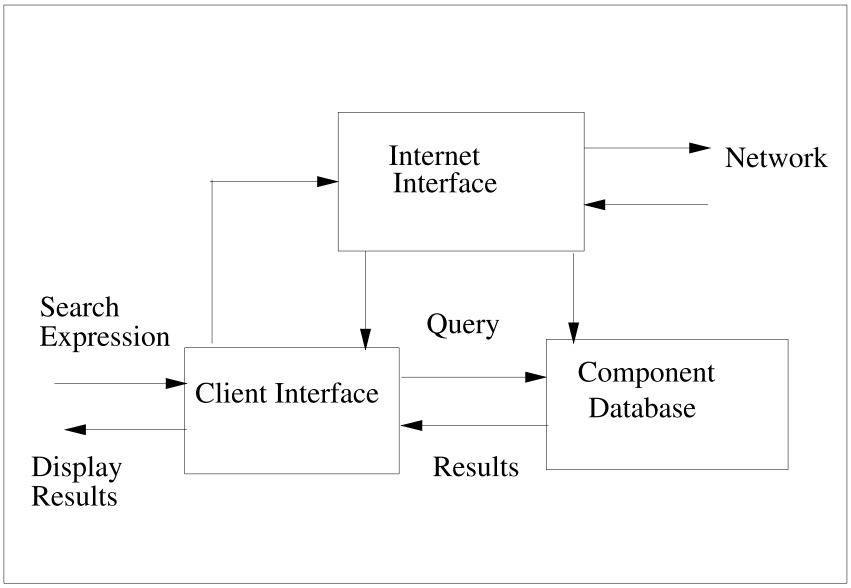

Figure 1.1: Block diagram of XCodelib

XCodelib system.

1.3.1 Client Interface

The Client Interface is the user interface for XCodelib. The Client Interface guides the programmer in finding reusable components, both by hierarchical and query-based search techniques. It accepts the search queries formulated by the user and translates them into queries against the Component Database. The search expressions which can be expressed in XCodelib via the Client Interface are quite general and have few arbitrary restrictions. The results returned by a search are displayed by the Client Interface. Only relevant information is displayed. There exist multiple modes for displaying library components. The Client Interface is uniform across all libraries that it can reach.

1.3.2 Component Database

The Component Database houses the local library components and caches components from remote libraries. It does the indexing and retrieval of components. It receives queries from the Client Interface and searches for components satisfying that query. It then returns the results to the Client Interface. It is the blackbox which lies between the Client Interface and the network. The Component Database is a standard commercial relational database management system (ORACLE).

1.3.3 Internet Interface

The Internet Interface transparently searches across the Internet, for specified components when they are not locally available. Selected components are transparently downloaded from the remote sites and cached in the Component Database. The Internet Interface forms the backend of the XCodelib system. The Internet Interface is actually a family of interfaces. Additional members can be added to the family to increase the effectiveness of XCodelib.

Chapter 2

Problem Definition

This chapter presents a detailed problem definition and the solution adopted in the XCodelib system. It gives an informal definition of the CODE model of programming and presents an overview of how reusability is enabled by the CODE 2.0 model. It will also detail the approach to reusability in CODE 2.0 implemented by the XCodelib system.

2.1 Overview

As discussed in the previous chapter the two main goals of the XCodelib system are to support reuse in the CODE 2.0 domain and secondly to present a uniform interface to accessing local and remote libraries, which required rendering Internet library access transparent.

2.1.1 Reuse in CODE 2.0

CODE 2.0 is a graphical retargetable parallel programming language, designed to support reuse of program components. Parallel programs in CODE 2.0 are cre-

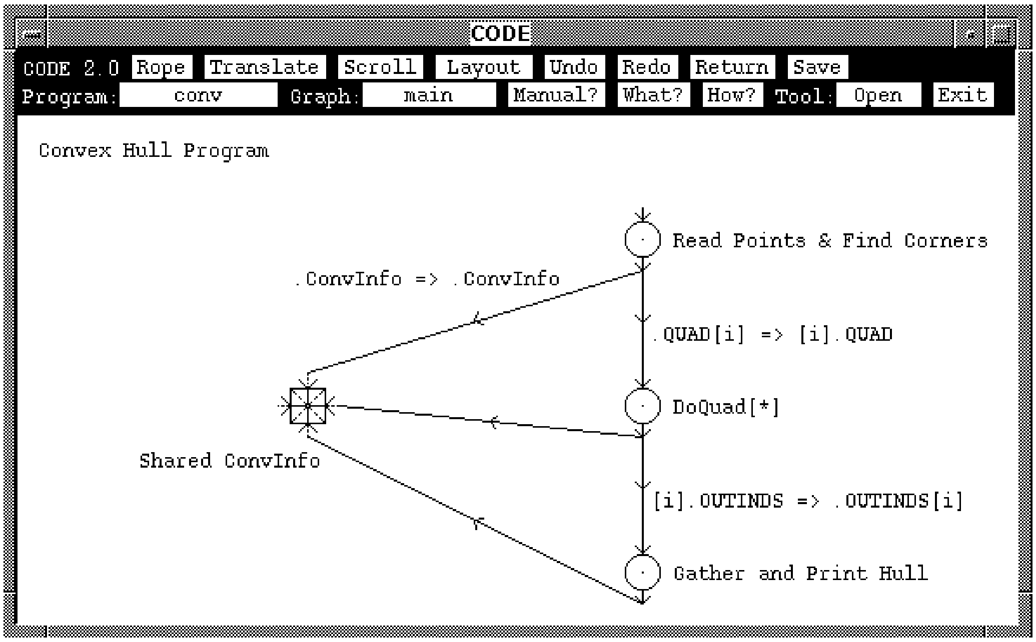

Figure 2.1: An Example CODE 2.0 Graph

ated by composing basic units of computation, which are sequential in nature, and defining realtionships amongst them. CODE 2.0 is based upon the *task* model of parallel computation [1]. A task model is characterized by the use of the task as the natural unit of parallel program composition. A task is a sequential computation that executes atomically and hence interacts with other tasks only at the begining and end of it's execution. Sets of tasks are composed into a parallel structure. For specifying parallel programs, the primitive components must be abstractions that are specifically tailored to the problem domain. CODE 2.0 specifies tasks as nodes in a graph. The sequential computation in a node is represented using a standard sequential language such as C. There is no attempt to reinvent sequential programming. This also facilitates reuse by making existing sequential program components reusable in CODE 2.0 programs.

CODE 2.0 is easy to use because it rests upon an expressive declarative model of parallel computation and because it has a graphical interface. Its programming model integrates familiar techniques of sequential programming with parallel composition. Users create parallel programs by drawing a graph showing communication structure and annotating it. Figure 2.1 shows an example CODE graph. Nodes represent computations and arcs represent the flow of data from one computation to another. Users specify the sequential computation a node performs. Such a node is called a Unit of Computation node (UC node) in CODE 2.0.

The sequential computation specified in a Unit of Computation Node is simply a piece of code, usually a function or a procedure, written in a sequential language such as C. Figure 2.2 shows a sample specification of a sequential computation given in a UC Node. The language used to specify the computation resembles a subset of C, but is not C. It is in fact rather limited. It is expected in CODE 2.0, that all substantial sequential computations will be encapsulated in ordinary sequential functions (usually written in C or some other sequential language which can be interfaced with C) that are defined external to CODE 2.0. Notice the calls to *FindHull* and *PrintList*. These are C functions written seperately. These external functions written in C can be compiled and linked in with the CODE 2.0 program. CODE 2.0 has facilities to specify the function signatures, invoke the functions via function calls in UC nodes, compile and link them with the parallel program. If a programmer can identify a previously written component which performs a desired function then the CODE 2.0 model makes, embbedding that component into a CODE program very easy.

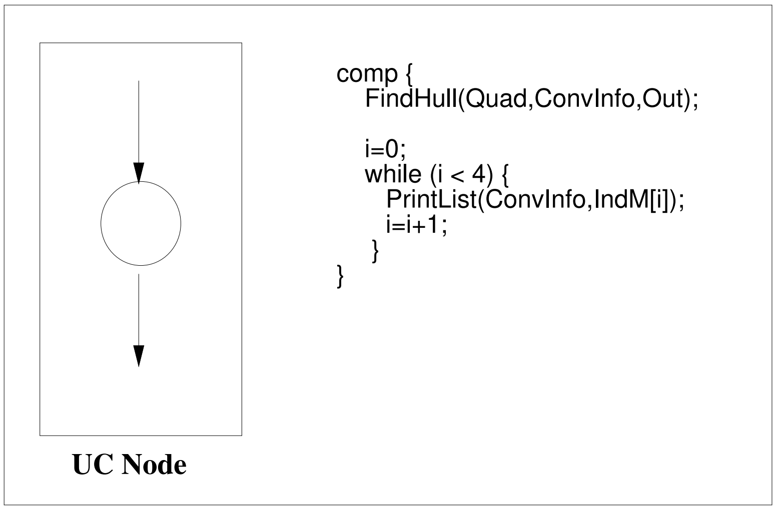

Figure 2.2: A sample sequential computation specified in a UC node

2.1.2 Uniform Access to Libraries

As the notion of an electronic global community matures, there is increased access to previously coded and tested software, which is available across the Internet. Any programming language/system of the future should exploit this abundant resource of available software. The only problem is that there are no standard conventions/interfaces imposed on free software. As a result, there are myraid conventions followed for definition and storage of components in software libraries. This makes the task of identification and accessing of software components over the Internet in any sort of uniform fashion non-trivial. The need is therefore to provide an interface which makes these differences transparent to the user.

2.2 Approach

The CODE 2.0 computation model supports program component reuse at all levels. In increasing order of granularity these are: the UC (node) level, the subgraph

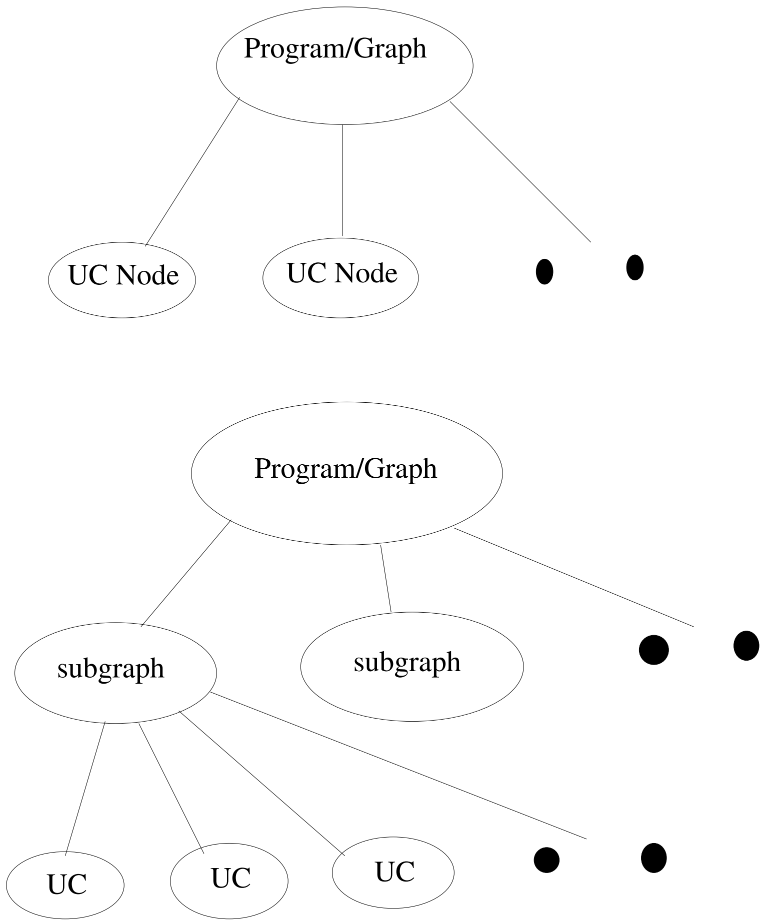

Figure 2.3: Hierarchical composition of components in CODE 2.0.

level and the graph level. The graph level is synonymous with the program level. Figure 2.3 shows the hierarchy in a CODE 2.0 program. Several nodes, which represent tasks are composed into a graph or a subgraph. Several subgraphs are composed into a graph/program.

