A Node Search Tool
for the
Distributed Collaborative Writing Aid

Liang Chen

Technical Report 96-04

In Partial Fulfillment of the Requirement
for the Degree of
Master of Computer Science and Engineering

Advisor: Dr. Kai H. Chang

Committee:
Dr. James Cross
Dr. Liam Murthy

Auburn University
Auburn, AL 36830
May 15, 1996
Abstract

Conventional computer systems, especially software systems, have been designed to support the single user applications. But this situation has begun to change. With the recent development of computer networks and the widespread deployment of networked workstations, cooperative computing environments for geographically distributed users have become feasible. The software package, Distributed Collaborative Writing Aid (DCWA) is one of such systems. It is designed for a group of people to contribute to the writing of a document or program.

In a group working environment, the document that the members are creating can be complicated and should be organized in a hierarchical tree structure. For a member to locate a desired portion of the document, the search process can be tedious. In order to help the member to perform the search efficiently, a search tool must be provided. This report describes a node search facility for the DCWA. The tool has been fully developed and tested. It will be integrated into the rest of the environment in the near future.
Table of Contents

1. Introduction ........................................................................................................................ 1

2. An Overview of the Node Search Facility ................................................................. 2

3. Detailed Description of the Project ............................................................................ 3
   3.1 Architecture Analysis ................................................................................................. 3
   Figure 1 ........................................................................................................................... 3
   3.2 Detailed Analysis ....................................................................................................... 4
   Figure 2 ........................................................................................................................... 4

4. Design of the Node Search Tool .................................................................................. 6
   4.1 Node Attributes ......................................................................................................... 6
   4.2 Query ...................................................................................................................... 7
      1). New Query .......................................................................................... 8
      2). Revised Query .................................................................................... 9
   Figure 3 - 6 ................................................................................................................. 7
   4.3 The Node Search Window ....................................................................................... 11
   Figure 7 ...................................................................................................................... 11
   4.4 Operation of the Node Search Tool ......................................................................... 13
   Figure 8 - 11 ............................................................................................................. 13

5. More Detailed Examples .............................................................................................. 14
   An Example .................................................................................................................. 14
   Figure 12 - 18 ............................................................................................................ 14

6. Conclusion and Future Work ..................................................................................... 16
   6.1 Benefits .................................................................................................................. 16
   6.2 Limitations and Future Work ................................................................................. 17

7. References ..................................................................................................................... 18
1. Introduction

Traditional computer applications have been designed to be run by one user at a time who does work in a single medium, such as ASCII text, and very little regard has been given to the fact that people often work together. If group members need to work on the same item, for example a document, in most cases, they must work together to prevent inconsistency. With the recent development of computer networks and the widespread deployment of networked workstations, cooperation process for geographically distributed users has become feasible. This leads to the generation of the new multi-user computer applications software packages known as Computer Supported Cooperative Work (CSCW). The overall purpose of CSCW is to support distributed, real-time, multimedia collaborative work. That is, it will allow a group of people working at remote sites, connected via Internet, to collectively work on multimedia documents at the same time, with changes made by one of the users reflected on the other users’ workstations in real-time. The tool, Distributed Collaborative Writing Aid (DCWA) [1] is one of them.

In a collaborative writing environment, the document that a group of members are creating can be complicated and is represented in a file structure which resembles a tree hierarchy. In order to help a user to locate a desired portion of the document quickly, a search facility must be provided. The emphasis of this report is on the node search facility for the DCWA’s distributed database, which is crucial to make the entire system work efficiently.
2. An Overview of the Node Search Facility

In group writing, especially in the case of "division of labor" writing [2], each writer only concentrates on his/her own part. It has been observed that collaborators need to know the details of a node written by another in order to use the node appropriately. If the node contains C program code, for example, the tediousness of the code often gives rise to the need for a reader to consult with the writer directly for clarification. In this case, an automatic search facility of the nodes’ attributes would help a collaborator gain knowledge about what the others have written without having to actually read their work or ask them directly. It would also help a user quickly locate the node(s) he/she wants to edit. In fact, in most commercial databases, the search facilities are a common feature and this feature should be available in a CSCW environment too. The use of node attributes and search techniques distinguishes the DCWA from other CSCW systems.

