references, to over a hundred, in the case of a review article (see [Fox 87b]). A document also may be cited by a large number of documents; for example, Dijkstra's letter [Dijkstra 68] on GOTO's has generated much controversy and still generates discussion (see editorial comment after [Klein 88]), so it is referenced many, many times. Furthermore, a document may typically have as many as twenty to thirty terms as part of its representation.

This potentially large number of links cannot be effectively displayed on a graphic screen. Therefore, the number of nodes that can appear around a node of interest is reduced and the organization of the screen is made regular so that the information can be effectively managed and displayed. The maps are organized as a square grid (see figure 5.3), which is not displayed. Only one object is displayed in a cell whether it is a link or a node. This limits the number of new nodes that can be displayed around a central node to a maximum of seven, since every node (but the initial node in the map) has an input link. Nodes in a neighborhood are drawn one cell away, with the links and their labels being drawn in the adjacent cells. Different symbols indicate different kinds of elements; circles represent documents, octagons represent concepts, squares represent lists of documents, hexagons represent journal issues, and diamonds are connectors. These symbols are shaded to indicate their status. Black indicates a relevant element, gray indicates an element that has been viewed by the user, diagonal bars indicate a node that is recommended by the BE. An oversize square around another node is a mark that the user can place to indicate a node that he might wish to come back to. In addition, the nodes are labeled; document nodes have the document number, concept nodes have the concept, list nodes have the number of items that they represent.

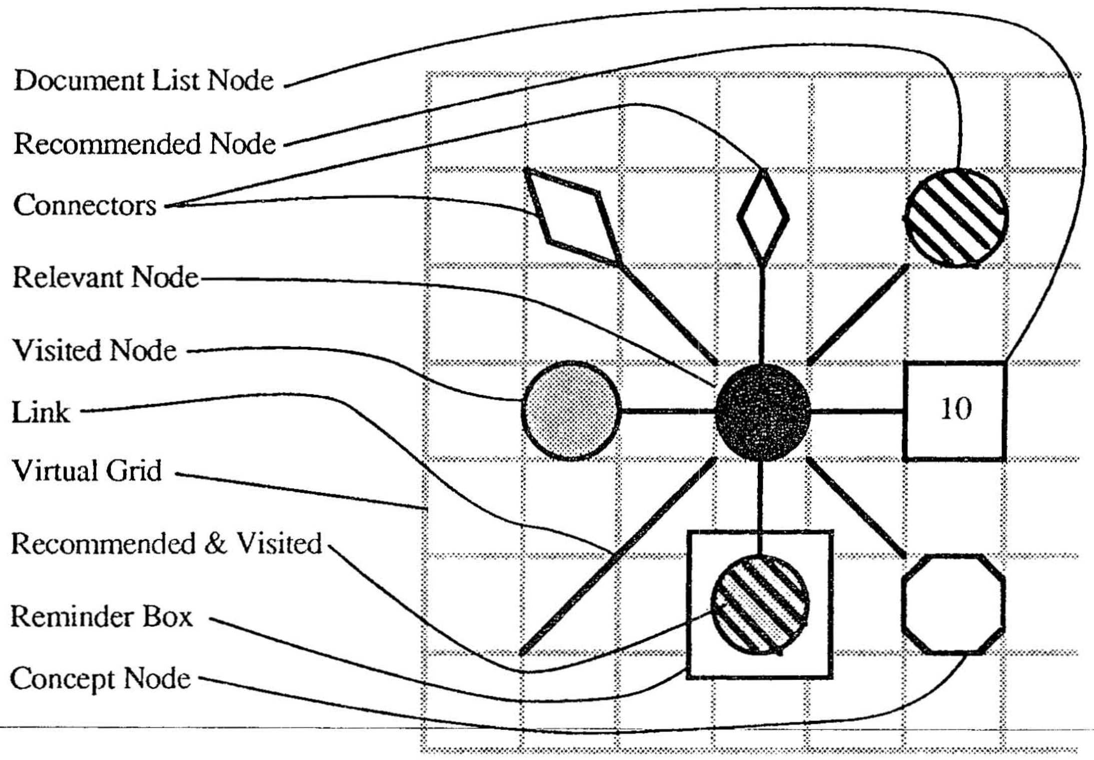


Figure 5.3: Grid for browsing maps showing different node markings, but without labels.

Organization of the browsing maps in this way was done with two considerations in mind, cognitive economy and machine efficiency. It is desirable not to overwhelm the user with a mass of nodes, such as can be the case even for an average document. It is not unreasonable to assume that a document might contain 10 terms and 10 references, making 20 links to other items. To display this many nodes requires a great deal of space on the screen. If a node is represented by a 1 inch diameter circle, the 20 related nodes have to be drawn on an approximately 6 inch diameter circle around the center node. Furthermore, consideration must be given to the space needed for labeling the links.

When the user traverses a link to examine the contents of another node and desires to see the neighborhood, the link must be extended and space must be found to draw the neighborhood. Initially, this is not a problem, but as the user continues to examine different nodes, it becomes very difficult to manage the space. Furthermore, many of the nodes may not be examined, so their presence only contributes to the visual clutter.

The actual number of nodes shown depends on the connectivity of the node and the user model. For example, a term that is found in only two documents, and has no nearest neighbors or domain knowledge connections will only have two links. A system expert will have a maximum of seven nodes shown to him, where a system novice will only have four. Consequently, the browsing expert must make a choice about the nodes that are likely to be useful, and what node it considers to be the most useful. This is accomplished using heuristics that are described in section 5.3.2.

There are times, due to the way that the user moves through the maps, that it may not be possible to expand the neighborhood of a node and still keep it linked to the originating node. For example, if the user takes a breadth-first approach to viewing the nodes, the neighborhood around the first node will be filled quickly. In this case, a connector is used. The node to be expanded is replaced by a connector, and its neighborhood is drawn somewhere else on the map where there is sufficient open space. It is drawn with a connector on its input node. If the user selects the connector, the map is moved to the location of the neighborhood. If the connector on the input link to the new neighborhood is selected the map is moved back to the neighborhood of the originating node.

As mentioned previously the user can elect to go off in a direction of his own choosing. To do so, the user selects from the choices available in the content menus of the windows that display a document or a concept. In the case of a document, the user can look at the reference list, citation list, nearest neighbor list, or journal issue list. If one of these is selected, a node representing it will be placed on the maps. This is shown in figure 5.4. The reference list for the selected paper is:

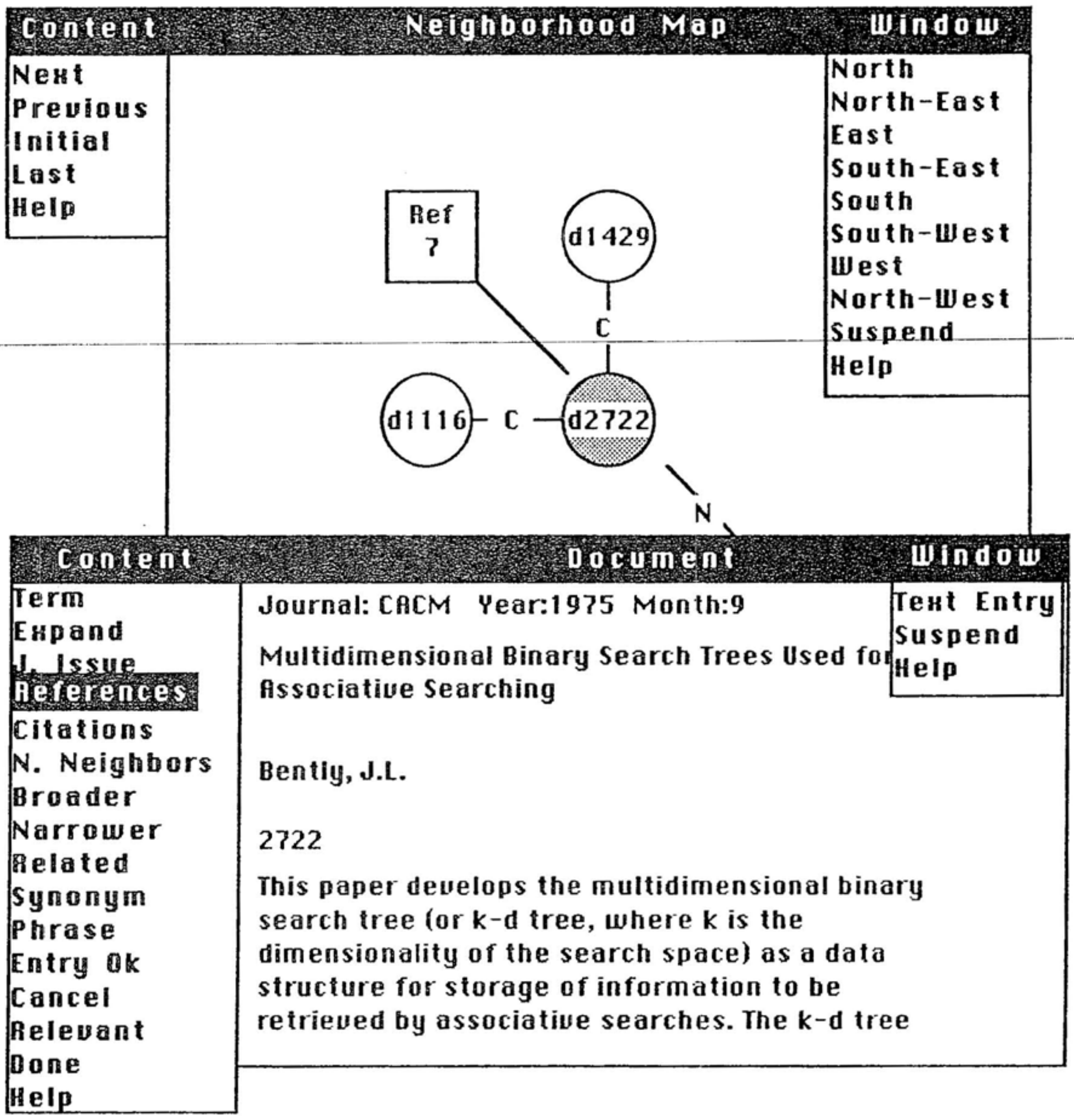
1. Friedman, J.H., Bently, J.L., Finkel, R.A. *An algorithm for finding best matches in logarithmic time*. Stanford CS Rep. 75-482.
2. Blum, M., Floyd, R.W., Pratt, V., Rivest R.L., Tarjan, R.E. *Time bounds for selection*. Stanford CS Rep. 73-349.
3. Finkel, R.A., & Bently, J.L "Quad Trees: a data structure for retrieval on composite key." *Acta Informatica* **4**, 1(1974), 1-9.

4. Knuth, D.E. *The Art of Computer Programming, Vol I: Fundamental Algorithms*. Addison-Wesley, Reading MA, 1969.

5. Knuth, D.E. *The Art of Computer Programming, Vol III: Sorting and Searching* Addison-Wesley, Reading MA, 1973.

6. McCreight, E. Computer Science 144A midterm examination, spring quarter, 1973, Stanford University.

7. Rivest, R.L. *Analysis of Associative Retrieval Algorithms*. Stanford CS Rep. 74-415.



Journal: CACM Year:1975 Month:9

Multidimensional Binary Search Trees Used for Associative Searching

Bently, J.L.

2722

This paper develops the multidimensional binary search tree (or k-d tree, where k is the dimensionality of the search space) as a data structure for storage of information to be retrieved by associative searches. The k-d tree

Figure 5.4: User chooses the `References` selection.

This list would be displayed in a document-list window with a banner labeling it as `Reference List: Document 2722`. None of these papers are available in the current collection, so the user cannot go anywhere. If any of the papers turned out to have an interesting title, which the user selected for viewing, the document list node would be expanded and the document node drawn around it. He then could choose the `j.issue` option to find that the issue contained primarily papers that are related by the fact that they were the top entries in the 1975 Forsythe student paper competition. The only citations in the current collection are already displayed on the map.

In the case of a concept, he can choose the `Select` option from the content menu and then select the concept he wishes to examine. A window with the new concept then appears and a concept node is added to the map.

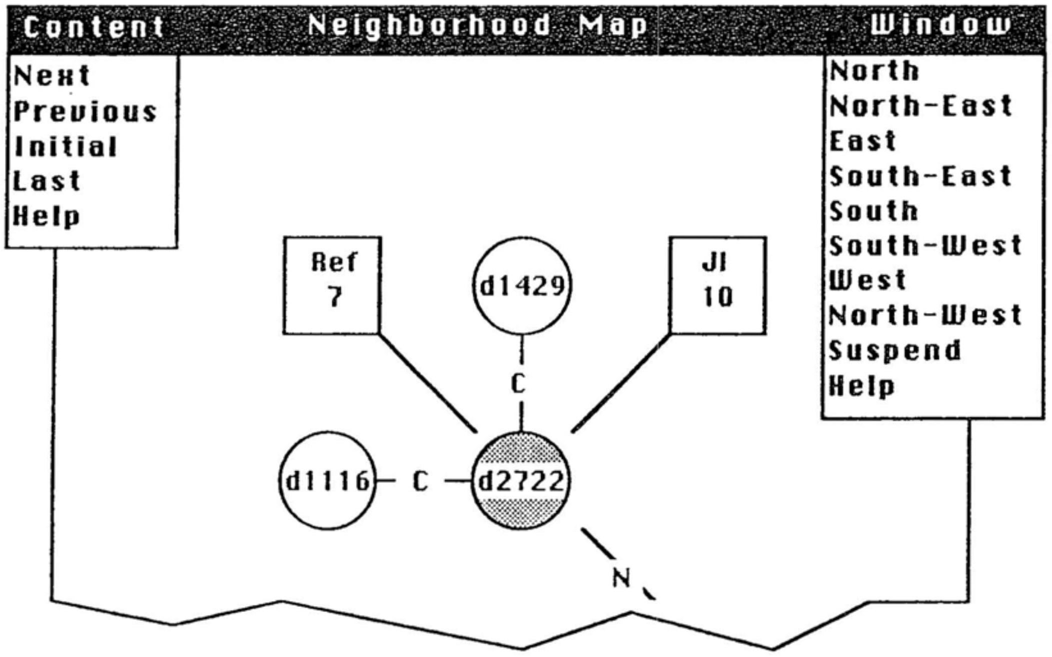


Figure 5.5: Neighborhood Map showing the addition of Reference and Journal Issue nodes.

5.3.2 Browsing Heuristics

The browsing heuristics make use of the evidence provided by the links and the request model to recommend nodes that are strongly related to the node currently in view. Those that are related by multiple pieces of evidence are selected as the most promising to examine. None of the heuristics infer recommendations by looking more than one link away. The heuristics are divided into two kinds, concept heuristics and document heuristics.

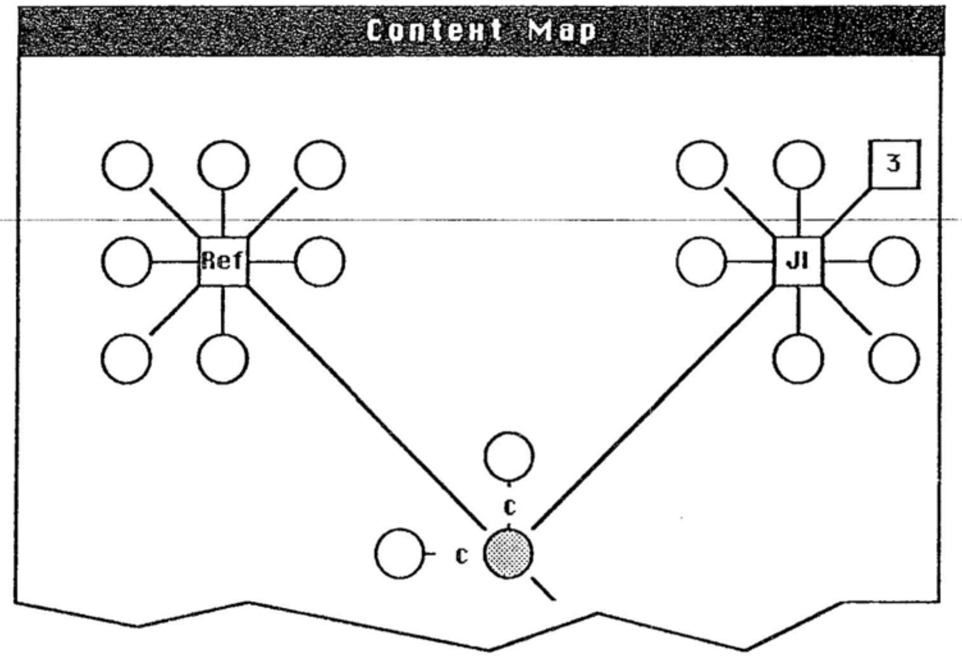


Figure 5.6: Context Map showing configuration if the Reference and the Journal Issue Nodes are expanded (Content and Window menus not shown, but are the same as a neighborhood map).