CODE 2.0 supports macro-level parallelism, where the granularity of the parallel program is restricted to that at the UC node level. The ability of CODE 2.0 to call external routines from the UC nodes, is very useful when parallelizing old programs. Subroutines from these programs can be called directly. Since there already exists a large body of already coded sequential programs, the ability to incorporate these programs by making function calls, makes CODE 2.0 very viable in supporting reuse. Keeping the above issues in mind, the XCodelib system addresses the problem of program component reuse at the unit of computation i.e UC node level. The sequential computation specified in a node forms the reusable component of XCodelib. Moreover the nature of existing software components facilitates this approach to reusability. Most of the existing components are specified as sequential programs, which exactly matches the nature of the units of computation. Also one of the aims of XCodelib is to search for both general and domain specific libraries. Specifying reusable components at the subgraph and graph level would restrict components to only those created by CODE 2.0 programmers. To provide a more generic specification, it seems more feasible to specify reusable program components at the lowest level of granularity. This component nature defined in XCodelib exactly matches that of components available in remote libraries. XCodelib provides a uniform interface in accessing system local and remote components. It uses existing Internet browsers to access the maximum possible number of software components. XCodelib also makes the differences in definition and storage of software components transparent to the user. The approach is to create a system which separates the user-interface, local component management and network search functionally. The three are then interfaced together to work as a whole.

Chapter 3

Client Interface

This chapter provides an overview of the Client Interface of the XCodelib system along with the key issues involved in the design of the user-interface. The Client Interface forms the frontend of the XCodelib system. This graphical interface is a major factor in making XCodelib a user-friendly system. The Client Interface bridges the communication gap between the user and the system. It accepts search queries from the user and displays the results of searches. The Client Interface also allows the user to browse the component database and formulate incremental search queries. The Client Interface displays only necessary and sufficient information to perform domain specific searches for reusable components. The Client Interface integrates graphical and textual modes for viewing and understanding components. Graphs (trees) are used to represent the hierarchy in components and text is used to view and understand the functionality and source code of components. The next sections will present the nature of reusable components in XCodelib, modes of operation of XCodelib, details of the component display, and an outline of the download operation.

3.1 Nature of Components

The XCodelib system is a component library interface to CODE 2.0. As described earlier it targets reuse at the lowest level of granularity in the CODE 2.0 system. A reusable component in CODE 2.0 and XCodelib is simply the sequential computation specified in a CODE 2.0 UC node. These reusable components are mostly present as sequential language source programs across the Internet. Reusable components are typically available as files or directories of files. If a component is present as a file, it will contain sequential language function definitions. If the component is a directory, it will contain context related source files and sub-directories of source files. The files contain the source code for program functions which can be invoked through UC nodes in a CODE 2.0 program. CODE 2.0 has facilities to compile and link these source files with the CODE 2.0 parallel programs. XCodelib enables the CODE 2.0 programmer to identify reusable components to embed in UC nodes. Once the components have been located, XCodelib allows the programmer to view the component structure and content, so that the programmer can understand the component. If the programmer finds the component useful, XCodelib allows the programmer to download the component to a user specified directory. The programmer can then modify the component to suit his/her parallel program and CODE 2.0 allows the modified component to be compiled and linked in with the parallel program.

3.2 Modes of Operation

The XCodelib system allows the user to browse the component database and formulate search queries incrementally. Hence there are two main modes of operation, namely the browse mode and the search mode.

3.2.1 Browse Mode

In this mode, the programmer can browse the system local component database hierarchically. All the components currently present in the database can be viewed. At the topmost level, the database appears as a forest of components. The components are displayed either as files or directories. Different icons are used to make this distiction between components. Since the Client Interface supports both graphical and textual modes for viewing components, selecting a directory component to view, causes the hierarchical tree nature of the component to be displayed, whereas selecting a file component causes the source code contained in the file to be displayed. This mode allows a top-down traversal of the XCodelib component database. The hierarchical directory structure of components is traversed down to the leaf nodes which are files. This mode allows up and down traversals from any level in the component hierarchy.

3.2.2 Search Mode

Search mode enables finding of desired reusable components. Search expressions formulated by the user are translated by the Client Interface into queries against the component database. The components returned from running the database queries are displayed as results. The number of components found is also displayed. XCodelib allows quite general search expressions. There are few arbitrary restrictions in formulating search expressions. Formulating search expressions is very easy in XCodelib because of the simple grammar used. Search expressions are composed mainly of keywords and logical operators. This is a very easy and natural way of

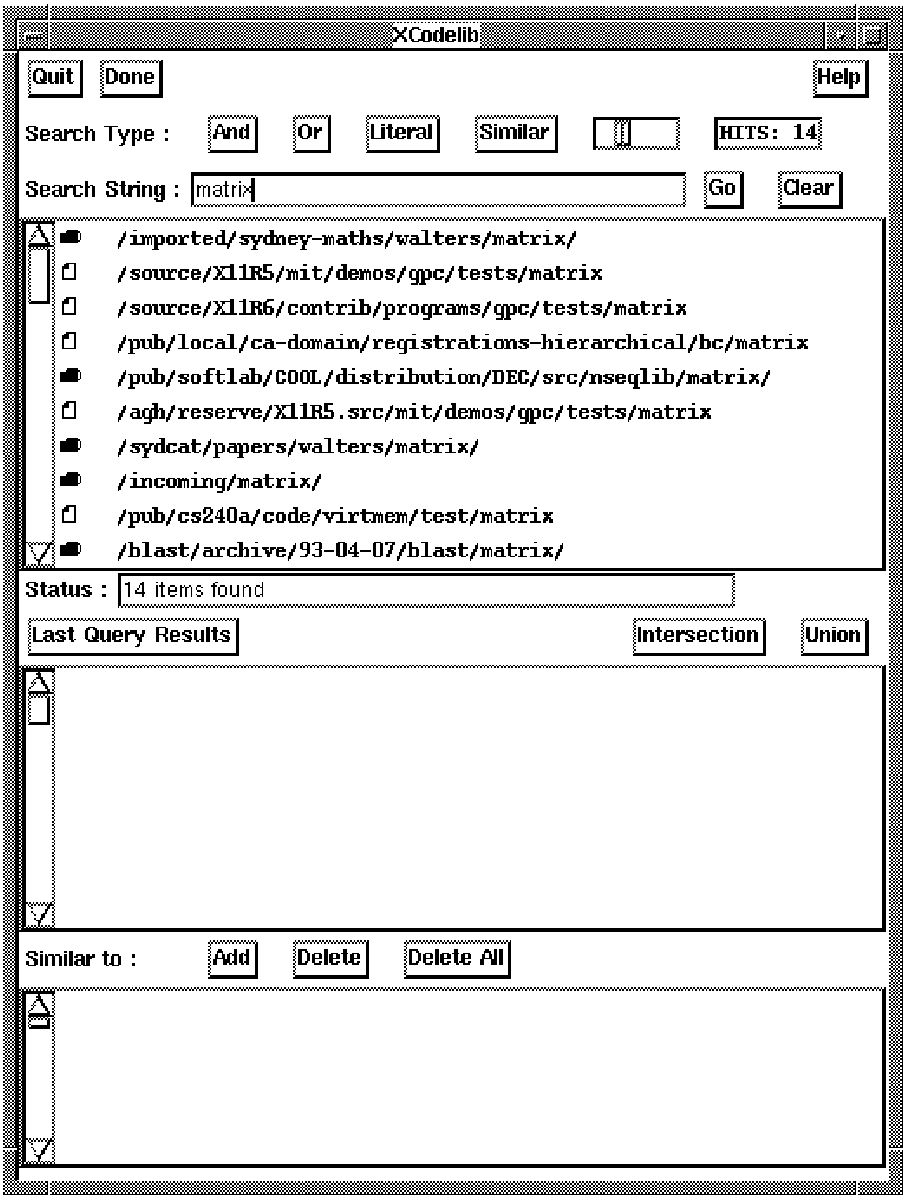

Figure 3.1: Different types of Search in XCodelib

composing search expressions. XCodelib also allows for incremental composition of search queries. This is done by buffering the results of the previous query and applying query level operators on the results of the last and current queries. The query level operators support the operations of union and intersection. These operations help the user in building context sensitive queries incrementally. The searches for reusable components are done uniformly and transparently in XCodelib i.e the user is unaware if the search is being done locally or across the Internet. Also, the user is unaware if the components displayed as the result of a search are local or remote. Figure 3.1 displays the search window of the XCodelib system and the results of a literal search. The components displayed are the results of a remote search. Three different types of searches are possible in XCodelib:

- Regular Search. This is the most general type of search and is simply an AND/OR search. The keywords in the search expression are connected by the logical operator of conjunction in an AND search, and of disjunction in an OR search. This kind of search is only done on the local component database.

- Literal Search. This is a very specific search. The search query is composed of a single search keyword which itself is the search expression. This search is done on both the local component database and across the Internet.

- Similar Search. This search is useful in retrieving components which are very much related to the target search string. Once a search is run, some of the resulting components can be marked and a similarity search can then be run. In this search, all components in the database which are similar to those marked are returned and displayed. The similarity arises from similar component names. This type of search is only done on the local database.

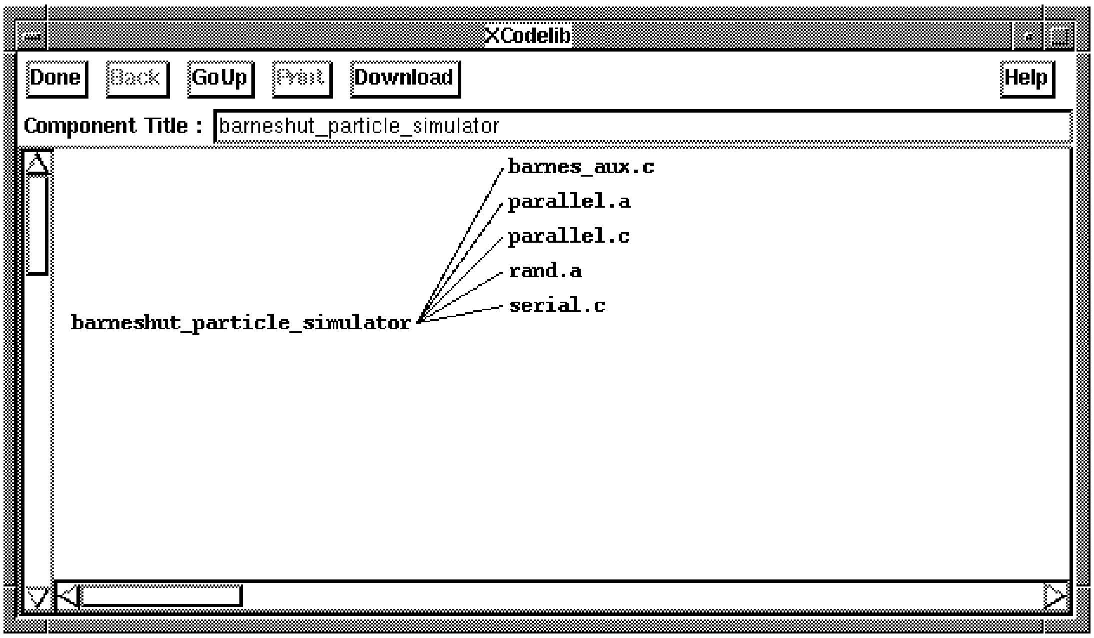

Figure 3.2: Component hierarchy display in XCodelib

Greater detail on the different types of searches is provided in the next chapter.

3.3 Component Display

Once a reusable component has been located through browse or search mode, the component can be displayed to understand the structure and content of the component. XCodelib integrates both graphical and textual modes of component display. Since the nature of components in XCodelib is either that of files or directories, selecting a component for viewing causes either the graphical or textual view to come up depending on whether the component is a file or a directory. If the component is a directory then the structure of the component i.e the hierarchy of the component is displayed in the form of a tree. The programmer can then select for viewing any file or sub-directory in the component hierarchy. The programmer can also traverse

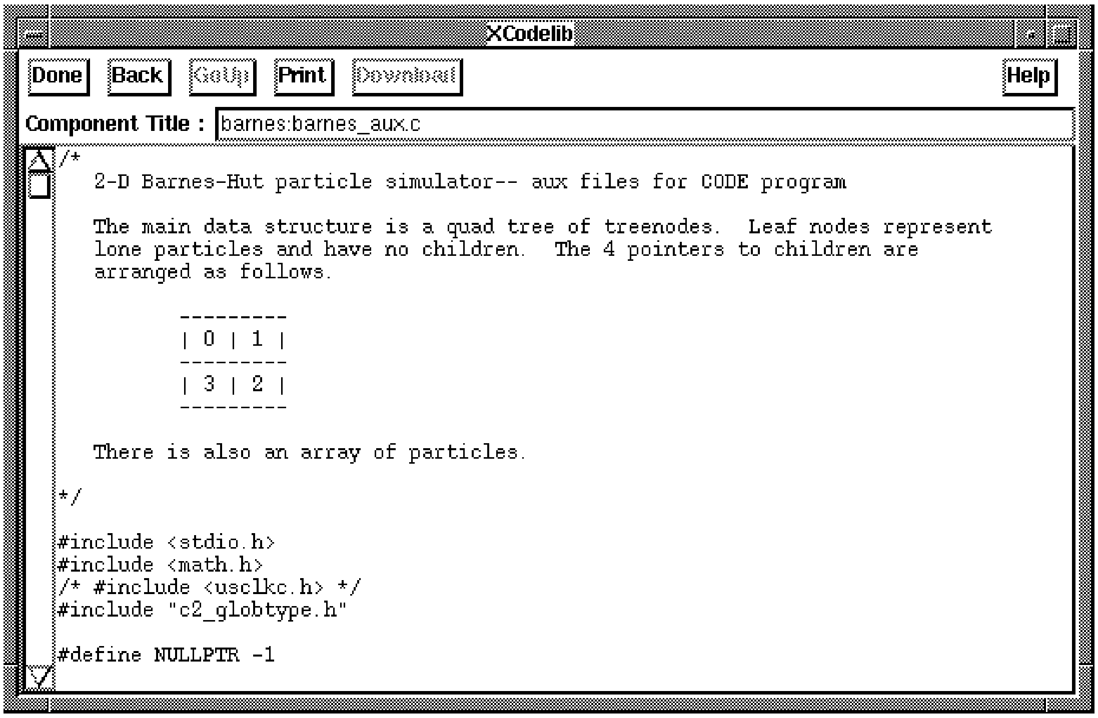

Figure 3.3: File Component Display in XCodelib

up and down the directory component tree by moving to sub-directories and/or parent directories. Figure 3.2 shows a directory component display. The component displayed is that for the Barneshut Particle Simulator Algorithm. The files contain the serial and parallel code for the algorithm.