A user can view his/her own or other members’ working areas by clicking on the nodes in the organizational hierarchy structure. In addition to the actual node content, each node in the file structure also contains attributes that describe the nature of the node. Typical descriptions can be a series of key words or concepts of the node content. A search tool that allows a user to specify the node descriptions can locate a desired node quickly. In this system, the node specification can be any logical combinations of the node attributes and the located nodes are color-coded in the structure for easy identification.
3. Detailed Description of the Project

The purpose of this project is to develop an easy-to-use, X/Motif-based query tool to access nodes from a database file. This is achieved by building an operational and easy to use front-end Graphic User Interface (GUI). The database is organized in a hierarchical tree and each node contains attributes that describe its nature. This tool will help a user access the information in the database, without ever worrying about the commands required to retrieve the information. Anyone with a knowledge of windows, menus, buttons, mouse and keyboard should be able to use the tool.

3.1 Architecture Analysis

The DCWA Prototype can be divided into five independent processes. The first is the DCWA daemon process. It is running in the background of all machines which may potentially be used by the group members. The DCWA daemon is responsible for the asynchronous start of the software, as well as establishing any initial socket connections [3].

After starting the daemon process, DCWA creates a controller (CR) process, which forks off three child processes on the host, namely the Graphical User Interface (UI), the Database (DB), and the Network Access (NA) processes. Communication between these sibling processes is accomplished through UNIX pipes and, because pipes are unidirectional, a pair of pipes must be established between every pair of processes. Communication with the other user’s host is accomplished via an Internet socket. Therefore, prior to forking its child processes, the CR creates all the pipes, as well as the socket. After forking its child processes, the CR suspends itself until the user wants to quit, when these connections are torn down.
When another group member starts DCWA, similar processes are created on his/her host, which results in the configuration shown in Figure 1. The responsibilities of the node search facility are to accept user defined queries from the User Interface(UI) and to search through the selected files in the Database(DB) to process the queries and display the results. The responsibilities also include saving the results in the default output files so that they can be used by the later processes.
3.2 Detailed Analysis

The search procedure can be divided into three steps: First, identify the files to be searched (i.e., search() function in the program). For a new query, the default looked up file is the ".com" file. For any other type of queries, the user may choose to look up either the ".com" file or an intermediate file. The intermediate file contains the information about those nodes that have satisfied previous queries. Second, search the user defined queries in the selected file (i.e., do_search() function). This is a detailed implementation of step one. In this step, the tool will compare the input keywords with each attribute in the selected file from the first step. Actually, this is the kernel step of the search tool. Third, display and save the hit nodes (i.e., cp() function). A pop-up window will show the attributes of the matched nodes according to the user’s requirements. Figure 2 illustrates a flowchart of the detailed design.

![Flowchart of the detailed design](image-url)
Step 1: Identify the files to be searched

Normally, a search would start by searching the original complete document, the ".com" file. The results of the first query are stored in a temporary file, hit0.tmp. This temporary file can be used in the subsequent search and more temporary files hit*.tmp (*=0,1,2...) will be generated. When the user is satisfied with the search results, he may stop the search and the latest version of the hit*.tmp would contain the search results. Searching from the results of the previous search is accomplished by the Pre_query function (explained later). However, if the user is not satisfied with the current search, he can always start the search over again.

Step 2: Query Search

The responsibilities of this step is to process the user defined query in the selected database file. Each database file (".com" or "hit*.tmp" file) normally contains many nodes. When the do_search() function performs the search, a file pointer will move down from line to line for comparison and will return the nodes that contain the user specified keywords or do not contain the keywords, if the Not_relation query is selected.

When a node is determined to have matched the desired keyword value, the file pointer must be moved back to the position of the first attribute of the node. However, due to the variant size of the attribute values, the move back size for the file pointer is undetermined. A simple solution is to copy all the attributes of a node to a temporary file, say, "temp.tmp". The keyword comparison process is then done in this file. If the node contains the specified keywords, the contents of "temp.tmp" is copied to the output file, for example, "hit0.tmp". If no match is found, the information of the next node will be copied to the temporary file "temp.tmp" and the search process repeats until it reaches the last node of
the "com" file. At the end, the output file contains all the hit nodes. If the output file is empty, that means no nodes were hit and a message window will pop up to tell the user "No Match" in the search.