5.3.2.1 Concept Heuristics

The concept heuristics contain one underlying assumption; since documents are the fundamental units of information in the system, the user should be directed towards documents rather than other concepts. Consequently, when a concept occurs in 4 to 7 documents (depending on the user model) that have not been seen, the recommended nodes will be those documents.

In determining what concepts to show if the previous condition does not hold, the most important links between concepts are the *nearest neighbor* and the *synonym* links. In the domain knowledge there should be relatively few synonym links, since these represent a very strict view of synonymy. Similarly, for any single word concept there should be few nearest neighbors, since only at most the top five most similar ones are saved. The next most important are the *related-to* and *phrase* links. The following are the heuristics of choosing concepts.

1. For any term (single-word concept) that has been marked relevant and occurs in 4 to 7 documents (depending on the user model) that have not been viewed select those documents as the neighborhood. If this is not the case, perform the subsequent steps.

2. Retrieve concepts connected by nearest-neighbor-to links. These are concepts on the nearest neighbor list of the *current* concept, and are given a weight of 3.

3. Retrieve concepts connected by nearest-neighbor-from links. These are concepts that have the current concept on *their* nearest neighbor list, and also are given a weight of 3.

4. Retrieve concepts connected by *synonym* links. These are given a weight of 2.

5. Retrieve concepts connected by *related-to* links. These are given a weight of 1.

6. Retrieve concepts connected by *narrower* links. These are given a weight of 1.

7. Retrieve concepts connected by *phrase* links. These are given a weight of 1.

8. If there are any tied scores, order by document frequency giving higher preference to those in fewer documents.

Once the list has been determined, the BE will take the top 4 to 7 as determined by the user model and send them to the interface manager for display. The highest ranked will marked as the recommended one.

While a user is viewing a concept, he may decide to look at the documents in which it occurs. The BE will act differently depending on the frequency of the concept in the collection and the user specific values determined by the UMB. The selection of the documents to display as nodes is determined by the frequency of the concept in the document and by the date of the document; the newest is given preference. If the number of documents is less than the number of nodes specified by the user model, all of the documents are displayed as nodes connected to the concept (figure 5.7b) and the links are marked by the concept frequency. If the number of documents is more than the number specified by the UMB or if there is no room around the node, then some of the documents are displayed as individual nodes and the remaining are represented by a document list node (box) that is marked with the number of documents it represents (figure 5.7c). If the user decides to examine the documents represented by the document list node, then that node is extended and as many document nodes as possible are shown with the remaining documents again being represented by a box node (figure 5.7d). If the concept occurs in more than 13 documents and the user is a system expert, the system asks if he really wants to look at that many documents. The value of 13 is chosen because of the organization of the map. Figure 5.8 shows the number of nodes that can be displayed around a document list node and one extension of it; there are 13 nodes surrounding the two document list nodes. If any more documents are on the list a third document list node would be required. This extra node would be shown far to the right, in this case, of the original node of interest, and would lead the user away from the immediate neighborhoods of the original node. If the user does want to look at that many documents, the document list nodes are expanded as before. If the user is a system novice, he simply is not allowed to look at that many documents, and a message informing him of that fact will be displayed in the system messages window.

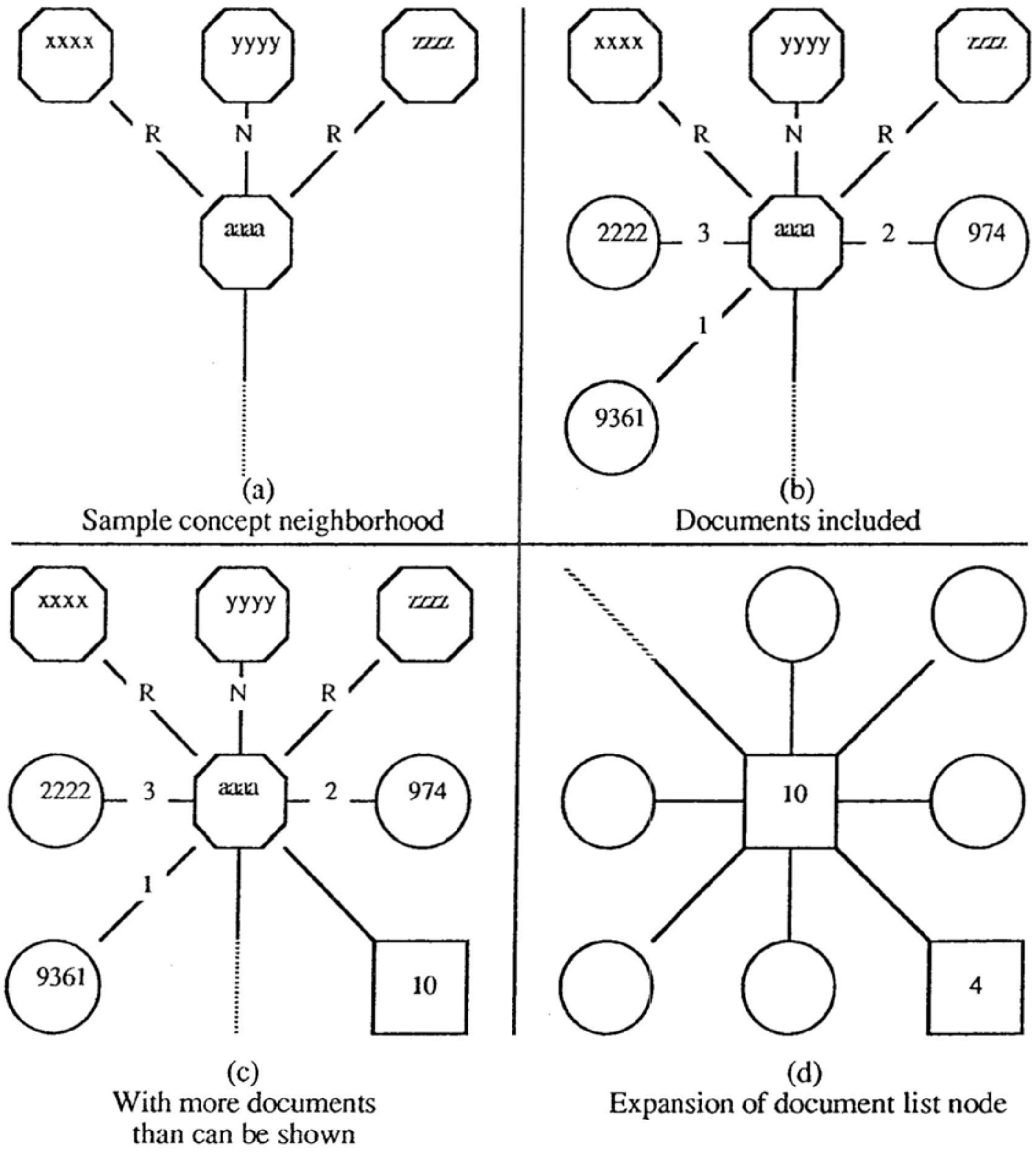

xxxx
yyyy
zzzz
R
N
R
aaaa
(a)
Sample concept neighborhood
2222
3
2
974
1
9361
(b)
Documents included
10
(c)
With more documents than can be shown
4
(d)
Expansion of document list node

Figure 5.7: Example term neighborhoods (node markings are: R = Related, N = Nearest Neighbor, <number> = occurrences of a concept in a document).

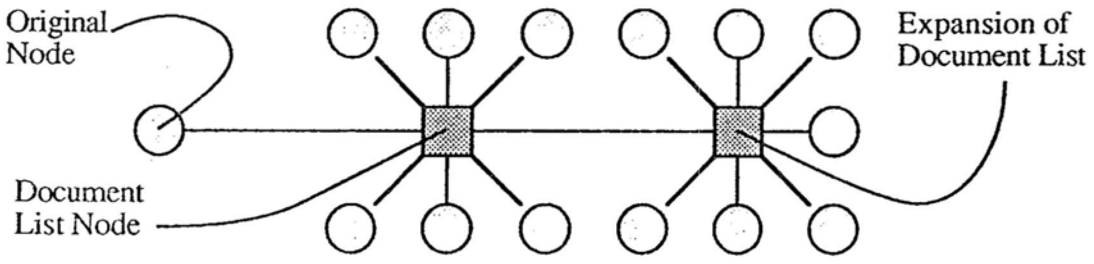


Figure 5.8: Expansion of a document list.

5.3.2.2 Document Heuristics

The document heuristics are similar to the concept ones. The most important links are the nearest neighbor links (to and from, both given a weight of 3). After this is the citation link (given a weight of 1), and then the reference link (given a weight of 1). However, since a document can potentially have many references and be cited many times, an evaluation of the documents connected by the citation links must be made. A document that is cited many times is given a higher value than one cited few times. This is based on studies of citation patterns [Salton 83] indicating that citation of a document is a relatively rare occurrence. Consequently, a document that is referenced many times is significant. The quantification of this significance for the browsing heuristics is:

1. Cited 2 to 3 times, an additional weight of 1
2. Cited 4 to 5 times, an additional weight of 2
3. Cited 6 to 9 times, an additional weight of 3
4. Cited 10+ times, an additional weight of 4.

The small numbers are due to the particular test collection being used to develop the system. It has information only about documents in the collection. A document could be cited many times, but since the citations are from other than the CACM, they are not available to the system. In a production system they would be tuned to more accurately reflect the frequency of citation in the document collection. If there are any tied scores, they are resolved by ranking on the actual number of citations.

An evaluation of a document by the size of the reference list is not so straightforward. A document with a large reference list may be good if the user is a novice in the domain of interest, since it might be assumed that the article is a survey of some field. However, a document that has only a few references may also be valuable, if it cites the current document of interest. This implies that the citing document may be closely related. If a document with few references is cited by the current document nothing can really be determined about its potential value. So, if the user is a domain novice, a document with a large (10, because of the test set) reference list is given an additional weight of 2.

In both cases, nodes that have been viewed are not included in the list of recommended nodes. It is assumed that the searcher can remember what he has examined in a particular session, and if not, then he can access them by looking at the list of documents judged relevant, the lists of search results, or by retracing his path.

The user is not limited to viewing the nodes that are displayed on the neighborhood map. When he selects a node for viewing from the map, a window containing the textual information appears. He may decide, for example, to examine the reference list of a document node. A list of titles appears, from which he can choose any one. If he chooses one that is not on the map, a node will be put up representing the document chosen. In this way, the map always maintains the path that the user has taken. Should the user care to backtrack, he can scroll a map window back over a node marked with a reminder box, select it for view, and move in a new direction.

5.3.3 Browsing Model

The model in the STM by the BE is the "path" that the user has taken through the network. Each node on this path represents a node that the user has viewed in the course of the session. And with each node the recommendations made by the BE when the node was first visited are saved. The path is connected, even though the user may have entered and left browsing several times, so while the entry and exit points may not be connected in the concept/document database, they are in the path model. This allows a user to retrace his steps through all the nodes that he has seen while browsing in an entire session. The IM provides four commands for retracing:

Last – moves the user to the last node viewed; the end of the path.

Next – moves the user to the next node on the path, if not at the end.

Previous – moves the user to the previous node on the path, if not at the beginning.

First – moves the user to the first node viewed; the beginning of the path.

New nodes are added to the path only when the user examines a new node. While this sounds obvious, it means that the path does not record *every* move that the user makes. For example, a path at some point while browsing consists of the nodes A through E, and the user has retraced his steps back to node C. From this point, the user moves to a new node, F. The path would consist of the nodes A through F and not A, B, C, D, E, D, C, F. If the user desires to trace back the exact path, he has the maps as references.

When the session is finished (not suspended), the path is saved as part of the session history, except that the recommendations are not retained. If the user comes back to the particular session and there have been no changes to the concept/document database the recommendations will be similar. The documents already seen will not be considered as candidates for recommendation during the new session. If the session is suspended, all the context information including the path and the maps is retained, so the user would see exactly the same scenes as he did previously.

5.4 Browsing Implementation

The implementation aspects of browsing that have not been discussed previously in relation to the interface manager or the implementation of the system architecture lie in the construction and maintenance of the browsing maps, and the structure of the browsing model.

Recall that the organizational principle behind the browsing maps is a square grid upon which all the nodes and other symbols are drawn. These symbols are stored in a hash table that is keyed by the grid coordinates. The reason for this is two-fold. First, it allows the interface manager to immediately determine if there is a symbol on the screen in any one of the cells by hashing on the coordinates. If there is a hit, then the cell is already taken. Second, there is no way ahead of time that the IM can know how much the user will browse and, therefore, it has no idea how much space to allocate for the grid storage. Since hash tables in Common Lisp are automatically extended when they become full, the system need not limit the user's ability to browse because of space considerations, and can allocate a small table initially.

The records that are stored in the map table have one of the following structures:

```
(Defstruct (Node
                (:Conc-Name N-)
                (:Predicate Node?)
   ;;  Icon Types: Doc, Concept, Doc-List,
   ;;  Journal-Issue
   Icon
   ;;  Values:(A)uthor, (C)oncept,
   ;;                   (D)ocument
   ;;                   (JI) - Journal Issue,
   ;;                   (R)eferences, (Ci)tations
   Value
   ;;  a pointer to the content of the node or the
   ;;  ids of a node.
   Content
   ;;  Where the node was developed from
   Predecessor
   ;;  A list of links emanating from this node
   (Successors (Make-Array '(8)))
   ;;  Has the node been visited, relevant,recommended
   Status
   ;;  Text that goes on the icon
   N-Label ; label on the neighborhood map
   C-Label ; label on the context map
   ;;  Where th node is on the maps
   Coordinates
   ;;  Whether the node has been marked as interesting
   ;;  This means a reminder box is around it
   (Marked nil))
```

```
(Defstruct (Connector
                 (:Conc-Name Connect-)
                 (:Predicate Connector?)
   Coordinates
   Source
   Destination)
```

The map table can be saved if the session is suspended, and when resumed the map can be reconstructed. When the map table is saved, the pointer to the content is replaced with the identifier of the object pointed to. For example, with a document, the document number is put in; with a document list, a list of documents is put in; with a concept, the text of the concept is put in.

Since the information that determines the contents of the browsing map is stored in the map table, the structure of the browsing model is simple. It consists of a list of coordinates of the cells that the user has been to. Any information required can be obtained by using these coordinates to key into the hash table.

5.5 Summary

Browsing is potentially an important technique for retrieving documents from large knowledge bases. Its advantages are that a user receives immediate feedback from the structure of the knowledge base and exerts complete control over the outcome of the search. The primary disadvantage is that it is easy to get lost in a complex network of nodes representing documents and concepts. Furthermore, there is no guarantee that browsing will be as effective as a more conventional search. These disadvantages can be avoided by providing facilities for controlling the browsing and for using the information derived during browsing in conventional search techniques.

CHAPTER 6

EXAMPLE SCENARIOS

6.1 Introduction

In this chapter, four example scenarios are presented to demonstrate the operation of I^3R, and to show how the system as a whole operates. Before these scenarios are presented, the difficulty of evaluating highly interactive systems like I^3R is discussed. It should be noted that I^3R is a prototype system, consequently parts of the interface are not as well developed as those of a production system would be. Furthermore, the screens shown in this chapter were composed using a drawing program to facilitate easy inclusion into the text, and are taken from screen dumps of the system while operating. The only differences between the screens seen in the text and those seen by the user are in the typefaces and in the sizes of the windows and symbols.

6.2 Evaluation

Evaluation of a highly interactive system such as I^3R presents a number of challenges. The first is a matter of retrieval effectiveness; how will the addition of the facilities provided by I^3R help increase the performance of the system. One way to answer this question is to notice that, since the system relies on retrieval techniques that have a sound theoretical and empirical basis, it will not perform any worse than those techniques. Furthermore, since Boolean query retrieval systems do not perform as well as statistically based techniques, I^3R will perform better than a Boolean system. This, however, is inadequate since it only tells us the minimum performance to be expected, does not consider the amount of effort that the user must expend to use the system, and provides no real way to compare the effectiveness of this system with others.

Major difficulties arise when considering how to evaluate a system like I^3R that has a variety of different functions that all contribute to the operation of the system. One difficulty is getting genuine needs and another is getting genuine users to interact with the system. Use of the standard test collections only provides a partial foundation for testing. In the test queries, there is no indication of the user's interest in recall or precision. One possibility for ascertaining this is to count the number of relevant documents; many relevant documents implies that the query is recall oriented, and few implies that the query is precision oriented. This, however, is a simplistic categorization. It ignores the possibility that a query may be recall oriented, but there is little information on the topic in the database, so there will not be not many relevant documents. Or, the opposite case, in which the user is looking for a specific piece of information that may be contained in many documents. One possible solution to this problem is to develop an new set of queries for one of the collections, such as the CACM collection, since it is not too large, and have the query authors indicate their interest in either an exhaustive or a specific search, as well as supplying other pertinent information This would include additional domain knowledge. A possible way to get this kind of information need description would be to extend the dialogue analysis technique used by Belkin et al. past the presearch interview stage to include retrieval with the user present,with the intermediary, helping him evaluate the results of the search.