If a file is selected for viewing from the directory component display or if the component is itself a file component then the textual view of the source code contained in the file is displayed. Figure 3.3 shows a file component display. The file displayed is an auxiliary file for the CODE 2.0 program of the Barneshut Particle Simulator. There also exists a facility to print the component source file. This textual view of the program files helps in understanding the code contained in the components, which helps in analyzing the components, listing them etc. The component display helps the programmer in understanding the structure and content of the component. Viewing the source code contained in the component helps to understand the algorithm used in coding the component. This goes a long way in helping the programmer to understand the component and deciding upon the utility value of the component. Since components on the Internet fall under the domain of free software, there are no compulsions on programmers to follow any guidelines, provide documentation etc. The XCodelib approach of viewing component directories and files helps the programmer in viewing, not only program source files but also in viewing any documentation present (for eg: README files), if at all. This approach to locating and understanding components takes into account the existing nature of free software available on the Internet, and tries to maximize access to all such software.

3.4 Downloading of Components

XCodelib provides facilities to download useful components to programmer specified directories. XCodelib has a uniform interface for downloading of files be they local component database files or files from across the Internet. A component can be selected for downloading by the programmer. Whenever a file is selcted for viewing, be it a file component or a file from a component directory, it gets automatically selected for downloading. If the file does not prove to be of interest it must be explicitly de-selected by the programmer. Entire component directories or individual files of interest can be selected for downloading. Once all the files/directories of interest have been selected they can be downloaded to a programmer specified directory. The Client Interface constantly keeps the user informed of the progress of the download operation. This feature is especially useful, when downloading files across the Internet.

Chapter 4

Component Database

The Component Database is that block of the XCodelib system which lies inbetween the Client Interface and the Internet. It is a library of software components. It stores local components and also caches components from remote libraries. This chapter presents a detailed design and an in-depth functionality of the Component Database. The next section presents the organization of the Component Database. An overview of the different operations possible on the Component Database is also described. Various types of searches or queries are detailed. Different ways of populating the database are also outlined.

4.1 Organization of the Database

The Component Database physically consists of two databases. An Oracle database and a software database. Each component in the database is identified by two attributes, a Component Name and a Component Path. The Component Name is used to identify the component and the Component Path is used to locate the component in the software database. This two-attribute tuple used to uniquely

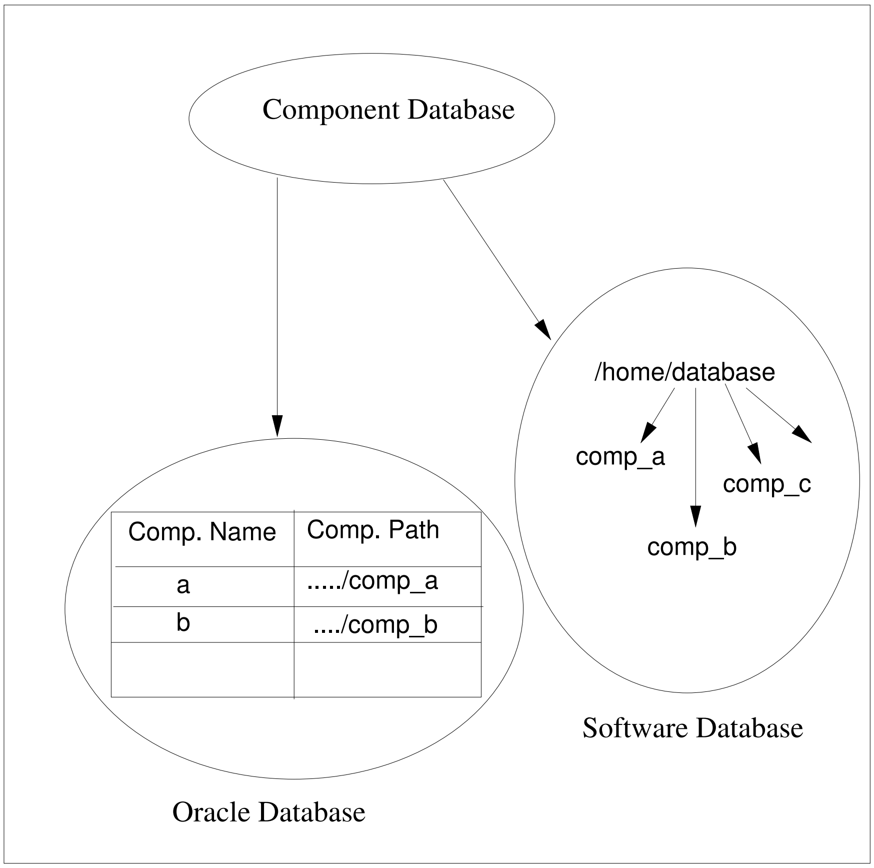

Figure 4.1: Organization of the Component Database

identify each component is housed in the Oracle database. The component itself resides in the software database. The software database is essentially a forest of components all located under a *database* directory in the UNIX file system. The Component Name is used to retrieve the corresponding Component Path from the Oracle database. The Component Path is then used to locate and retrieve the component from the software database. Figure 4.1 shows the organization of the Component Database.

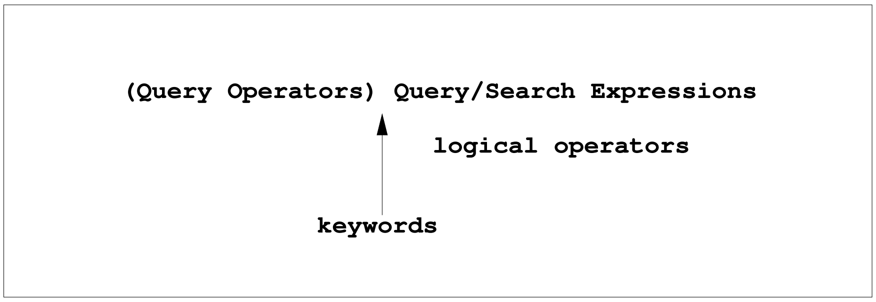

Figure 4.2: Organization of Search Expressions in XCodelib

4.2 Operations on the Database

Two main operations can be performed on the Component Database. These are, entry operations which help in populating the database, so that it functions like a cache and query operations which help in retrieving library components from the database. Both these operations go towards meeting the requirements of the XCodelib system.

4.3 Types of searches

The different types of searches are implemented as query operations on the database. The various types of searches in XCodelib help the user in formulating quite general search expressions with few arbitrary restrictions. There are three different types of searches in XCodelib. The first is the Regular Search which is simply an AND/OR search. Next is the Similarity Search and lastly the Literal Search. Figure 4.2 shows the organization of search expressions.

4.3.1 Regular Search

A Regular Search query in XCodelib is formulated in terms of keywords in a compositional expression. A keyword is the basic unit in a search expression. The logical operators of conjunction (AND) and disjunction (OR) are applied to the keywords. A combination of keywords and logical operators forms a search expression. The query-operators, UNION and INTERSECTION can be applied to search queries. By default, the ALL query-operator applies to all search queries.

```
letter = {A,B,C,.......,Z,a,b,......,z}

digit = {0,1,2,....,9}

specialchars = {_,/,.....}

keyword = letter(letter|digit|specialchars)*

LOP -> AND

      |OR

QOP -> UNION

      |INTERSECTION

      |ALL

expr -> expr LOP expr

       |keyword

query -> QOP expr

        |expr
```

For an AND search to be successful, all the keywords in the search expression must be present in the component name (one of the attributes of the component). Similarly, for an OR search to be successful, atleast one of the keywords in the search expression must be present in the component name. XCodelib also facilitates incremental building of search queries. This is done by buffering the results of the last query executed, and doing a UNION or INTERSECTION operation on the results of the last and current queries. This incremental building of search queries helps the user in narrowing (INTERSECTION) or broadening (UNION) the context of the search. The Regular Search is only carried out against the local component database and is not a remote search. An OR search done remotely leads to an explosion of results, and cannot be restricted to context.

4.3.2 Similar Search

A Similarity Search in XCodelib is run by marking some of the results of the current query, and finding other components similar to those marked. The criterion for deeming a component similar to a marked component, is the similarity in the component names. The component name of a marked component is taken, and all permutations and combinations of the keywords in the marked component name are used to find other similar components. Different AND and OR searches are run on the permutations and combinations of the keywords of the marked component, and the results are combined and displayed. Instead of making the user formulate complex search expressions, Similarity searches in XCodelib are used to perform complex search queries. For example, search expressions like the following can be undertaken by similarity searches.

```
search keywords : (key1 key2 key3)
search expression : (key1 and key2) or (key2 and key3) or (key1 and key3)
```

Figure 4.3 shows an example of a "Similar Search". In the figure all components similar to 'barnes' (the marked component) have been located. Similarity searches are also done only against the local component database and are not done remotely.

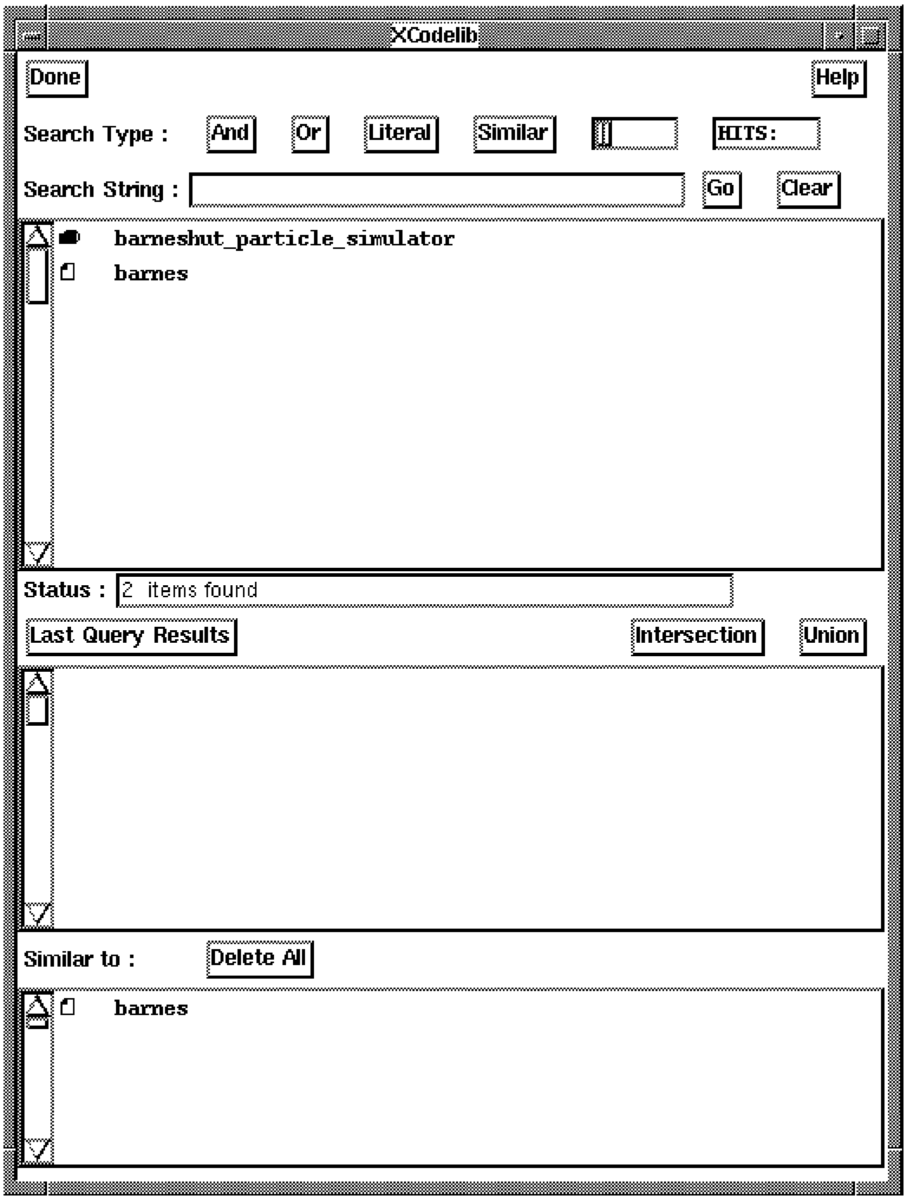

Figure 4.3: Similarity Search in XCodelib

4.3.3 Literal Search

A Literal Search query in XCodelib is formulated in terms of a single keyword, which itself is the search expression. This search returns matches on any component whose name matches that of the search expression exactly. This type of search is well suited for remote searches. Since there are huge numbers of components on the Internet, a literal search is the appropriate first step in locating exactly the components needed in the least amount of time. Figure 3.1 shows an example of a literal remote search. The search expression is 'matrix'. Remote file or directory components having the name 'matrix' are displayed as results.

```
LOP -> LITERAL

QOP -> ALL

expr -> keyword LOP

query -> QOP expr

        |expr
```