**Step 3: Display and Save Results**

The third step is to display and to save the hit nodes. A user may want to display just some parts of the hit nodes. For this reason, a flag array (i.e., display[]) is created to control the display. Only those attributes that have been set on by the flag array are displayed. However, if a user does not specify the attributes to be displayed, all the attributes would be displayed. The latest output file hit*.tmp holds all the attributes of the hit nodes. It should be noted that when a new search is completed, the contents of the display window will be updated to reflect the results.

**4. Design of the Node Search Tool**

**4.1 Node Attributes**

The basis for modeling semantics in the DCWA database is to assign attributes to each node. Through attributes, *meanings* are assigned to the node. A node can then be related to other nodes through relational operations such as classification, hierarchy, and inheritance [1]. The following is the list of default attributes that are maintained for each node.

*Node Id*: An integer

*Name*: A character string label for the node.

*Node Level*: One of Root, Intermediate, or Leaf.

*Creation Time*: A number of type time_t;

*Creator*: A character string holding a user name.
4.2 Query

A user may use a queried search on the logical structure to limit his/her personal view of the tree.

There are two types of queries, new and revised. A new query searches the group’s entire logical view
for matching nodes while a revised query only searches the nodes that match the previous query (either new or revised). In either case, a "hit list" of the matched nodes will be returned to the User Interface (UI) and the user may then select a node from the hit list to read or edit. The user can also make another query to further limit the view. If a query does not produce any nodes that the user is interested in, the user may start over again by making a new query which will search the entire logical view.

1). **New Query Search:**

A new query is actually equivalent to a Structured Query Language (SQL) statement and is sent to the database for process. A user can specify a query by using the following elements:

- **Select:** It allows a user to choose a list of attributes to be displayed. This is accomplished by clicking on the desired attributes under the "Display For" dialog box.

- **From:** Database files to be searched (".com" file by default in this project)

- **Where:** User defined keywords. The search process is based on these keywords. This is accomplished by typing in the keywords in the query dialog box.

For example:

- **Select:** Node Name, Creator, Description, Keyword, User defined Attributes (from the "Display For" dialog box)

- **From:** all ".com" database files

- **Where:** Node Id=3 (typed in query dialog box)

For the attributes specified in the SELECT field, their associated values will be returned to the User Interface (UI). In addition, the node id is always sent to the UI as the first attribute value. This allows
the UI to identify the nodes in the hit list and to communicate with the DataBase (DB) component.

The other attributes are sent to the UI in the order that the user specified in the SELECT field of the query. Note that the nodes shown in the hit list are no longer arranged in a tree structure since such an organizational structure does not exist. That is, they are simply a list of nodes which meet the query criteria.

2). Revised Query Search:

The user may make a revised query search based on what was obtained from the previous query. A revised query is one of the following: AND, OR and NOT. An And_relation query (also called Pre_query) will extract matched nodes from the immediately previous query search. An Or_relation query will append matched nodes to the the results of the previous query. The Not_relation query itself contains three subtypes: New_Not query, And_Not query, and Or_Not query.

*New_Not query* is similar to the New query except that the hit nodes do NOT contain the user defined keywords. *And_Not query* and *Or_Not query* are similar to the And_relation and Or_relation queries except that the hit nodes do NOT contain the user defined keywords. A revised query is defined similar to a new query, except:

- The SELECT field may be different from what it was in the previous query. If the user still wants to display an attribute’s value that was displayed in the previous query, the attribute must be listed again.

- The WHERE field will normally be different. If it is different, the resulting hit list is ANDed, ORed or NOTed to the previous hit list to form the new hit list. If it is the same, the hit list will remain the same.
The functionality of these queries are further explained by the following examples.

1. A new query:

   - Select: Node Id, Node Name, Description, Keyword (From "Display For"
     dialog box)
   - From: All ".com" database files
   - Where: Node Name=n1 (typed in the query dialog box)

   A sample result is shown in Figure 3. This result is used as the lookup file for the next query.