The problem of getting genuine users is also very difficult. One possibility is to form pseudo-users, much as Oddy did in evaluating THOMAS [Oddy 1974]. The aforementioned dialogues could be analyzed to determine how users would react to specific situations arising in the course of a session. This analysis would produce rules that describe this behavior. The advantage of this approach is that the users remain constant, making the test relatively repeatable. This approach has a number of problems. First, it is not possible to determine all behaviors of the users using different facilities. Second, it is not possible to determine the reaction of the user to new facilities that were not present when the sessions were being developed.

The alternative is to test using real users; this too has many difficulties. It is impossible to give one user the same need to use on a system as novice for testing various combinations of facilities, since the user will have learned something about the problem the first time he uses the system. This will affect how he uses the system to deal with the need in subsequent sessions. This learning process, however, can be used as training for the user. In subsequent sessions, the user can take different system experience stereotypes, progressing from novice to expert, so a test subject can participate in more than a single experiment. This points out, however, the need for a great number of test subjects for experiments to test all the possible combinations of facilities that a system like I^3R has to offer. These combinations include not only the use or non-use of a particular expert, like the domain knowledge expert for example, but the use of different heuristics in implementing the expert. This situation is exacerbated by the requirement to have a statistically large enough sample to reduce the occurrence of any biases. Although this kind of testing requires a significant investment of resources (ie. perhaps thousands of test subjects), it is in the opinion of experts [Spark Jones 88], the only legitimate way to proceed. This kind of testing is far beyond the current resources available for the I^3R research.

6.3 Scenarios

With the difficulties of evaluating a highly interactive system in mind, this section of the chapter presents example scenarios to indicate how the system works. These scenarios are meant to be illustrative of the flexibility of the architecture, and show how the it assists the user to find the information he desires. The query used in scenarios one, two, and three is taken from the 52 test queries that are a part of the CACM test collection. This means that there is an established set of relevance judgements for the query. The query used in scenario four was developed by the author, who having a background in computer systems was qualified to make relevance judgements.

6.3.1 Scenario One

The first scenario demonstrates the basic operation of the system. The user is a domain and a system novice. In this scenario, much of the internal operation of the system will be shown. Each section will summarize what happens during the cycle. There are a number of occasions where the user will be thinking about what his response should be. In this case the system simply "spins." This points out a difference in the implementation of the experts from traditional rule based systems. Typically, when the system has nothing to do, (ie. no rules to fire) that signals the end of processing. In I^3R it means that the system has nothing to do for the moment, but that does not means that more information of interest to an expert is not forthcoming from the user. The system ends when it reaches the end state.

6.3.1.1 Cycle 1

The first part of every system cycle is the operation of the control expert to determine the state of the system. The initial state of the system is $Start. The CE uses rule 2 that recognizes that it is in `$Start` and changes it to $CRS (conduct retrieval session). Since this is not a leaf state, one that has an associated priority list of experts, the CE continues to operate. Since a state has been changed, the rules that are associated with state change are active. The rule selected changes the CE state form $CRS to $CU (characterize user). This is a leaf state, so a priority list for the experts is established, which consists of one expert, the user model builder, UMB. Figure 6.1 shows schematically the portion of the plan that the scheduler has traversed.

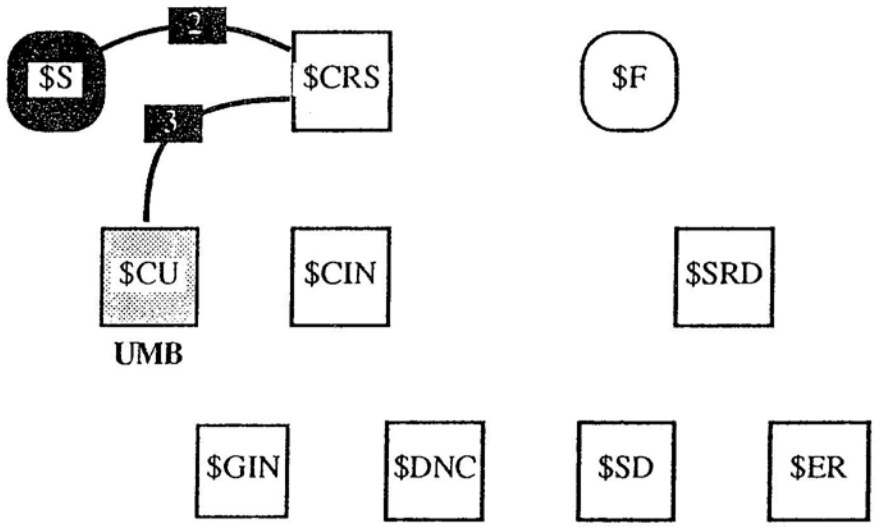
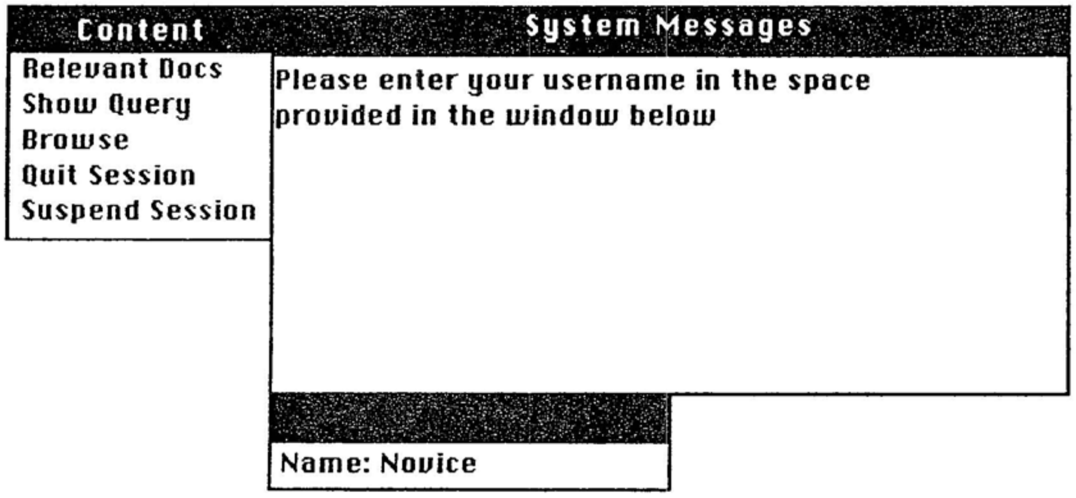


Figure 6.1: First portion accomplished of the CE plan. The transitions taken are shown in bold lines and marked with the rule that they correspond to. A state or goal that has been "satisfied" is filled with black; partially satisfied goals are filled with varying shades of gray.

The UMB recognizes that there is no user model, so it sends a message,

```
(IPM :Message-No 1 :Msg-Id UMB-User-Name :Choices 1
:Value-Type Name :Values Nil),
```

to the interface manager, IM, telling it to ask for the user's name. Figure, 6.2, shows the initial state of the interface with the user being prompted for his name.



Please enter your username in the space provided in the window below

Name: Novice

Figure 6.2: Initial state of the interface.

6.3.1.2 Cycle 2

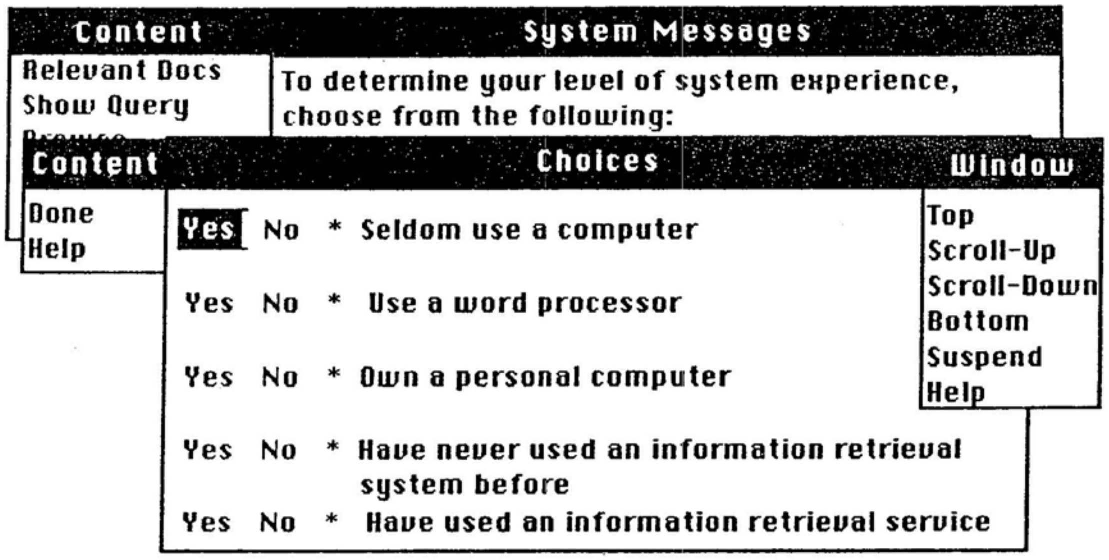
The user enters his name and keys `return`. From this action, the IM returns the message with the `:Values` field filled in with the user's name, `Novice`. The CE runs through another cycle looking for either a change of state or a change of the condition of the STM. Since neither of these has happened, at least nothing the CE is interested in, it remains in the same state, so the priority list remains unchanged.

As part of determining which rule to fire, the UMB searches the system for a file containing the user's model, which has the name `Novice.model`. Since there is no such file, it builds an empty model, which means that there are no previous session records and no user specific domain knowledge.

The UMB begins the process of asking the user questions to determine the appropriate stereotypes, by sending a message with the first group of questions. The message is

```
(IPM :Message-No 2 :Msg-Id UMB-System-Experience
:Choices Any :Value-Type Choice-List :Values (UMB-
Seldom UMB-Word-Proc UMB-Own-PComp UMB-Never-IR,
UMB-Used-IR UMB-Freq-IR))
```

Each of the values in the list following the keyword `:Values,` refers to a question that is stored in a table of questions that the IM accesses. The display derived from this message is shown in figure 6.3. In this figure, only the first five questions are shown; the rest are viewed by selecting `Scroll Down` from the window menu.



To determine your level of system experience, choose from the following:

[✓ Yes] No * Seldom use a computer

Yes No * Use a word processor

Yes No * Own a personal computer

Yes No * Have never used an information retrieval system before

Yes No * Have used an information retrieval service

Figure 6.3: System prompting the user to answer questions that will determine the appropriate stereotypes. These choices determine system expertise.

6.3.1.3 Cycle 7

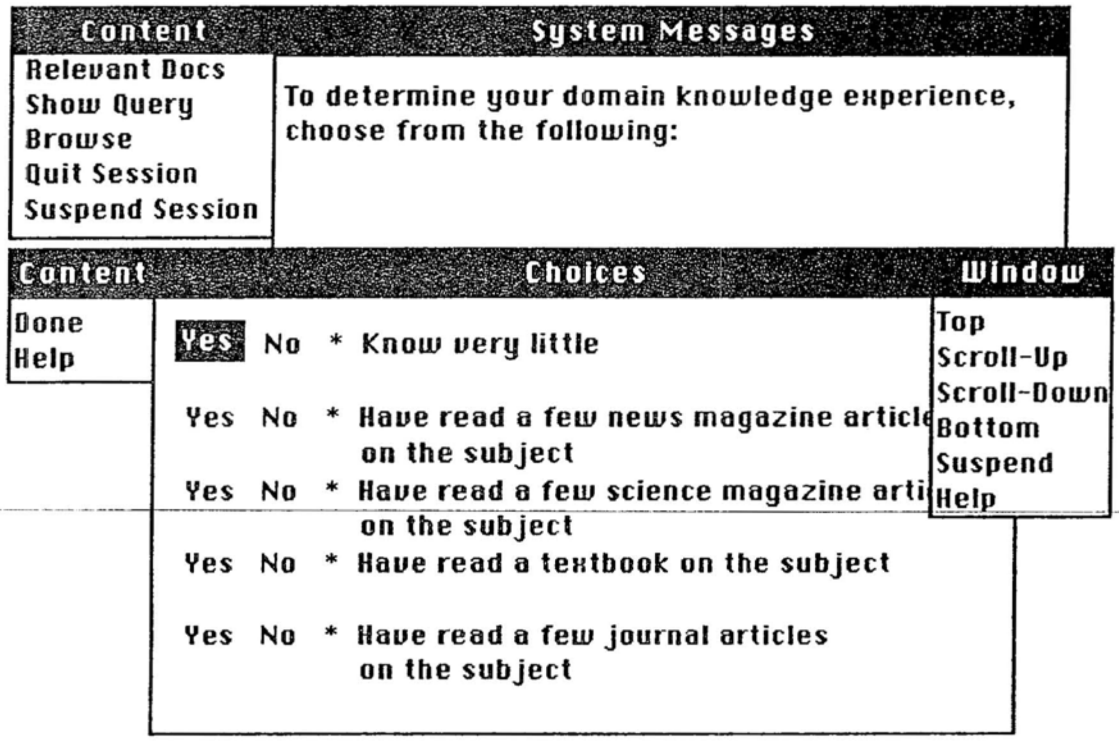
A number of cycles have passed while the user determines the choices he will select, and indicates that he is done with the question. Since the determination of the user stereotypes is not complete the CE will stay in the $CU state. Upon receiving the message back with the user's selection placed in the `:Values` field (in this case he selects only the first choice, (`UMB-Seldom` **`Yes`**), indicating that he seldom uses a computer) the UMB determines that he is a system novice, and posts this on the STM place `*User-Model*`, which then has the following list bound to it, `((System-Type Novice))`.

6.3.1.4 Cycle 8

Upon recognizing the establishment of the system-type stereotype, the UMB sends a message to the IM with choices for the user that will indicate his domain knowledge expertise. This message is:

```
(IPM :Message-No 3 :Msg-Id UMB-Domain-Experience
:Choices Any :Value-Type Choice-List :Values (UMB-
Know-Little UMB-Read-NewsMag UMB-Read-SciMag UMB-
Read-Text UMB-Read-Jrnl UMB-Write-Jrnl UMB-Write-
Text)))
```

The display derived from this message is shown in figure 6.4



Content | System Messages

Relevant Docs
Show Query
Browse
Quit Session
Suspend Session

To determine your domain knowledge experience, choose from the following:

Content | Choices | Window

Done
Help

Yes No * Know very little

Yes No * Have read a few news magazine articl on the subject

Yes No * Have read a few science magazine arti on the subject

Yes No * Have read a textbook on the subject

Yes No * Have read a few journal articles on the subject

Top
Scroll-Up
Scroll-Down
Bottom
Suspend
Help

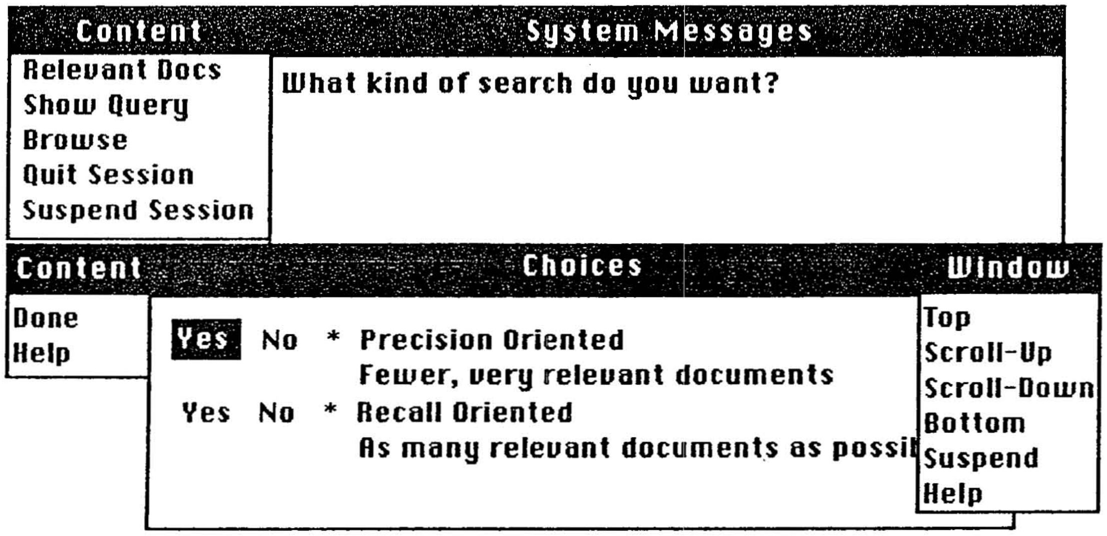
Figure 6.4: These choices determine domain knowledge expertise.

6.3.1.5 Cycle 14

The user responds by selecting `Know very little`. The symbol for this message along with the response, `(UMB-Know-Little Yes)` is placed in the `:Values` field of the message and returned. From this response the User Model Builder determines that the user is a domain novice, and adds `(Domain-Type Novice)` to the list bound to the STM place `*User-Model*` resulting in the list `((Domain-Type Novice) (System-Type Novice))`.