4.4 Populating the Component Database

The Component Database of the XCodelib system functions as a cache, it grows with increasing amounts of system usage. Components are added to the database via two modes. Firstly, remote components which are selected for downloading by the user are not only downloaded to the user's working directory but are also added to the Component Database. This is done automatically by the system and is transparent to the user. This transparent adding of remote components makes the Component Database behave like a "cache of software components". The idea being that if a remote component was useful to one user then the probability of it being useful to

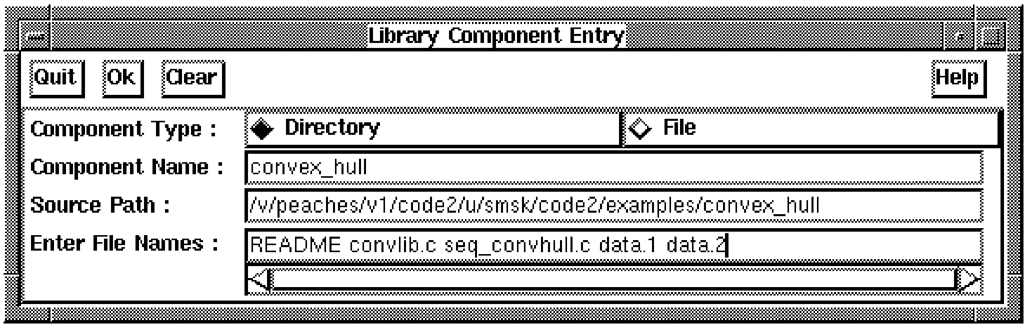

Figure 4.4: Adding Components to the Database

another is high, and so it must be added to the database. The other mode of adding components to the database, is when the CODE 2.0 programmer explicitly adds components coded by him/her to the database/library of components, promoting software reuse. Figure 4.4 shows how components can be explicitly added to the database by a CODE 2.0 programmer. The programmer must indicate whether the component to be added is a file or directory component. A meaningful name must also be given to the component. The names of files which belong to that component and the directory where the files are located must also be specified. XCodelib then adds the component along with its accompanying files to the database.

Chapter 5

Internet Interface

The Internet Interface forms the backend of the XCodelib system. It is a family of interfaces which access components over the Internet. The Internet Interface supports the following operations : searching for remote components, display of remote components, and downloading of remote components. The following sections detail the various operations supported by the Internet Interface.

5.1 Searching for Remote Components

The Literal Search in XCodelib is used to search for remote library components. XCodelib uses *archie* to search for remote components. The literal search expression is passed to *archie* as the search string. *archie* is an Internet archive service [3]. It allows searching of indexes of the available files on public servers on the Internet. *archie* returns the results of the search and these are formatted and displayed by XCodelib. Figure 3.1 shows the results of a remote search. The results show the various components satisfying the search expression "matrix". These components are located at different public server sites, but this fact is not readily apparent from the results. This Internet search is limited in that it can search for components only at those Internet hosts which provide the service of 'Annonymous ftp', and not all hosts. Since a large number of public servers on the Internet provide the service of 'Annonymous ftp', the search domain becomes extremely large and Internet searches return a very large number of components. To restrain this search there is a facility provided by which a user can indicate the maximum number of components to be returned.

5.2 Displaying of Remote Components

The remote components obtained as results of a literal search are displayed. As the components are either files or directories of files, selecting a component for viewing causes either the graphical or textual view to come up, depending upon the type of the component. These graphical or textual views are exactly the same for both local and remote components. The user is unable to distinguish whether a component is local or remote, by merely looking at the views. If the component is a directory, then the structure of the component i.e the hierarchy of the component is displayed in the form of a tree. This is done on-line in real time. The interface scripts access the component at the remote site, determine its structure and then display it locally. Traversal up and down the directory component tree by moving to sub-directories and/or parent directories is also done in real-time. Figure 3.2 shows a directory component display. Similarly, if the component is a file then the textual view of the source code contained in the file is displayed. This is also done by accessing the component at the remote site and then displaying it's contents. Figure 3.3 shows a file component display. Viewing of files under a directory component is done in a similar fashion. XCodelib maintains a uniform display for both local and remote

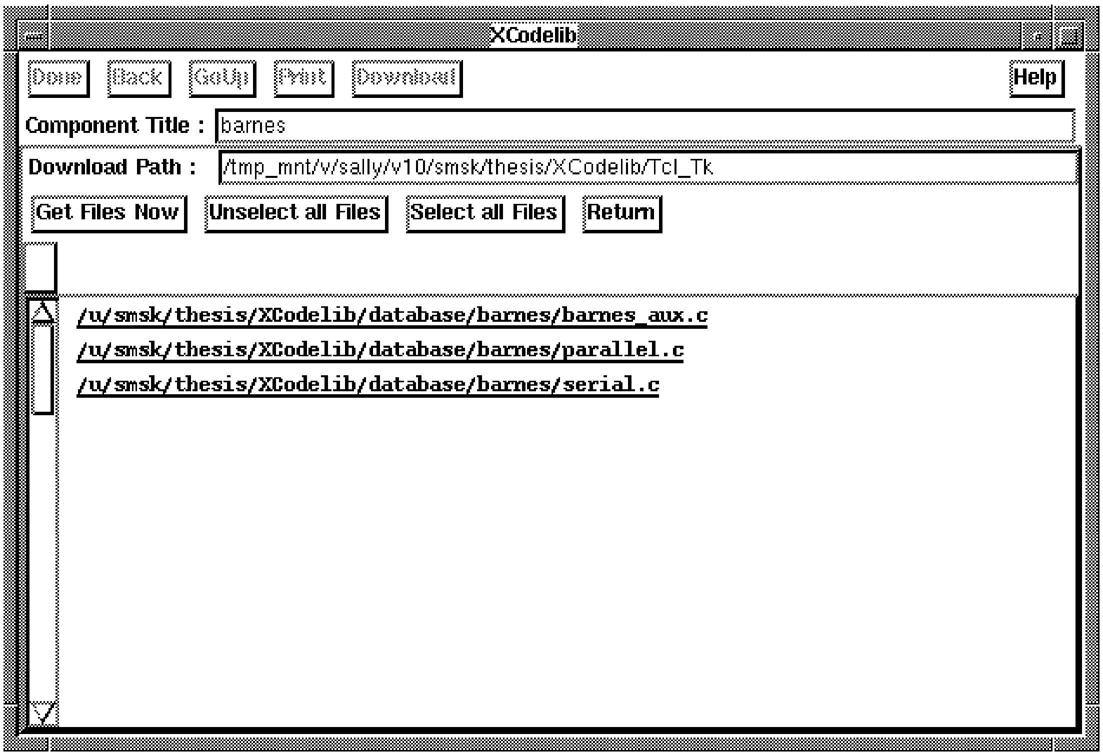

Figure 5.1: Download Operation in XCodelib

components, which satisfies one of the main goals of the system.

5.3 Downloading of Remote Components

XCodelib provides facilities to download useful components to programmer specified directories. Components can be selected for downloading by the programmer. Whenever a component is selected for viewing, it also gets automatically selected for downloading. The idea being that, if the component was of sufficient interest to be viewed, then it has a high probability of being used. Entire component directories or individual files of interest can be selected for downloading. Since downloading remote files/directories can be a time consuming operation, especially if the files are large in size, XCodelib provides facilities by which the user is constantly informed of the progress of the downloading operation. These remote components are not only downloaded to the programmer specified directories but they are also cached in the component database to ensure that the next time a user wants to access the same component, it does not have to be a time consuming remote access, rather it is a quick local access. Figure 5.1 shows the download operation for remote components.

Chapter 6

Implementation

This chapter presesnts an indepth description of the implementation details of the XCodelib system. Different blocks of XCodelib have been implemented using various tools. Tcl and the Tk toolkit, Oracle 7.0, oratcl, expect and ftp are some of the tools which have been used in the implementation of XCodelib. The various tools used are described. The flow of control and data through the XCodelib system is presented in the following section. Data flow through different types of searches is also described.

6.1 Control Flow through XCodelib

This section gives a detailed description of the control and data flow through the XCodelib system. Figure 1.1 depicts the building blocks of the system. It also shows a high level version of the data flow through the system. The Client Interface accepts search expressions from the user, these are converted into queries against the database for a local database search, or they are converted into network search expressions to do a remote Internet search. The Component Database runs the search queries and returns the results to the Client Interface. Similarly, the Internet Interface runs the remote search and returns the results to the Client Interface. The results, returned to the Client Interface by the Component Database and the Internet Interface, are formatted and displayed. User selection of components is again passed to the Component Database or Internet Interface depending upon whether the component is local or remote. These interfaces then act upon the selection to retrieve the contents of the component, which are again displayed via the Client Interface. Figure 6.1 depicts an indepth picture of the flow of control in XCodelib.

XCodelib has two modes of operation : browse or search mode. In the browse mode, all the components in the component database are displayed. Since components in XCodelib are either directories or files, selecting a component for display causes the directory component display or the file component display to come up. The directory component display shows all the files under that directory. Selecting a file for viewing causes a file display to come up. If found to be interesting, individual files or the entire directory component can be selected for downloading. Similarly a file component could also be selected for downloading. Once the components of interest have been selected for downloading, the download option can chosen to actually download the components. In the search mode, various types of searches can be run against the component database. And, Or, Similar, Union and Intersection searches are run against the local component database. Only Literal searches are executed across the Internet.