2. Pre_query (And_Relation query):

   - Select: Node Id, Node Name, Description, Keyword
   - From: All "hit*.tmp" database files
   - Where: Keyword=Toyota

   A sample result is shown in Figure 4. This result is also used as the lookup file for the next query.

3. Or_Relation query:

   - Select: Node Id, Node Name, Description, Keyword
   - From: All ".com" database files
   - Where: Description=ship

   A sample result is shown in Figure 5. This result is used as the input file for the next query too.
4. **And_Not_Relation query:**

- Select: Node Name, Topic, Description
- From: All "hit*.tmp" database files
- Where: Keyword=Toyota

A sample result is shown in Figure 6.
4.3 The Node Search Window

The search window includes four major parts (See Figure 7). They are: A query type selection box, an attribute option_menu (hidden in the option_menu button), five user input query dialog boxes, and a display selection box. A user may specify a query by clicking a certain toggle button from the query type selection box. If the user tries to trace some specific attributes in the database, he/she may specify the attributes by clicking the option_menu button and then choosing them from a pull-down option_menu (which contains all the attributes of nodes). The format of the display is also flexible. A user can select whatever he/she wants to display by selecting a single or multiple attributes from the "Display For" box.

By default, if the user does not specify any query toggle button, the program will accept the query in the "User Input Query" dialog box. The search will then be based on this query and go through all the attributes of all nodes in the selected file.

The lower part of the node search window consists four query dialog boxes. It has two columns: "Attributes" and "Query". The Attributes column accepts attributes from the "Attributes for Query" option_menu. That is, when a user select a specific attribute from the option_menu, this attribute is passed to the Attributes column automatically. The user can then type in the desired value in the Query column.
A user can specify up to four attributes in the query table. Note that, whenever the user types in a value in anyone of the four dialog boxes, its corresponding toggle button at the left-most side of the query table is set on automatically, and the other query toggle buttons would be set off. Thus, the program knows which attribute should be focused on and guarantees that no conflicts will happen. If the user tries to use more than four different attributes in the query table, an error message will pop up and suggest the user to use the "User Input Query" dialog box or to "Reset" the query table.

Figure 7. The Node Search Tool Window
4.4 Operation of the Node Search Tool

The search tool window is accessible through the "Node_Search" button in the main window. Once the user clicks the "Node_Search" button, the search window pops up immediately (see Figure 7). A search session should proceed as follows:

**Step 1:** Select the query type, which includes New Query, Pre_Query, OR_Relation Query and Not_Relation Query. The first time search should use New Query, since no pre-query exists. If the user chooses the Pre_Query or OR_Relation Query, an error message will pop-up to notify the user to change the selection. However, if the user doesn’t make a choice, by default, the program will assume that the user is going to make a new query. This step is shown in Figures 8 and 9.

**Step 2:** Select the attribute(s) that the user tries to query. The user can simply click on the "Attributes for Query" option_menu button and choose an attribute from a list of node attributes. The selected attribute is then shown in the "attributes" column in the lower left part of the window. In Figure 8, for example, "keywords" has been selected.

**Step 3:** After the user selects the attribute, he/she may specify the desired attribute value in the "Query" column of the table (lower right part of the window). The corresponding toggle button on that line is then automatically set on. This means the search is going to focus on that attribute. In the example of Figure 10, "Toyota" has been specified as the desired keyword value.

**Step 4:** The user may specify the attributes he/she would like to display. The user can specify by
choosing either a single or multiple attributes. However, if the user doesn’t specify any attribute, all
the attributes will be displayed by default. Figure 10 indicates that some attributes in the "Display For"
dialog box have been selected.

**Step 5:** So far, the user has finished building the query and the display format. If the search is case
sensitive, the user can click the "Case Matters" toggle button to turn it on. At this time, clicking on the
"Search" button would start the search process and the result is displayed. This step is shown in Figure
11 and the result of the search is similar to that shown in Figure 4.
5. More Detailed Examples

The Node Search Tool has been successfully implemented, and it has accomplished the anticipated result. Its main purpose is to help a user locate a desired node in the document quickly. It makes use of the meanings (or attributes) assigned to the object structure. The attributes of the nodes may actually form a semantic network which is separate from the organizational (or node) structure. The separation of these two structures can prevent the disorientation a hypertext (where organizational information and semantic information are mixed up) user normally suffers [4].