6.3.1.6 Cycle 15

In a similar manner as the system and domain stereotypes were determined, the UMB sends a message to the IM with two choices for the search orientation, `UMB-Precision` and `UMB-Recall`. The display for making these selections is shown in figure 6.5.



What kind of search do you want?

Yes No * Precision Oriented
Fewer, very relevant documents
Yes No * Recall Oriented
As many relevant documents as possi

Figure 6.5: These choices determine search orientation.

The user selects the **`Precision`** choice which is then transmitted back to the experts.

6.3.1.7 Cycle 24

The UMB receives the message and posts the final stereotype on the user model, so `*User-Model*` now has the following list bound to it: `((Search-Type Precision) (Domain-Type Novice) (System-Type Novice))`.

6.3.1.8 Cycle 25

The UMB now evaluates the stereotypes that it has determined apply to the user and determines other parameters that also apply. These are the expectations of the number of relevant documents, and the number of searches that it will take to find them, that the control expert uses to determine the course of the rest of the session, which in this case are 2 searches and 5 relevant documents. The stereotypes also limit the choices that the user has from which to choose in many of the interface windows, and affect the number of nodes shown on the browsing maps, which are four for a system novice and seven for a system expert. They also affect the number of documents that will be judged relevant before the system initiates another search and whether or not the system asks the user before initiating one. For a domain novice the number of documents is 3 and for a domain expert it is 5. The STM place `*User-Model*` now has the following list bound to it: `((Searches 2) (Relevant 5) (Candidates 4) (Search-Type Precision) (Domain-Type Novice) (System-Type Novice))`. The UMB also marks the user model as finished.

6.3.1.9 Cycle 26

Since the user model is now complete, the focus should shift from characterizing the user to characterizing the information need. This shift is now done by the CE. It recognizes that the UMB has marked the user model as being finished. This causes a rule to fire that takes the system from $CU back to $CRS, and marks the state $CU as finished or satisfied. The next rule recognizes this and takes the system from $CRS to $CIN, characterize information need. This state has two substates, neither of which at present is satisfied, so the next CE rule that fires takes the system to $GIN, get initial need. This state is a leaf state, and its priority list consists of one expert, the request model builder (RMB). This is shown in figure 6.6.

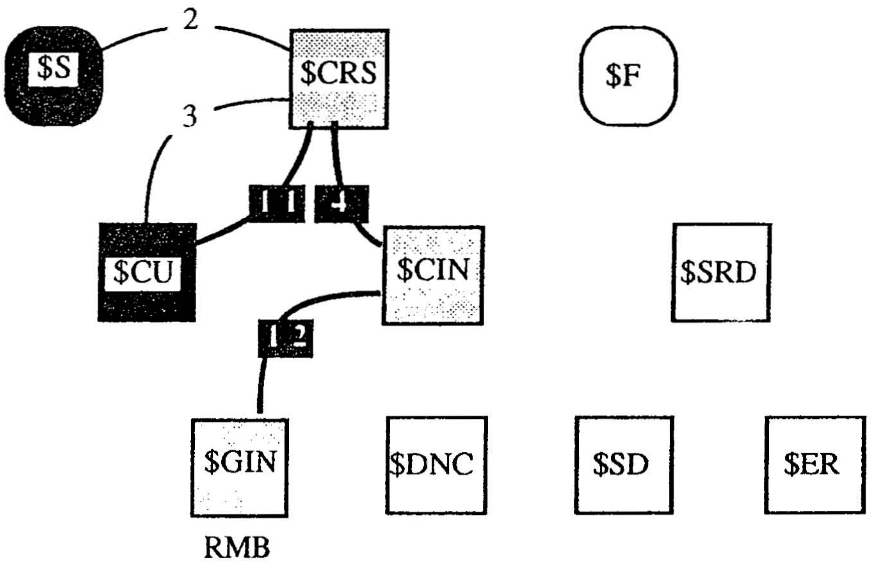
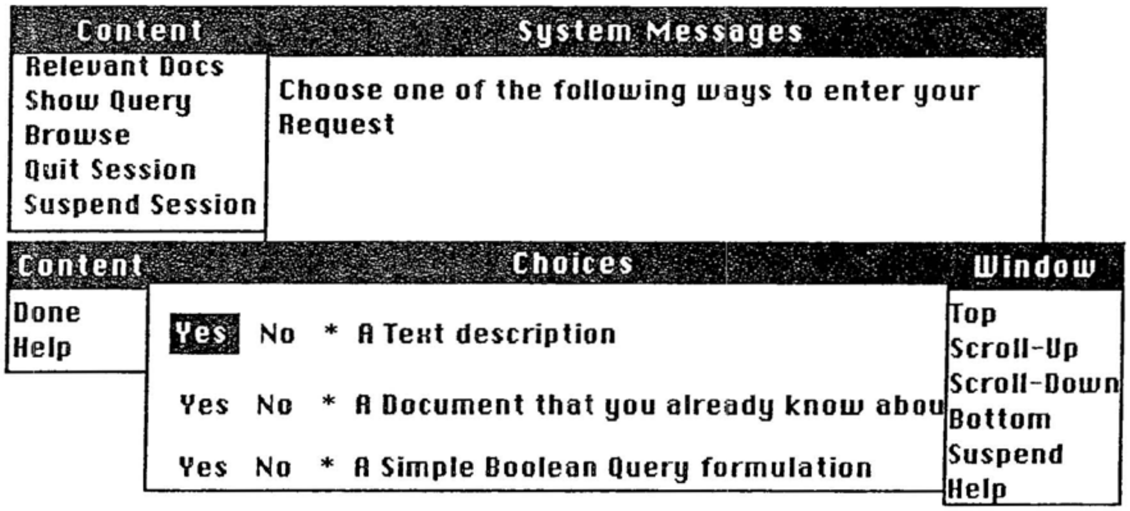


Figure 6.6: New portion of CE Plan accomplished. CE rules 4, 11, and 12 fired to move the system to the new state.

The RMB now responds to the fact that a new session record was posted without an initial need. This happens because when the user model built and installed the new session record, it had on its list of interested experts for this action the RMB, so it sent a message to the RMB. This message was not cleared during any of the previous cycles because the RMB was not active. The response is to send a message to the IM asking it to ask the user how he would like to enter his query. The actual message sent is:

```
(IPM :Message-No 5 :Msg-Id RMB-Get-Query-Entry-Form
:Choices 1 :Value-Type Choice-List :Values (RMB-
Text RMB-Document RMB-Simple-Boolean))
```

Three choices are available, and these are presented in figure 6.7. Had the user been a system expert a fourth choice, a complex Boolean query, would have been made available. In this case, the user decides to enter his query as text. One thing to note in the previous message is the value for `:Choices`. In previous messages, this field had the value `Any`, indicating that the user could choose all of the choices; in the recent message, this value is `1` indicating that the user can only choose one.



Choose one of the following ways to enter your Request

[✓ Yes] No * A Text description

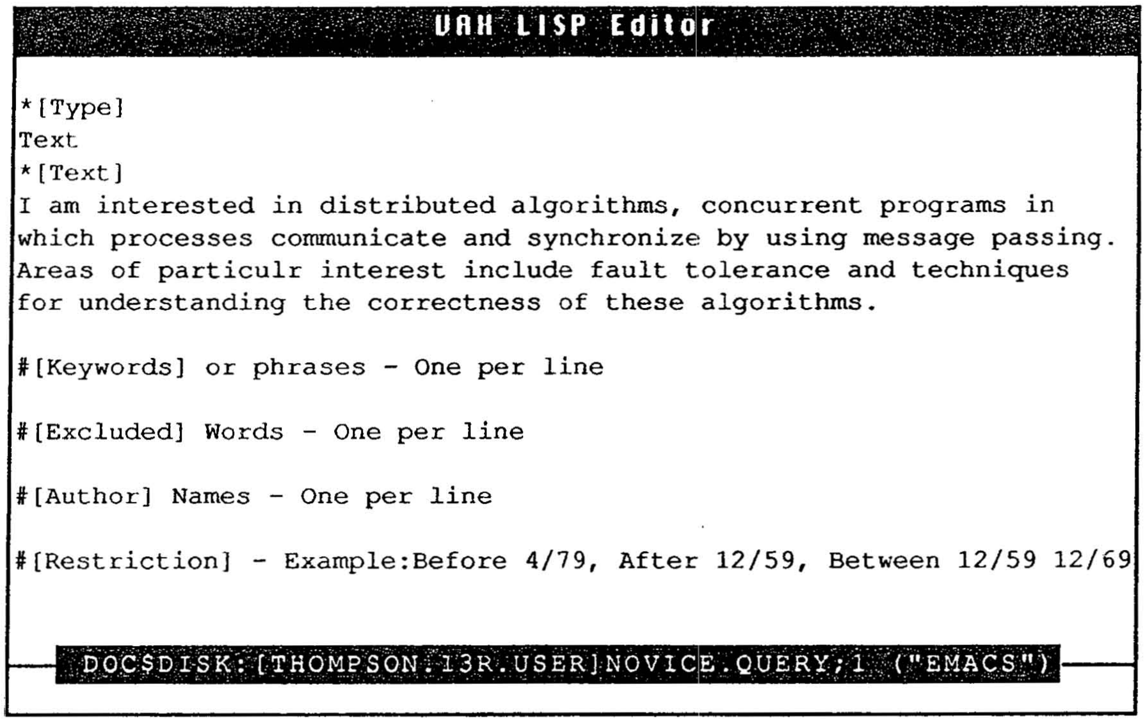
Yes No * A Document that you already know abou

Yes No * A Simple Boolean Query formulation

Figure 6.7: System asks the user for the kind of input form to initially specify his query.

6.3.1.10 Cycle 27

The RMB responds to the user's selection of input form by sending the IM a message to display that form. The IM interprets the message by generating a file with the name `NOVICE.QUERY` and copying the appropriate form into it from the file `TEXT.FORM`. The IM then transfers control to the VAX Lisp Editor with the query file as input. An editor window appears with the file consisting of an empty form, and the user types in his query. This is shown in figure 6.8. When the user is done with entering his query, he saves it back into the file and exits the editor, returning control to the interface manager. The IM sends back the message that the RMB sent to it with the `:Values` field filled with the query.



```
*[Type]
Text
*[Text]
I am interested in distributed algorithms, concurrent programs in
which processes communicate and synchronize by using message passing.
Areas of particulr interest include fault tolerance and techniques
for understanding the correctness of these algorithms.

#[Keywords] or phrases - One per line

#[Excluded] Words - One per line

#[Author] Names - One per line

#[Restriction] - Example:Before 4/79, After 12/59, Between 12/59 12/69
```

```
DOC$DISK:[THOMPSON.13R.USER]NOVICE.QUERY;1 ("EMACS")
```

Figure 6.8: The user has entered his query in a free text format.

6.3.1.11 Cycle 28

At this point, the RMB processes the query by indexing it (note this query is used as the example for indexing in chapter two), and putting on the STM place `*Term-Weights*` under the property `Representations`; any phrases that have been extracted from the query are processed and put on the property `Tuples` in the form of a list of pairs or triples of term numbers. The RMB then signals that the initial request model has been formed.

6.3.1.12 Cycle 29

The indication that the RMB gives that the initial request model has obtained from the user causes the CE to mark the state $GIN complete and move the system back to the $CIN state. Since $CIN has another state, $DNC (develop need context), that has not been completed, it goes to that state. Figure 6.9 shows the state of the CE's plan.

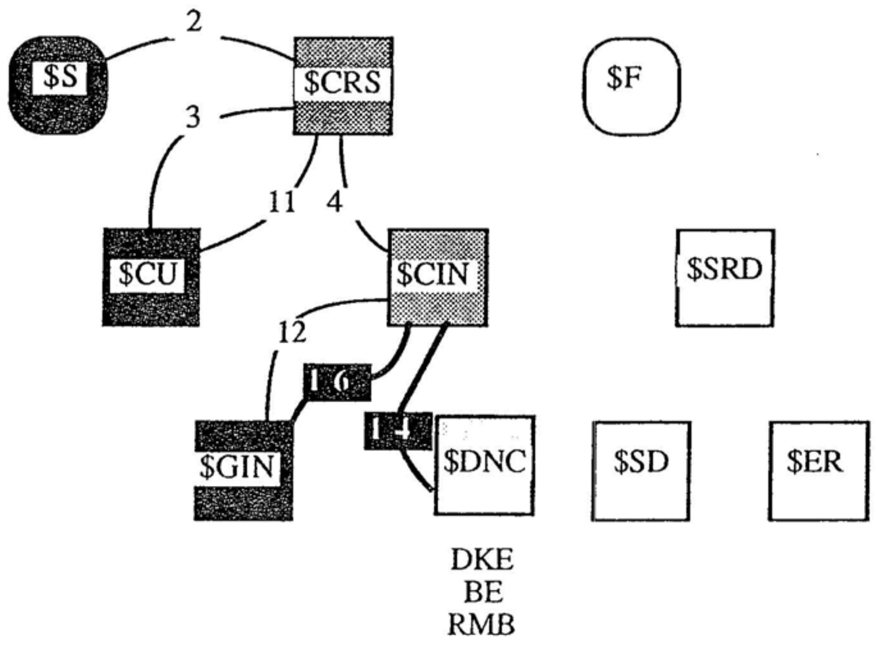
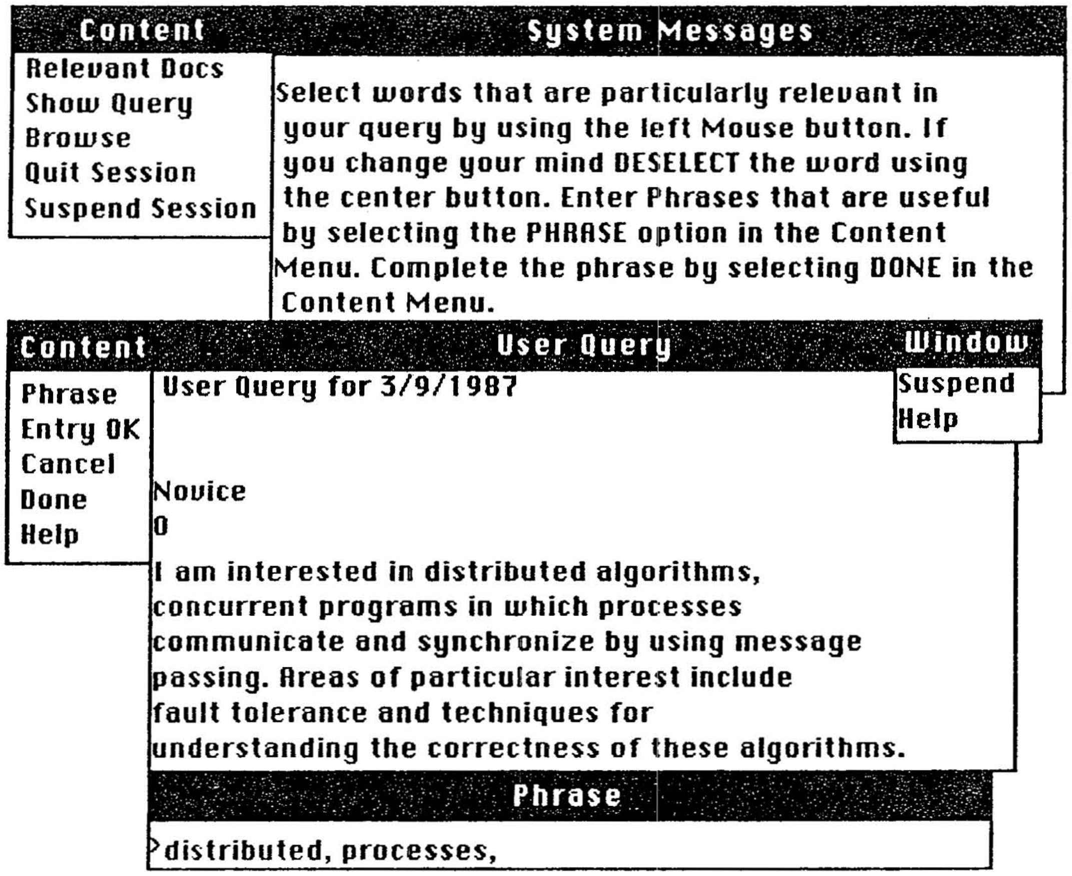


Figure 6.9: The CE's plan after CE operation in cycle 29.

This also marks the first time during the operation of the system that there is more than one expert on the priority list. The priority list has both the domain knowledge expert (DKE) and the RMB on it. The DKE will look for more concepts that might be related to those already in the request model and the RMB will take any concepts approved by the user and place them in the request model.

The first activity on the part of the DKE is to have the user pick out important phrases from the initial request. It posts a message on the *IM-TO* place that tells the IM to redisplay the query so that the user can highlight important words or phrases. This activity is show in figure 6.10.