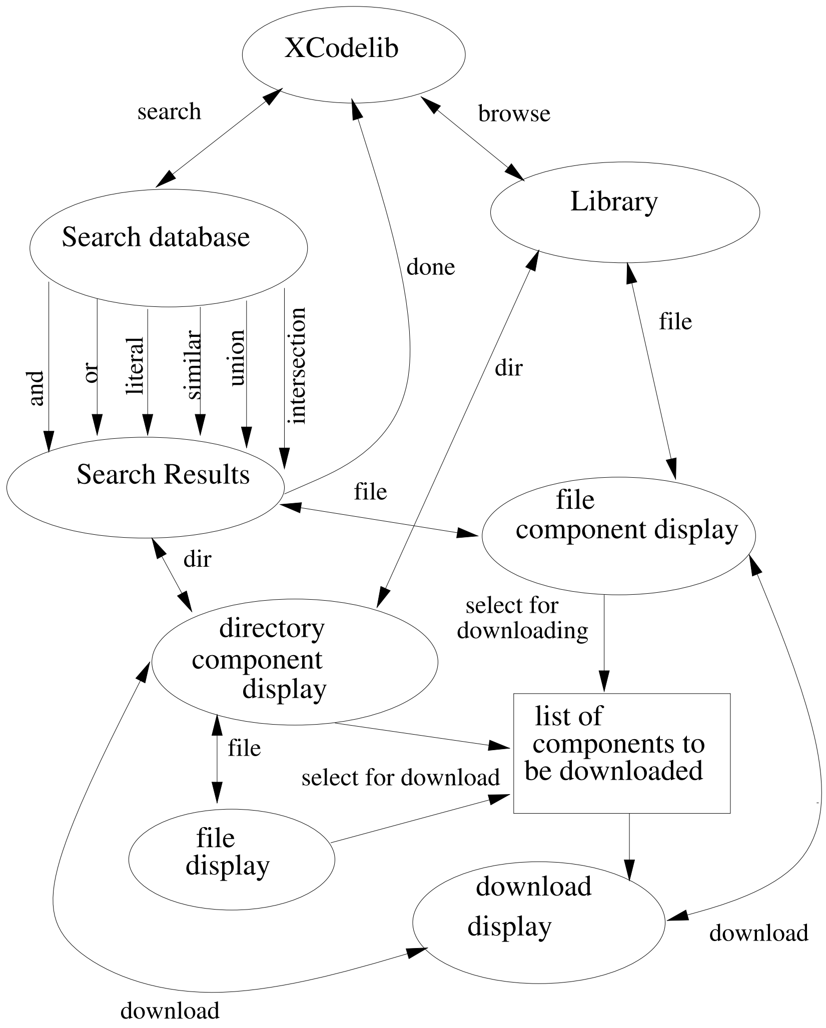

Figure 6.1: Flow of Control in XCodelib

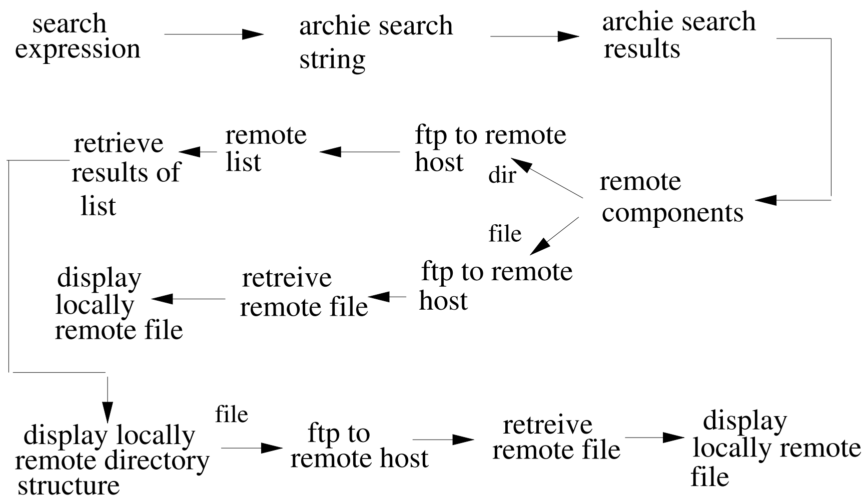

Figure 6.2: Flow of Data in a Remote Search

6.2 Data Flow through various Search Types

This section details the organization and flow of data through the different search types in XCodelib. The different types of search have been broadly grouped as remote search and database search.

6.2.1 Remote Search

A Literal search is a remote search in XCodelib. This search is not done on the local database rather it is done across the Internet. XCodelib uses the *archie* internet archive facility to conduct remote searches. The search expression is passed to the *archie* client. The *archie* client then returns the results of the remote search to XCodelib, which are then formatted and displayed. Figure 6.2 shows the flow of data during a remote search.

Once a remote component has been selected for viewing, the Internet Interface runs the file transfer protocol (ftp) to the remote host. If the remote component

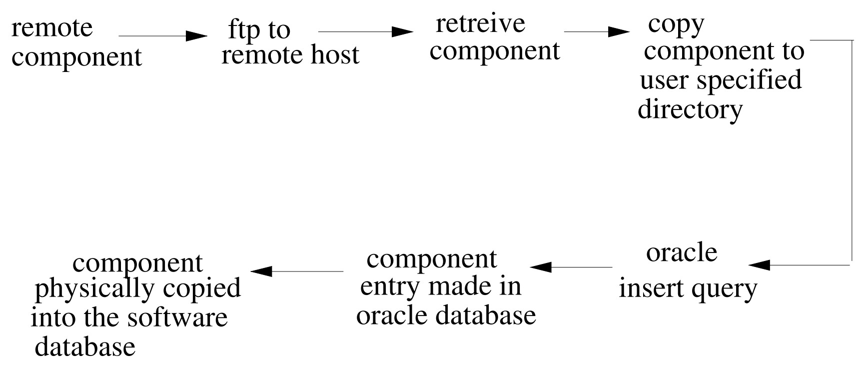

Figure 6.3: Flow of Data in a Remote Download

is a directory, then it's structure is retrieved and displayed locally, whereas if it is a file then it's contents are retrieved and displayed locally. If a file under a directory component is selected for viewing then the Internet Interface scripts retrieve the file contents from the remote host and these are displayed. Remote components which are selected for viewing are also automatically selected for downloading. During downloading, the selected components are not only copied to the user specified directory but are also copied into the Component Database. Figure 6.3 shows the flow of data during a remote download operation.

The remote component is copied into the Component Database by making an entry into the Oracle Database and physically copying the component in it's entirety to the software database.

6.2.2 Database search

And/Or searches are database searches in XCodelib. In a database search, the search expression is converted into an Oracle query and run against the Oracle Database. The results of the Oracle query are formatted and displayed. If a directory component is selected for viewing, then the Component Path is used to list the

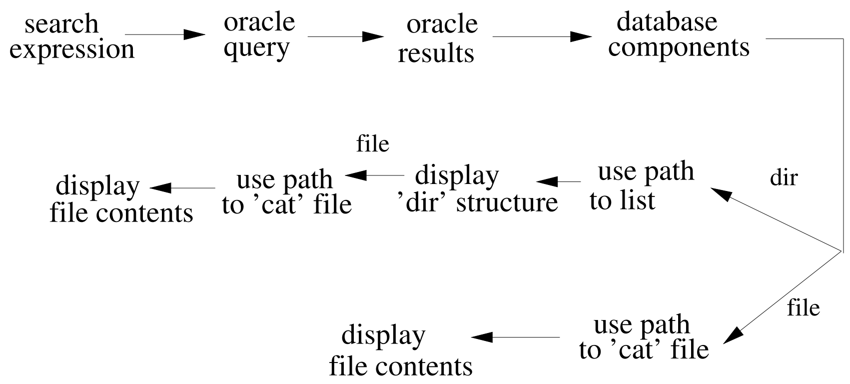

Figure 6.4: Flow of Data in an And/Or Search

component and display it's structure. If a file component is selected, then the Component Path is used to view it's contents. Figure 6.4 shows the flow of data in an And/Or search.

Similarity searches are also database searches. Similarity searches are just a combination of And/Or searches. The results of all the individual And/Or searches are combined and displayed as the results of a similar search. In a similar search, a component is marked. The object is to find other components in the database, similar to those marked. The example given below illustrates how the similar search works.

```
marked component name : word1_word2_word3

word1 -> key1, word2 -> key2, word3 -> key3

sub_search1 : key1 OR key2 OR key3

sub_search2 : key1 AND key2 AND key3

sub_search3 : key1 AND key2

sub_search4 : key1 AND key3

sub_search5 : key2 AND key3

similar search : sub_search1 U sub_search2
```

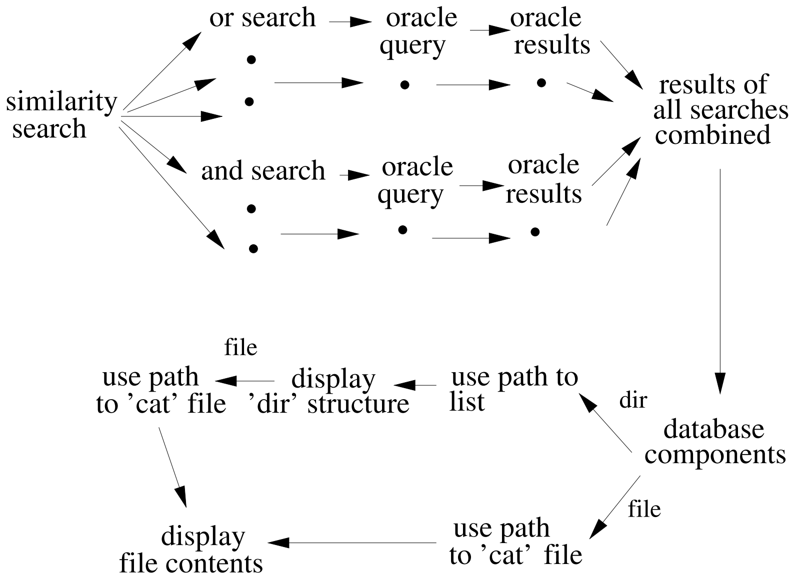

Figure 6.5: Flow of Data in a Similar Search

```
U sub_search3 U sub_search4 U sub_search5
```

Figure 6.5 shows the flow of data in a similar search. The results of all the sub-searches are combined together, formatted and displayed.

6.3 Tools Used

The XCodelib system has been implemented using various tools. These tools have enabled a quick and efficient implementation of the various concepts formulated during design. *Tcl* (tool command language) is the tool used to implement the XCodelib system. Tcl is a simple scripting language for controlling and extending applications [4]. Tcl is an embeddable command language. It is an interpreted language and it's interpreter is a library of C procedures that can be easily incorporated into applications, and each application can extend the core Tcl features with additional commands for that application. *Tk*, one of the most useful extensions to Tcl [5] is a toolkit for the X Window System. Together, Tcl and Tk provide a programming system for developing and using graphical user interface (GUI) applications. Tk is the tool used to implement the Client Interface of XCodelib, which is a GUI. *Oracle 7.0*, relational database management system [6] has been used to implement part of the Component Database. A commercial database system has been used as the Component Database to ensure scalability. Since, only basic database functionality is required, it could have been programmed into the application, instead of using a commercial RDBMS. But as the XCodelib system is used more and more often, the component database will grow over time and scalability would become a real issue leading to a degrade in performance. To improve performance, indexing operations would also have to be programmed, so rather than duplicating all DBMS functionality, it is more feasible to use an actual RDBMS. *Oratcl*, a Tcl interface to the Oracle Database server, is another useful Tcl extension [7]. Oratcl is an extension to Tcl that provides access to a Oracle Database server. Oratcl adds additional Tcl commands that login to an Oracle Server, pass SQL (Structured Query Language) code, read results etc. Oratcl is used in XCodelib to communicate between Tcl and SqlPlus (the query language of the Oracle Database server). *archie* [3] clients are used to do remote searches across the Internet. archie is an Internet archive service. It allows searching of indexes of available files on public servers on the Internet. *expect* scripts are used for controlling the interactive ftp (file tranfer protocol) [8]. *expect* is a tool for controlling interactive programs. *expect* uses Tcl as a language core. *FTP*(file transfer protocol) is used to access remote components and download them. Annonymous FTP is the principal way in which software is distributed on the Internet [3]. Annonymous ftp allows downloading of any file from every Internet host, which has been set up to offer this service. All the tools used to implement the XCodelib system were very easy to learn and use. They made implementation of a complex system quick and easy. Using tools, instead of conventional programming languages, to implement the system reduced development time by a large factor. Building XCodelib from assembling component tools was an excercise in reuse. XCodelib was developed for supporting reuse in CODE 2.0, but developing the system itself was also an exercise in reuse. All of the tools used were readily available. Tcl was an excellent choice to implement the system. It is an extensible language and a large body of supporting extensions have already been coded and are readily available.

Chapter 7

Related Work

This chapter presents some related work in the field of component reuse and library management tools. In certain respects, there is a lot of work related to XCodelib, in others there is practically none. All topics related to XCodelib, fall under four different cateegories : component reuse tools, libraries, library browsers and network browsers. The following sections present further details on the various categories.

7.1 Component Reuse Systems

ROPE 1.2 [9], (software Reuse in Parallel programming Environments), was an earlier attempt to support reuse in the CODE 2.0 domain. ROPE addressed the fundamental problems of finding, understanding, modifying and composing reusable components. But unlike XCodelib, ROPE addressed the problem of reusability at a much higher level of granularity (graph level) in CODE 2.0. Component identification was done by a structured relational classification method. Design analysis methods were used to understand components. Components were modified by employing techniques for composing and decomposing graphs. The resulting system was complex and difficult to use. Also, ROPE did not address the problem of finding reusable components across the network. XCodelib is a second generation attempt at solving the same problem. XCodelib is much simpler and easy to use. It also supports remote searches.

7.2 Component Libraries

There exists a large body of work done in the field of component libraries. There are various reusable class libraries developed for C++. *libg++* is one such C++ class library [10]. Keith Gorlen's NIHCL library [11] is also a C++ class library. But as opposed to XCodelib, all these libraries are specifically C++ class libraries and target the data structures domain. XCodelib is also a library/database of components, but it also supports browsing of components with a view to understanding them, modifying them, reusing them etc.

7.3 Browsers

Centerline's Object Center [12] set of development tools, used for producing C++ applications, use browsers for supporting C++ programming. A *Man* Browser contains the documentation and eases the learning curve of users by providing quick access to explanations of operations of the ObjectCenter environment. A *Project* Browser provides developers with a method of viewing the hierarchy of the component files of the project, allowing varying depth of component relationships. The *Inheritance* Browser portrays the relationships between the objects that are a part of the system and indicates the level of the class structure. The *Cross Reference* Browser displays the hierarchy of the C++ program itself, or a subset of it. A *Data*

Browser is used for analyzing the structures, variables and objects within the system. XCodelib also allows component browsing to determine structure and contents, but it is used for supporting understanding of components so as to appropriately reuse them, as opposed to a debugging purpose as in ObjectCenter.