An Example

To illustrate the capability of the tool, let us trace the executions of some sample queries. For example, in a writing application, a section may correspond to a segment of text that mentions an engine design, and all the nodes of the section are organized in a hierarchical tree structure. This logical structure is presented in Figure 12. Suppose a user wishes to get all the nodes about "engine" design. What the user needs to do is to click the "New Query" toggle button, then input the keyword "Engine" and click "Search" button. The search will go through all the attributes of each node in the database file and extract the nodes that match the user input query. The search window and the search results are shown in Figure 13.

Suppose the user tries to retrieve the engine design about "automobile" in the second search and just wants to display "Node Name", "Creator", "Topic", "Description", and "Keywords". He/She can type "auto" in the query dialog box and select the five attributes that he/she wants to display from the "Display For" selection box. Because this search is based on the first search, the user should select "Pre_Query" button this time. The result will be displayed in a pop-up window. Again, if the user tries
Figure 12. A real-world example of the DCWA
Figure 13. New Query ---- "Engine"
to search all the nodes about "ship" engine design and append the results to the previous one. He/she then can just select the "Or_Relation Query" button and type "ship" in query dialog box. The results will be appended to the previous results. The windows and results of these search are shown in Figures 14 and 15.

Figure 14. Pre_Query ---- "Engine" AND "Auto"
So far, we have retrieved all the nodes that are related to the auto or ship engine design. Suppose the user likes to continue to search the nodes that also contain the "sun-roof" feature. The search must be based on the previous two searches. The user just needs to press the "Pre_Query" button, then selects attribute "keywords" from the pull-down option menu, and then types "sun-roof" in the associated table query dialog box and clicks the "Search" button. The search will search the "keywords" attribute of those nodes form the previous two queries and extract the matched nodes. This search is illustrated
The user can restart a new query at anytime. Suppose the user wants to start a new query by clicking the "New Query" button, and tries to retrieve the engine design about plane. He/She can type in "plane" in the query dialog box and then click the "Search" button. If the user also wants all the nodes about plane that do NOT have the "anti-lock brakes" feature, he/she should select the "And_Not_Relation" query and type in the keywords "anti-lock brakes". The results are shown in Figures 17 and 18.
Figure 17. New Query — “Plane”

Figure 18. And_Not Query — “Plane And_Not "AstrickBrushes"”
6. Conclusion and Future Work

The distributed cooperative writing tool, DCWA, will be useful to any group of people who have a need to coauthor documents. The DCWA database organizes both textual and graphical information according to their structural relationships. Users may navigate through the files by using the search facility. The search facility can limit the user’s view to only those nodes which meet the query criteria.

6.1. Benefits

The experience gained from the design and implementation of the Node Search Tool indicates the following benefits:

- The search tool has a friendly user interface. Anyone who has a basic knowledge of window can grasp it easily. The on-line help provides very useful references.

- The search can be conducted as a general node search or a specific node attribute search. The result display provides the same flexibility as well.

- The software of the search tool includes many exception-handling functions which make it reliable.

- The Node Search Tool is relatively independent from the other parts of the DCWA. This increases the maintainability and modifiability of the tool.

6.2. Limitations and Future Work:

There still exist some limitations of the tool. They are:
The tool is only a search tool, and the user cannot update the nodes’ attributes through it.

Some assumptions are predefined.

- The database files are in the same working directory as the node search tool.
- The size of each attribute of a node is less than 1K.
- The maximum depth level of the Pre_Query that a user can reach is 10.

File access (i.e., read/write) is frequent, thus affects the performance of the tool, especially for a large database file.

Currently, the information about the hit nodes is simply displayed in a window and saved in a file. Integration is still to be done so that the hit nodes can be highlighted in the logical view structure.

This will provide the user easy access to the nodes.

The DCWA project is still in progress. Most of these limitations should be overcome after it is integrated with the rest of the DCWA.

7. Reference


1993.