Select words that are particularly relevant in your query by using the left Mouse button. If you change your mind DESELECT the word using the center button. Enter Phrases that are useful by selecting the PHRASE option in the Content Menu. Complete the phrase by selecting DONE in the Content Menu.

User Query for 3/9/1987

Novice
0
I am interested in distributed algorithms, concurrent programs in which processes communicate and synchronize by using message passing. Areas of particular interest include fault tolerance and techniques for understanding the correctness of these algorithms.

>distributed, processes,

Figure 6.10: User selecting phrases and important words.

This is one place where the stereotypes have limited the user's options. If the user were a system expert, he would have more relationships to choose from. In this session, the user highlights the phrases "fault tolerance," and "message passing." He also decides that "concurrent processes" and "distributed processes" should be considered phrases by selecting their component words.

The RMB does not post a rule on the agenda for execution this cycle.

6.3.1.13 Cycle 140

A number of cycles have passed while the user has been selecting phrases from the query display. These phrases are stored in the record that represents the query in the interface manager. They are stored until the user selects `Done` from the content menu and then they are put in the `:values` field of the message record and returned. The evaluation portion of the message is:

```
(((0 nil) (correctness (Phrase fault tolerance 1.0)
synchronize (Phrase message passing 1.0) (Phrase
concurrent processes 1.0) (Phrase distributed pro-
cesses 1.0)))
```

This format is the same as that used when the user evaluates the result of a search. The first item, `(0 nil)`, is the evaluation of the query, and is a result of using the structures for the evaluation of a document to get domain knowledge from the query. This query is considered a document with the number zero. The next part of the list consists of important words and domain knowledge connections, in this case only phrases. The value of 1.0 that is associated with the phrases is a remnant of the initial implementation of the domain code, when consideration was given to associate strengths with the connections in a manner similar to RUBRIC [Tong 83]. They are not used in the current system.

The DKE will take the content of the message, extract the phrases and put them in the user's domain knowledge. In doing this, it checks to see if any of the words are already in the request model, so it can avoid stemming the words and making access to the LTM to get information such as the term number.

The RMB will take the list and convert the phrases into tuples. It does this by retrieving the term numbers from the domain knowledge models.

6.3.1.14 Cycle 141

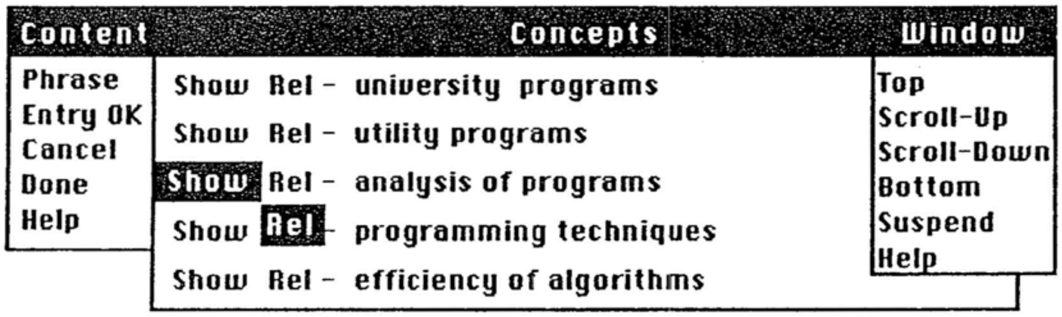
At this point in the session the DKE will be examining the global domain knowledge to find concepts that are related to the concepts in the request model that have not been checked already. Since this is the initial phase of domain knowledge search, none of them have been, but as they are used, they are marked as having been checked. The DKE looks for synonyms that are related, and finds none.

6.3.1.15 Cycle 142

The DKE looks for phrases that contain words that are in the request model. It finds six and posts a message to the IM with those six for evaluation by the user. They are shown in figure 6.11. The RMB has no activity at this time.

6.3.1.16 Cycle 209

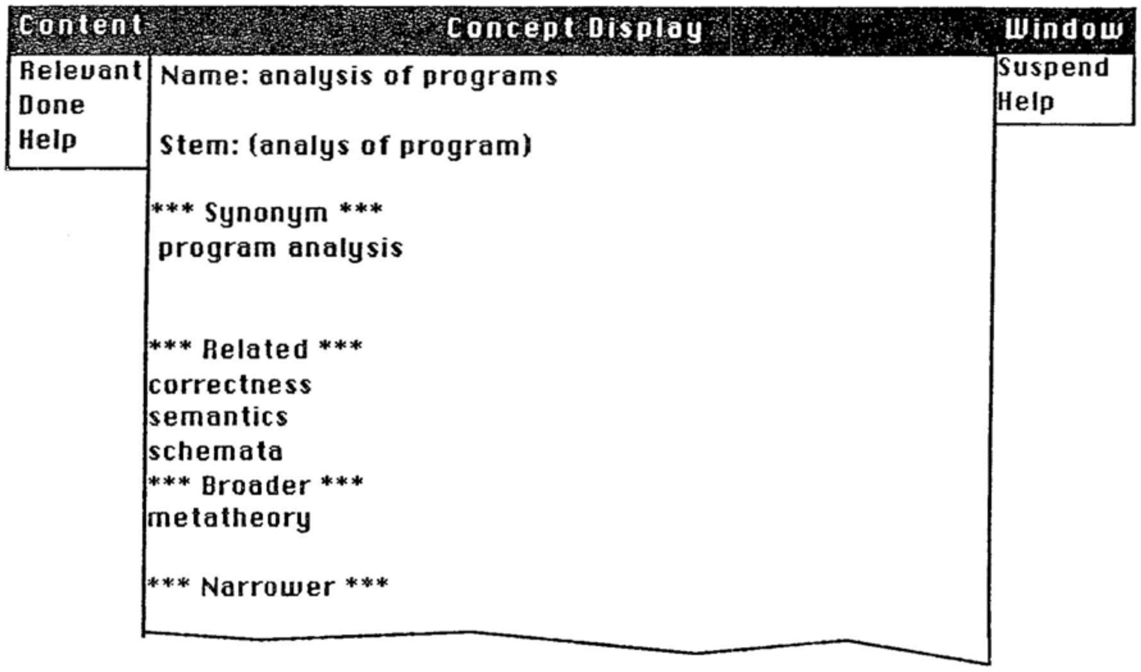
A large number of cycles pass as the user evaluates the phrases presented to him by the interface manager. The user selects `programing techniques` as relevant to his need.



Show Rel - university programs
Show Rel - utility programs
Show Rel - analysis of programs
Show Rel - programming techniques
Show Rel - efficiency of algorithms

Figure 6.11: Concepts presented for user evaluation.

The interface manager in response to the user selecting `Show` displays the information about the concept `analysis of programs`, which is shown in figure 6.12.



Name: analysis of programs

Stem: (analys of program)

*** Synonym ***
program analysis

*** Related ***
correctness
semantics
schemata
*** Broader ***
metatheory

*** Narrower ***

Figure 6.12: Information about `analysis of programs`.

The user decides that `analysis of programs` and `programming techniques` are relevant to his information need. These are transmitted back to the experts when he selects `Done` from the content menu.

The DKE takes the phrases that have been generated by the user and puts them into the user's domain knowledge model.

The RMB takes the phrases that have been selected by the user and puts the component words into the request model and also converts them to tuples, adding them to the tuple list.

6.3.1.17 Cycles 210 – 216

In these next few cycles the DKE looks for domain knowledge using the phrase, synonym, related, broader, and narrower links. In this case, no more additional concepts are found. Since no new information from the user has been obtained, the RMB performs no actions.

6.3.1.18 Cycle 217

Since the search for additional domain knowledge has been completed the focus of the system now changes. The state $DNC is marked complete as well as $CIN. The CE goes back to $CRS, which still has one subgoal that has not been been completed, $SRD (search for relevant documents). This state also has two substates, $SD (search for documents) and $ER (evaluate results). The CE puts the system into state $SD, which has only the search controller on its priority list. The transitions are show in figure 6.13

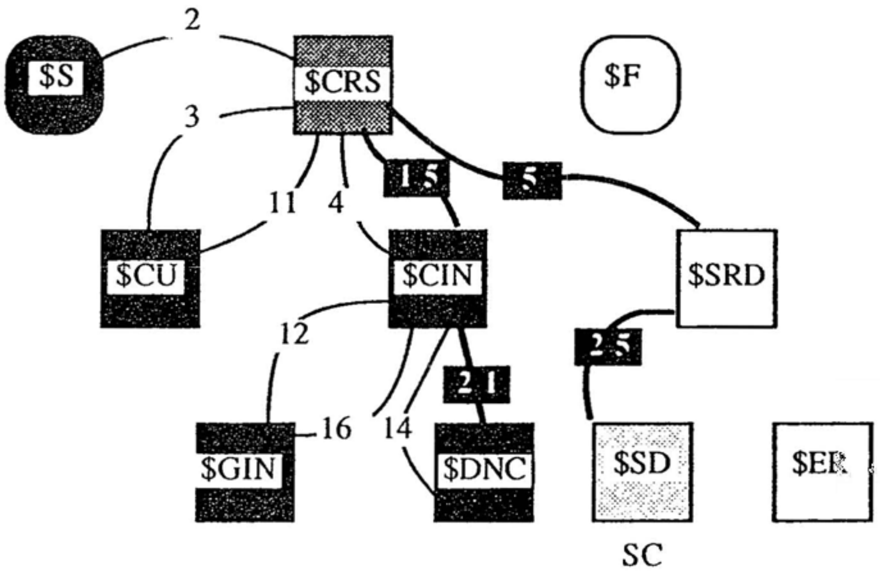
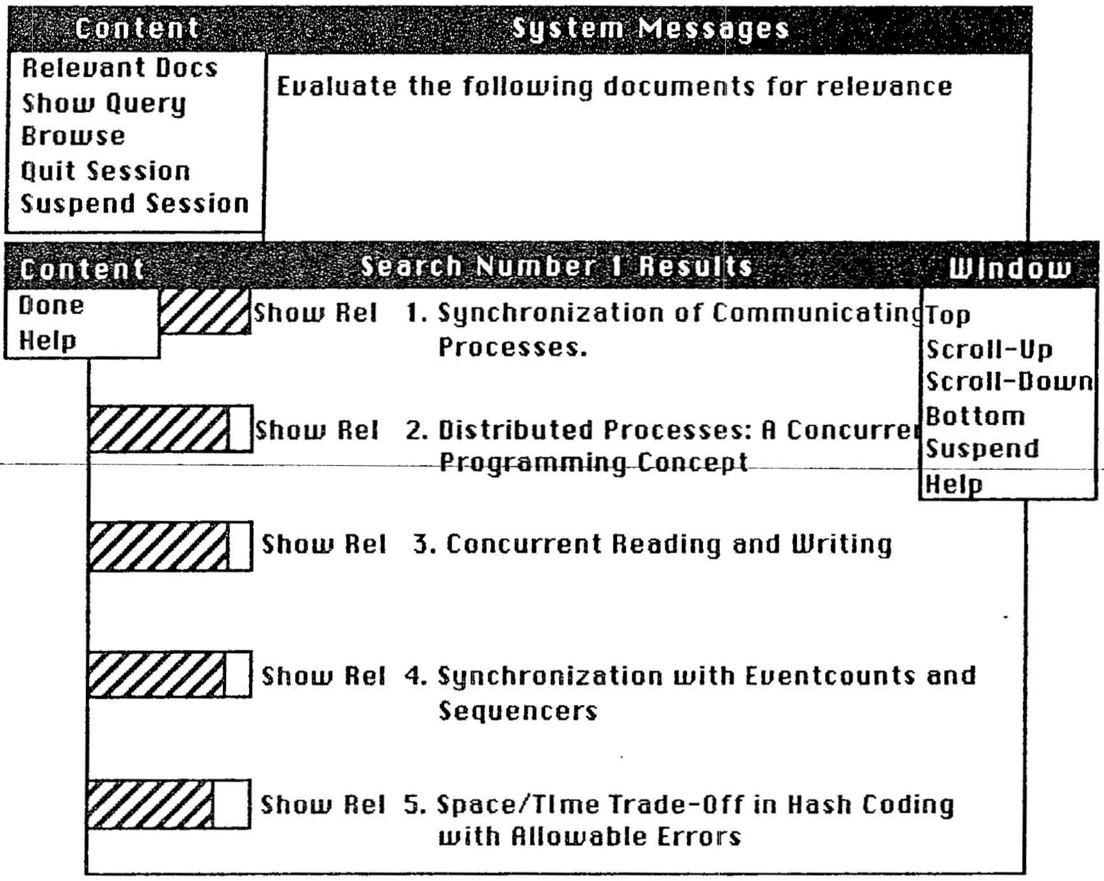


Figure 6.13: CE moves from $DNC to $SD using control expert rules 21, 15, 5 and 25, making the search controller active.

The SC now invokes a search. From the information that the user is precision oriented, the SC chooses to use an initial probabilistic search, which means the term weights are computed using the inverse document frequency weight. In all of the searches, a correction factor is added to increase the score of documents that have terms that are dependent, which are derived from the phrases that the user generates.

The SC generates a file with the search parameters that consist of the kind of search, the number of relevant documents found so far, the term numbers and their collection frequency, the tuples generated from the phrases, and any documents that have been seen by the user. In the case of this search, the number of relevant documents is zero, and no documents have been seen as yet. When the search is completed, the results are sent to the interface manager for display. The initial display is shown in figure 6.14.



Evaluate the following documents for relevance

1. Synchronization of Communicating Processes.
2. Distributed Processes: A Concurre Programming Concept
3. Concurrent Reading and Writing
4. Synchronization with Eventcounts and Sequencers
5. Space/TIme Trade-Off in Hash Coding with Allowable Errors

Figure 6.14: Top five documents of initial search.

The bar to the left indicates the relative relevance of the documents. The first document is the basis of the measurement. The length of the bar is simply <document score>/<first document score>. To see the other documents retrieved, the user selects one of the scroll options from the `window` menu on the right. The rest of the results are:

```
6.  On Computer Enumeration of Finite Topologies
7.  Proving Monitors
```

8. Proving the Correctness of Heuristically Optimized Code

9. On Multiprogramming, Machine Coding, and Computer Organizations

10. The Next 700 Programming Languages

11. An Information Algebra - Phase I Report-Language

12. Programming Systems and Languages 1965-1975

13. Logic and Programming Languages

14. Distributed Packet Switching for Local Computer Networks

15. Secure Communication over Insecure Channels

16. A Computer Analysis Method For Thermal Diffusion

17. Verifying Properties of Parallel Programs: An Axiomatic Approach

18. An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processes

19. The Expanding World of Computers

20. Exclusive Simulation of Activity in Digital Networks

6.3.1.19 Cycle 218

With the result of the first search done, the CE marks the state $SD as satisfied and moves back to $SRD. Since $SRD has one substate still not satisfied, $ER, it moves the system to that state. The priority list for $ER is BE, RMB, DKE. The browsing expert is first on the priority list since it is given priority to interact with the user. The RMB and DKE are in a passive role; accepting and interpreting input from the user. Figure 6.15 shows the moves of the control expert.