7.4 Network Browsers

There are a large number of information services on the Internet. These information services allow the user to find files, prowl through the Internet, search indexed material and browse the entire network through hypertext links. archie [3] allows a user to search indexes to locate files that are available on public servers. Currently archie indexes about 1200 servers and 2.5 million files. XCodelib uses archie to perform remote searches. gopher [3] is a lookup tool that lets a user prowl through the Internet by selecting resources from menus. It lets the user browse through the Internet's resources regardless of type. Wide Area Information Service (WAIS) [3] helps the user search indexed material. The user can search for particular words or phrases. WAIS returns a list of online files that contain those words. The World Wide Web [3] is another menu based service that helps a user access different resources. However, the Web is based on a flexible "hypertext" model that allows cross-references, or links between related resources. All of the above allow network browsing. XCodelib allows remote searches through archie but also allows browsing of remote components to view structure and contents. XCodelib also downloads remote components to a local library/database and functions like a cache.

All of the above mentioned tools are related in the fact that they provide some of the functionality of XCodelib. Jack Dongarra's *xnetlib* [13] is the most closely related previous work. *xnetlib* allows searching a database of components and downloading them over the network. *xnetlib* is a tool for software distribution. It uses an X Window interface to its large collection of public-domain mathematical software. But unlike XCodelib it does not allow searching for remote components, nor does it allow viewing of local and remote components to understand their structure and contents.

Chapter 8

Conclusions and Future Work

A library system for component reusabilty is designed and implemented. It demonstrates the effectiveness of reuse in the CODE 2.0 system. XCodelib meets all the goals outlined in its inception. It effectively supports reuse in the CODE 2.0 domain and also has a uniform interface for searching local and remote libraries. It allows the CODE 2.0 programmer access to the large body of software available over the Internet. Furthermore, XCodelib also takes into account issues of scalability and effectively deals with them, so that these issues will not become a problem over time. XCodelib also demonstrates the ease and efficiency with which tools can be used to build large and complex systems.

Although an unrestrained network search is reasonable in the current implementation, future versions of XCodelib must try to restrain the Internet search domain to get more context sensitive results. Since the Internet Interface is a collection of scripts, these can be enhanced to do more productive and efficient remote searches. XCodelib uses *archie* to do remote searches. Since the search domain for archie is extremely large, the Internet Interface scripts can be modified to duplicate the functionality of archie in a restricted search domain. The similarity search can also be enhanced by maintaining more information about each component. Finally, although XCodelib was developed for the CODE 2,0 system, it could be used to support reuse in other domains. There is nothing in the XCodelib implementation which is CODE 2.0 specific and so it can be used in a generic fashion to support reuse. The objectives and strategy used in the development of XCodelib could be used in other systems.

Appendix A

User's Manual for XCodelib

A.1 Introduction

Identification and accessing of reusable components, is a critical step in the development of CODE 2.0 programs. The basic philosophy behind XCodelib is providing a uniform interface to both local and remote libraries of reusable components.

XCodelib can guide the user through the process of finding reusable software components. It supports three major steps for reusability : finding the components, understanding the components, downloading the components.

The above steps are executed through the querying and browsing capabilities of XCodelib. The browsing mechanism helps a user determine the contents of the library. The querying mechanism helps a user find the components that are useful. The component viewing mechanism helps a user to understand the components and the downloading mechanism helps a user to download the required components. Figure A.1 shows the main XCodelib window. A user can select the search or the library option.

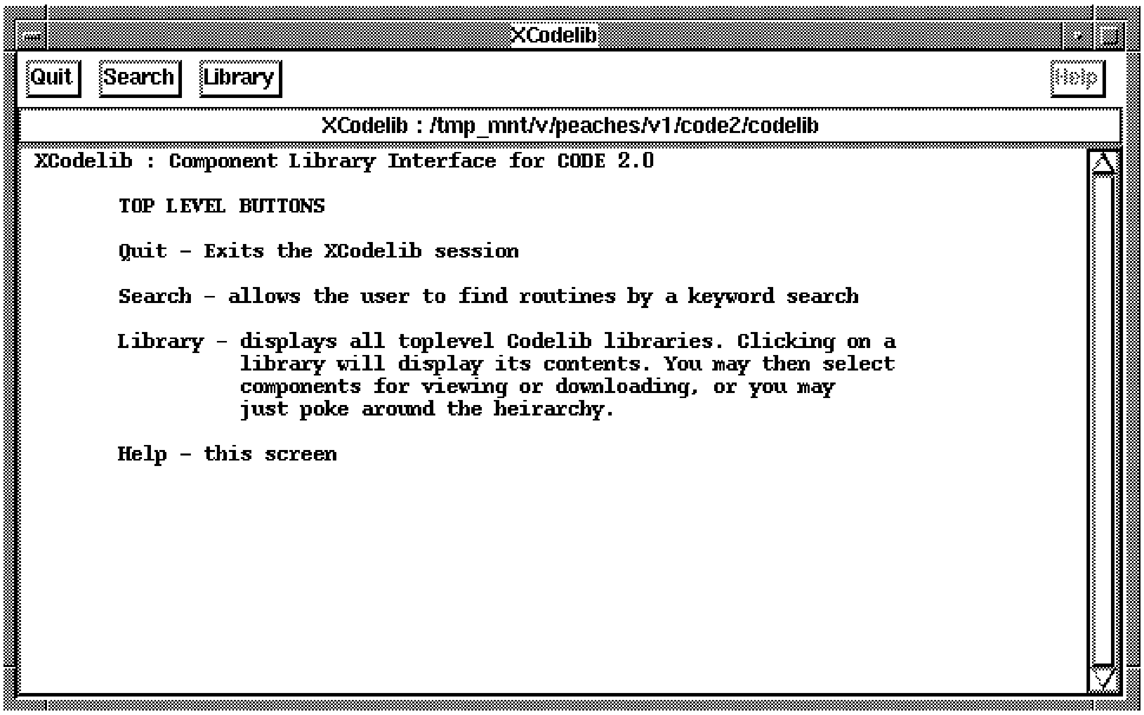

Figure A.1: XCodelib Window

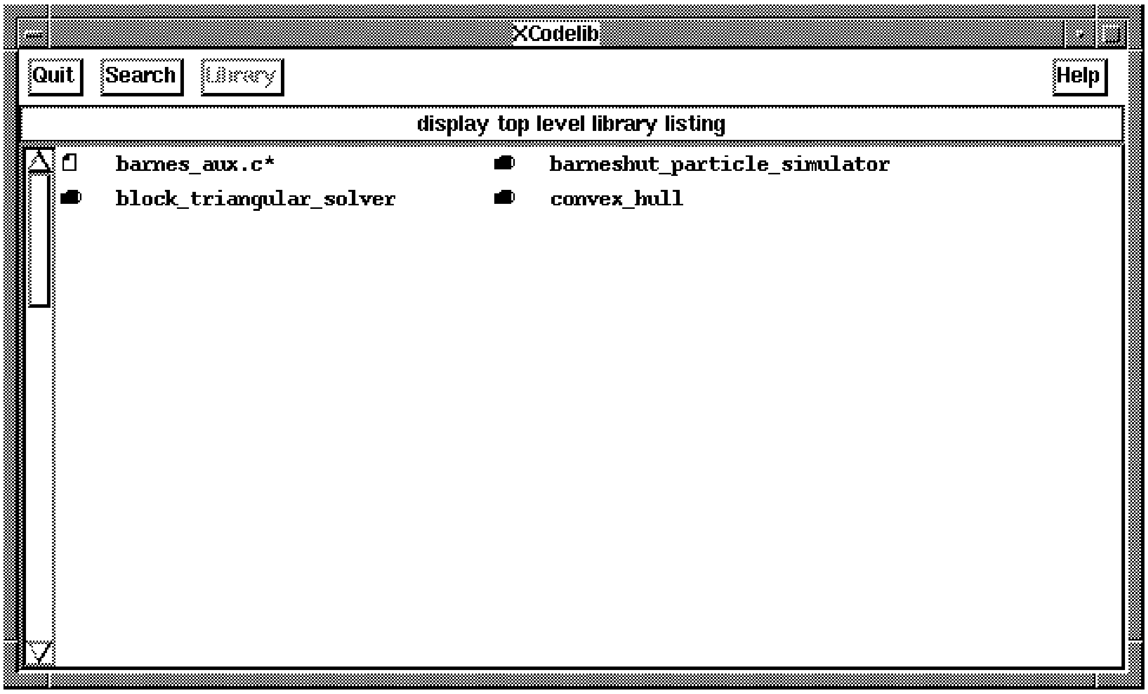

Figure A.2: Toplevel library listing in XCodelib

A.2 Browsing in XCodelib

Selecting the 'library' option from the main window brings the user to the 'browse' window. The 'browse' window displays a toplevel library listing. All the components in the component database are displayed. Figure A.2 shows the browse window of XCodelib. In A.2 four components are listed. Different icons are used to distinguish between directory and file components. Whenever the cursor moves over a component, XCodelib highlights the component. Any component can be selected for viewing by double-clicking the left mouse button on the component. This causes the component display window to come up. Clicking on a directory or file component causes the component directory display window or the component file display window to come up respectively.

A.3 Querying in XCodelib

Selecting the 'search' option from the main window brings the user to the 'search' window. The search window displays the various types of searches which can be performed in XCodelib. A search type of And/Or/Literal can be selected by clicking on the specific button. A serach type has to be selected before a search can be run. A search string of keywords can then be specified and the button 'Go' can be pressed to run the search. In case of an 'AND' search all the keywords in the search expression will be connected by the logical AND operator. Similarly in the case of an 'OR' search all the keywords in the search expression would be connected by the operator of logical OR. The 'Literal' search is a remote search. The search expression in a Literal search must be a literal string. Figure A.3 shows an example of a Literal search. The search expression is the literal string 'matrix'. In case of a Literal search the user can set the number of hits i.e the maximum number of components which can be returned as the results of the search. If the user does not specify the maximum number of hits, the default chosen by XCodelib is five. XCodelib reports the status of each search by specifying the number of components returned as results. The user can select 'Clear' to clear the results of the previous search and to start a new search. The components which are reported as the results of a search can be marked 'similar' by double-clicking the middle mousebutton on the component in the upper search window. This causes the marked components to appear in the bottommost window i.e the similar window. These marked components can be cleared from the similar window by clicking on the 'Delete All' button. Once the marked components have been finalised, the user can set the search type to be 'similar', by clicking on the 'Similar' button. When the user clicks on the 'Similar' button, the previous search results are cleared from the upper search window. Now

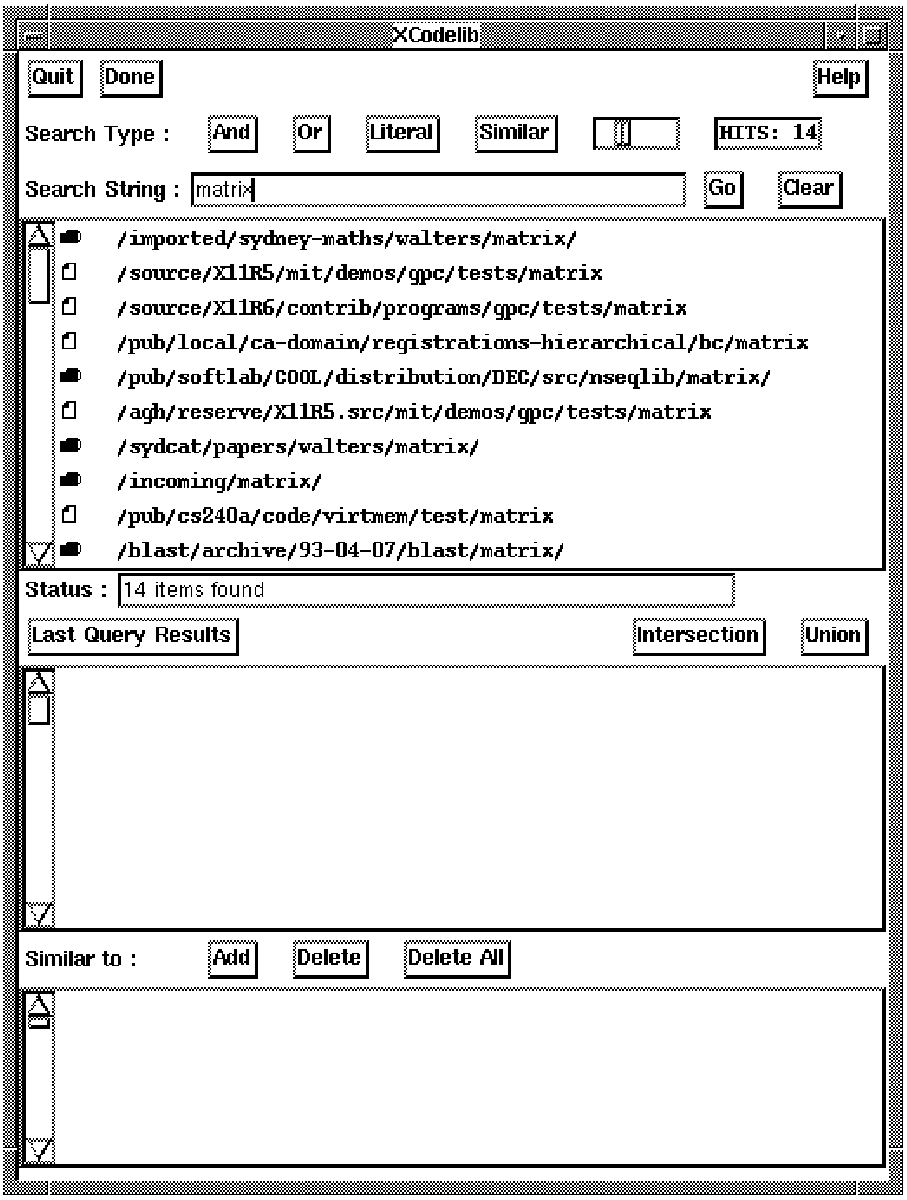

Figure A.3: Different types of search in XCodelib when the user clicks on 'Go' the similarity search is run, and results are reported in the upper search window. Figure A.4 shows an example of a similarity search. All components similar to 'barnes' have been located. The previous query results can be obtained by clicking on the 'Last Query Results' button. This causes XCodelib to display the results of the previous query in the middle window. The 'Union' or 'Intersection' button can then be clicked to select the UNION or INTERSECTION operations respectively. The 'Go' button can then be clicked to obtain the results of the UNION or INTERSECTION search. Figure A.5 shows the results of applying the UNION operator.

A.4 Component Display in XCodelib

Selecting a component for viewing (i.e by double-clicking the left mousebutton on it) causes either the directory component display window or the file component display window to come up depending upon whether the component selected for viewing is a directory or file component. Figure A.6 shows a directory component display and Figure A.7 shows a file component display. In A.6, which is a directory component display, the user can traverse down the directory tree, by doubleclicking the left mousebutton on the nodes of the tree. The user can go up the directory tree by clicking on button 'GoUp' in the component display window. When the user is done viewing a component the 'Done' button in the component display window or the 'Back' button in the file component display window can be selected to return to the library or search window. Selecting a file for viewing from the directory component display causes it to be automatically selected for downloading. XCodelib indicates that a file is selected for downloading by underlining it. The file can be deselected from being downloaded by double-clicking the middle mousebutton on the under-

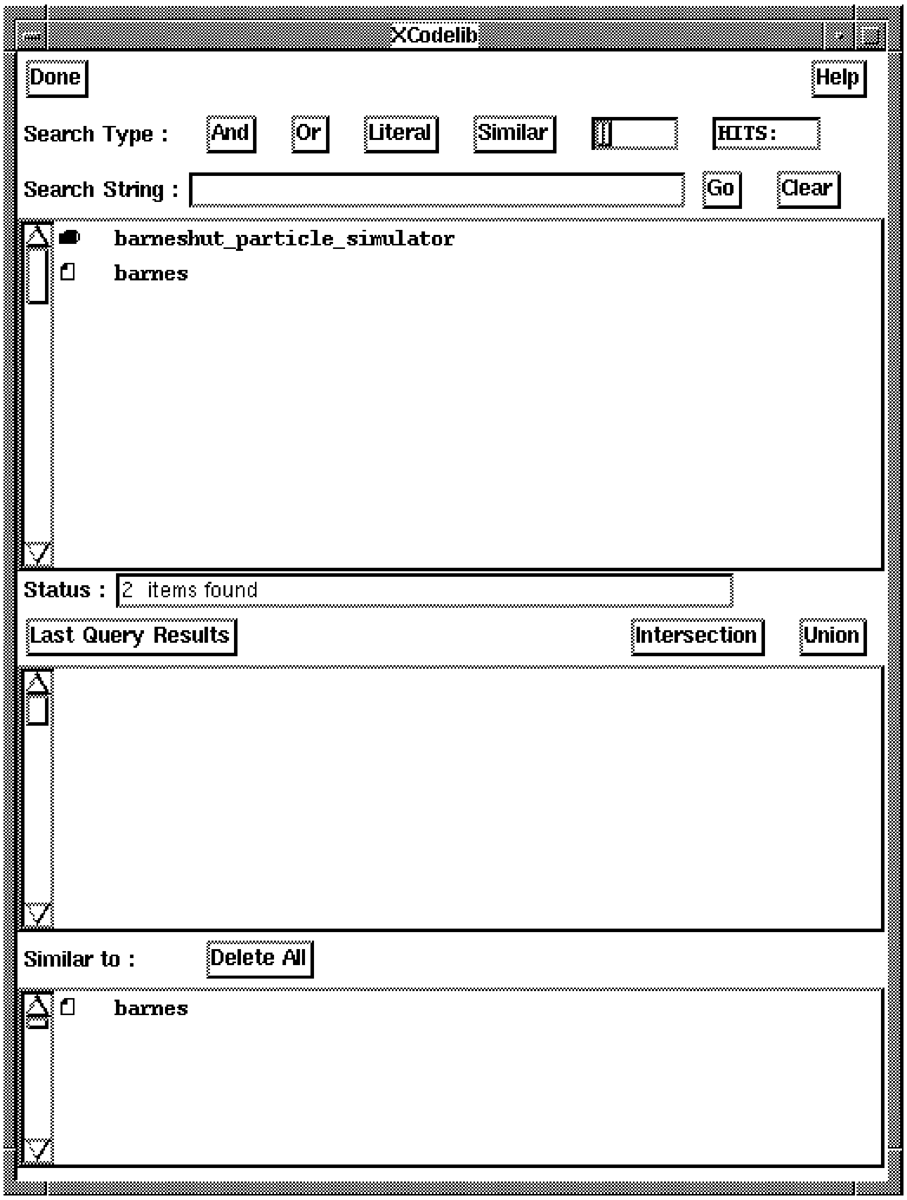

Figure A.4: Similarity search in XCodelib

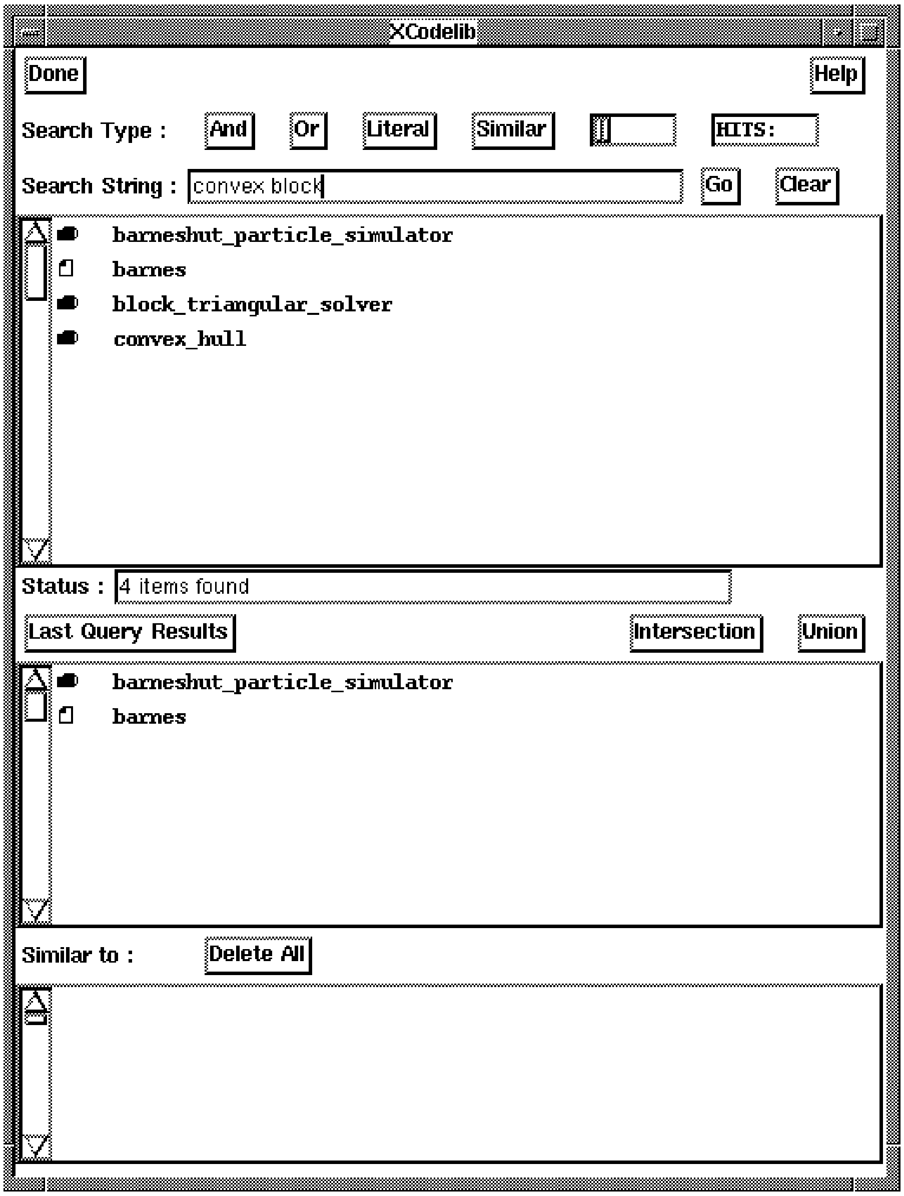

Figure A.5: Union operation on Queries

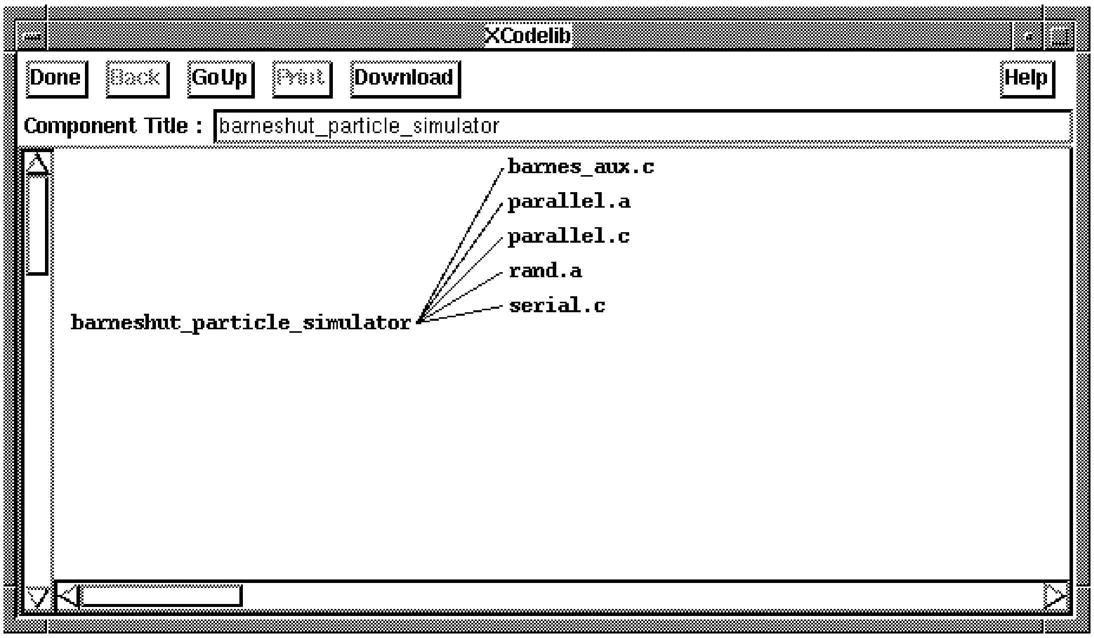

Figure A.6: Component hierarchy display in XCodelib

lined component. This causes the underline to disappear, indicating that the file has been deselected. An entire directory component can be selected for downloading by clicking the left mousebutton under the component name. XCodelib underlines the component name and all the files under that component to indicate that it has been selected for downloading. Similarly, an individual file under a component directory can aslo be selected for downloading by clicking the left mousebutton on the filename. Figure A.8 shows an entire component directory being selected for downloading. The entire component can be deselected by clicking with the middle mousebutton under the component name. Once the selections for downloading have been finalised, the user can click on the 'Download' button to actually download the selected files. When a file component is selected for viewing it also gets automatically selected for downloading. The user can click on the 'Download' button to download the file component.