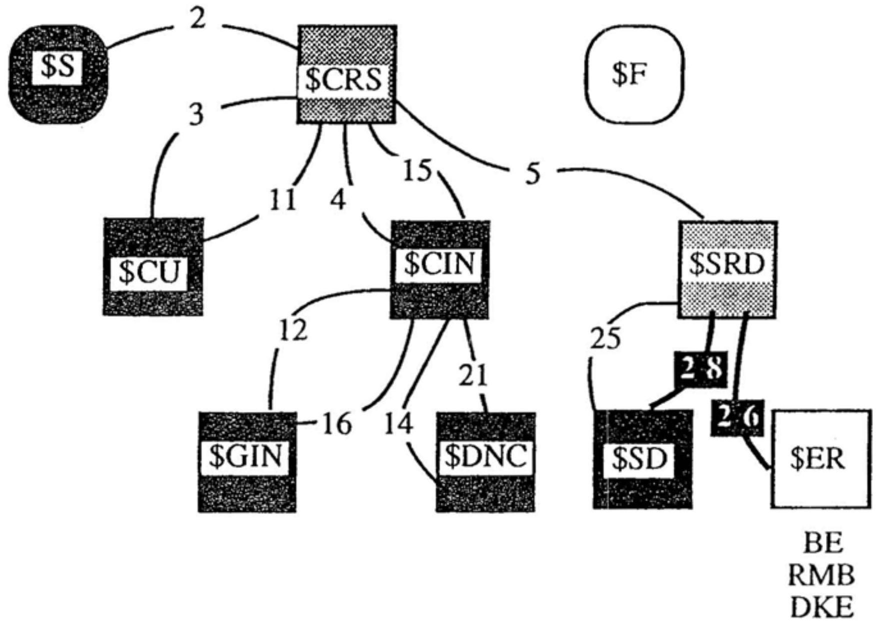
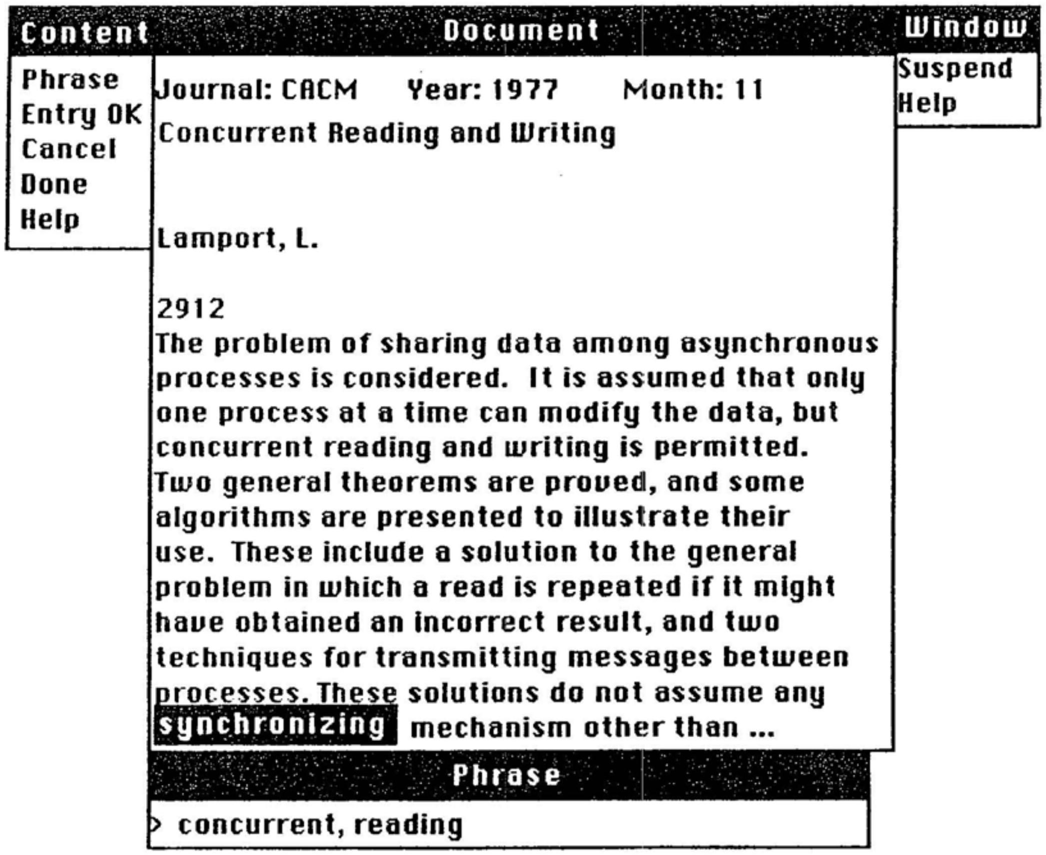


Figure 6.15: The control expert moves the system to state $ER, evaluate results, for evaluation of the search results.

The only activity by the experts during this cycle is the RMB taking the documents and putting them on the `*Doc-Evals*` STM place.

6.3.1.20 Cycles 220 – 785

The primary activity during these cycles of the system is the user evaluating the search results. The same domain knowledge entry options are available to the user that were shown when he was asked to highlight important information in the query. This information is also entered into the user's domain knowledge model and the request model. It is very important at this stage that the user select important words and phrases in the documents that he views. These words are the basis of the relevance feedback process, since these are the only words that are added to the request model. The system does not take all of the words in the relevant documents and add them to the request model. If the user fails to select additional words from the documents that he determines to be relevant, the system will redisplay the search results and prompt him to select some words. Figure 6.16 shows one of the user's judgements.



Journal: CACM Year: 1977 Month: 11
Concurrent Reading and Writing

Lamport, L.

2912
The problem of sharing data among asynchronous processes is considered. It is assumed that only one process at a time can modify the data, but concurrent reading and writing is permitted. Two general theorems are proved, and some algorithms are presented to illustrate their use. These include a solution to the general problem in which a read is repeated if it might have obtained an incorrect result, and two techniques for transmitting messages between processes. These solutions do not assume any synchronizing mechanism other than ...

> concurrent, reading

Figure 6.16: User makes relevance judgements of documents terms and phrases in the retrieved documents.

In this search, the user selects four documents as relevant, the first three shown in figure 6.16 and the 17th document on the list. The important words and phrases are deadlock, receivers, senders, synchronization, concurrent writing, concurrent reading, and asynchronous process.

6.3.1.21 Cycle 786

After the user has indicated that he is done with evaluating the search results by selecting `Done` from the content menu, the interface manager sends back the search result message with the user's evaluations.

The RMB adds the selected terms and words that compose the phrases that are new to the request model. The phrases are be converted into tuples.

The DKE puts the phrases, if they are new, into the user's domain knowledge.

6.3.1.22 Cycle 787

The user has indicated that only four of the twenty documents are relevant. This fails to meet the expectation that the user will find five relevant documents, so the CE will, using an exception transition, shift the system back to the state $SD to search for additional documents. Figure 6.17 shows the change in state.

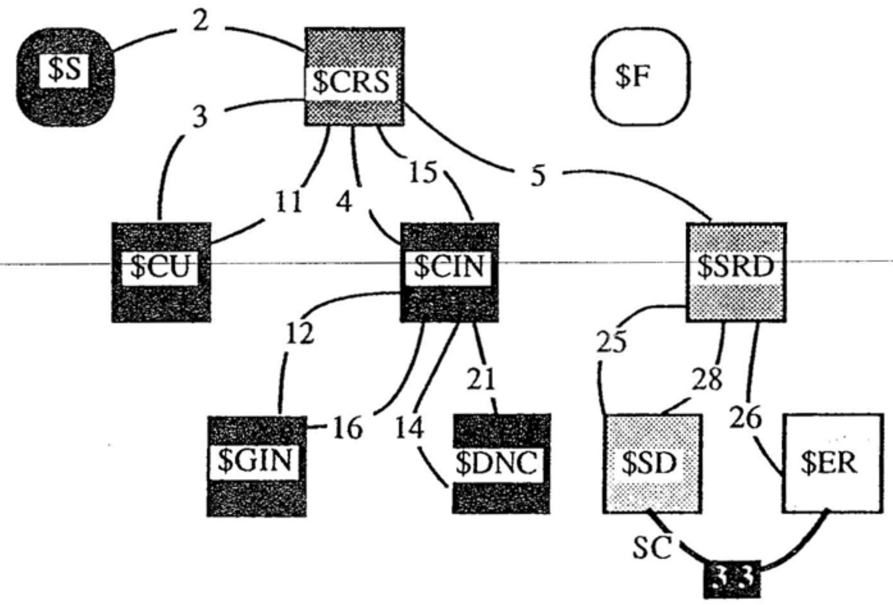


Figure 6.17: The exception transition back to $SD to enable the search controller.

The first thing that the SC does is to evaluate the results of the previous search in order to have a basis for selecting the next search. The first search had a precision of 20 percent (4 out of 20 documents).

6.3.1.23 Cycle 788

The SC now fires of the second search. Since the previous search had a precision level greater that 15 percent the SC chooses to use the same search method. However, the request model has been refined by the addition of new terms and phrases and the search method can use the full probabilistic weight (equation 2.4 in chapter two) rather than the idf estimate (log(N) - log(n) +1). The results of this search are the following documents.

1. The Executive System Implemented as a Finite-State Automation
2. Three Criteria for Designing Computing Systems to Facilitate Debugging
3. A Large Semaphore Based Operating System
4. An Alternative to Event Queues for Synchronization in Monitors
5. Formal Verification of Parallel Programs
6. A language for Formal Problem Specification
7. Synchronizing Processors with Memory-Content-Generated Interrupts.
8. Synchronization in a Parallel-Accessed Data Base
9. Concurrent Control with "Readers" and "Writers"
10. Signature Simulation and Certain Cryptographic Codes.
11. The design of the Venus Operating System
12. A New Solution of Dijkstra's Concurrent Programming Problem
13. Productivity of Multiprogrammed Computers - Progress in Developing an Analytic Prediction Method
14. Internal and Tape Sorting Using the Replacement-Selection Technique
15. Comments on Prevention of System Deadlocks
16. A Modular Computer Sharing System
17. Monitors: An Operating System Structuring Concept
18. PUFFT-The Purdue University Fast FORTRAN Translator
19. On-the-Fly Garbage Collection: An exercise in Cooperation
20. A Note on Data Base Deadlocks

6.3.1.24 Cycle 789

Since the SC has performed its second search the CE moves the system to the $ER state; figure 6.18 shows this.

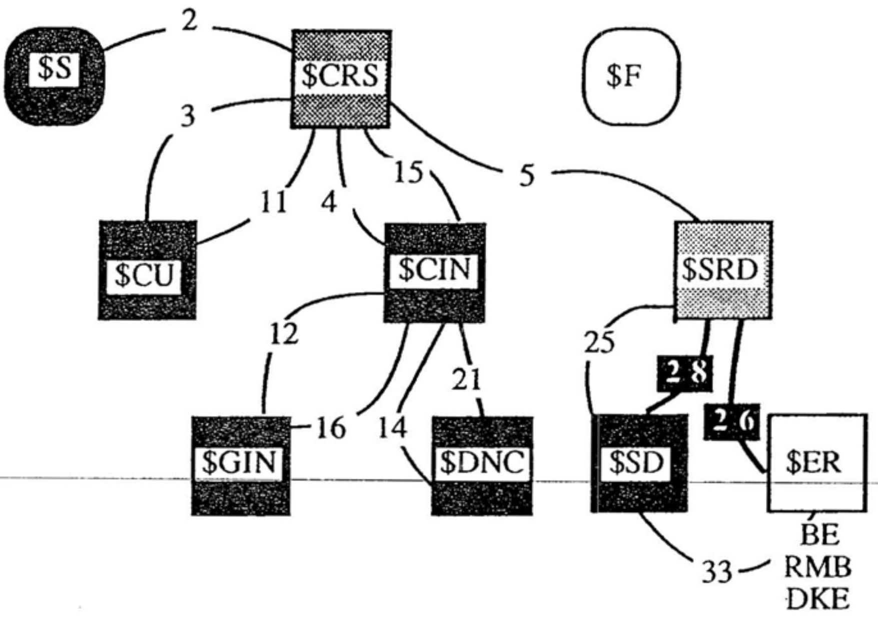


Figure 6.18: CE moving back to $ER

The RMB takes the documents retrieved and puts them into the document evaluations.

6.3.1.25 Cycles 790 – 1448

During this time the user is evaluating the search results. The user finds two more documents, 5, and 6, that he feels are relevant. From these two documents, the user selects two phrases, parallel programs and communicating parallel processes as important concepts.

6.3.1.26 Cycle 1449

As before, in cycle 786, the RMB puts the component words of the phrases and any selected terms into the request model, and the DKE puts any new phrases into the user's domain knowledge.

6.3.1.27 Cycle 1450

At this point, the number of relevant documents is six which fulfills the expectation that the control expert has for finding five documents in two searches. The CE marks $ER as satisfied and moves to $SRD. Now, this state has both of its substates satisfied so the CE marks it as satisfied too, and moves back up to $CRS which also is recognized to be complete. The CE then moves the system the $Finish state, shown in figure 6.19. The system then tells the user that it has found enough documents, and that he may continue the session at a later date. The system writes out the user model with the new session record and the session comes to an end.

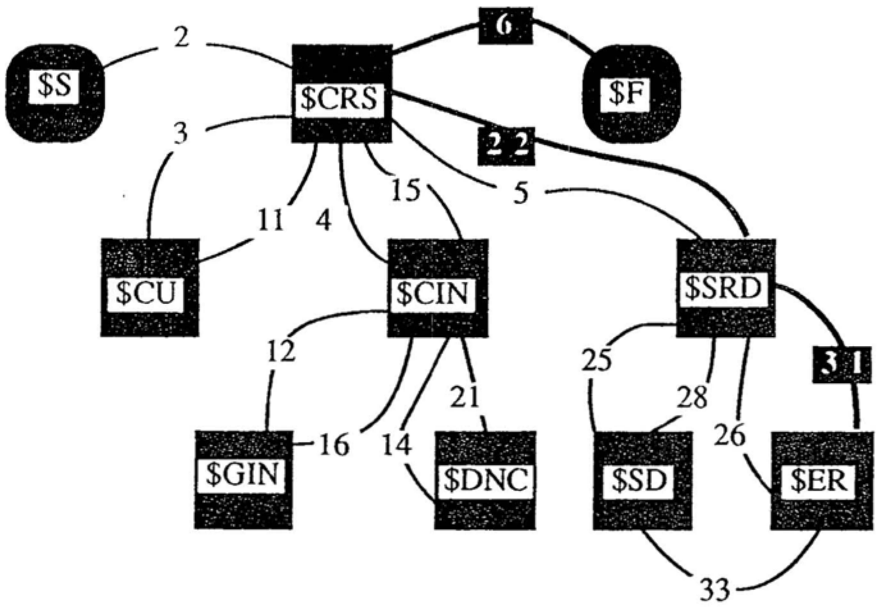


Figure 6.19: The CE moves the system to the $Finish state.

There are a number of behaviors not observed in this first scenario. If, after the two searches, the user had failed to find the expected number of relevant documents, the CE would have recognized the need to do more work in developing the query. To accomplish this, the CE would have caused the states $CIN and $DNC to be unsatisfied, and then the system would move back to the $DNC state where the DKE could begin to look for additional domain knowledge. It would use the new concepts added by the user in evaluating the search results as starting points for new spreading activations in the domain knowledge.

This kind of behavior differs from simple backtracking. Simple backtracking can be characterized in the following way. Consider the problem of the Knight's tour, where the object is for the knight chess piece to visit every square on a chessboard. The piece moves until it cannot move further. If it has not visited every square, it backs up a move to try an alternative move. If every move has been exhausted, it backs up two moves and tries alternatives and so forth until it finds a path through every square. The CE returns to a previous state, but does not retract any information other than the fact that a state has been satisfied; it does not throw out the domain knowledge collected from the user the first time it was in the $DNC state. It is a recognition that it does not have enough information.

6.3.2 Scenario Two

Scenario two is variation on scenario one; its purpose is to show the behavior of the system when another one of the exception transitions is taken. In figure 4.13 there is a transition from $SRD to $CIN, which is taken when the system has made the expected number of searches, but has not found the expected number of relevant documents. In order to show the system taking this transition, the selection of relevant documents in the search results is changed slightly. In the first search, only two documents, numbers one and three are evaluated as being relevant. The same phrases and important words are selected from each document. The results of the second search are slightly different, mostly the order of the documents is changed. In search two, only document five is chosen to be relevant.

The effect of this is to lower the precision of the searches to 0.1 and 0.05 respectively for searches one and two. This causes the SC to evaluate them as marginally effective rather than as complete failures. If a search is a complete failure, the SC will try a different search technique the next time. So, for example, if the first search was a probabilistic one and a failure, the next one would be a cluster search. In this case, where two searches in a row were only marginally effective, the SC will choose a cluster search for its third search.

The CE uses rule 30 to go back to the $SRD state which recognizes that the expectation of the number of searches has been met. Once in state $SRD, it recognizes that the expectation of the number of searches has been met, but the expectation on the number of relevant documents has not been met. This causes the CE to first raise the search expectation by one, then to unsatisfy the states $DNC and $CIN, and finally to put the system into the $CIN state. Once in the $CIN state, the CE recognizes that the $DNC state is not satisfied, so it moves the system to the $DNC state, where it can again search for any potentially relevant domain knowledge. This CE activity is summarized in figure 6.20.

Once in the state where it can interact with the user, the DKE begins to look for new domain knowledge and informs the user of what it is happening by the message shown in figure 6.21.

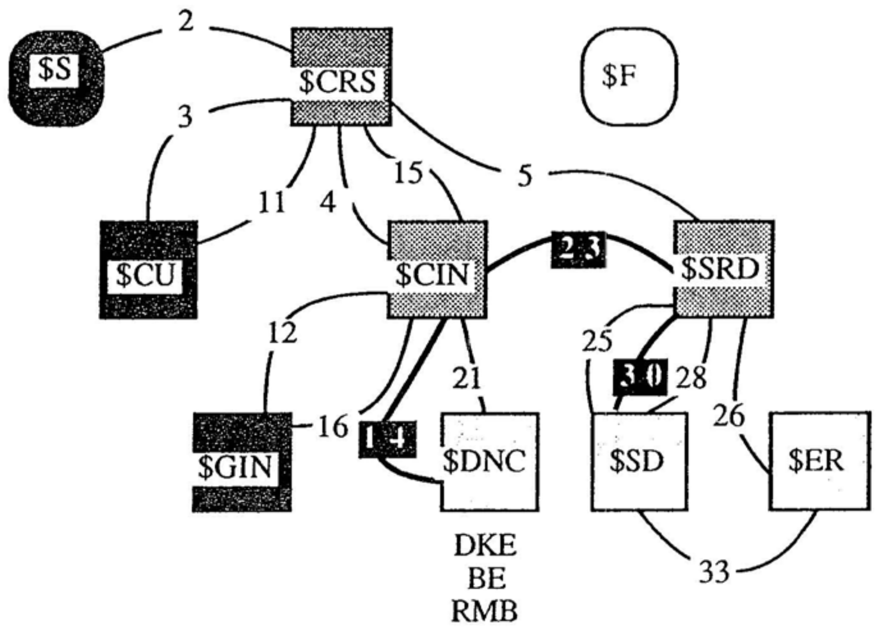
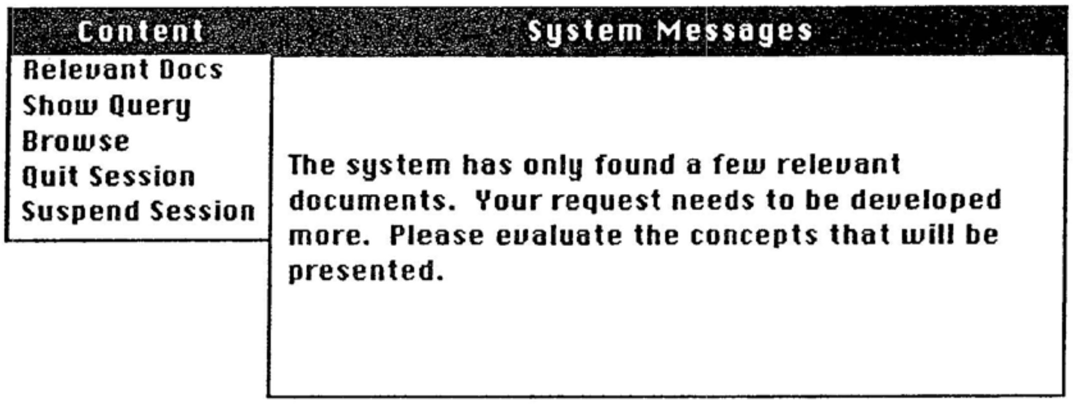


Figure 6.20: CE moving back to the state $DNC to allow the DKE to search the domain knowledge for other concepts.



The system has only found a few relevant documents. Your request needs to be developed more. Please evaluate the concepts that will be presented.

Figure 6.21: Message advising the user on the next activity.

The DKE now searches the domain knowledge using the words that the user has added to the request model from the document that have been evaluated previously. The added terms are: “deadlock,” “reading,” “writing,” ”receivers,” “senders,” “synchronization,” and “parallel.” This search produces the following new concepts: “process management,” “deadlock avoidance,” “input output,” “parallel algorithms,” “parallel processors,” and “parallel rewriting systems.” The only one of interest is “parallel algorithms.”

Once these are presented to the user and evaluated the CE will put the system back into the $SD state, as shown by figure 6.22.

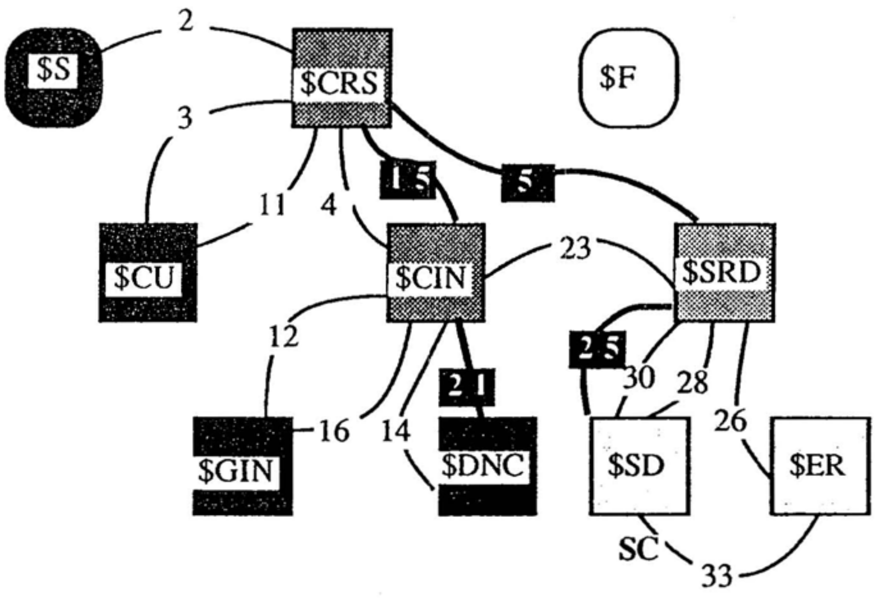


Figure 6.22: CE move system state to $SD to allow the SC to make another search.

The results of the search are the following documents:

```
1.  A General-Purpose Display Processing and Tutorial
    System

2.  A Model for a Multifunctional Teaching System

3.  Read-Backward Polyphase Sorting

4.  A comparison Between the Polyphase and Oscillating
    Sort Techniques

5.  The ALCOR Illinois 7090/7094 Post Mortem Dump

6.  Recursive Solution of a Class of Combinatorial Prob-
    lems: An Example

7.  Polyphase Sorting with Overlapping Rewind

8.  Sorting on a Mesh-Connected Parallel Computer

9.  Merging with Parallel Processors

10. Mechanization of Tedious Algebra: The Newcomb Opera-
    tors of Planetary Theory
```

11. Mechanization of Tedious Algebra: The Coefficients of Theoretical Chemistry

12. Computing Connected Components on Parallel Computers

13. Fast Parallel Sorting Algorithms

14. A Policy driven Scheduler for a Time-Sharing System

15. A Practical Approach to Managing Resources and Avoiding Deadlocks

16. Dynamic Computation of Derivatives

17. A Simple Automatic Derivative Evaluation Program

18. Thoth, A Portable Real-Time Operating System

19. A Multiprogramming Monitor for Small Machines

20. A Language for Describing the Functions of Synchronous Systems

21. SIMULA - An Algol-based Simulation Language

None of the documents are, in the opinion of the user, particularly relevant to his query, so he decides to review the results of the previous searches. This can be done by selecting the partial search results windows, which are generated when the user signifies that he is done in the content menu. Instead of disappearing like the rest of the windows, they are redrawn as shown in figure 6.23. By placing the pointer on the single line and clicking the left mouse button, the window reappears as it was originally shown (figure 6.9).

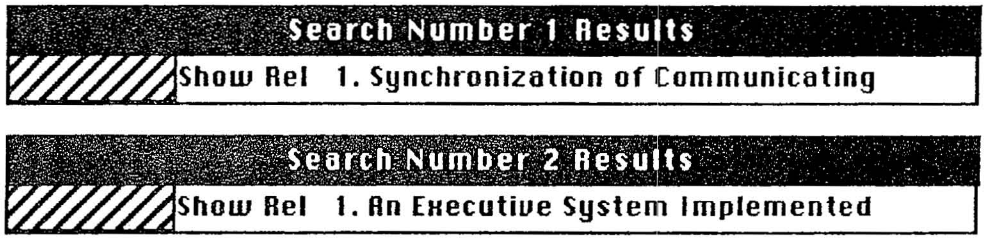

Search Number 1 Results
Show Rel
1. Synchronization of Communicating
Search Number 2 Results
Show Rel
1. An Executive System Implemented

Figure 6.23: Search results windows after the user is done with the second search.

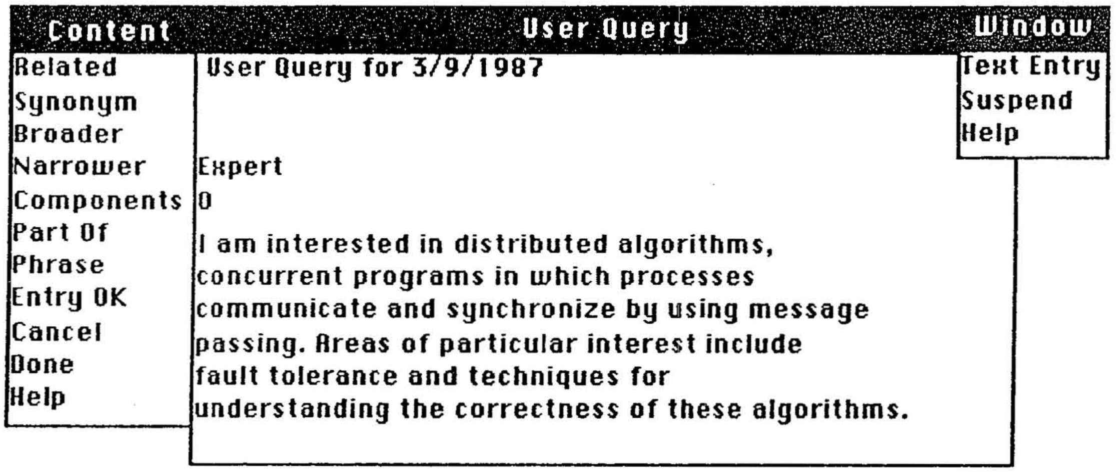
The user decides that several more of the documents from search one and search two are relevant. In so doing, the expectation on the number of relevant documents is met, as well as the expectation on the number of searches. This will cause the CE to recognize that the session is over and will put the system into the $Finish state as before (figure 6.19).

6.3.3 Scenario Three

The third scenario demonstrates the extra choices available when the user is a system and domain expert. In order to simulate expertise the user's domain knowledge has been expanded in the area of operating systems. This scenario emphasizes the differences in the operation from the basic novice operation, so many of the details presented in the previous scenario are omitted, particularly the cycle by cycle operation of the system.

After the user has entered his name, the system, as before, asks him questions about the nature of his experience in the domain, computer and IR systems, and the type of search. In this case, he is familiar with the domain, has used an IR service before, and is interested in a recall oriented search. This sets up the control expert's expectations, which are 20 relevant documents in 2 searches.

After entering his query (the same one as in the first scenario), the system presents it to him for elaboration (figure 6.24).



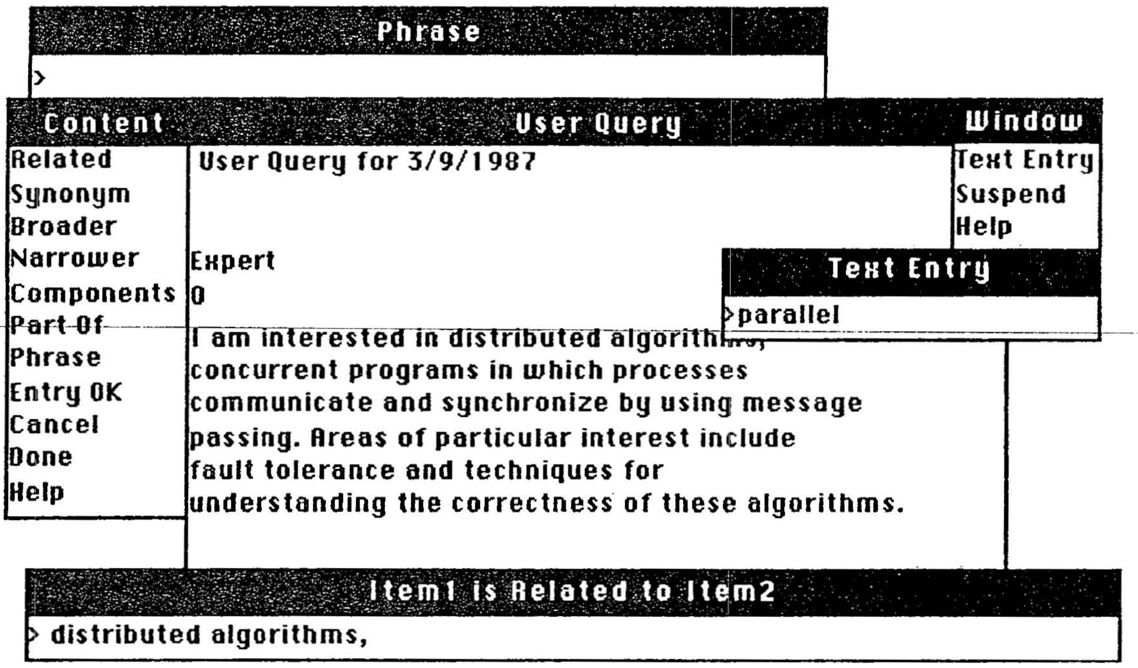
User Query for 3/9/1987

Expert
0

I am interested in distributed algorithms, concurrent programs in which processes communicate and synchronize by using message passing. Areas of particular interest include fault tolerance and techniques for understanding the correctness of these algorithms.

Figure 6.24: Query elaboration with more choices for the expert user.

Besides indicating *phrases*, the user can indicate *broader*, *narrower*, *synonym*, *related*, *component*, and *part-of* relationships. Furthermore, the expert user is not limited to the words in the original query. If he should think of words that might apply that are not in the original query, he can add them by selecting the `Text Entry` selection from the window menu on the right. Figure 6.25 shows the user entering two phrases, `distributed algorithms` and `parallel algorithms`, that are *related* to each other.



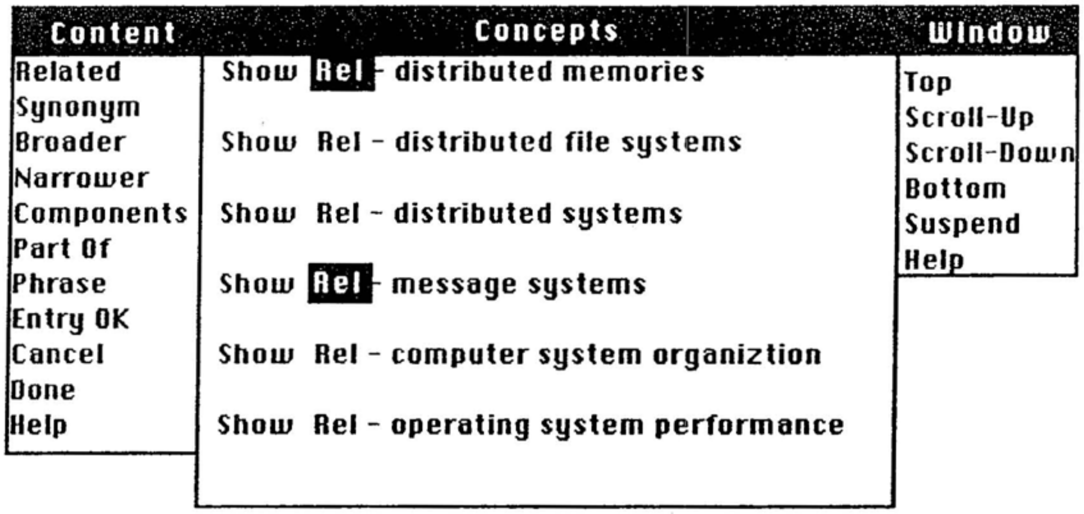
User Query for 3/9/1987

Expert
0
I am interested in distributed algorithms,
concurrent programs in which processes
communicate and synchronize by using message
passing. Areas of particular interest include
fault tolerance and techniques for
understanding the correctness of these algorithms.

> distributed algorithms,

Figure 6.25: Domain knowledge entry by a domain and system expert.

As before, the system shifts to a state where the DKE presents candidate concepts to the user for approval and addition to the request model. Figure 6.26 shows the concepts that are taken from the user's domain knowledge model. The DKE searching the global DK will retrieve the same concepts as shown in figure 6.5 in the first scenario.



Show Rel - distributed memories

Show Rel - distributed file systems

Show Rel - distributed systems

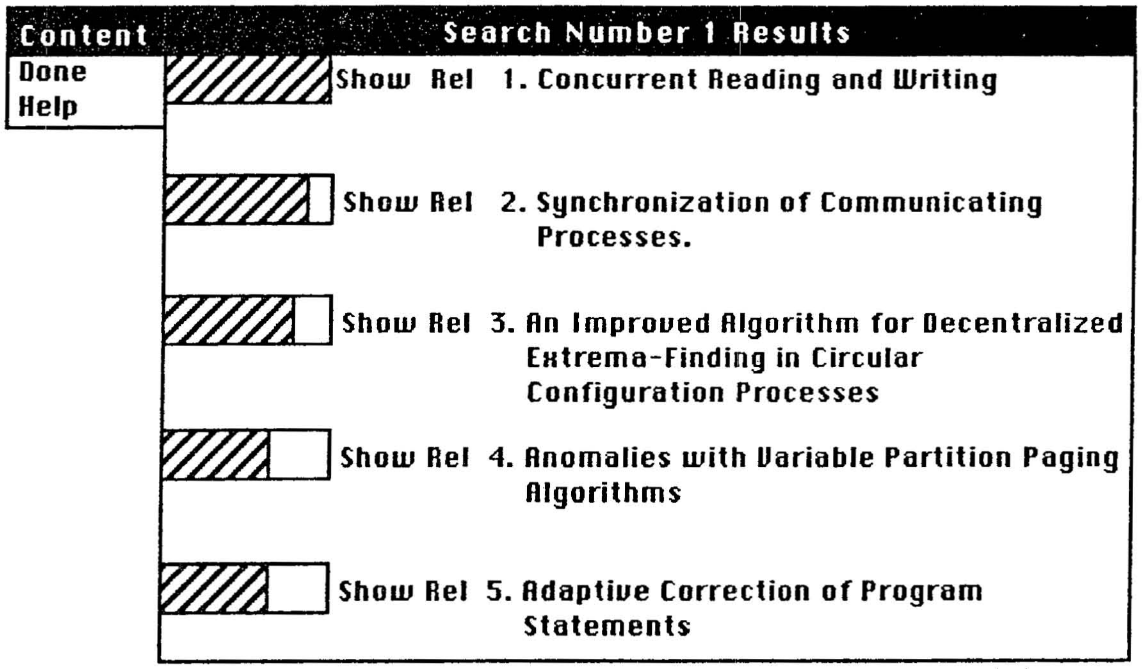
Show Rel - message systems

Show Rel - computer system organiztion

Show Rel - operating system performance

Figure 6.26: Concepts from the user's domain knowledge.