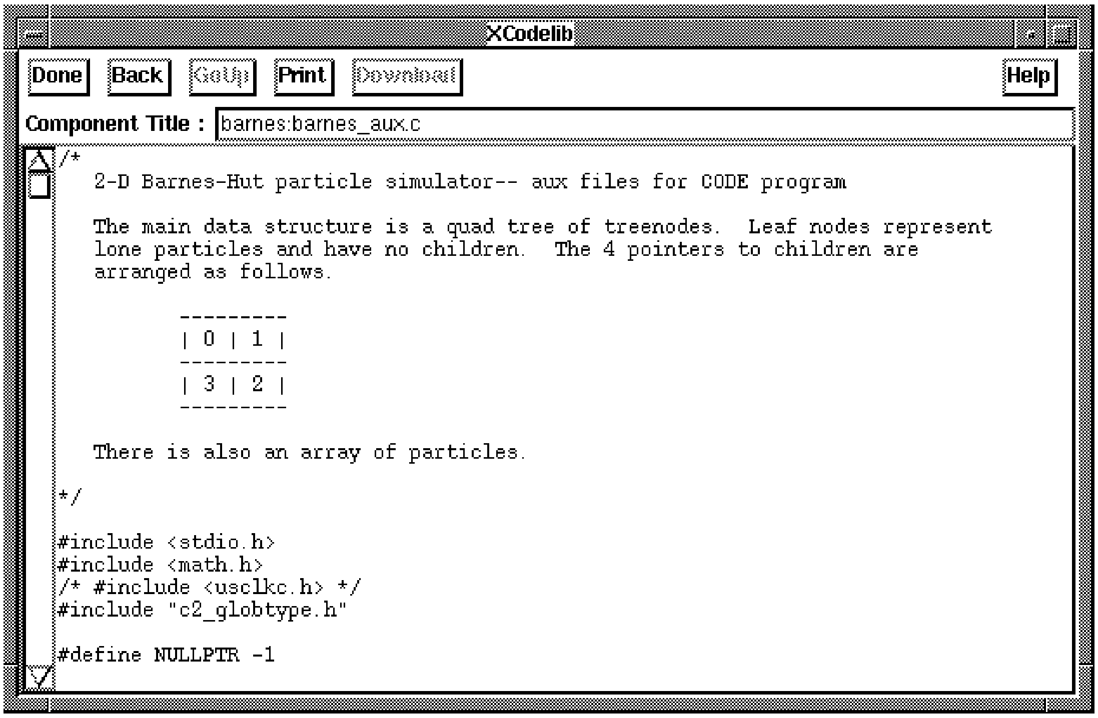

Figure A.7: File Component display in XCodelib

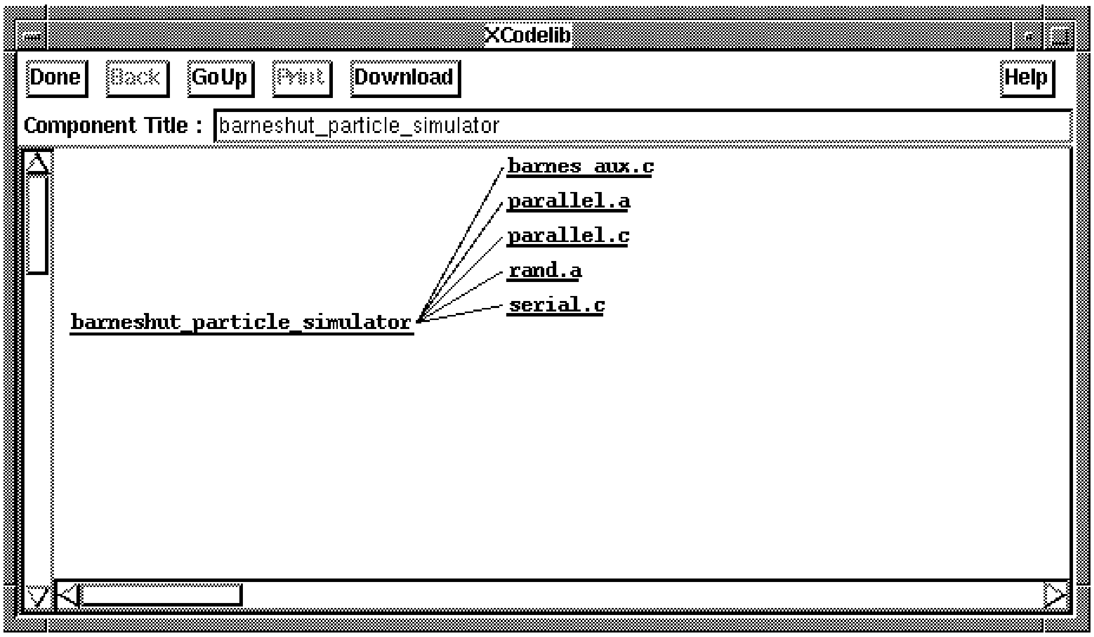

Figure A.8: Component directory selected for downloading

A.5 Downloading Components in XCodelib

Selecting the 'Download' option from the file or directory component display causes the 'download' window to come up. Figure A.9 show the download window of XCodelib. The user can specify the download path (i.e the directory to which the files are to be downloaded). If a remote directory component is being downloaded then the download path must contain as its final directory, the name under which the component is to be stored in the software database. The user can also select and unselect all the files to be downloaded by choosing the 'Select all Files' and 'Unselect all Files' options respectively. When the files which have to be downloaded are finalised, the user can select the 'Get Files Now' option to actually download the files to the directory specified in the download path. XCodelib indicates the progress of the download operation by a growing horizontal bar. Once all the components have been downloaded the user can click on 'Return' to return to the component display.

A.6 Adding Components to the Database

XCodelib allows a programmer to add components to the database. To do so the user must,

- remote login to oracle
- change directory to /v/peaches/v1/code2/codelib
- read the README file
- set the environment variables mentioned in the README file.
- source entry

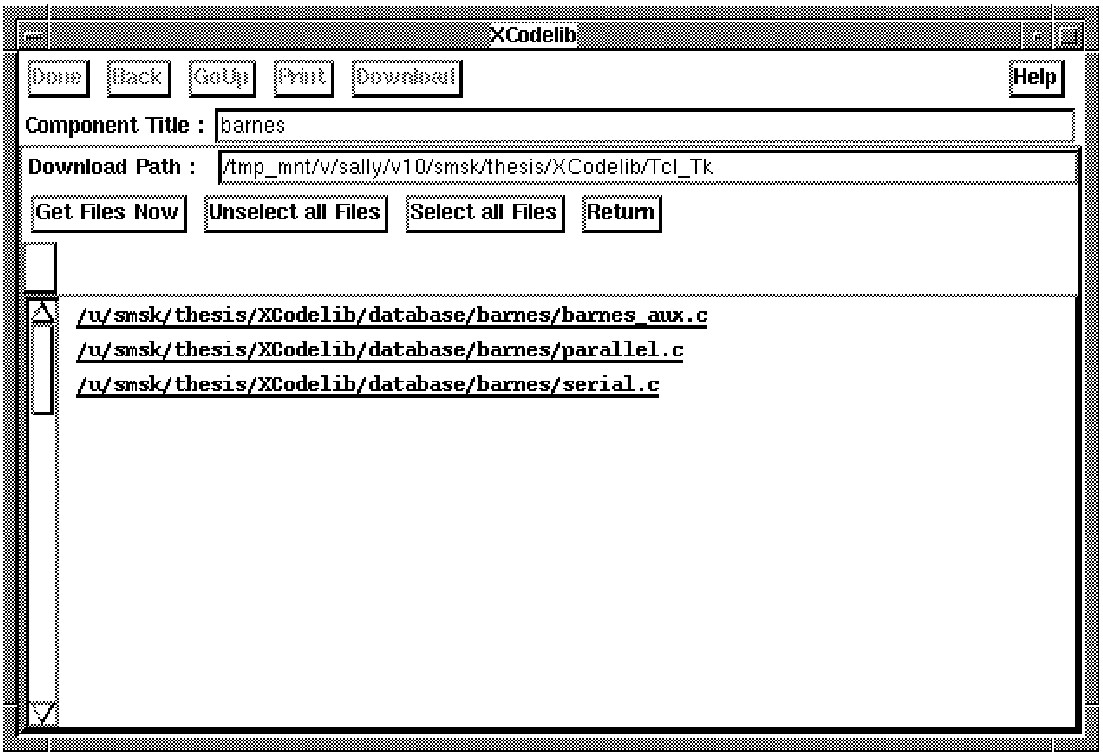

Figure A.9: Download operation in XCodelib

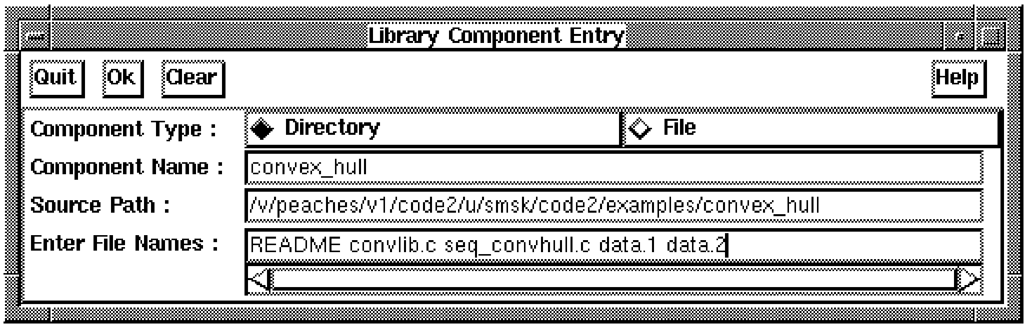

Figure A.10: Adding Components to the Database

A 'Library Component Entry' window comes up. Figure A.10 shows the 'Library Component Entry' window. Depending upon whether the component is a file or directory component, the user must slect the 'Directory' or 'File' option. Then the user must enter the component name, the source path (i.e the path where the files to be added are to be found) and the file names. Once all details of a component have been entered the user must click on the 'Ok' button to commit them. The user can select 'Clear' to clear all the entries and a new component can be added. More than one component can be added at a time. 'Quit' must be selected to quit the application.

A.7 Miscellaneous

The Xcodelib system is present under the fillowing directory /v/peaches/v1/code2/codelib. To run the system, an account on the oracle@cs machine is needed. To execute XCodelib, the user must,

- remote login to oracle
- change directory to /v/peaches/v1/code2/codelib
- read the README file

- set the environment variables mentioned in the README file
- source codelib

Bibliography

[1] P. Newton. A Graphical Retargetable Parallel Programming Environment and it's Efficient Implementation. *Ph.D. thesis*, University of Texas at Austin, Dept. of Computer Sciences, 1993.

[2] J.C. Browne. Formulation and Programming of Parallel Computers: a Unified Approach. *Proc. Int'l Conf. on Parallel Processing*, 1985.

[3] Harley Hahn and Rick Stout. The Internet Complete Referance. *Osborne McGraw-Hill*, 1994.

[4] John K. Ousterhout. Tcl: An Embeddable Command Language. *Proc. of the Winter 1990 USENIX Conferance*, 1990.

[5] John K. Ousterhout. Tcl and the Tk Toolkit. *Addison-Wesley Proffesional Computing Series.*, 1994.

[6] ORACLE7 Server SQL Language Reference Manual. *Oracle Corporation*, 1992.

[7] Tom Poindexter and US WEST Adv. Technologies, Inc. Oratcl - Tcl Interface to Oracle Database Server, 1993.

[8] Don Libes. expect : Scripts for Controlling Interactive Processesa. *Computing Systems, Vol.4,No.2*, University of California Press, Berkeley, CA, 1991.

[9] T. Lee. Software Reuse in Parallel Programming Environments *Ph.D. thesis*, University of Texas at Austin, Dept. of Computer Sciences, 1990.

[10] D. Lea. libg++, the GNU C++ library. *Proc. of the USENIX C++ Conference*, 1988.

[11] K. Gorlen. An Object-Oriented Class Library for C++ Programs. *USENIX C++ Papers*, 1987.

[12] Centerline's ObjectCenter 2.0, the C++ Programming Environment.

[13] J. Dongarra, T. Rowan and R. Wade. Software Distribution using Xnetlib. Oak Ridge National Laboratory. 1993.

Vita

Seema Yeshwant Khedekar was born on October 26, 1969 in Bombay, India, the daughter of Sudha and Yeshwant Khedekar. After completing her high school in Auxilium Convent, Bombay, India, in 1985 she continued junior college studies in D. G. Ruparel College, Bombay, India. She then joined the Computer Engineering program in Vivekanand College, under the University of Bombay, and received her degree in Bachelor of Engineering in August, 1991. Thereafter she worked as a Software Design Engineer in Godrej Mfg. Co. Ltd, Bombay, India for one year. Ms Khedekar then proceeded to The University of Texas at Austin in the Fall of 1992 to join the graduate program in Computer Sciences.

Permanent Address: 1/15 Surya Coop Housing Society
Sector 3, 7C/7D, Vashi
New Bombay 400 703, INDIA

This thesis was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society.