After evaluation of the domain knowledge, the system by means of the search controller initiates two searches. This activity demonstrates the modifiability of the system. Originally, the system only initiated one search (probabilistic) for this set of user stereotypes. It was considered that a more appropriate response would be to initiate two searches (probabilistic and cluster using citation links), since the user is interested in retrieving as many documents as possible, and these two retrieve different sets of documents. All that was required to accomplish this change in behavior was the modification of one rule in the search controller. The results are shown in figure 6.27.


Content — Search Number 1 Results

Done
Help

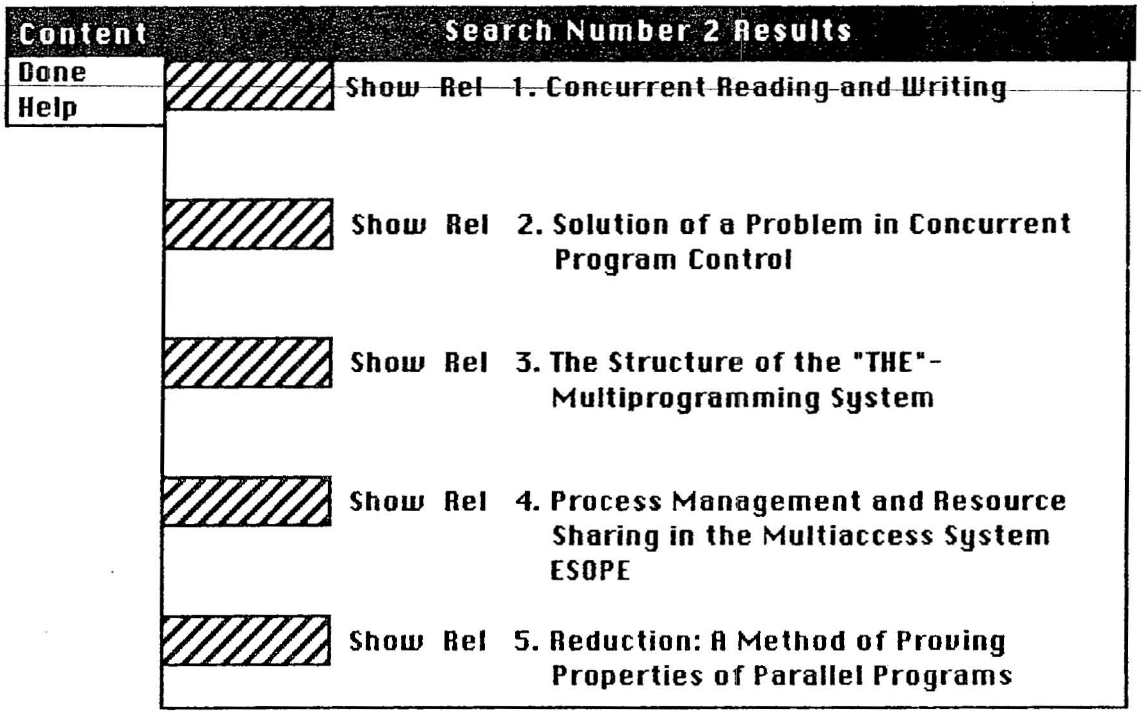
Show Rel 1. Concurrent Reading and Writing

Show Rel 2. Synchronization of Communicating Processes.

Show Rel 3. An Improved Algorithm for Decentralized Extrema-Finding in Circular Configuration Processes

Show Rel 4. Anomalies with Variable Partition Paging Algorithms

Show Rel 5. Adaptive Correction of Program Statements


Content — Search Number 2 Results

Done
Help

Show Rel 1. Concurrent Reading and Writing

Show Rel 2. Solution of a Problem in Concurrent Program Control

Show Rel 3. The Structure of the "THE"-Multiprogramming System

Show Rel 4. Process Management and Resource Sharing in the Multiaccess System ESOPE

Show Rel 5. Reduction: A Method of Proving Properties of Parallel Programs

Figure 6.27: Results of the first two searches (window menus not shown).

Notice that documents 1 through 5 of search one have the same relative relevance, this is because they are members of a single cluster. Document 2 in search one is document

6 in search two. The utility of using the cluster search is clearly demonstrated, since many of the documents in the clusters of search two do not appear in the results of search one.

6.3.4 Scenario Four

The fourth scenario shows the system's operation with the inclusion of the browsing capability. The previous scenarios were, in fact, developed before the browsing capability was added to the system. The work needed to add the BE was to develop and code the heuristics and to extend the IM to manage the browsing maps. Most of the coding effort was done in support of the latter effort.

In this scenario, a different query is used.

```
"I want articles on various tree data structures, I
am especially interested in analysis of adding and
deleting items from them."
```

The user in this case is an expert, so the greatest number of options are made available to him. The results of the first search are the following documents.

```
1.  Faster Retrieval from Context Trees (Corrigendum)
2.  Performance of Height Balanced Trees
3.  A Compiler-Building System Developed by Brooker and
    Morris
4.  Conversational Access to a 2048-Word Machine
5.  Structured Data Structures
6.  Multidimensional Binary Search Trees Used for Asso-
    ciative Searching
7.  A Comparison of Simulation Event List Algorithms
8.  Cellular Arrays for the solution of Graph Problems
9.  COKO III: The Cooper-Koz Chess Program
10. Fen-An Axiomatic Basis for Program Semantics
11. Abstraction Mechanisms in CLU
```

12. Blocks-A New Data Type in SNOBOL4
13. Accommodating Standards and Identification of Programming Languages
14. Optimizing Binary Trees Grown With a Sorting Algorithm
15. A Relational Model of Data for Large Shared Data Banks
16. File Structures Using Hashing Functions
17. Syntax-Directed Documentation for PL 360
18. Algebraic Simplification: A Guide for the Perplexed
19. The SMART Automatic Document Retrieval System - An Illustration
20. A Record and File Partitioning Model

At this point in the session, the experienced user is allowed to browse; a novice user is allowed to browse only after the system has failed to retrieve its expected number of documents in the expected number of searches. Browsing is signaled by the selection of the Browse option from the Content Menu that is associated with the System Messages window (for example, see fig 6.1). Selection of this option tells the system that the user is done with the search results. The interface manager sends the evaluation of the search to the experts for their use. This causes the RMB to update the request model by adding new terms (if any) from the documents that were marked relevant by the user. Evaluation by the SC of its search performance is delayed until the system enters the $SD or $Finish state. If he has made any connections between terms the DKE will establish them in the user's domain model.

If the user selects browsing before he has marked any of the documents of the search as relevant, the search will initially be categorized as a complete failure. But, if the user subsequently marks documents that were in the search as relevant, the search controller will adjust its evaluation of the success of the search.

The system then tells the user to select a document as the initial point from which to begin. The user may choose a document from a variety of sources. He may select a relevant document by getting to the relevant document list, or may redisplay the result of a particular search. Indication of the starting document is made by selecting `Show`.

Commencement of browsing is indicated by a message in the system messages window and by the appearance of the neighborhood and context maps on the screen. Figure 6.28, shows the initial configuration of the neighborhood map. In this instance, the user selected document two, "Performance of Height-Balanced Trees" (document number 2889).

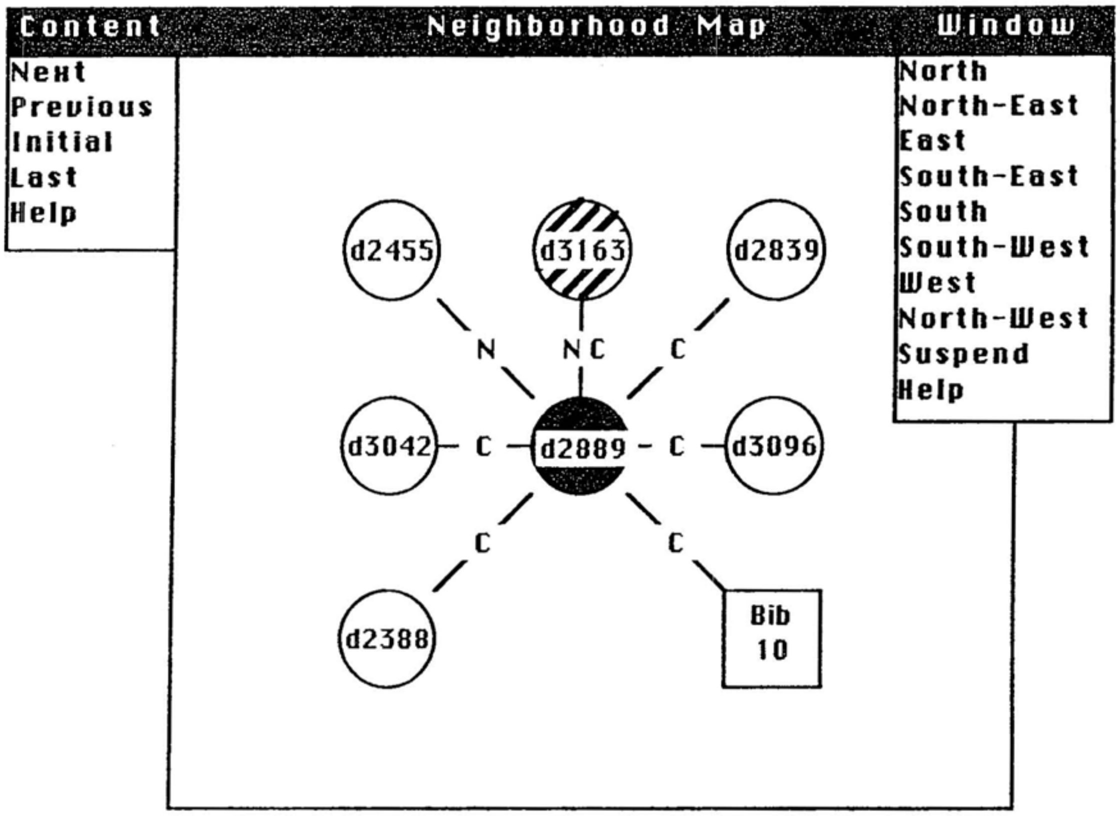


Figure 6.28: Initial display on the Neighborhood Map (the document 2889 and Context window is not shown).

In the center of the map is the document that the user selected as the starting point. The surrounding nodes are documents that the browsing expert has decided are likely to be interesting, based on its heuristics. One node is marked as the most likely to be interesting, which in this case is document 3163, being selected by having a nearest neighbor and a citation link to the current node. The boxes represent lists of documents that cannot be shown but are related to the current document. The relationship is marked on the link and the number of items that the box represents is in the center of it. The menu to the right labeled `Window` allows the user to scroll around the context map. The content menu selections allow the user to trace the path that he has chosen. The selections mean:

`Next` – select the recommended node for viewing.

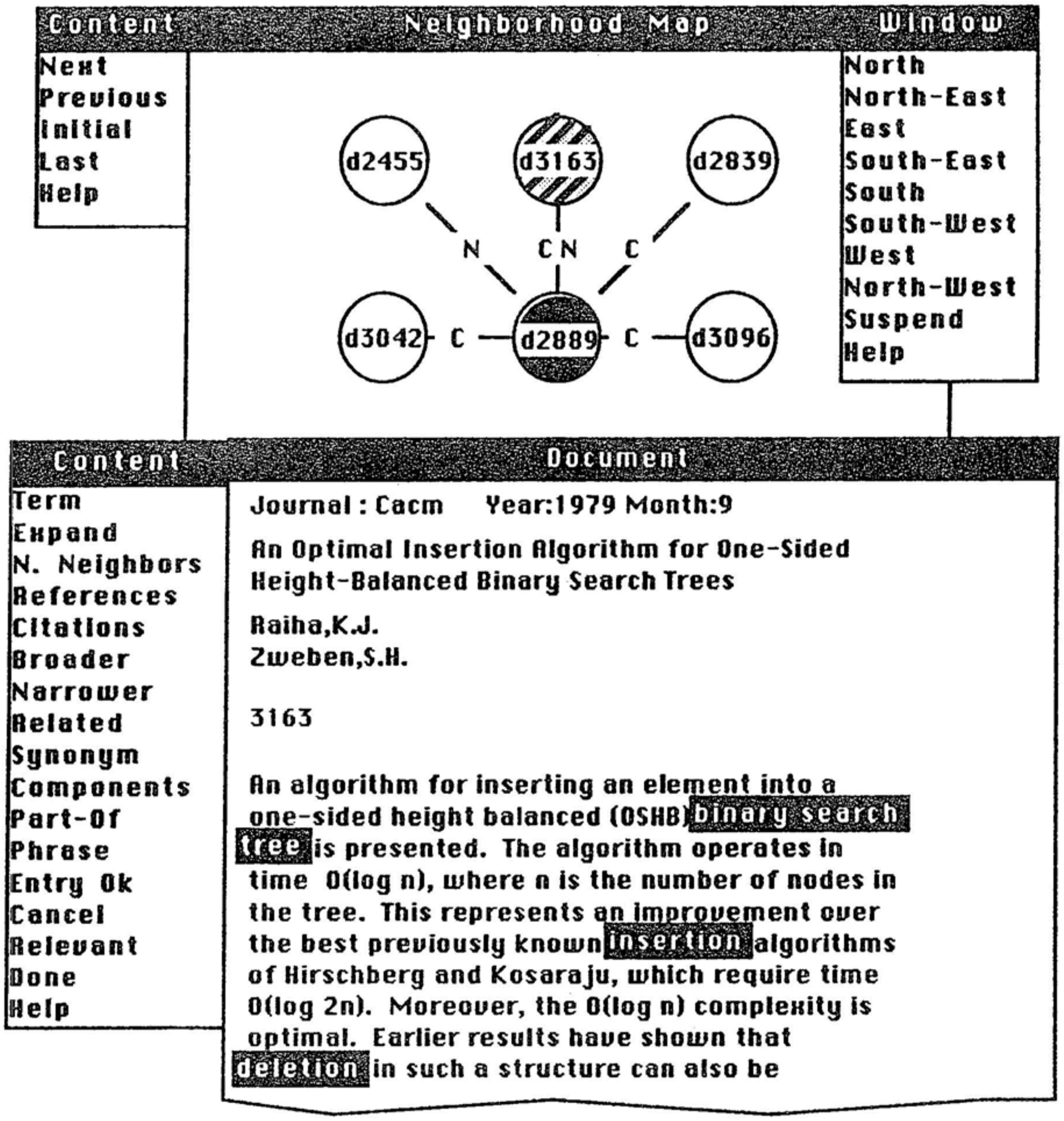
`Previous` – go to the previously viewed node .

`Initial` – go to the first node displayed.

`Last` – go to the last viewed node.

Here, the display shows that there are a number of documents that the system judges as likely to be interesting, and that there are more documents connected than can be shown.

When the user selects one of the nodes with the pointing device, the document that it represents appears on the screen (figure 6.29) and the node is shaded to indicate that it has been visited. When the user indicates that a node is relevant and says that he is done evaluating it, the evaluations are added to the request model and any domain knowledge is added to his domain knowledge model. After a number of documents are judged relevant the system will, depending on the user stereotype, seek to initiate a search. If the user is a system novice, as well as a domain novice, the search controller will automatically be given control to initiate a search. If the user is system expert, the system will indicate, via the system messages window and choice selection window, that it can make a search based on



Journal : Cacm   Year:1979 Month:9

An Optimal Insertion Algorithm for One-Sided
Height-Balanced Binary Search Trees

Raiha,K.J.
Zweben,S.H.

3163

An algorithm for inserting an element into a
one-sided height balanced (OSHB) binary search
tree is presented. The algorithm operates in
time O(log n), where n is the number of nodes in
the tree. This represents an improvement over
the best previously known insertion algorithms
of Hirschberg and Kosaraju, which require time
O(log 2n). Moreover, the O(log n) complexity is
optimal. Earlier results have shown that
deletion in such a structure can also be

Figure 6.29: User selects a recommended node to view its contents, and selects terms that are particularly relevant or interesting. He also selects the concept "AVL," which is not shown.

the new information and requests the user's permission to do so. The number of new documents judged relevant that causes a search to be performed is determined by the stereotypes set by the UMB.